

## Bilan du projet

Gestion des documents de la médiathèque

Livres DVD Revues Parutions des revues Commande de livre Commandes de dvd Commandes de revues

Recherches

Saisir le titre ou la partie d'un titre :  Ou sélectionner le genre :  X

Saisir un numéro de document :  Rechercher Ou sélectionner le public :  X

Ou sélectionner le rayon :  X

Id	Titre	Auteur	Collection	Genre	Public	Rayon
00017	Catastrophes au Brésil	Philippe Masson		Policier	Ados	Jeunesse romans
00007	Dans les coulisses du musée	Kate Atkinson		Roman	Tous publics	Littérature étrangère
00003	Et je danse aussi	Anne-Laure Bondoux		Comédie	Tous publics	Littérature française
00019	Guide Vert - Iles Canaries		Guide Vert	Voyages	Tous publics	Voyages
00020	Guide Vert - Irlande		Guide Vert	Voyages	Tous publics	Voyages
00008	Histoire du juif errant	Jean d'Omesson		Roman	Adultes	Littérature française
00025	L'archipel du danger	Ayrolles - Masbou	De cape et de crocs	Bande dessinée	Adultes	BD Adultes
00004	L'armée furieuse	Fred Vargas	Commissaire Adamsberg	Policier	Adultes	Policiers français étrangers

Informations détaillées

Numéro de document : 00017 Code ISBN :

Image :

Titre : Catastrophes au Brésil

Auteur(e) : Philippe Masson

Collection :

Genre : Policier 10014

Public : Ados 00004

Rayon : Jeunesse romans JN002

Chemin de l'image :

Boutons

Ajouter Modifier Supprimer

## Sommaire

<b>Informations sur le projet</b>	<b>3</b>
<b>Mission 1</b>	<b>4</b>
<b>Mission 2</b>	<b>8</b>
Tâche 1 /	8
Tâche 2 /	17
<b>Mission 4</b>	<b>23</b>
<b>Mission 5</b>	<b>26</b>
Tâche 1 /	26
Tâche 2 /	29
Tâche 3 /	30
<b>Mission 6</b>	<b>31</b>
Tâche 1 /	31
Tâche 2 /	33
Tâche 3 /	35
<b>Mission 7 :</b>	<b>36</b>
<b>Bilan final :</b>	<b>37</b>

# **Informations sur le projet**

## **1. Contexte**

Le projet **mediatekdocument** est une application web conçue pour gérer et mettre à disposition des livres, DVD et revues. Ce site s'adresse à des utilisateurs souhaitant avoir accès à des documents dans un cadre structuré. L'objectif principal est de permettre une consultation efficace des documents disponibles.

## **2. Existant**

**Authentification** : Une authentification est demandée lors de l'arrivée sur le site web.

**Structure du site** : Le site est organisé autour des fonctionnalités de mise à disposition des livres, DVD et revues. Et tout ça sécurisé par une demande d'authentification à l'entrée du site web.

## **3. Langages et technologies utilisées**

**IDE** : Netbeans, Visual Studio

**Framework** : Symfony, C#

**Langages** :

- Backend : PHP
- Frontend : C#

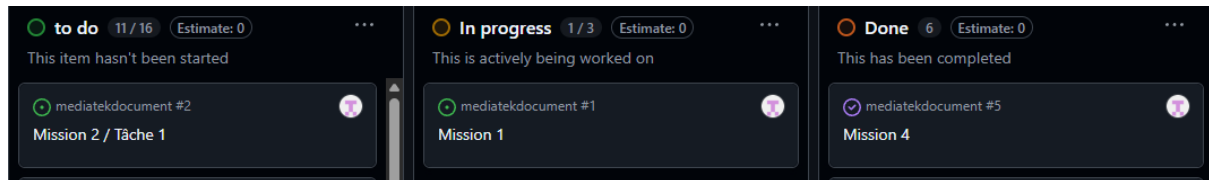
**Base de données** : MySQL

**Serveur** : WAMP

## Mission 1

Pour cette mission, il m'était demandé dans un premier temps de gérer l'ajout d'un livre en local et dans la base de données.

Voici la capture d'écran de mon kanban avec la mission dans la colonne In Progress :



Temps estimé : 8h

Temps réel : 10h

La première tâche que j'ai effectuée a été la création des boutons dans le Form :

Ensuite, j'ai codé la partie concernant le clique sur le bouton ajouter qui devait me ramener vers une nouvelle page Form que j'ai réalisé pour renseigner les différentes informations d'un livre :

```
1 référence | Mathis2111, il y a 18 jours | 1 auteur, 1 modification
private void btnAjouterLivres_Click(object sender, EventArgs e)
{
    FrmAjoutDocument frmAjoutDocument = new FrmAjoutDocument();
    if (frmAjoutDocument.ShowDialog() == DialogResult.OK)
    {
        Livre nouveauLivre = frmAjoutDocument.NouveauLivre;
        if (controller.AjouterLivre(nouveauLivre))
        {
            lesLivres.Add(nouveauLivre);
            RemplirLivresListeComplete();
            MessageBox.Show("Livre ajouté avec succès !", "Succès", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            MessageBox.Show("Erreur lors de l'ajout du livre.", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}
```

Voici l'affichage du Form pour ajouter un livre :

Ensuite j'ai codé le deuxième Form pour y renseigner le contenu des différents TextBox :

```
2 références | Mathis2111, il y a 18 jours | 1 auteur, 1 modification
public Livre NouveauLivre { get; set; }
private readonly FrmMediatekController controller = new FrmMediatekController();
private readonly BindingSource bdgGenres = new BindingSource();
private readonly BindingSource bdgPublics = new BindingSource();
private readonly BindingSource bdgRayons = new BindingSource();

1 référence | Mathis2111, il y a 18 jours | 1 auteur, 1 modification
public FrmAjoutDocument()
{
    InitializeComponent();
    ChargerComboBox();
}

1 référence | Mathis2111, il y a 18 jours | 1 auteur, 1 modification
private void btnAjouter_Click(object sender, EventArgs e)
{
    string idGenre = ((MediaTekDocuments.model.Categorie)cbxGenre.SelectedItem).Id;
    string idPublic = ((MediaTekDocuments.model.Categorie)cbxPublic.SelectedItem).Id;
    string idRayon = ((MediaTekDocuments.model.Categorie)cbxRayon.SelectedItem).Id;

    lblIdGenre.Text = idGenre;
    lblIdPublic.Text = idPublic;
    lblIdRayon.Text = idRayon;

    NouveauLivre = new Livre (
        txtBoxID.Text,
        txtBoxTitre.Text,
        txtBoxImage.Text,
        txtBoxISBN.Text,
        txtBoxAuteur.Text,
        txtBoxCollection.Text,
        lblIdGenre.Text,
        cbxGenre.Text,
        lblIdPublic.Text,
        cbxPublic.Text,
        lblIdRayon.Text,
        cbxRayon.Text
    );

    this.DialogResult = DialogResult.OK;
    this.Close();
}

3 références | 0 modification | 0 auteur, 0 modification
public void RemplirComboCategorie(List<Categorie> lesCategories, BindingSource bdg, ComboBox cbx)
{
    bdg.DataSource = lesCategories;
    cbx.DataSource = bdg;
    if (cbx.Items.Count > 0)
    {
        cbx.SelectedIndex = -1;
    }
}

1 référence | 0 modification | 0 auteur, 0 modification
private void ChargerComboBox()
{
    RemplirComboCategorie(controller.GetAllGenres(), bdgGenres, cbxGenre);
    RemplirComboCategorie(controller.GetAllPublics(), bdgPublics, cbxPublic);
    RemplirComboCategorie(controller.GetAllRayons(), bdgRayons, cbxRayon);
}
```

Tout cela permet d'ajouter un livre localement, mais le but était également de l'ajouter dans la base de données, j'ai donc ajouté une méthode dans le controller qui amenait vers la classe Access, et cette classe m'a permis de me connecter à l'API en appelant un Web Service que j'ai du créé permettant à partir d'un livre donné, de l'ajouter dans différentes tables de la BDD :

```
private function AjouterLivre(?array $champs): int {

    if ($champs === null) {
        $jsonData = file_get_contents('php://input');
        $champs = json_decode($jsonData, true);
    }

    if (empty($champs)) {
        error_log("Erreur : Le tableau \$champs est vide.", 3, "C:\\mon_application.log");
        return 0;
    }

    try {
        error_log("Début de la transaction", 3, "C:\\mon_application.log");
        $this->conn->updateBDD("START TRANSACTION");

        $champsDocument['id'] = $champs['Id'];
        $champsDocument['titre'] = $champs['Titre'];
        $champsDocument['image'] = $champs['Image'];
        $champsDocument['idGenre'] = $champs['IdGenre'];
        $champsDocument['idPublic'] = $champs['IdPublic'];
        $champsDocument['idRayon'] = $champs['IdRayon'];
        $result1 = $this->insertOneTupleOneTable("document", $champsDocument);

        $champsLivres_dvd['id'] = $champs['Id'];
        $result2 = $this->insertOneTupleOneTable("livres_dvd", $champsLivres_dvd);

        $champsLivre['id'] = $champs['Id'];
        $champsLivre['auteur'] = $champs['Auteur'];
        $champsLivre['isbn'] = $champs['Isbn'];
        $champsLivre['collection'] = $champs['Collection'];
        $result3 = $this->insertOneTupleOneTable("livre", $champsLivre);

        $this->conn->updateBDD("COMMIT");

        return $result1 + $result2 + $result3;
    } catch (Exception $e) {

        $this->conn->updateBDD("ROLLBACK");
        return 0;
    }
}
```

Le résultat de l'action devait permettre d'ajouter un livre en local et dans la BDD.

La deuxième tâche était de pouvoir gérer la modification d'un livre, j'ai donc pris appui sur le Form existant pour rendre les textBox modifiables une fois le bouton modifier pressé, puis un bouton valider apparaît pour pouvoir valider une fois les modifications ajoutées. Quand le bouton valider est pressé, la modification se fait en local et également dans la base de données grâce à un Web service qui permet de modifier un champs dans la base de données en fonction de son ID :

```
private function ModifierLivre(?array $champs): int {
    if ($champs === null) {
        $jsonData = file_get_contents('php://input');
        $champs = json_decode($jsonData, true);
    }

    if (empty($champs)) {
        error_log("Erreur : Le tableau \$champs est vide.", 3, "C:\\mon_application.log");
        return 0;
    }

    try {
        error_log("Début de la transaction", 3, "C:\\mon_application.log");
        var_dump($champs);
        $this->conn->updateBDD("START TRANSACTION");

        $champsId['id'] = $champs['Id'];
        $champId = implode("", $champsId);
        $champsDocument['titre'] = $champs['Titre'];
        $champsDocument['image'] = $champs['Image'];
        $champsDocument['idGenre'] = $champs['IdGenre'];
        $champsDocument['idPublic'] = $champs['IdPublic'];
        $champsDocument['idRayon'] = $champs['IdRayon'];
        $result1 = $this->updateOneTupleOneTable("document", $champId, $champsDocument);

        $champsLivre['auteur'] = $champs['Auteur'];
        $champsLivre['isbn'] = $champs['Isbn'];
        $champsLivre['collection'] = $champs['Collection'];
        $result3 = $this->updateOneTupleOneTable("livre", $champId, $champsLivre);

        $this->conn->updateBDD("COMMIT");

        return $result1 + $result3;
    } catch (Exception $e) {
        $this->conn->updateBDD("ROLLBACK");
        return 0;
    }
}
```

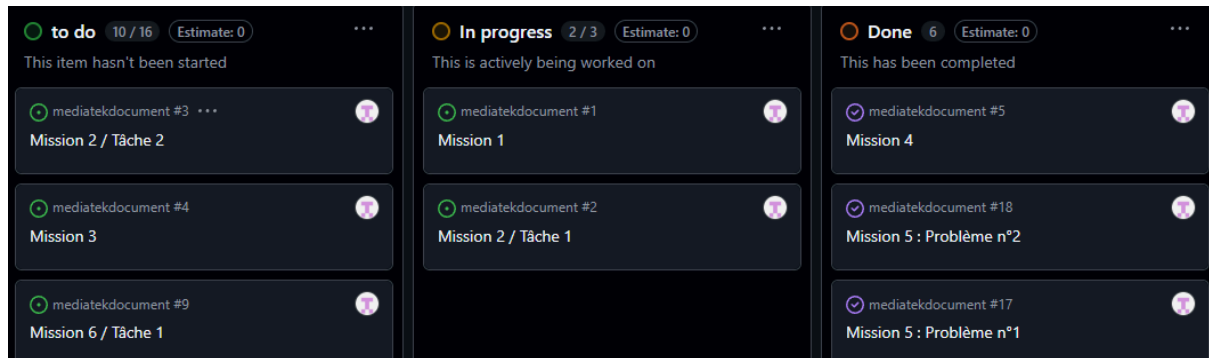
En ce qui concerne la suppression des livres, dvd et revues ainsi que les boutons pour les DVD et revues, je n'ai pas eu le temps de les réaliser et ils seront codés et disponibles pour les épreuves.

## Mission 2

### Tâche 1 /

Pour cette tâche il m'était demandé d'ajouter un onglet pour la gestion des commandes pour les livres et les DVD.

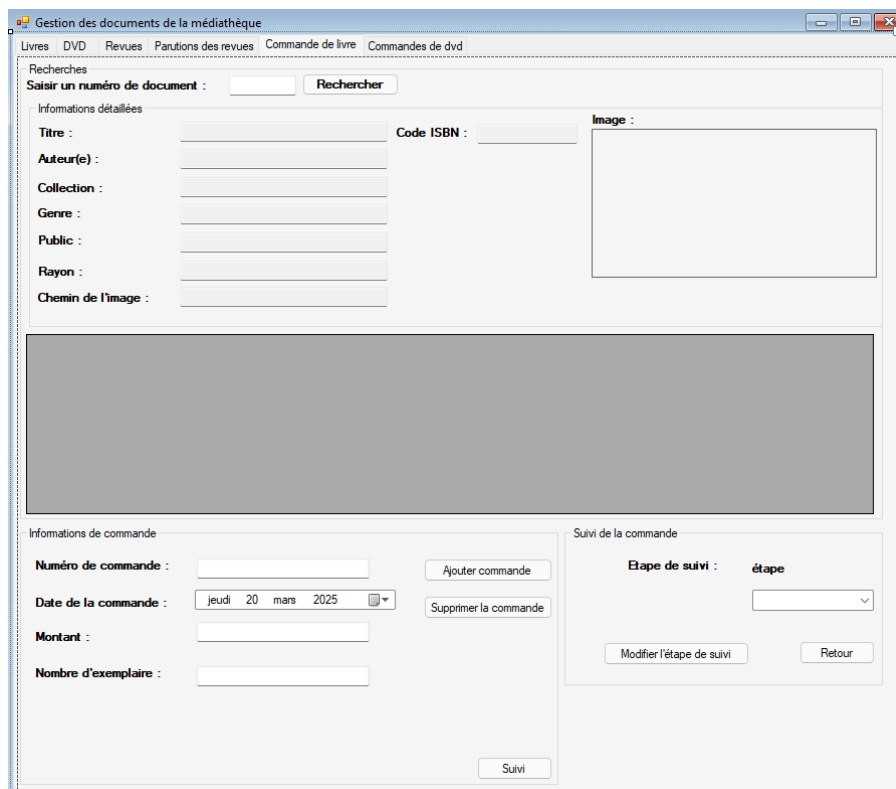
Voici la capture d'écran de mon kanban :



Temps estimé : 8h

Temps réel : 16h

La première tâche que j'ai du réaliser a été la création du nouvel onglet avec les différents visuels :





Une fois l'onglet créé et modélisé, j'ai commencé par créer les classes Listes CommandeDocument et Suivi, qui contiendraient les informations des différentes tables. Ensuite j'ai créé une instance de ces classes pour pouvoir les utiliser. Ensuite, j'ai créé une méthode pour qu'au moment de l'ouverture de l'onglet, certains groupBox soient inutilisables, et que certaines variables puissent contenir les livres et les suivis. Ensuite j'ai codé les différentes réactions des groupBox en fonction des manipulations de certains boutons et de certains groupBox :

```

/// <summary>
/// Masque la groupBox des suivis
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | 0 modification | 0 auteur, 0 modification
private void gbxInfosCommandeLivre_Enter(object sender, EventArgs e)
{
    gbxEtapeSuivi.Enabled = false;
}

1 référence | 0 modification | 0 auteur, 0 modification
private void btnInfosCommandeLivreAnnuler_Click(object sender, EventArgs e)
{
    gbxEtapeSuivi.Enabled = true;
    gbxInfosCommandeLivre.Enabled = false;
}

```

Ensuite, j'ai développé la méthode "RemplirLivresListe2(List lesCommandesDocument)" qui se charge de remplir la grille avec les données des commandes de document fournies en paramètre. Enfin, j'ai implémenté la méthode "AfficheReceptionCommandesLivre()" pour mettre à jour l'affichage des commandes lorsque l'utilisateur saisit un numéro de livre dans le champ de recherche. Cette méthode récupère les commandes correspondant au livre saisi et met à jour la datagrid en appelant la méthode "RemplirLivresListe2(lesCommandesDocument)" avec la liste des commandes récupérées :

```

/// <summary>
/// Mise à jour de la liste des commandes de livre
/// </summary>
5 références | 0 modification | 0 auteur, 0 modification
private void AfficheReceptionCommandesLivre()
{
    string idDocument = txbLivresNumRecherche2.Text;
    lesCommandesDocument = controller.GetCommandesDocument(idDocument);
    RemplirLivresListe2(lesCommandesDocument);
}

```

Lorsque l'utilisateur clique sur le bouton "rechercher", la procédure événementielle "btnLivresNumRecherche2\_Click" est déclenchée pour afficher les commandes liées au livre recherché dans le champ "txbLivresNumRecherche2.Text". Cette méthode commence par vérifier si un numéro de document a été saisi dans le champ de recherche en utilisant la condition "if (!txbLivresNumRecherche2.Text.Equals(""))". La saisie d'un numéro de document est rendue obligatoire pour effectuer la recherche. Ensuite, la méthode "Find()" est utilisée pour retrouver le livre correspondant à l'ID saisi dans le champ de recherche dans la liste "lesLivres". Si le livre existe, la méthode "AfficheReceptionCommandesLivre()" est appelée pour afficher la liste des commandes associées à ce livre. Enfin, la zone d'affichage des informations de commande concernant ce livre est activée, permettant à l'utilisateur d'effectuer des ajouts, des modifications ou des suppressions de commande :

```
private void btnLivresNumRecherche2_Click(object sender, EventArgs e)
{
    if (!txbLivresNumRecherche2.Text.Equals(""))
    {
        Livre livre = lesLivres.Find(x => x.Id.Equals(txbLivresNumRecherche2.Text));
        if (livre != null)
        {
            AfficheReceptionCommandesLivre();
            gbxInfosCommandeLivre.Enabled = true;
            AfficheReceptionCommandesLivreInfos(livre);
        }
        else
        {
            MessageBox.Show("numéro introuvable");
        }
    }
    else
    {
        MessageBox.Show("Le numéro de document est obligatoire", "Information");
    }
}
```

Une fois les commandes du livre affichées avec succès, j'ai ajouté une fonction pour afficher les détails du livre recherché, remplissant les champs appropriés avec les informations telles que le titre et les auteurs. Cette fonction est déclenchée lors du clic sur le bouton de recherche :

```
/// <summary>
/// Affichage des informations du livre
/// </summary>
/// <param name="livre">Le livre</param>
/// <reference> 0 modification | 0 auteur, 0 modification
private void AfficheReceptionCommandesLivreInfos(Livre livre)
{
    txbCommandeLivresTitre.Text = livre.Titre;
    txbCommandeLivresAuteur.Text = livre.Auteur;
    txbCommandeLivresIsbn.Text = livre.Isbn;
    txbCommandeLivresCollection.Text = livre.Collection;
    txbCommandeLivresGenre.Text = livre.Genre;
    txbCommandeLivresPublic.Text = livre.Public;
    txbCommandeLivresRayon.Text = livre.Rayon;
    txbCommandeLivresCheminImage.Text = livre.Image;
    string image = livre.Image;
    try
    {
        pictureBox1.Image = Image.FromFile(image);
    }
    catch
    {
        pictureBox1.Image = null;
    }
    AfficheReceptionCommandesLivre();
}
```

Puis, pour récupérer l'id de l'étape de suivi d'une commande, j'ajoute une fonction "GetIdSuivi(string libelle)" qui récupère la liste des objets "Suivi" et trouve l'id correspondant au libellé donné :

```
private string GetIdSuivi(string libelle)
{
    List<Suivi> lesSuivisCommande = controller.GetAllSuivis();
    foreach (Suivi unSuivi in lesSuivisCommande)
    {
        if (unSuivi.Libelle == libelle)
        {
            return unSuivi.Id;
        }
    }
    return null;
}
```

Ensuite, j'ai créé la procédure "dgvCommandesLivre\_RowEnter" pour afficher les informations de la commande sélectionnée dans la datagrid dans la groupBox dédiée. En utilisant des variables pour stocker les données de la ligne sélectionnée, je les affecte ensuite aux champs correspondants. Pour garantir une meilleure expérience utilisateur, j'ajoute une condition pour empêcher la modification de l'étape de suivi si la commande est déjà réglée, en appelant la méthode "GetIdSuivi" pour obtenir l'ID correspondant au libellé "réglée" :

```
private void dgvCommandesLivre_RowEnter(object sender, DataGridViewCellEventArgs e)
{
    DataGridViewRow row = dgvCommandesLivre.Rows[e.RowIndex];

    string id = row.Cells["Id"].Value.ToString();
    DateTime dateCommande = (DateTime)row.Cells["dateCommande"].Value;
    double montant = double.Parse(row.Cells["Montant"].Value.ToString());
    int nbExemplaire = int.Parse(row.Cells["NbExemplaire"].Value.ToString());
    string libelle = row.Cells["Libelle"].Value.ToString();

    txtBoxNumCommande.Text = id;
    txtBoxNombreExemplaire.Text = nbExemplaire.ToString();
    txtBoxMontant.Text = montant.ToString();
    dtpCommande.Value = dateCommande;
    lblEtapeSuivi.Text = libelle;

    if (GetIdSuivi(libelle) == [?] (variable locale) string libelle)
    {
        cbxCommandeLivresLibelleSuivi.Enabled = false;
        btnReceptionCommandeLivresModifierSuivi.Enabled = false;
    }
    else
    {
        cbxCommandeLivresLibelleSuivi.Enabled = true;
        btnReceptionCommandeLivresModifierSuivi.Enabled = true;
    }
}
```

Puis, j'ai construis la procédure événementielle

"dgvLivresListe2\_ColumnHeaderMouseClick" pour permettre le tri des colonnes dans l'ordre inverse de la chronologie lors du clic sur le haut des colonnes :

```
private void dgvLivresListe2_ColumnHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)
{
    string titreColonne = dgvCommandesLivre.Columns[e.ColumnIndex].HeaderText;
    List<CommandeDocument> sortedList = new List<CommandeDocument>();
    switch (titreColonne)
    {
        case "Date de commande":
            sortedList = lesCommandesDocument.OrderBy(o => o.DateCommande).Reverse().ToList();
            break;
        case "Montant":
            sortedList = lesCommandesDocument.OrderBy(o => o.Montant).ToList();
            break;
        case "Nombre d'exemplaires":
            sortedList = lesCommandesDocument.OrderBy(o => o.NbExemplaire).ToList();
            break;
        case "Suivi":
            sortedList = lesCommandesDocument.OrderBy(o => o.Libelle).ToList();
            break;
    }
}
```

Pour ajouter une nouvelle commande de livre, j'ai créé la procédure

"btnReceptionCommandeLivresValider\_Click". Cette procédure vérifie que les champs obligatoires sont remplis avant d'insérer une nouvelle commande de livre dans la base de données. Ensuite, elle appelle les méthodes "CreerCommande" et "CreerCommandeDocument" pour enregistrer les données et met à jour l'affichage de la grille des commandes de livres :

```
private void btnReceptionCommandeLivresValider_Click(object sender, EventArgs e)
{
    if (!txtBoxNumCommande.Text.Equals("") && !txtBoxNombreExemplaire.Text.Equals("") && !txtBoxMontant.Text.Equals(""))
    {
        string id = txtBoxNumCommande.Text;
        int nbExemplaire = int.Parse(txtBoxNombreExemplaire.Text);
        double montant = double.Parse(txtBoxMontant.Text);
        DateTime dateCommande = dtpCommande.Value;
        string idLivreDvd = txbLivresNumRecherche2.Text;
        string idSuivi = lesSuivis[0].Id;

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

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

        if (idCommandeLivreNonExistante)
        {
            if (controller.CreerCommande(commande))
            {
                controller.CreerCommandeDocument(id, nbExemplaire, idLivreDvd, idSuivi);
                MessageBox.Show("La commande " + id + " a bien été enregistrée.", "Information");
                AfficheReceptionCommandesLivre();
            }
        }
        else
        {
            MessageBox.Show("Le numéro de la commande existe déjà, veuillez saisir un nouveau numéro.", "Erreur");
        }
    }
    else
    {
        MessageBox.Show("Tous les champs sont obligatoires.", "Information");
    }
}
```

Dans la fonctionnalité "Modifier", j'ai mis en place une procédure sur le clic du bouton "Modifier", "btnReceptionCommandeLivresModifierSuivi\_Click", qui permet de modifier l'étape de suivi de la commande. Les règles de modification sont appliquées en fonction de l'étape actuelle de la commande. Les données sont récupérées depuis les champs de saisie et la sélection de la comboBox des étapes de suivi. Un nouvel objet "CommandeDocument" est créé avec ces données, et une vérification est effectuée pour s'assurer que la modification est autorisée. Ensuite, la méthode "ModifierSuiviCommandeDocument" du controller est appelée pour effectuer la modification, et l'affichage de la grille des données est mis à jour :

```
private void btnReceptionCommandeLivresModifierSuivi_Click(object sender, EventArgs e)
{
    string id = txtBoxNumCommande.Text;
    int nbExemplaire = int.Parse(txtBoxNombreExemplaire.Text);
    double montant = double.Parse(txtBoxMontant.Text);
    DateTime dateCommande = dtpCommande.Value;
    string idLivreDvd = txtLivresNumRecherche2.Text;
    string idSuivi = GetIdSuivi(cbxCommandeLivresLibelleSuivi.Text);

    try
    {
        string libelle = cbxCommandeLivresLibelleSuivi.SelectedItem.ToString();

        CommandeDocument commandeDocument = new CommandeDocument(id, dateCommande, montant, nbExemplaire, idLivreDvd, idSuivi, libelle);
        if (MessageBox.Show("Voulez-vous modifier le suivi de la commande " + commandeDocument.Id + " en " + libelle + " ?", "Confirmation de modification"))
        {
            controller.ModifierSuiviCommandeDocument(commandeDocument.Id, commandeDocument.IdSuivi);
            MessageBox.Show("L'étape de suivi de la commande " + id + " a bien été modifiée.", "Information");
            AfficheReceptionCommandesLivre();
        }
    }
    catch (NullReferenceException)
    {
        MessageBox.Show("La nouvelle étape de suivi de la commande doit être sélectionnée.", "Information");
    }
}
```

Par la suite, j'ai créé la procédure "btnSupprimerCommandeLivres\_Click", qui supprime la commande de livre sélectionnée dans la datagrid si elle n'est pas encore livrée. Une condition "if" vérifie si une ligne est sélectionnée avant de supprimer, et une autre condition permet la suppression seulement si le libellé est "en cours" ou "relancée" :

```
private void btnSupprimerCommandeLivres_Click(object sender, EventArgs e)
{
    if (dgvCommandesLivre.SelectedRows.Count > 0)
    {
        CommandeDocument commandeDocument = (CommandeDocument)bdgCommandesLivre.List[bdgCommandesLivre.Position];
        if (commandeDocument.Libelle == "en cours" || commandeDocument.Libelle == "relancée")
        {
            if (MessageBox.Show("Voulez-vous vraiment supprimer la commande " + commandeDocument.Id + " ?", "Confirmation de suppression"))
            {
                controller.SupprimerCommandeDocument(commandeDocument);
                AfficheReceptionCommandesLivre();
            }
        }
        else
        {
            MessageBox.Show("La commande sélectionnée a été livrée, elle ne peut pas être supprimée.", "Information");
        }
    }
    else
    {
        MessageBox.Show("Une ligne doit être sélectionnée.", "Information");
    }
}
```

Certaines de ces méthodes appellent le controller pour avoir des données sur certaines tables ou pour effectuer certaines actions, pour cela, la première méthode que j'ai dû créer a été "GetCommandesDocument(string idDocument)" cette méthode devait retourner

l'ensemble des commande pour un ID de document donné, j'ai donc créer la méthode dans access pour appeler le Web Service créé dans mon API :

```
public List<CommandeDocument> GetCommandesDocument(string idDocument)
{
    string jsonIdDocument = convertToJson("Id", idDocument);
    List<CommandeDocument> lesCommandesDocument = TraitementRecup<CommandeDocument>(GET, "commandedocument/" + jsonIdDocument, n
    return lesCommandesDocument;
}
```

Voici la capture d'écran du Web Service :

```
/**
 * récupération de toutes les commandes d'un document
 * @param string $champs id du document concerné
 * @return lignes de la requete
 */
public function selectAllCommandesDocument(?array $champs) : ?array {
    $sql = "SELECT cd.*, c.dateCommande, c.montant, s.libelle
    FROM commandedocument cd
    JOIN commande c ON cd.id = c.id
    JOIN suivi s ON cd.idSuivi = s.id
    WHERE cd.idLivreDvd = :idLivreDvd";
    $champNecessaire['idLivreDvd'] = $champs['Id'];

    return $this->conn->queryBDD($sql, $champNecessaire);
}
```

Par la suite j'ai du faire de même pour obtenir tous les suivis, ainsi que toutes les commandes cette fois pour un ID donné :

```
private function getAllSuivis() : ?array{
    $requete = "SELECT s.* FROM suivi s";
    return $this->conn->queryBDD($requete);
}

private function selectAllCommande(?array $champs) : ?array{
    $champNecessaire['id'] = $champs['Id'];
    $requete = "SELECT * FROM commande WHERE id = :id";
    return $this->conn->queryBDD($requete, $champNecessaire);
}
```

Ensuite, j'ai dû écrire la méthode pour ajouter une commande dans la table "commande" à partir des informations fournies, puis j'ai fait de même pour ajouter une commande dans la table "commandedocument" :

```

private function AjouterCommande(?array $champs): int {
    if ($champs === null) {
        $jsonData = file_get_contents('php://input');
        $champs = json_decode($jsonData, true);
    }

    if (empty($champs)) {
        return 0;
    }

    try {
        $this->conn->updateBDD("START TRANSACTION");
        $champsCommande['id'] = $champs['Id'];
        $champsCommande['dateCommande'] = $champs['DateCommande'];
        $champsCommande['montant'] = $champs['Montant'];
        $result1 = $this->insertOneTupleOneTable("commande", $champsCommande);

        $this->conn->updateBDD("COMMIT");

        return $result1;
    } catch (Exception $e) {

        $this->conn->updateBDD("ROLLBACK");
        return 0;
    }
}

private function AjouterCommandeDocument(?array $champs): int {
    if ($champs === null) {
        $jsonData = file_get_contents('php://input');
        $champs = json_decode($jsonData, true);
    }

    if (empty($champs)) {
        return 0;
    }

    try {
        $this->conn->updateBDD("START TRANSACTION");
        $champsCommandeDocument['id'] = $champs['Id'];
        $champsCommandeDocument['nbExemplaire'] = $champs['NbExemplaire'];
        $champsCommandeDocument['idLivreDvd'] = $champs['IdLivreDvd'];
        $champsCommandeDocument['idSuiwi'] = $champs['IdSuiwi'];
        $result1 = $this->insertOneTupleOneTable("commandedocument", $champsCommandeDocument);

        $this->conn->updateBDD("COMMIT");

        return $result1;
    } catch (Exception $e) {

        $this->conn->updateBDD("ROLLBACK");
        return 0;
    }
}

```

Ensuite, j'ai dû écrire le Web Service pour modifier l'état d'une commande dans la table "commandedocument" :



```

private function ModifierEtatCommande(?array $champs): int {
    if ($champs === null) {
        $jsonData = file_get_contents('php://input');
        $champs = json_decode($jsonData, true);
    }

    if (empty($champs)) {
        return 0;
    }

    try {
        $this->conn->updateBDD("START TRANSACTION");
        $champsId['id'] = $champs['Id'];
        $champId = implode("", $champsId);
        $champsEtatCommande['idSuiivi'] = $champs['IdSuiivi'];

        $result1 = $this->updateOneTupleOneTable("commandedocument", $champId, $champsEtatCommande);

        $this->conn->updateBDD("COMMIT");

        return $result1;
    } catch (Exception $e) {

        $this->conn->updateBDD("ROLLBACK");
        return 0;
    }
}

```

Pour finir, le dernier Web Service devait supprimer une commande en fonctions des informations fournies, cette suppression devait se faire dans les tables “commande” et “commandedocument” :

```

private function SupprimerCommandeDocument(?array $champs): int {
    if ($champs === null) {
        $jsonData = file_get_contents('php://input');
        $champs = json_decode($jsonData, true);
    }

    if (empty($champs)) {
        return 0;
    }

    try {
        $this->conn->updateBDD("START TRANSACTION");

        $champsCommandeDocument['id'] = $champs['Id'];
        $champsCommandeDocument['nbExemplaire'] = $champs['NbExemplaire'];
        $champsCommandeDocument['idLivreDvd'] = $champs['IdLivreDvd'];
        $champsCommandeDocument['idSuiivi'] = $champs['IdSuiivi'];
        $result1 = $this->deleteTuplesOneTable("commandedocument", $champsCommandeDocument);

        $champsCommande['dateCommande'] = $champs['DateCommande'];
        $champsCommande['id'] = $champs['Id'];
        $champsCommande['montant'] = $champs['Montant'];
        $result2 = $this->deleteTuplesOneTable("Commande", $champsCommande);

        $this->conn->updateBDD("COMMIT");

        return $result1 + $result2;
    } catch (Exception $e) {

        $this->conn->updateBDD("ROLLBACK");
        return 0;
    }
}

```

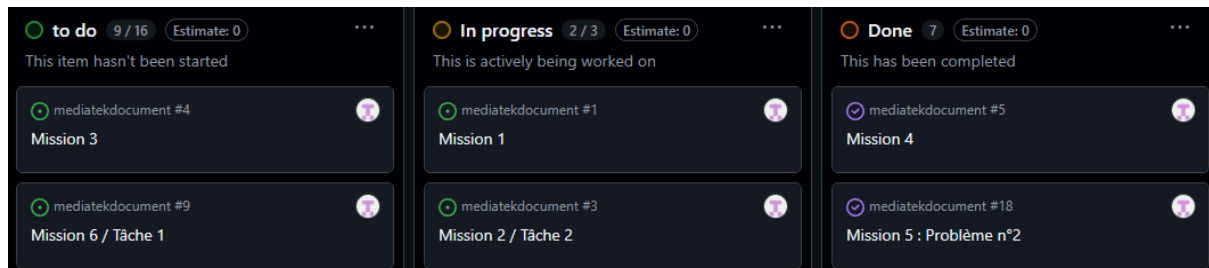
En ce qui concerne les DVD, j’ai repris exactement les mêmes méthodes mais en modifiant le visuel de l’onglet.



## Tâche 2 /

Pour cette deuxième tâche, il m'était demandé de gérer les commandes de revues, il devait être possible d'ajouter ou de supprimer une commande.

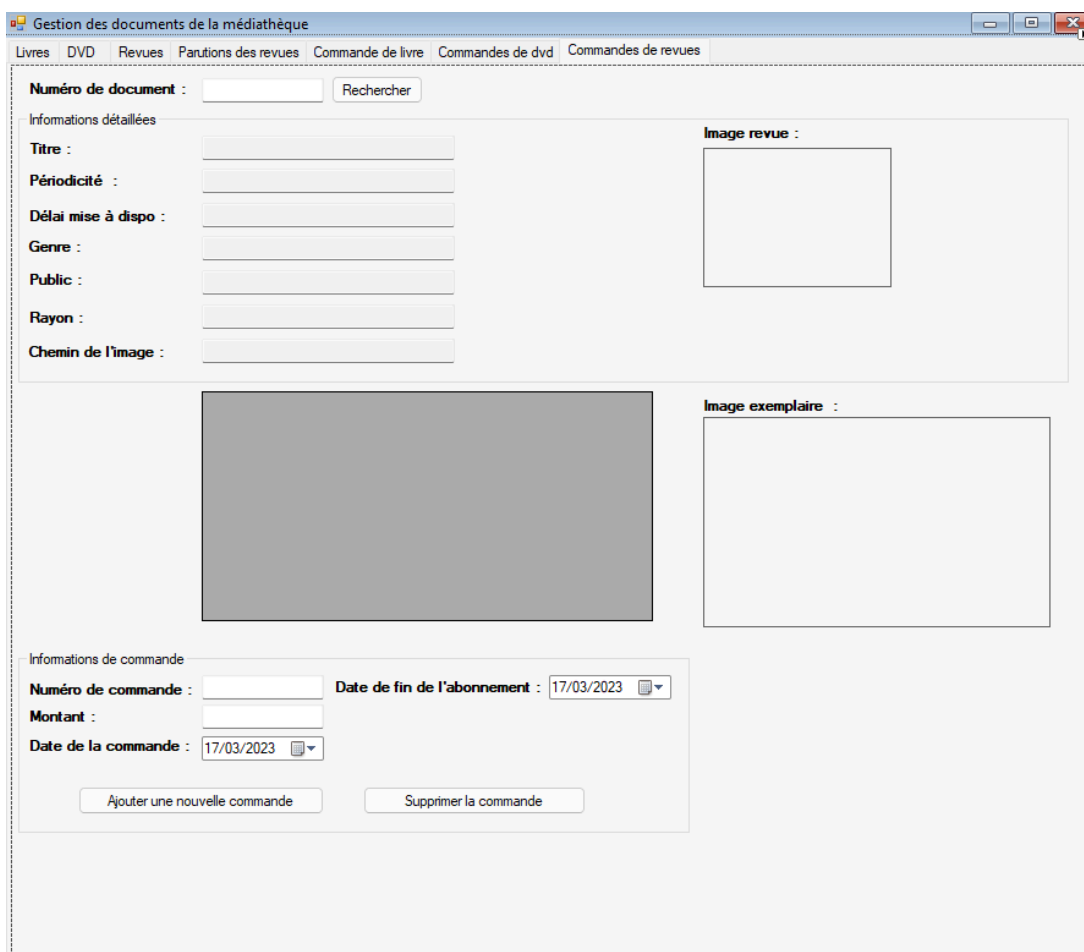
Voici la capture d'écran de mon kanban :



Temps estimé : 4h

Temps réel : 5h

Pour commencer, j'ai ajouté un nouvel onglet en reprenant l'affichage des parutions de revues, puis j'ai ajouté un GroupBox "Information de commande", qui a pour but de gérer les commandes des revues. J'ai donc remplacé les informations des revues dans le DataGridView par les informations de la commande en fonction de la revue recherchée par son numéro d'identification :



Puis j'ai repris la façon de gérer les commandes pour un livre, en l'adaptant pour une revue. J'ai donc réécrit les méthodes, en changeant leur contenu. Pour la méthode "btnRechercheDocRevue\_Click", elle permet au moment de son appel de vérifier si le contenu du textBox qui contient le numéro de la revue n'est pas vide, puis cherche la revue dont le numéro d'identification est égal au contenu du textBox. Ensuite, si la revue est trouvée, la méthode pour afficher le contenu de la revue dans les textBox en dessous du numéro recherché est appelée, le groupBox affichant les informations de la commande devient utilisable et les informations de la commande s'affiche :

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

Par la suite, la méthode "dgvAbonnementRevue\_RowEnter" va permettre de modifier l'affichage des informations d'une commande en fonction de la revue sélectionnée dans le DataGridView :

```
private void dgvAbonnementsRevue_RowEnter(object sender, DataGridViewCellEventArgs e)
{
    DataGridViewRow row = dgvAbonnementsRevue.Rows[e.RowIndex];
    string id = row.Cells["Id"].Value.ToString();
    DateTime dateCommande = (DateTime)row.Cells["dateCommande"].Value;
    double montant = double.Parse(row.Cells["Montant"].Value.ToString());
    DateTime dateFinAbonnement = (DateTime)row.Cells["DateFinAbonnement"].Value;

    txbCommandeRevueNumero.Text = id;
    txbCommandeRevueMontant.Text = montant.ToString();
    dtpCommandeRevue.Value = dateCommande;
    dtpCommandeRevueAbonnementFin.Value = dateFinAbonnement;
}
```

La méthode "dgvAbonnementsRevue\_ColumnHeaderMouseClick" va permettre d'afficher les colonnes et de faire un tri sur les colonnes en fonction de l'en-tête cliquée :

```
private void dgvAbonnementsRevue_ColumnHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)
{
    string titreColonne = dgvAbonnementsRevue.Columns[e.ColumnIndex].HeaderText;
    List<Abonnement> sortedList = new List<Abonnement>();
    switch (titreColonne)
    {
        case "Date de commande":
            sortedList = lesAbonnementsRevue.OrderBy(o => o.DateCommande).Reverse().ToList();
            break;
        case "Montant":
            sortedList = lesAbonnementsRevue.OrderBy(o => o.Montant).ToList();
            break;
        case "Date de fin d'abonnement":
            sortedList = lesAbonnementsRevue.OrderBy(o => o.DateFinAbonnement).Reverse().ToList();
            break;
    }
    RemplirAbonnementsRevueListe(sortedList);
}
```

Ensuite, la méthode “btnReceptionCommandeRevueValider\_Click” va permettre au moment du clique sur le bouton ajouter une nouvelle commande, de remplir différentes variables à l’aide du contenu des textBox sur les informations de la commande concernant une revue, de passer ces variable dans un nouvel objet de type Commande, puis d’appeler le Web Service correspondant pour vérifier parmi les commande, si l’une d’elles contient l’id récupéré. Si aucune n’a le même id, on va créer la commande dans les tables commande et commandedocument grâce à deux Web Service, si la commande existe, on va afficher un message :

```
private void btnReceptionCommandeRevueValider_Click(object sender, EventArgs e)
{
    if (!txtbCommandeRevueNumero.Text.Equals("") && !txtbCommandeRevueMontant.Text.Equals(""))
    {
        string idRevue = txtbCommandesRevueNumRecherche.Text;
        string id = txtbCommandeRevueNumero.Text;
        double montant = double.Parse(txtbCommandeRevueMontant.Text);
        DateTime dateCommande = dtpCommandeRevue.Value;
        DateTime dateFinAbonnement = dtpCommandeRevueAbonnementFin.Value;

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

        var idCommandeRevueExistante = controller.GetCommandes(id);
        var idCommandeRevueNonExistante = !idCommandeRevueExistante.Any();

        if (idCommandeRevueNonExistante)
        {
            if (controller.CreerCommande(commande))
            {
                controller.CreerAbonnementRevue(id, dateFinAbonnement, idRevue);
                MessageBox.Show("La commande " + id + " a bien été enregistrée.", "Information");
                AfficheReceptionAbonnementsRevue();
            }
        }
        else
        {
            MessageBox.Show("Le numéro de la commande existe déjà, veuillez saisir un nouveau numéro.", "Erreur");
        }
    }
    else
    {
        MessageBox.Show("Tous les champs sont obligatoires", "Information");
    }
}
```

La méthode “ParutionDansAbonnement” va permettre de retourner vrai si la date de parution est entre la date de commande et la date de fin d’abonnement. Puis on va appeler cette méthode dans la méthode “VerificationExemplaire” qui va vérifier si aucun exemplaire n’est rattaché à un abonnement de revue :

```

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

/// <summary>
/// Vérifie si aucun exemplaire n'est rattaché à un abonnement de revue
/// </summary>
/// <param name="abonnement">L'abonnement</param>
/// <returns></returns>
1 référence | 0 modification | 0 auteur, 0 modification
public bool VerificationExemplaire(Abonnement abonnement)
{
    List<Exemplaire> lesExemplairesAbonnement = controller.GetExemplairesRevue(abonnement.IdRevue);
    bool datedeparution = false;
    foreach (Exemplaire exemplaire in lesExemplairesAbonnement.Where(exemplaires => ParutionDansAbonnement(abonnement.DateCommande, at
    {
        datedeparution = true;
    }
    }
    return !datedeparution;
}

```

Puis j'ai créé le test unitaire de la méthode "ParutionDansAbonnement" en lui passant en paramètre une dateCommande inférieure à la dateFinAbonnement et une dateParution supérieur à la dateFinAbonnement et inférieur à la dateFinCommande :

```

public class ParutionTest
{
    /// <summary>
    /// Test la méthode ParutionDansAbonnement()
    /// </summary>
    [TestMethod]
    0 référence | 0 modification | 0 auteur, 0 modification
    public void TestParutionDansAbonnement()
    {
        DateTime dateCommande = new DateTime(2025, 10, 3);
        DateTime dateFinAbonnement = new DateTime(2025, 11, 3);
        DateTime dateParution = new DateTime(2025, 10, 4);
        bool resultatAttendu = true;

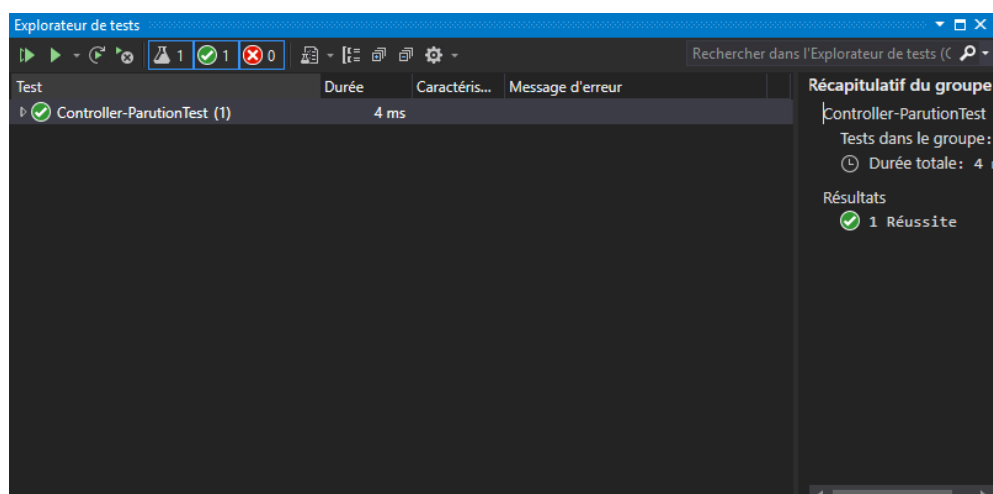
        bool resultatActuel = ParutionDansAbonnement(dateCommande, dateFinAbonnement, dateParution);

        Assert.AreEqual(resultatAttendu, resultatActuel);
    }

    1 référence | 1/1 ayant réussi | 0 modification | 0 auteur, 0 modification
    public bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFinAbonnement, DateTime dateParution)
    {
        return (DateTime.Compare(dateCommande, dateParution) < 0 && DateTime.Compare(dateParution, dateFinAbonnement) < 0);
    }
}

```

Le résultat attendu était le suivant :



La dernière méthode de cet onglet a pour but de supprimer une commande, pour cela on va donc vérifier qu'une commande est sélectionnée dans le DataGridView, puis on va appeler le Web Service qui va venir supprimer l'abonnement sélectionné. Puis on va afficher la liste de commande mise à jour :

```
private void btnSupprimerCommandeRevue_Click(object sender, EventArgs e)
{
    if (dgvAbonnementsRevue.SelectedRows.Count > 0)
    {
        Abonnement abonnement = (Abonnement)bdgAbonnementsRevue.Current;
        if (MessageBox.Show("Souhaitez-vous confirmer la suppression de l'abonnement " + abonnement.Id + " ?", "Confirmation de la suppression"))
        {
            if (VerificationExemplaire(abonnement))
            {
                if (controller.SupprimerAbonnementRevue(abonnement))
                {
                    AfficheReceptionAbonnementsRevue();
                }
                else
                {
                    MessageBox.Show("Une erreur s'est produite.", "Erreur");
                }
            }
            else
            {
                MessageBox.Show("Cet abonnement contient un ou plusieurs exemplaires, il ne peut donc pas être supprimé.", "Information");
            }
        }
    }
    else
    {
        MessageBox.Show("Une ligne doit être sélectionnée.", "Information");
    }
}
```

Pour finir, il nous a été demandé de créer une page qui contiendrait un DataGridView avec les informations des abonnements se terminant dans moins de 30 jours. Pour ce faire, j'ai créé un Form, en y ajoutant le visuel puis j'ai passé en paramètre de la méthode FrmAlerteFinAbonnement l'appel du controller. Puis dès l'ouverture de l'alerte, je vais venir appeler mon Web Service qui va me retourner les abonnements qui arrivent à échéance dans les 30 jours qui viennent, puis je vais venir remplir le DataGridView avec les "abonnementAEcheance" :

```
private readonly BindingSource bdgAbonnementsAEcheance = new BindingSource();
private readonly List<Abonnement> lesAbonnementsAEcheance = new List<Abonnement>();
1 référence | 0 modification | 0 auteur, 0 modification
public FrmAlerteFinAbonnement(FrmMediatekController controller)
{
    InitializeComponent();
    lesAbonnementsAEcheance = controller.GetAbonnementsEcheance();
    RemplirAbonnementsAEcheance(lesAbonnementsAEcheance);
}

/// <summary>
/// Remplissage de la grille des abonnements qui se terminent
/// </summary>
/// <param name="lesAbonnementsAEcheance">Liste des abonnements à échéance</param>
2 références | 0 modification | 0 auteur, 0 modification
private void RemplirAbonnementsAEcheance(List<Abonnement> lesAbonnementsAEcheance)
{
    bdgAbonnementsAEcheance.DataSource = lesAbonnementsAEcheance;
    dgvAbonnementsAEcheance.DataSource = bdgAbonnementsAEcheance;
    dgvAbonnementsAEcheance.Columns["dateCommande"].Visible = false;
    dgvAbonnementsAEcheance.Columns["montant"].Visible = false;
    dgvAbonnementsAEcheance.Columns["idRevue"].Visible = false;
    dgvAbonnementsAEcheance.Columns["id"].Visible = false;
    dgvAbonnementsAEcheance.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
    dgvAbonnementsAEcheance.Columns[0].HeaderCell.Value = "Date de fin d'abonnement";
    dgvAbonnementsAEcheance.Columns[1].HeaderCell.Value = "Titre";
}
```

Ensuite, j'ai créé les colonnes et le tri sur les colonnes au moment du clic sur les en-têtes :

```
private void dgvAbonnementsAEcheance_ColumnHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)
{
    string titreColonne = dgvAbonnementsAEcheance.Columns[e.ColumnIndex].HeaderText;
    List<Abonnement> sortedList = new List<Abonnement>();
    switch (titreColonne)
    {
        case "Titre":
            sortedList = lesAbonnementsAEcheance.OrderBy(o => o.Titre).ToList();
            break;
        case "Date de fin d'abonnement":
            sortedList = lesAbonnementsAEcheance.OrderBy(o => o.DateFinAbonnement).Reverse().ToList();
            break;
    }
    RemplirAbonnementsAEcheance(sortedList);
}
```

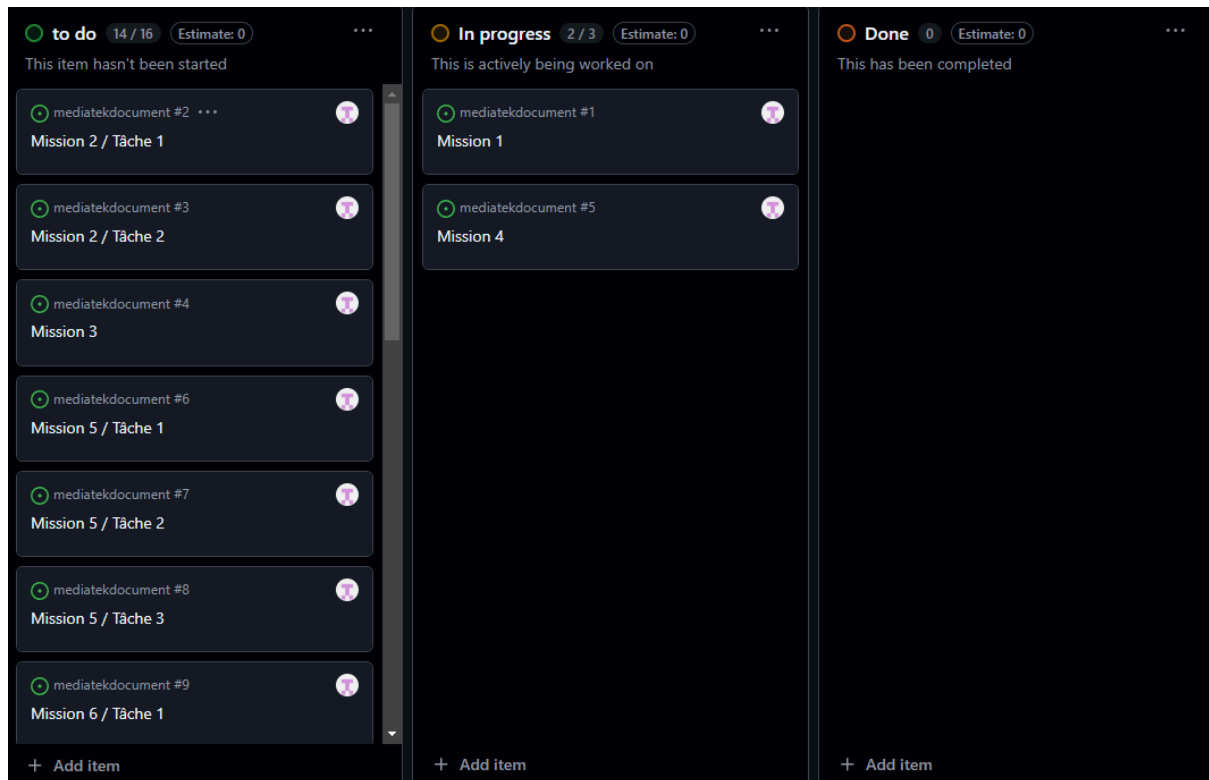
Puis le clic sur le bouton OK va venir fermer cette fenêtre. J'ai appelé le Form d'alerte au moment du clic sur le bouton Valider de ma page d'authentification, cette alerte s'affiche que si l'utilisateur correspond au service "Administratif" ou "Administrateur" :

```
if (nomService == "Culture")
{
    MessageBox.Show("Les droits ne sont pas suffisants pour accéder à cette application", "Refus", MessageBoxButtons.OK, Message
}
if (nomService == "Administrateur" || nomService == "Administratif")
{
    frmAlerteFinAbonnement.ShowDialog();
    FrmMediatek frmMediatek = new FrmMediatek();
    frmMediatek.ShowDialog();
}
if (nomService == "Prêts")
{
    FrmMediatek frmMediatek = new FrmMediatek();
    frmMediatek.GérerVisibilitéBoutons(false);
    frmMediatek.ShowDialog();
}
```

## Mission 4

Il m'était demandé d'ajouter une page d'authentification dès le lancement du programme, suivant la personne authentifié et son service, la personne avait des droits différents de consultations des documents.

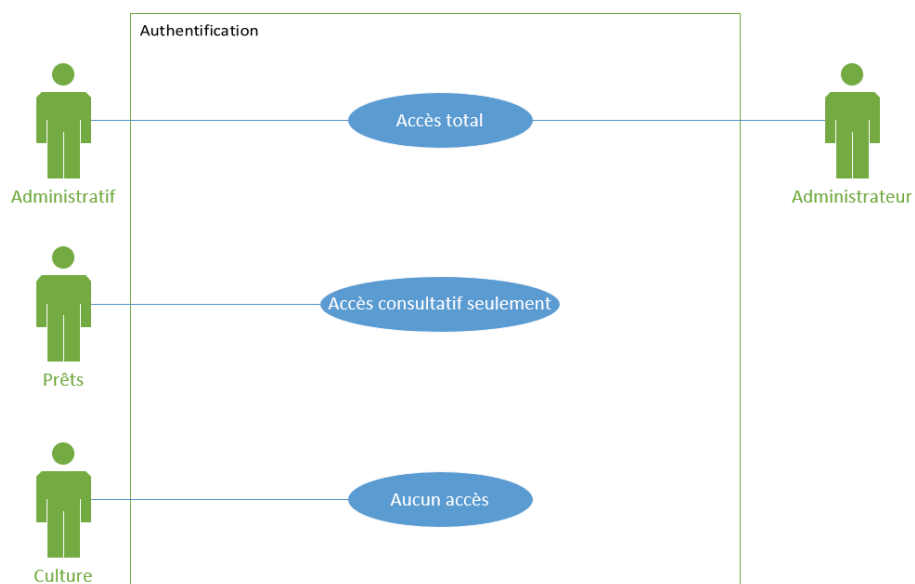
Voici la capture d'écran de mon Kanban :



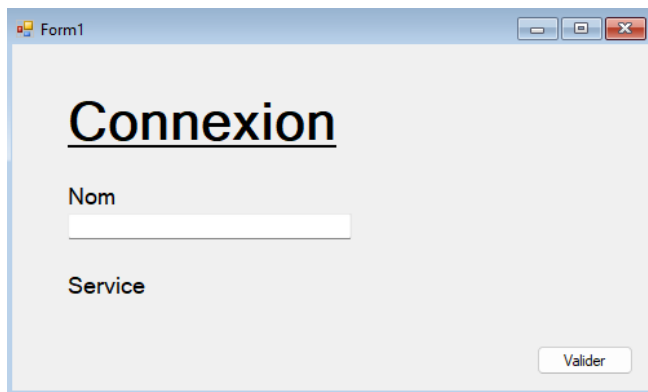
Temps estimé : 4h

Temps réel : 8h

Voici le diagramme de cas d'utilisation pour cette mission :



La première tâche que j'ai effectué a été de créer la page d'authentification au format Form :



Une fois conceptualiser, j'ai créé les deux tables qui m'étaient demandées (Utilisateurs et Services) dans la base de données, et j'y ai inséré des données. voici l'exemple avec les services :

service	id
Administratif	1
Prêts	2
Culture	3
Administrateur	4

L'étape suivante a été de coder l'action de cliquer sur le bouton valider, car au moment du clique, suivant la personne authentifié les droits d'accès à la page principale ne sont pas les mêmes. J'ai donc créer un web service qui permet en fonction du nom d'utilisateur donné, de retourner son service associé :

```
/**
 * Récupère le service d'un utilisateur en fonction de son nom
 * @param ?array $champs
 * @return array|null
 */
private function getServiceByUserName(?array $champs) : ?array {
    if ($champs === null) {
        $jsonData = file_get_contents('php://input');
        $champs = json_decode($jsonData, true);
    }

    if (!is_array($champs) || empty($champs)) {
        throw new InvalidArgumentException("Erreur 1 : aucun nom d'utilisateur fourni.");
    }
    if (!array_key_exists('utilisateur', $champs)) {
        throw new InvalidArgumentException("Erreur 2 : aucun nom d'utilisateur fourni.");
    }
    $nomUtilisateur = $champs['utilisateur'];
    $requete = "SELECT service.service
                FROM utilisateur
                JOIN service ON utilisateur.service_id = service.id
                WHERE utilisateur.nom = :nom";

    $resultat = $this->conn->queryBDD($requete, ["nom" => $nomUtilisateur]);

    if ($resultat === false) {
        throw new RuntimeException("Erreur lors de la récupération du service.");
    }

    return $resultat;
}
```

Dès  
que les



tests de ce web service ont été concluants avec Postman, j'ai pu intégrer ce web service à mon application. Pour ce faire, j'ai tout d'abord créé une classe "service" qui contient les données qu'on lui passera au retour de l'appel de mon WS. Puis j'ai créé une méthode dans la classe Access pour retourner les services des utilisateurs :

```
/// <summary>
/// Retourne les services d'un utilisateur
/// </summary>
/// <param name="nomUtilisateur">nom d'utilisateur du service concerné</param>
/// <returns>Liste d'objets Service</returns>
1 référence | 0 modification | 0 auteur, 0 modification
public List<Service> GetServiceByUserName(string nomUtilisateur)
{
    String jsonNomUtilisateur = convertToJson("utilisateur", nomUtilisateur);
    List<Service> lesServices = TraitementRecup<Service>(GET, "service/" + jsonNomUtilisateur, null);
    return lesServices;
}
```

Par la suite, j'ai appelé cette méthode dans le Controller qui permettra de faire le lien entre l'appel et le renvoi des informations :

```
/// <summary>
/// getter sur la liste des services
/// </summary>
/// <returns>Liste d'objets Services</returns>
1 référence | 0 modification | 0 auteur, 0 modification
public List<Service> GetServiceByUserName(string nomUtilisateur)
{
    return access.GetServiceByUserName(nomUtilisateur);
}
```

Une fois mon WS raccroché à mon application, j'ai pu coder les différents droits suivant les noms des personnes connectés :

```
public void btnValider_Click(object sender, EventArgs e)
{
    lesServices = controller.GetServiceByUserName(txtBoxNom.Text);

    if (lesServices.Count > 0)
    {
        string nomService = lesServices[0].TypeService;

        lblNomService.Text = nomService;

        if (nomService == "Culture")
        {
            MessageBox.Show("Les droits ne sont pas suffisants pour accéder à cette application", "Refus", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
        if (nomService == "Administrateur" || nomService == "Administratif")
        {
            FrmMediatek frmMediatek = new FrmMediatek();
            frmMediatek.ShowDialog();
        }
        if (nomService == "Prêts")
        {
            FrmMediatek frmMediatek = new FrmMediatek();
            frmMediatek.GérerVisibilitéBoutons(false);
            frmMediatek.ShowDialog();
        }
    }
    else
    {
        MessageBox.Show("Utilisateur inconnu ou aucun service associé", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
```

Pour le service Prêts, j'ai écrit une méthode dans la classe FrmMediatek pour gérer la visibilité des différents objets :

```
public void GérerVisibilitéBoutons(bool afficher = true)
{
    grpBoutonsLivres.Visible = afficher;
    grpBoutonsDVD.Visible = afficher;
    grpBoutonsRevue.Visible = afficher;
    grpReceptionRevue.Visible = afficher;
    grpReceptionExemplaire.Visible = afficher;
}
```

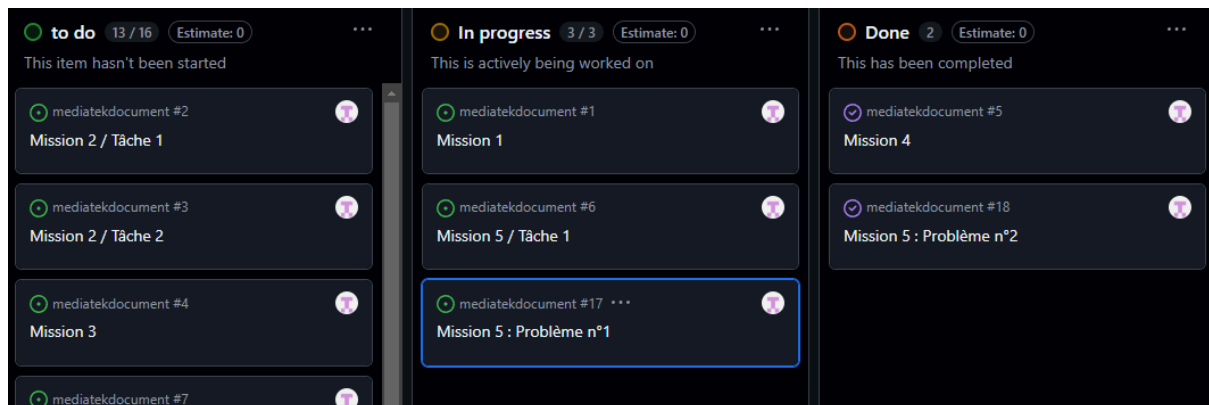
## Mission 5

### Tâche 1 /

#### Problème 1

Il m'était demandé de modifier la classe "App.config" et "Access" de mon application, pour permettre de sécuriser les informations de connexion à mon API.

Voici la capture d'écran de mon Kanban :



Temps estimé : 1h

Temps réel : 30 minutes

Pour corriger ce problème, j'ai d'abord regardé l'application Habilitation pour m'inspirer de sa méthode de connexion à la base de données. J'ai donc modifier dans mon projet la classe "App.config" pour y ajouter les différentes informations de connexion telles que l'adresse de connexion, le login ainsi que le mot de passe. Voici ma classe après les modifications :

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
  </configSections>
  <appSettings>
    <add key="API_URI" value="http://localhost/rest_mediatekdocuments/" />
    <add key="API_USER" value="admin" />
    <add key="API_PASSWORD" value="adminpwd" />
  </appSettings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
  </startup>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="Newtonsoft.Json" publicKeyToken="30ad4fe6b2a6aeed" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-13.0.0.0" newVersion="13.0.0.0" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

Ensuite, j'ai modifié la méthode "Access" dans ma classe Access pour y appeler grâce à "ConfigurationManager.AppSettings", ce qui m'a permis d'aller chercher mes informations de connexion dans ma classe "AppConfig". Voici ma classe après les modifications :

```

public class Access
{
    /// <summary>
    /// adresse de l'API
    /// </summary>
    private static readonly string uriApi = ConfigurationManager.AppSettings["API_URI"];
    /// <summary>
    /// instance unique de la classe
    /// </summary>
    private static Access instance = null;
    /// <summary>
    /// instance de ApiRest pour envoyer des demandes vers l'api et recevoir la réponse
    /// </summary>
    private readonly ApiRest api = null;
    /// <summary>
    /// méthode HTTP pour select
    /// </summary>
    private const string GET = "GET";
    /// <summary>
    /// méthode HTTP pour insert
    /// </summary>
    private const string POST = "POST";
    /// <summary>
    /// méthode HTTP pour update

    /// <summary>
    /// Méthode privée pour créer un singleton
    /// initialise l'accès à l'API
    /// </summary>
    1 référence | Mathis2111, il y a 29 jours | 1 auteur, 1 modification
    private Access()
    {
        try
        {
            string apiUser = ConfigurationManager.AppSettings["API_USER"];
            string apiPassword = ConfigurationManager.AppSettings["API_PASSWORD"];
            string authenticationString = $"{apiUser}:{apiPassword}";

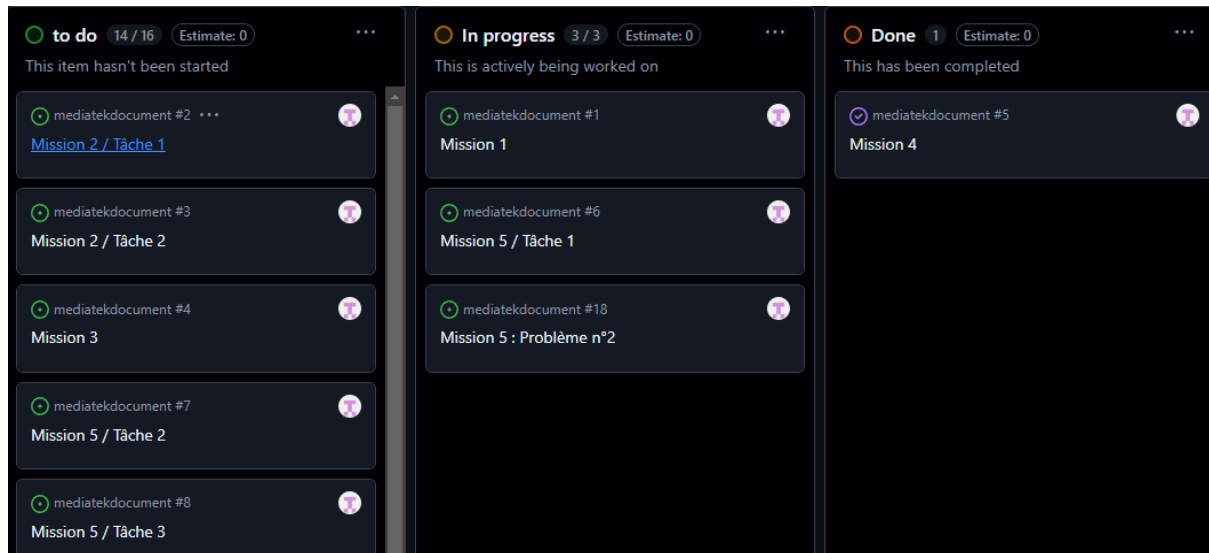
            api = ApiRest.GetInstance(uriApi, authenticationString);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Environment.Exit(0);
        }
    }
}

```

## Problème 2

Il m'était demandé pour ce problème 2, de rendre inaccessible les dossiers et fichiers contenus dans mon API en tapant cette URL : [http://localhost/rest\\_mediatekdocuments/](http://localhost/rest_mediatekdocuments/).

Voici la capture d'écran de mon Kanban :



Temps estimé : 1h

Temps réel : 30 minutes

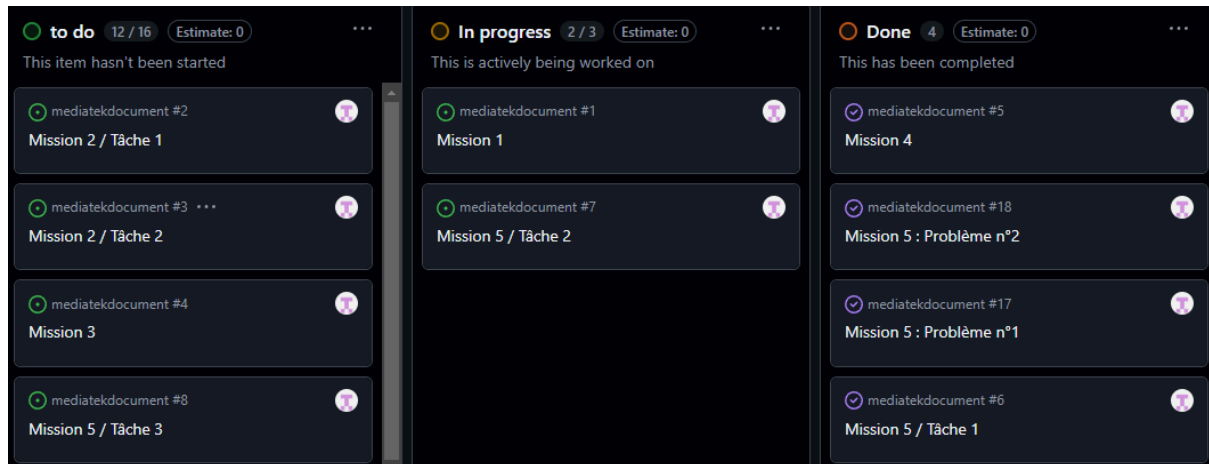
J'ai d'abord créé une issue dans mon kanban pour pouvoir visualiser la tâche. Ensuite j'ai fait des recherches sur chrome pour essayer de comprendre comment interdire l'accès public aux dossier de mon API depuis mon fichier htaccess. J'ai réussi à trouver une option qu'il fallait ajouter à la toute première ligne pour bloquer cet accès. Voici donc mon fichier htaccess après modification :

```
Options -Indexes
RewriteEngine on
RewriteCond %{REQUEST_METHOD} =GET
RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,L]
RewriteCond %{REQUEST_METHOD} =GET
RewriteRule ^([a-zA-Z0-9_]+)/({.*})$ src/index.php?table=$1&champs=$2 [B,L]
RewriteCond %{REQUEST_METHOD} =POST
RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,L]
RewriteCond %{REQUEST_METHOD} =PUT
RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,L]
RewriteCond %{REQUEST_METHOD} =PUT
RewriteRule ^([a-zA-Z0-9_]+)/([a-zA-Z0-9_]+)$ src/index.php?table=$1&id=$2 [B,L]
RewriteCond %{REQUEST_METHOD} =DELETE
RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,L]
RewriteCond %{REQUEST_METHOD} =DELETE
RewriteRule ^([a-zA-Z0-9_]+)/({.*})$ src/index.php?table=$1&champs=$2 [B,L]
```

## Tâche 2 /

Il m'était demandé de nettoyer le code ajouté excepté le nommage des objets qui se met automatiquement en minuscule.

Voici mon kanban avec la mission dans la colonne In Progress :



Temps estimé : 1h

Temps réel : 20 minutes

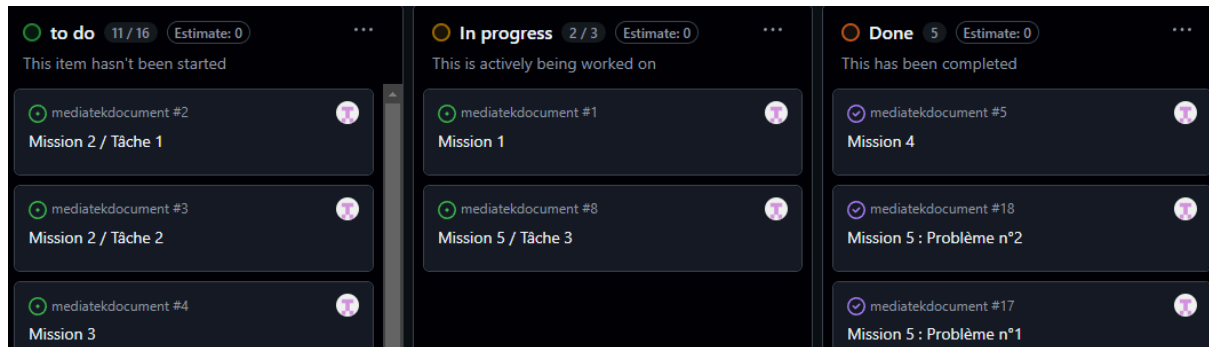
Je n'ai pas eu de modifications à effectuer et les seuls messages d'erreurs sont dus au nommage des objets graphiques. Voici une photo de l'analyse de Sonar Lint :



## Tâche 3 /

Pour cette tâche, il m'était demandé d'ajouter le code de configuration des logs, et de mettre des logs au niveau de chaque affichage console.

Voici mon kanban avec la tâche dans la colonne In Progress :



Temps estimé : 1h

Temps réel : 1h

La première chose que j'ai dû faire a été d'installer les bon package NuGet (serilog, serilog Console et serilog files. Une fois les packages installés, je me suis inspiré du projet Habilitation pour la configuration des logs, puis j'ai ajouté un log erreur à chaque Console. Voici une capture d'écran de la configuration des logs :

```
private Access()
{
    Log.Logger = new LoggerConfiguration()
        .MinimumLevel.Debug()
        .WriteTo.Console()
        .WriteTo.File("logs/log.txt", rollingInterval: RollingInterval.Day)
        .CreateLogger();
}
```

Puis voici un des logs que j'ai ajouté :

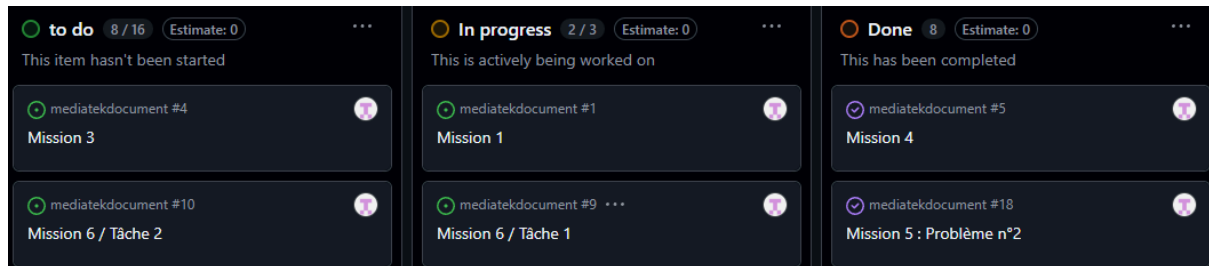
```
catch (Exception e)
{
    Log.Error("Erreur d'accès à l'API");
    Console.WriteLine(e.Message);
    Environment.Exit(0);
}
```

## Mission 6

### Tâche 1 /

Pour cette mission il m'était demandé d'effectuer les tests unitaires pour chaque classe du mode de l'application, puis d'effectuer les tests fonctionnels des méthodes de mon API.

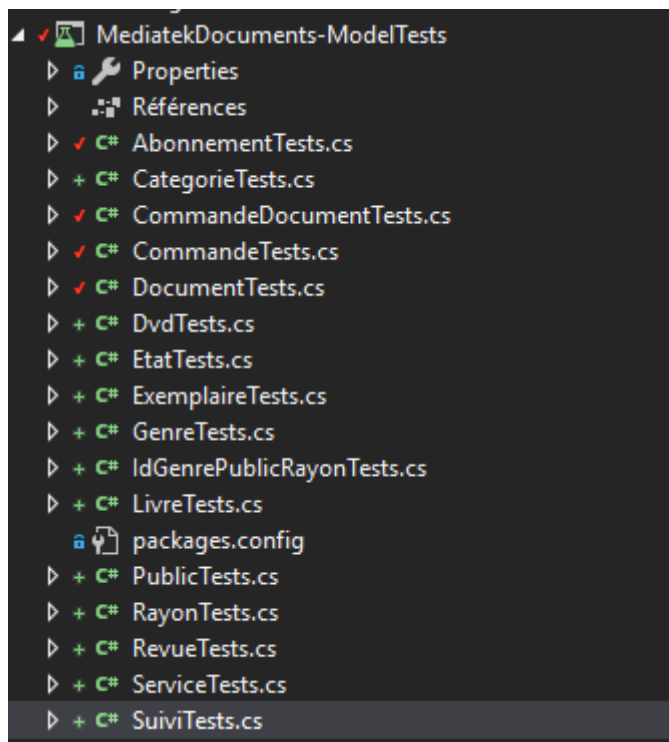
Voici une capture d'écran de mon Kanban :



Temps estimé : 5h

Temps réel : 5h

La première tâche que j'ai faite a été de créer un nouveau projet de test dans la solution qui contiendra toutes les classes de tests. Ensuite, j'ai ajouté la référence du dossier model dans le projet pour pouvoir utiliser les classes du model :



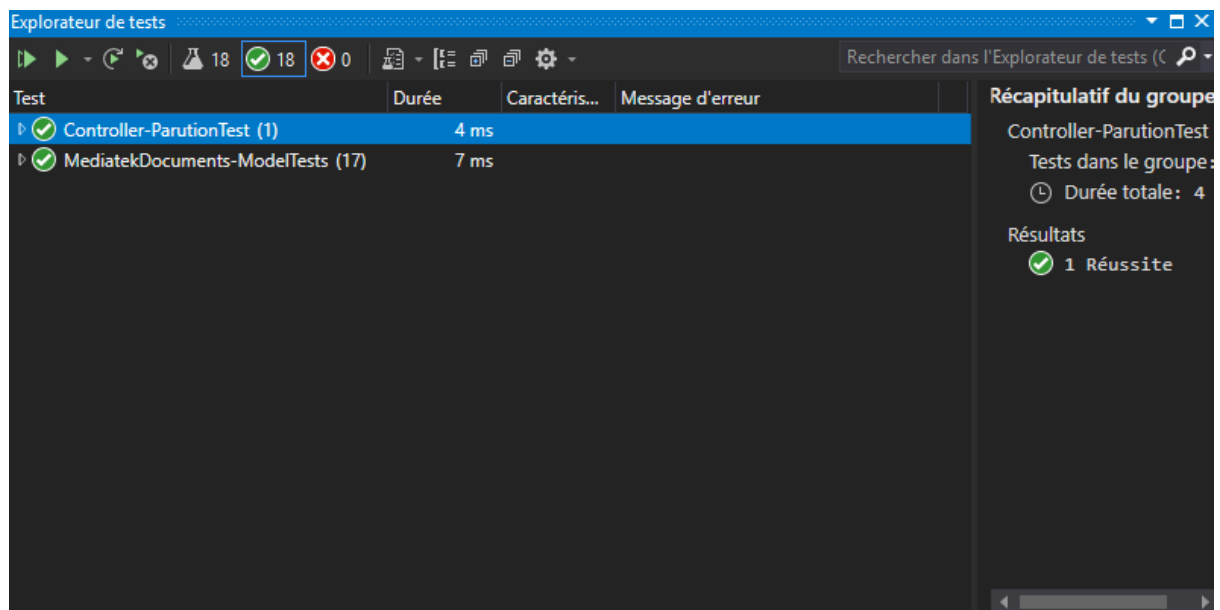
Puis j'ai écrit le premier test pour la classe Abonnement, j'ai créé un objet de type Abonnement en lui passant des informations et le résultat devait me retourner un message

me disant que l'id, la dateCommande, le montant, la dateFinAbonnement, l'idRevue et le titre étaient valorisés :

```
[TestClass]
0 références | Mathis2111, il y a 22 heures | 1 auteur, 1 modification
public class AbonnementTests
{
    private const string id = "00005";
    private static readonly DateTime dateCommande = new DateTime(2025, 2, 22);
    private const double montant = 30;
    private static readonly DateTime dateFinAbonnement = new DateTime(2025, 1, 30);
    private const string idRevue = "10003";
    private const string titre = "Challenges";

    private static readonly Abonnement abonnement = new Abonnement(id, dateCommande, montant, dateFinAbonnement, idRevue, titre);
    [TestMethod]
    0 références | Mathis2111, il y a 22 heures | 1 auteur, 1 modification
    public void AbonnementTest()
    {
        Assert.AreEqual(id, abonnement.Id, "devrait réussir : id valorisé");
        Assert.AreEqual(dateCommande, abonnement.DateCommande, "devrait réussir : date de commande valorisée");
        Assert.AreEqual(montant, abonnement.Montant, "devrait réussir : montant valorisé");
        Assert.AreEqual(dateFinAbonnement, abonnement.DateFinAbonnement, "devrait réussir : date de fin d'abonnement valorisée");
        Assert.AreEqual(idRevue, abonnement.IdRevue, "devrait réussir : idRevue valorisé");
        Assert.AreEqual(titre, abonnement.Titre, "devrait réussir : titre valorisé");
    }
}
```

J'ai reproduit le même test pour chaque classe du model en demandant la vérification des informations comprises dans les classes du model. Le résultat de ces tests devait donné cela :



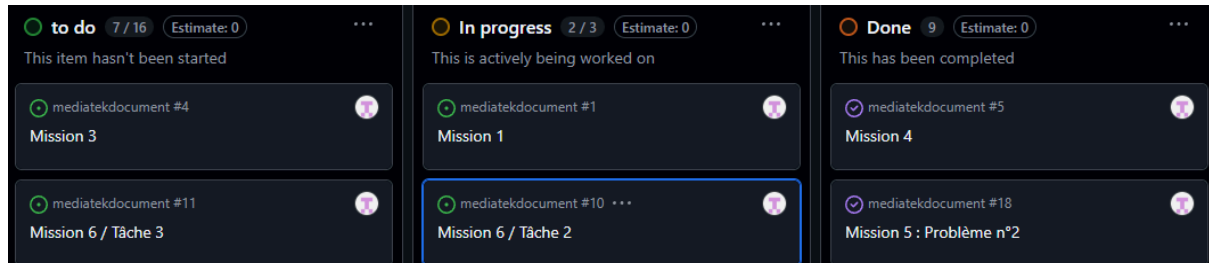
En ce qui concerne les tests de mon API sur Postman, je n'ai pas pu les réaliser car je n'ai pas la bonne version de Postman pour réaliser ces tests, je n'ai donc pas l'onglet "Tests".



## Tâche 2 /

Pour cette deuxième tâche il m'a été demandé de créer la documentation technique pour l'application et pour l'API.

Voici la capture d'écran de mon kanban :



Temps estimé : 1h

Temps réel : 1h

Pour cette deuxième tâche, j'ai commencé par insérer les commentaires normalisés pour chaque méthode de mon projet :

```
/// <summary>
/// Bouton de validation d'une modification pour un livre
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | Mathis2111, Il y a 1 jour | 1 auteur, 1 modification
private void btnValiderModifs_Click(object sender, EventArgs e)
{

/**
 * ajoute un abonnement pour une revue
 * @param array|null $champs
 * @return array|null
 */
private function AjouterAbonnementRevue(?array $champs): int {
```

Puis, j'ai suivi les différentes informations qui m'ont été donné, puis j'ai réussi à générer la documentation technique pour l'application en C# :

### Document Mediatek

#### ▼ Espaces de noms

<a href="#">Documents MediaTek</a>
<a href="#">MediaTekDocuments.controller</a>
<a href="#">MediaTekDocuments.dal</a>
<a href="#">Gestionnaire de documents MediaTek</a>
<a href="#">MediaTekDocuments.model</a>
<a href="#">MediaTekDocuments.view</a>

et pour l'API :

## documents rest\_mediatek

### Espaces de noms

- Compositeur
- Chargement automatique

- Graham Campbell
- Type de résultat

- PhpOption

- Dotenv

- Exception

- Chargeur

- Analyseur

- Dépôt

- Magasin

### Forfaits

- Application

### Rapports

## Documentation

### Table des matières

#### Forfaits

-  [Application](#)

#### Espaces de noms

-  [Compositeur](#)

-  [Graham.Campbell](#)

-  [PhpOption](#)

-  [Dotenv](#)

#### Interfaces

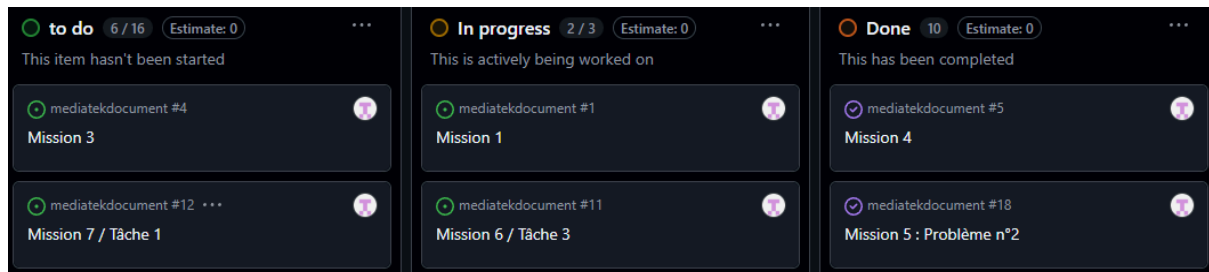
-  [Enfilable](#)

Puis j'ai transféré les documentations dans les dépôts respectifs.

### Tâche 3 /

Pour cette tâche il m'a été demandé de créer la documentation utilisateur sous forme de vidéo de présentation du projet.

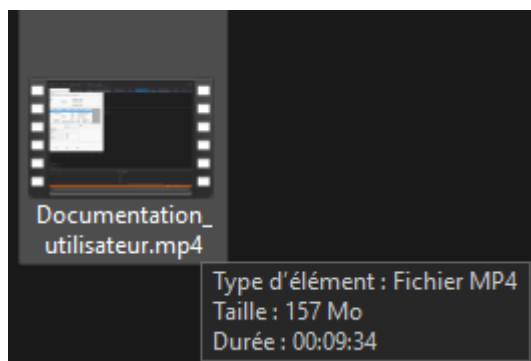
Voici une capture d'écran de mon kanban :



Temps estimé : 2h

Temps réel : 1h30

Pour cette vidéo j'ai utilisé mon application de capture d'écran et je l'ai passé en mode capture vidéo. Le temps de la vidéo est de 9 minutes et 34 secondes :



## **Mission 7 :**

Pour cette tâche, je n'ai pas pu déployer mon API ainsi que mon application car j'ai rencontré des erreurs qui ne m'ont pas permises de déployer. Ces erreurs ont modifié mon projet en local, j'ai donc dû trouver une solution pour réparer celles-ci, j'ai privilégié une application fonctionnelle en local plutôt qu'une application non fonctionnelle. Mon application est donc fonctionnelle en local mais pas en ligne. Je n'ai donc pas pu non plus gérer les sauvegardes des données.

## **Bilan final :**

Le projet Mediatekdocument a été une expérience particulièrement enrichissante qui m'a permis de découvrir et de comprendre en profondeur les différentes étapes nécessaires à la réalisation d'un projet applicatif qui comprend la gestion d'une API, du début jusqu'à la fin. Il m'a offert l'occasion de mettre en pratique mes connaissances tout en acquérant de nouvelles compétences liées à la gestion et au développement d'une application.

Ce projet m'a permis de m'initier à l'organisation et à la réalisation d'un projet, depuis l'analyse des besoins jusqu'à la mise en œuvre des solutions. Cela m'a permis de mieux comprendre les aspects techniques liés à la communication entre une API et une application. Et renforcer mes compétences dans la gestion des problématiques de liaison entre l'API et l'application. J'ai pu découvrir les enjeux et les difficultés liés aux tests, notamment les tests unitaires qui ont été compliqués à réaliser.

Bien que globalement positif, le projet m'a réservé quelques problèmes :

Problèmes de communication entre les Web Services et la base de données :  
J'ai rencontré des difficultés dues à l'utilisation de méthodes toutes faites qui ont été compliquées à comprendre.

Compréhension du déploiement d'une API :  
J'ai eu du mal à appréhender la manière dont l'API aurait pu être hébergée, et j'ai rencontré une grosse difficulté pour l'hébergement qui m'a poussé à ne pas en mettre en place.

J'en conclus que malgré ces obstacles, ce projet a été vraiment enrichissant pour moi. Il m'a permis de mieux comprendre les différentes facettes d'un projet applicatif, de l'analyse des besoins au développement, en passant par les tests et la gestion des erreurs. Ces expériences, bien que parfois difficiles, m'ont permis de développer une meilleure compréhension des consignes qui m'étaient demandées.