ATELIER PROFESSIONNEL

MEDIATEK DOCUMENTS

Compte-rendu

Réalisé par
Joseph-Nicolas YAZBEK

BTS SIO-SLAM
2ème année

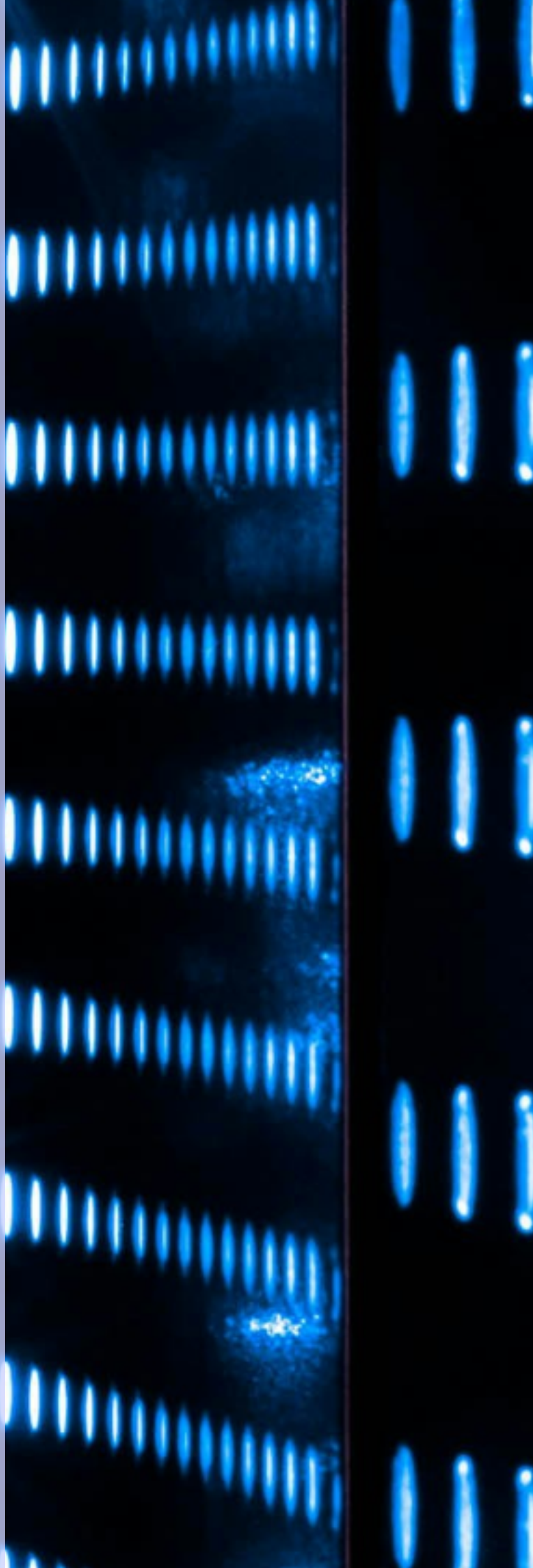


Table des matières

Mission / Contexte

L'objectif de cet atelier est de faire évoluer une application C# utilisant une API REST pour interagir avec une base de données.

La finalité de l'application est que les médiathèques puissent gérer leurs documents et leurs commandes.

L'application est déjà partiellement construite et permet déjà les consultations de documents, livres, DVD et revues. Il nous faut cependant la faire évoluer, ajouter des fonctionnalités, déployer la BDD et l'API, et enfin permettre l'installation du plusieurs postes.

Mission 1 : gérer les commandes

a) Tâche 1 : les commandes de livres ou de DVD

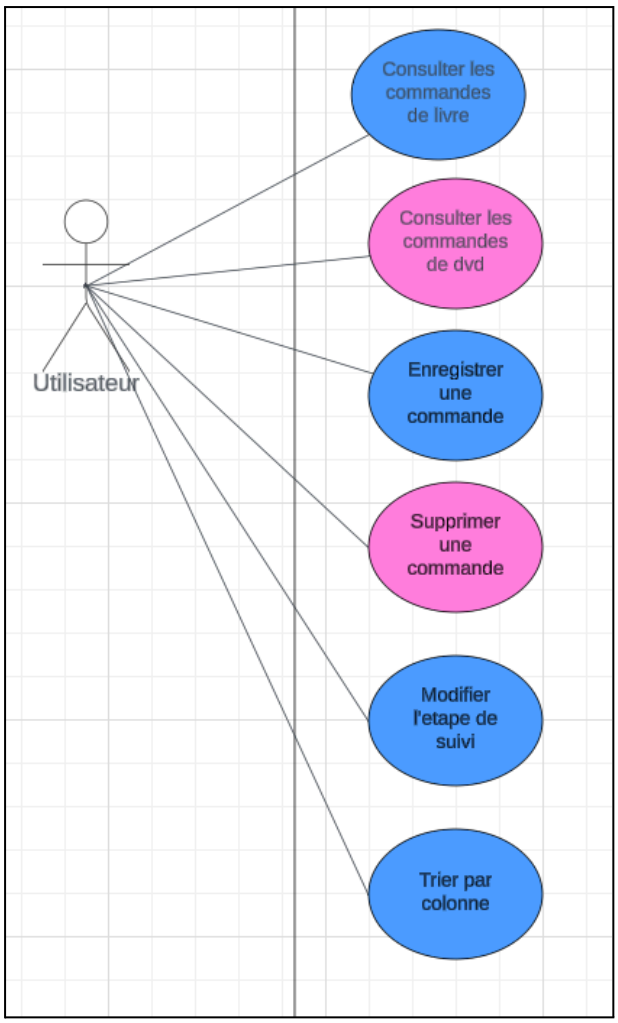
Gérer les commandes de livres ou de DVD #1

Open jnyazbek/mediatekdocuments Public

jnyazbek opened this issue on Jan 19 edited by jnyazbek · Edits

Tâche 1 : gérer les commandes de livres ou de DVD (8h)
Dans la base de données, créer la table 'Suivi' qui contient les différentes étapes de suivi d'une commande de document de type livre ou dvd. Relier cette table à CommandeDocument.
Créer un onglet (ou une nouvelle fenêtre) pour gérer les commandes de livres.
La charte graphique doit correspondre à l'existant.
Dans toutes les listes, permettre le tri sur les colonnes.
Dans l'onglet (ou la fenêtre), permettre la sélection d'un livre par son numéro, afficher les informations du livre ainsi que la liste des commandes, triée par date (ordre inverse de la chronologie). La liste doit comporter les informations suivantes : date de la commande, montant, nombre d'exemplaires commandés et l'étape de suivi de la commande.
Créer un groupbox qui permet de saisir les informations d'une nouvelle commande et de l'enregistrer. Lors de l'enregistrement de la commande, l'étape de suivi doit être mise à "en cours".
Permettre de modifier l'étape de suivi d'une commande en respectant certaines règles (une commande livrée ou réglée ne peut pas revenir à une étape précédente (en cours ou relancée), une commande ne peut pas être réglée si elle n'est pas livrée).
Permettre de supprimer une commande uniquement si elle n'est pas encore livrée. Faire un trigger qui réalise aussi la suppression dans la classe fille.
Créer le trigger qui se déclenche si une commande passe à l'étape "livrée" et qui crée autant de tuples dans la table "Exemplaire" que nécessaires, en valorisant la date d'achat avec la date de commande et en mettant l'état de l'exemplaire à "neuf". Le numéro d'exemplaire doit être séquentiel par rapport au livre concerné.
Créer un onglet pour gérer les commandes de DVD en suivant la même logique que pour les commandes de livres.
Toutes les sécurités seront mises en place pour éviter des erreurs de manipulation.
Créer le trigger qui contrôle la contrainte de partition de l'héritage sur Commande.

Temps estimé : 12h
Temps réel : 18h



Les commandes de livres ou de DVD seront gérées par la classe model CommandeDocument qui héritera de la classe Commande.

Il nous faudra bien sûr ajouter la table correctement dans la base de donnée :

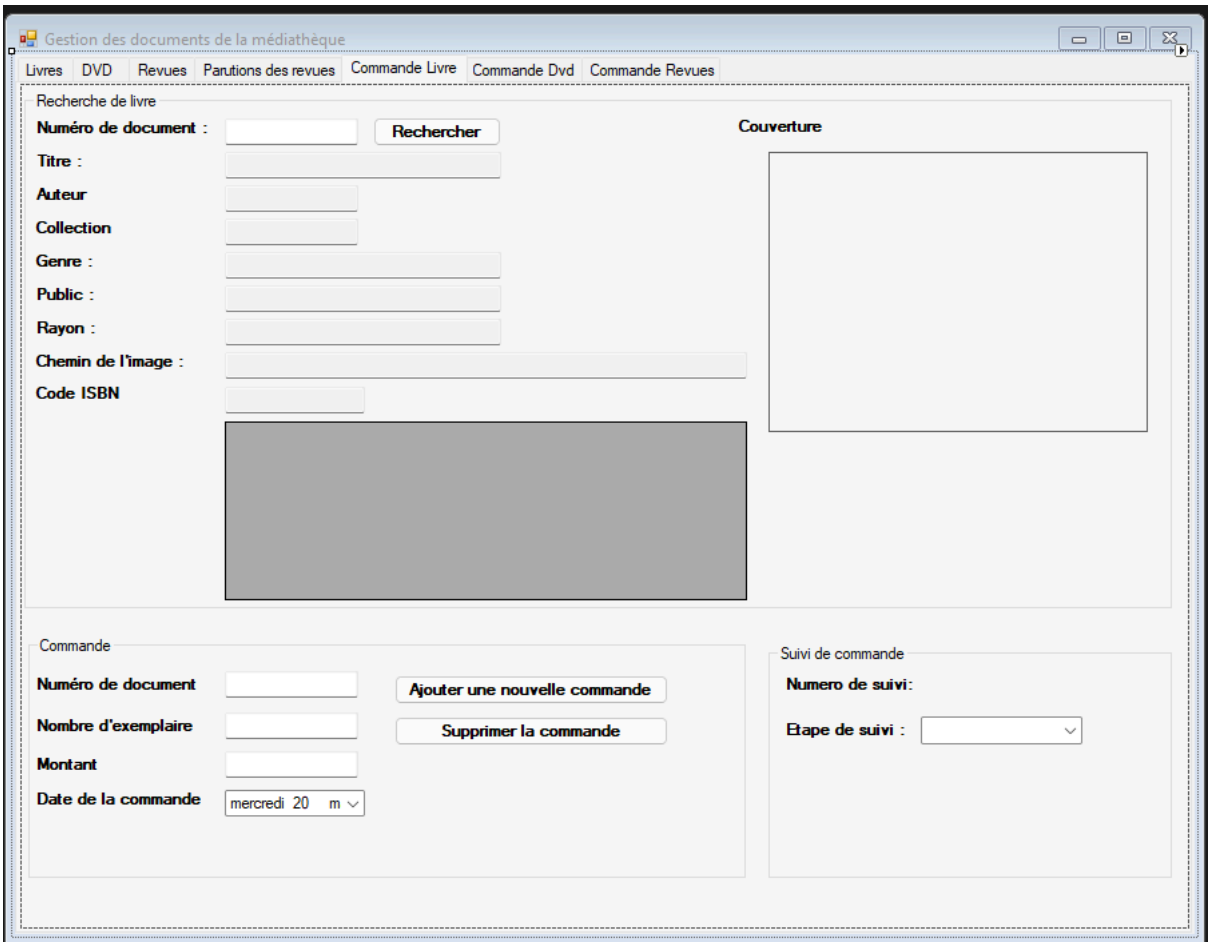
```
56 références
public class CommandeDocument : Commande
{
    /// <summary>
    /// Récupère ou définit le nombre d'exemplaires de la commande de document
    /// </summary>
    7 références
    public int NbExemplaire { get; set; }

    /// <summary>
    /// Récupère ou définit l'id du livreDvd de la commande de document
    /// </summary>
    5 références
    public string IdLivreDvd { get; set; }

    /// <summary>
    /// Récupère ou définit l'id de l'étape de suivi de la commande de document
    /// </summary>
    2 références
    public string IdSuivi { get; set; }




    /// <summary>
    /// Initialisation d'un nouvel objet CommandeDocument
    /// </summary>
    /// <param name="id">Id de la commande </param>
    /// <param name="dateCommande">Date de la commande </param>
    /// <param name="montant">Montant total de la commande</param>
    /// <param name="nbExemplaire">Nombre d'exemplaires de document</param>
    /// <param name="idLivreDvd">Id du LivreDvd correspondant à la commande de document</param>
    /// <param name="idSuivi">Id de l'étape de suivi correspondant à la commande de document</param>
    3 références
    public CommandeDocument(string id, DateTime date, double montant, int nbExemplaire, string idLivreDvd, string idSuivi, string libelle)
        : base(id, date, montant, idSuivi,libelle)
    {
        this.NbExemplaire = nbExemplaire;
        this.IdLivreDvd = idLivreDvd;
        this.Date = date;



    }
}
```

Parcourir | Structure | SQL | Rechercher | Insérer | Exporter | Importer | Privilèges | Opérations

Structure de table | Vue relationnelle

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	id	varchar(5)	utf8mb4_0900_ai_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
2	nbExemplaire	int			Oui	*NULL*			Modifier Supprimer Plus
3	idLivreDvd	varchar(10)	utf8mb4_0900_ai_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
4	date	datetime			Non	*Aucun(e)*			Modifier Supprimer Plus
5	montant	double			Non	*Aucun(e)*			Modifier Supprimer Plus
6	idsuivi	int			Oui	*NULL*			Modifier Supprimer Plus

Il nous faut commencer par l'interface graphique de l'onglet "Commande de livre" :



À l'aide de Windows Form, on crée une interface permettant de rechercher un livre à l'aide de son identifiant. S'afficheront alors les informations sur ce livre ainsi que les commandes associées si elles existent.

En cliquant sur l'une des commandes affichée dans la Datagridview, il nous est alors possible de modifier son étape de suivi dans le cadre "Suivi de commande".

Enfin dans le cadre Commande, il nous est possible d'ajouter une nouvelle commande en remplissant les informations adéquates.

1. **Recherche de livre et affichage des informations**

Pour effectuer une recherche, on inscrit le numéro de livre recherché puis on appuie sur le bouton recherche qui déclenche la méthode suivante :

```
/// <summary>
/// recherche les information sur le document sélectionné
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
2 références
private void buttonRechercheNumDoc_Click(object sender, EventArgs e)
{
    string numdoc = textBoxNumDoc.Text;
    Console.WriteLine("button recherche num doc " + numdoc);
    if (numdoc != null)
    {
        lesCommandesDocument = controller.GetCommandesDocuments(numdoc);
        foreach (CommandeDocument commande in lesCommandesDocument)
        {
            Console.WriteLine("ID: " + commande.Id + ", Nb Exemplaires: " + commande.NbExemplaire + ",
        }

        RemplirCommandesLivresListe(lesCommandesDocument);
    }


    if (!textBoxNumDoc.Text.Equals(""))
    {
        Livre livre = lesLivres.Find(x => x.Id.Equals(numdoc));
        if (livre != null)
        {
            AfficheLivreCommandeInfos(livre);
        }
        else
        {
            MessageBox.Show("numéro introuvable");
        }
    }
}
```

On récupère le numéro de livre dans la textBoxNumDoc et on apelle par l'intermédiaire d'une méthode dans notre controller, la méthode présente dans access qui est la classe communiquant avec la base de donnée :

```
/// <summary>
/// Retourne les commandes des documents
/// </summary>
/// <param name="idDocument">id du document concerné</param>
/// <returns>Liste d'objets CommandeDocument</returns>

1 référence
public List<CommandeDocument> GetCommandesDocument(string idDocument)
{
    Console.WriteLine("the id document passed in Access.getcommandedocument is "+ idDocument);
    String jsonIdDocument = convertToJson("id", idDocument);
    List<CommandeDocument> lesCommandesDocument = TraitementRecup<CommandeDocument>(GET, uriApi+"commandedocument/livre/"+idDocument);
    Console.WriteLine("GetCommandesDocument utilisé, the request is "+ uriApi + "commandedocument/livre/" + idDocument);
    return lesCommandesDocument;
}
```

La requête est donc envoyée à l'API qui la reçoit et la traite premièrement dans la classe : mediatekdocument.php grâce à la configuration du fichier .htaccess.

```
RewriteEngine on
# Règle pour les commandedocument par ID de livre
RewriteRule ^commandedocument/livre/([0-9]+)$ mediatekdocuments.php?table=commandedocument&id=$1 [L,QSA]
RewriteRule ^commandedocument/dvd/([0-9]+)$ mediatekdocuments.php?table=commandedocument&id=$1 [L,QSA]
RewriteRule ^abonnement/revue/([0-9]+)$ mediatekdocuments.php?table=abonnement&id=$1 [L,QSA]
RewriteRule ^abonnement/([0-9]+)$ mediatekdocuments.php?table=abonnement&id=$1 [L,QSA]
RewriteRule ^([a-zA-Z]+)$ mediatekdocuments.php?table=$1
RewriteRule ^([a-zA-Z]+)/({.*})$ mediatekdocuments.php?table=$1&champs=$2
RewriteRule ^([a-zA-Z]+)/([a-zA-Z0-9]+)/({.*})$ mediatekdocuments.php?table=$1&id=$2&champs=$3
RewriteRule ^commandedocument/([0-9]+)$ mediatekdocuments.php?table=commandedocument&id=$1 [L,QSA]
RewriteRule ^suivi/([0-9]+)$ mediatekdocuments.php?table=suivi&id=$1 [L,QSA]
RewriteRule ^commande/([0-9]+)$ mediatekdocuments.php?table=commande&id=$1 [L,QSA]
RewriteRule ^exemplaire/([0-9]+)$ mediatekdocuments.php?table=exemplaire&id=$1
RewriteRule ^utilisateur/([0-9]+)$ mediatekdocuments.php?table=utilisateur&id=$1
# Règle pour la route vide
RewriteRule ^$ mediatekdocuments.php?table=routevide [L,QSA]
```

La requête est donc prise en charge en tant que requête get :

```
case 'GET':
    // récupération des données
    // Nom de la table au format string
    $table = filter_input(INPUT_GET, 'table', FILTER_SANITIZE_SPECIAL_CHARS);

    // id de l'enregistrement au format string
    $id = filter_input(INPUT_GET, 'id', FILTER_SANITIZE_SPECIAL_CHARS);
    if($id!=''){
        $id= array('id'=> $id);
    }

    // nom et valeur des champs au format json
    $contenu = filter_input(INPUT_GET, 'contenu', FILTER_SANITIZE_SPECIAL_CHARS,FILTER_FLAG_NO_ENCODE_QUOTES);
    if($contenu != ""){
    $contenu = json_decode($contenu, true);}
    //print_r("id");
    //print_r($id);
    //print_r("fin id");
    $controle->get($table, $id);
    break;
```

On récupère les informations nécessaires dans la requête et on les passe en paramètre de la méthode get dans contrôle.

```php
/**
 * requete arrivée en GET (select)
 * @param string $table nom de la table
 * @param type $champs nom et valeur des champs de recherche
 */
public function get($table, $champs){
    //print_r("champs = ");
    //print_r($champs);
    $result = null;
    if (empty($champs)){

        $result = $this->accessBDD->selectAll($table);
    }else{
        //print_r("champs n'est pas vide");
        //print_r($champs);
        if (!is_array($champs)) {
        $champs = json_decode($champs, true);
        //print_r("champs converit en tableau");
        //print_r($champs);
        }
        $result = $this->accessBDD->select($table, $champs);
    }
    if (gettype($result) != "array" && ($result == false || $result == null)){
        $this->reponse(400, "requete invalide");
    }else{
        $this->reponse(200, "OK", $result);
    }
}
```

Si les champs sont vides, on utilise selectall qui retournent la totalité des lignes de la table, sinon, on utilise select qui utilise les champs pour la sélection.
Dans notre cas on utilise la recherche par ID : c'est donc la méthode suivante de la classe AccessBDD qui construit la requête SQL.

```php
public function selectCommandeDocumentByLivreId($idLivre) {
 // print_r(" selectCommandeDocumentByLivreId");
 $param = array(
 "idLivre" => $idLivre
 );
 $req = "SELECT cd.id, cd.nbExemplaire, cd.idLivreDvd, cd.idsuivi, cd.date, cd.montant, ";
 $req .= "s.libelle as libelle ";
 $req .= "FROM commandedocument cd ";
 $req .= "JOIN commande c ON cd.id = c.id ";
 $req .= "JOIN suivi s ON cd.idsuivi = s.id ";
 $req .= "JOIN livres_dvd ld ON cd.idLivreDvd = ld.id ";
 $req .= "WHERE cd.idLivreDvd = :idLivre ";
 $req .= "ORDER BY c.dateCommande DESC";

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

Cette réponse est récupérée par la méthode ayant permis l'envoi de la requête :

```
21 références
private List<T> TraitementRecup<T>(String methode, String message)
{
    List<T> liste = new List<T>();
    try
    {



        JObject retour = api.RecupDistant(methode, message);
        Console.WriteLine("Réponse brute de l'API :incoming");
        Console.WriteLine("Réponse brute de l'API : " + retour);

        // extraction du code retourné
        String code = (String)retour["code"];
        Console.WriteLine("le code reçu est " + code);

        if (code.Equals("200"))
        {
            DateTest dateTest = new DateTest();
         //parametre de deserialisation
            var settings = new JsonSerializerSettings
            {
                Converters = new List<JsonConverter> { new CustomDateTimeConverter(), new CustomBooleanJsonConverter() }
            };
            //recuperation de la string
            String resultString = JsonConvert.SerializeObject(retour["result"]);
            String resultString2 = JsonConvert.SerializeObject(dateTest.jsonTest);
            // Deserialize the JSON string into a list of objects using the defined settings
            liste = JsonConvert.DeserializeObject<List<T>>(resultString, settings);
            Console.WriteLine(resultString2 + "and resultstring is "+ resultString);
        }

        else
        {
            Console.WriteLine("code erreur = " + code + " message = " + (String)retour["message"]);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("Erreur lors de l'accès à l'API : " + e.Message);
        Environment.Exit(0);
    }
    return liste;
}
```

On récupère une Liste ne contenant les commandes concernant le livre dont l'ID a été recherché.
Cette liste est exploitée par la méthode suivante, qui remplit la datagridview avec les informations sur les commandes :

```
2 références
private void RemplirCommandesLivresListe(List<CommandeDocument> lesCommandesDocument)
{
    if (lesCommandesDocument != null)
    {
        bindingSourceCommandesLivre.DataSource = lesCommandesDocument;
        dataGridViewLivres.DataSource = bindingSourceCommandesLivre;
        foreach (CommandeDocument commande in lesCommandesDocument)
        {
            Console.WriteLine(" remplircommandeslivres liste id " + commande.Id + "date " + commande.Date + "libelle " + commande.Suivi.Libelle);
        }
        dataGridViewLivres.Columns["id"].Visible = true;
        dataGridViewLivres.Columns["idLivreDvd"].Visible = false;
        dataGridViewLivres.Columns["idSuivi"].Visible = false;
        dataGridViewLivres.Columns["Suivi"].Visible = false;
        dataGridViewLivres.Columns["Libelle"].Visible = false;
        dataGridViewLivres.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
        dataGridViewLivres.Columns["date"].DisplayIndex = 4;
        dataGridViewLivres.Columns["montant"].DisplayIndex = 1;
        dataGridViewLivres.Columns["id"].DisplayIndex = 0;
        dataGridViewLivres.Columns[4].HeaderCell.Value = "Date de commande";
        dataGridViewLivres.Columns[1].HeaderCell.Value = "Nombre d'exemplaires";

        dataGridViewLivres.Columns["LibelleSuivi"].HeaderText = "Suivi";


    }
    else
    {
        bindingSourceCommandesLivre.DataSource = null;
    }
}
```

Les informations sur le livre concernés sont également affichées dans les différent espaces prévus à cet effet grâce à :

```
/// <summary>
/// afficher les information sur le livre recherché
/// </summary>
/// <param name="livre"></param>
1 référence
private void AfficheLivreCommandeInfos(Livre livre)
{
    textBoxTitre.Text = livre.Titre;
    textBoxAuteur.Text = livre.Auteur;
    textBoxCollection.Text = livre.Collection;
    textBoxGenre.Text = livre.Genre;
    textBoxPublic.Text = livre.Public;
    textBoxRayon.Text = livre.Rayon;

    string image = livre.Image;
    try
    {
        pictureBoxLivre.Image = Image.FromFile(image);
    }
    catch
    {
        pictureBoxLivre.Image = null;
    }


}
```

Le paramètre livre ayant été recherché avec l'ID dans la liste des livres qui a été récupérée dans la BDD par l'API au préalable.

2. **Changement de l'étape de suivi**

En cliquant sur l'une des commandes dans la liste affichée, on déclenche la méthode suivante :

```
//permet de suivre le numero de commande selectionné
string selectedCommandeDocument = null;
/// <summary>
/// permet de selectionnner une commande
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void dataGridViewLivres_CellClick(object sender, DataGridViewCellEventArgs e)
{
    if (e.RowIndex >= 0)
    {
        // Accéder à la ligne où le clic a eu lieu
        DataGridViewRow row = dataGridViewLivres.Rows[e.RowIndex];


        string idcommande = row.Cells[3].Value.ToString();
        selectedCommandeDocument = idcommande;
        numeroSuivi.Text = idcommande;
        comboBoxSuivi.SelectedIndex = 2;
        try
        {
            string libelle = row.Cells[8].Value.ToString();
            switch (libelle)
            {
                case "En cours":
                    comboBoxSuivi.SelectedIndex = 0;
                    break;
                case "Relancée":
                    comboBoxSuivi.SelectedIndex = 1;
                    break;
                case "Livrée":
                    comboBoxSuivi.SelectedIndex = 2;
                    break;
                case "Réglée":
                    comboBoxSuivi.SelectedIndex = 3;
                    break;
            }
            Console.WriteLine("Vous avez cliqué sur la commande : " + idcommande);
            Console.WriteLine("Le suivi est  : " + libelle);
        }
        catch
        {
            Console.WriteLine("erreur de selected value");
        }

    }
}
```

On cherche et enregistre sans selectedCommandeDocument le numéro de la commande sur laquelle on a cliqué.

La méthode va également rechercher l'étape de suivi de la commande dans la datagridview et changer l'index sélectionné de la combobox du cadre suivi afin que l'option affichée reflète l'état actuel.

Du côté de ce cadre de suivi, il nous suffit alors une fois une commande sélectionnée de choisir une option afin que le changement se reflète dans la base de données.
En effet, changer l'option de la combobox déclenche la méthode suivante :

```
1 référence
private void comboBoxSuivi_DropDown(object sender, EventArgs e)
{
    canUpdateSuivi = true;
}
/// <summary>
/// Change l'etape de suivi dans la bdd
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void comboBoxSuivi_SelectedIndexChanged(object sender, EventArgs e)
{
    if (numeroSuivi.Text != null && numeroSuivi.Text != "" && canUpdateSuivi == true)
    {
        string numSuivi = numeroSuivi.Text;


        string libelle = comboBoxSuivi.Text;
        controller.ChangeSuivi(numSuivi, libelle);
        Console.WriteLine("The combobox selected suivi is " + libelle);
        canUpdateSuivi = false;

        foreach (CommandeDocument commande in lesCommandesDocument)
        {
            if (commande.Id == numSuivi)
            {
                commande.Suivi.Libelle = libelle;
                Console.WriteLine("commade document modifiée " + commande.Id);
                break;
            }

        }
        bindingSourceCommandesLivre.DataSource = lesCommandesDocument;
        bindingSourceCommandesLivre.ResetBindings(false);

        dataGridViewLivres.DataSource = bindingSourceCommandesLivre;
    }
}
```

En utilisant le numéro de suivi et la valeur nouvellement enregistrée on vas utiliser par l'intermédiaire du controller la méthode suivante dans Access :

```
/// <summary>
/// met a jour l'étape de suivi
/// </summary>
/// <param name="numSuivi"></param>
/// <param name="step"></param>
/// <returns></returns>
1 référence
public bool ChangeSuivi(string numSuivi, string step)
{
    try
    {
        var SuiviData = new
        {

            id = numSuivi,
            libelle = step
        };
        string jsonUpdateSuivi = JsonConvert.SerializeObject(SuiviData);
        List<Suivi> suivis = TraitementRecup<Suivi>(PUT, uriApi + "suivi", jsonUpdateSuivi);

        return (suivis != null && suivis
                .Count > 0);
    }
    catch(Exception ex)
    {
        Console.WriteLine("Erreur lors de la modification du suivi: " + ex.Message);
        return false;
    }

}
```

On envoie ainsi une requête post qui sera récupérée par la méthode suivante qui récupère les informations dans le jsonbody et l'URL de la requête :

```
case 'POST':

        //$input = filter_input_array(INPUT_GET, $args);
    $urlPath = $_SERVER['REQUEST_URI'];
    //print_r($urlPath);
    // Division l'URL en segments
        $urlSegments = explode('/', $urlPath);
         $jsonBody = json_decode(file_get_contents('php://input'), true);
         //print_r("jsonbody");
         //print_r($jsonBody);
         //print_r("jsonbodyfin");
    //debut new
        //$table = $input['table'] ?? '';
        // = $jsonBody['id'] ?? []; // Idem
        //print_r("id is ");
        //print_r($jsonBody['id']);
        //$champs = $jsonBody;
      //print_r($jsonBody);
         $table = $urlSegments[2];
         //print_r("table name is ");
         //print_r($table);
       if (is_array($jsonBody)) {
          //fin new

        $controle->post($table, $jsonBody);
    } else {
        // cas où $jsonBody n'est pas un tableau

        echo "Le corps de la requête JSON n'est pas un tableau.";
            }
        break;
```

Une fois les informations récupérées, on finit par créer la requête à l'aide de la méthode :

```php
/**
 * modification d'une ligne dans une table
 * @param string $table nom de la table
 * @param string $id id de la ligne à modifier
 * @param array $param nom et valeur de chaque champs de la li
 * @return true si la modification a fonctionné
 */
public function updateOne($table, $champs){
    if($this->conn != null && $champs != null){

        // construction de la requête
        $id =$champs["id"];
        $requete = "update $table set ";
        foreach ($champs as $key => $value){
            $requete .= "$key=:$key,";
        }
        // (enlève la dernière virgule)
        $requete = substr($requete, 0, strlen($requete)-1);
        //$champs["id"] = $id;
        $requete .= " where id=:id;";
      //    echo "SQL Query: " . $requete . "\n";
        //print_r($champs);
        return $this->conn->execute($requete, $champs);
    }else{
        return null;
    }
}
```

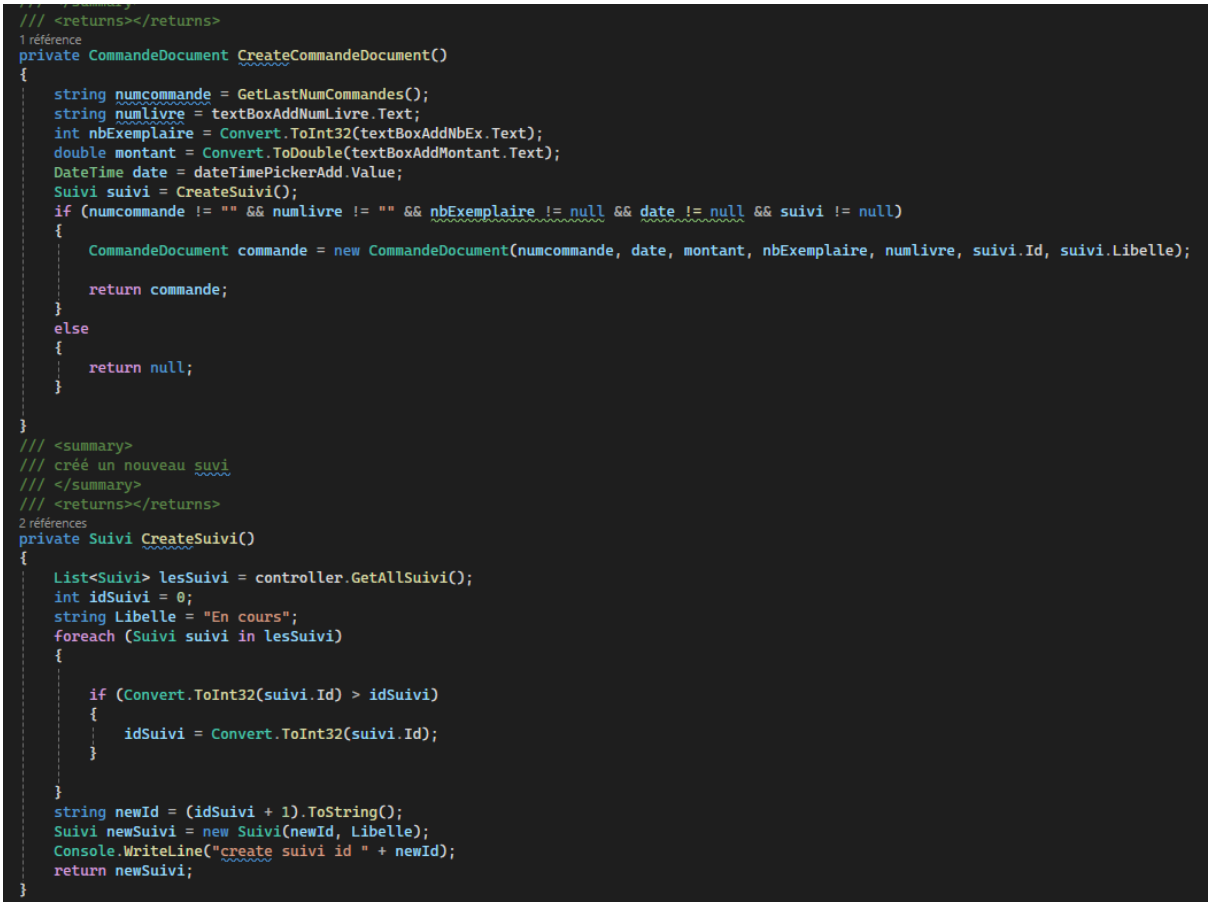
Le changement est à présent effectif dans la base de données.

3. Création de commande

Le cadre de création de commande fonctionne de la manière suivante : il suffit d'entrer les informations dans les champs prévus à cet effet : numéro de commande, nombre d'exemplaires, montant et date de commande.
En appuyant sur le bouton "ajouter une nouvelle commande", on déclenche la méthode suivante :

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

    if (!string.IsNullOrEmpty(textBoxAddNumLivre.Text) &&
        !string.IsNullOrEmpty(textBoxAddMontant.Text) &&
        !string.IsNullOrEmpty(textBoxAddNbEx.Text))

    {
        bool livreExsite = false;
        string numlivre = null;
        foreach (Livre livre in lesLivres)
        {
            if (textBoxAddNumLivre.Text == livre.Id)
            {
                livreExsite = true;
                numlivre = livre.Id;
                break;
            }
        }
        if (livreExsite == true)
        {
            Console.WriteLine("clik validé xxxxxxxxxxxxxxxxxxx");
            CommandeDocument commande = CreateCommandeDocument();

            controller.AddCommandeDocument(commande);

            textBoxNumDoc.Text = numlivre;
            buttonRechercheNumDoc_Click(buttonRechercheNumDoc, new EventArgs());
        }
        else
        {
            MessageBox.Show("Livre non exsitant.");
        }

    }
    else
    {
        MessageBox.Show("Veuillez remplir toutes les informations sur la commande.");

    }


}
```

Cette méthode permet de vérifier que les informations nécessaires sont présentes et correctes, auquel cas on utilise ensuite la méthode CreateCommande document pour créer les commandes puis une fois l'objet créée on l'ajoute à la BDDen utilisant la méthode addcommandedocument .



```
/// <returns></returns>
1 référence
private CommandeDocument CreateCommandeDocument()
{
    string numcommande = GetLastNumCommandes();
    string numlivre = textBoxAddNumLivre.Text;
    int nbExemplaire = Convert.ToInt32(textBoxAddNbEx.Text);
    double montant = Convert.ToDouble(textBoxAddMontant.Text);
    DateTime date = dateTimePickerAdd.Value;
    Suivi suivi = CreateSuivi();
    if (numcommande != "" && numlivre != "" && nbExemplaire != null && date != null && suivi != null)
    {
        CommandeDocument commande = new CommandeDocument(numcommande, date, montant, nbExemplaire, numlivre, suivi.Id, suivi.Libelle);

        return commande;
    }
    else
    {
        return null;
    }

}
/// <summary>
/// créé un nouveau suvi
/// </summary>
/// <returns></returns>
2 références
private Suivi CreateSuivi()
{
    List<Suivi> lesSuivi = controller.GetAllSuivi();
    int idSuivi = 0;
    string Libelle = "En cours";
    foreach (Suivi suivi in lesSuivi)
    {

        if (Convert.ToInt32(suivi.Id) > idSuivi)
        {
            idSuivi = Convert.ToInt32(suivi.Id);
        }

    }
    string newId = (idSuivi + 1).ToString();
    Suivi newSuivi = new Suivi(newId, Libelle);
    Console.WriteLine("create suivi id " + newId);
    return newSuivi;
}
```

Ces méthodes récupèrent les informations entrées dans l'interface pour créer la CommandeDocument et le Suivi associé que l'on utilise ensuite en paramètre dans la méthode suivante :

```
1 référence
public bool AddCommandeDocument(CommandeDocument commandeDocument)
{


    CommandeDocument commande = commandeDocument;
    Suivi suivi = commandeDocument.Suivi;
    Console.WriteLine("addcommande document verification " + suivi.Libelle);

    try
    {
        var commandeData1 = new
        {
            id = commande.Id,
            dateCommande = commande.Date.ToString("yyyy-MM-dd HH:mm:ss"),
            montant = commandeDocument.Montant

        };
        // Création de l'objet à envoyer
        var commandeData = new
        {
            id = commande.Id,
            nbExemplaire = commande.NbExemplaire,
            idLivreDvd = commande.IdLivreDvd,
            date = commande.Date.ToString("yyyy-MM-dd HH:mm:ss"), // Assurez-vous du format de date attendu par l'API
            montant = commande.Montant,
            idsuivi = commande.Suivi.Id
        };

        var suiviData = new
        {
            id = suivi.Id,
            libelle = suivi.Libelle

        };

        string jsonCreerCommande = JsonConvert.SerializeObject(commandeData1);
        // Sérialisation de l'objet en JSON
        string jsonCreerCommandeDocument = JsonConvert.SerializeObject(commandeData);
        Console.WriteLine("jsonCreerCommandeDocument: " + jsonCreerCommandeDocument);

        string jsonCreerSuivi = JsonConvert.SerializeObject(suiviData);
        Console.WriteLine("jsonCreerSuivi: " + jsonCreerSuivi);

        // Envoi de la requête POST
        List<CommandeDocument> listeSuivi = TraitementRecup<CommandeDocument>(POST, uriApi + "suivi" , jsonCreerSuivi);
        List<CommandeDocument> listeCommande = TraitementRecup<CommandeDocument>(POST, uriApi + "commande" , jsonCreerCommande);
        List<CommandeDocument> listeCommandeDocument = TraitementRecup<CommandeDocument>(POST, uriApi+"commandedocument" , jsonCreerCommandeDocument);

        // Vérifiez ici le succès de l'opération (peut-être en vérifiant la taille de la liste ou d'autres indicateurs)
        return (listeCommandeDocument != null && listeCommandeDocument
            .Count > 0);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Erreur lors de l'ajout d'une commande document: " + ex.Message);
        return false;
    }
}
```

On utilise les références présentes dans notre commande document pour créer des fichier json, un pour la commandeDocument et un pour le suivi associé.
La méthode envoie ensuite deux requêtes POST permettant d'ajouter ces éléments dans la BDD.

Enfin, le bouton “supprimer commande” permet de supprimer la commande sélectionnée dans la datagridview en envoyant une requête DELETE à l’API :

```php
/**
 * suppresion d'une ou plusieurs lignes dans une table
 * @param string $table nom de la table
 * @param array $champs nom et valeur de chaque champs
 * @return true si la suppression a fonctionné
 */
  public function delete($table, $champs){
    if($this->conn != null){
        // construction de la requête
        //print_r("champs  ");
        //print_r($champs);
        // print_r(" champs fin");
        $requete = "delete from $table where ";
        foreach ($champs as $key => $value){
            $requete .= "$key=:$key and ";
        }
        // (enlève le dernier and)
        $requete = substr($requete, 0, strlen($requete)-5);
        //print_r("apirequetedbut ");
        //print_r($requete);
        // print_r(" apirequetedbut fin");
        return $this->conn->execute($requete, $champs);
    }else{
        return null;
    }
}
```

4. Tri par colonne dans les datagridview

En cliquant sur les en-têtes de la datagridview on déclenche la méthode permettant le tri.

```
bool orderLivreASC = true;
1 référence
private void dataGridViewLivres_ColumnHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)
{


    string titreColonne = dataGridViewLivres.Columns[e.ColumnIndex].HeaderText;
    List<CommandeDocument> sortedList = new List<CommandeDocument>();
    switch (titreColonne)
    {
        case "Id":
            if (orderLivreASC == true)
            {
                sortedList = lesCommandesDocument.OrderBy(o => o.Id).ToList();
                orderLivreASC = false;
            }
            else
            {
                sortedList = lesCommandesDocument.OrderByDescending(o => o.Id).ToList();
                orderLivreASC = true;
            }
            break;
        case "Date de commande":
            if (orderLivreASC == true)
            {
                sortedList = lesCommandesDocument.OrderBy(o => o.Date).ToList();
                orderLivreASC = false;
            }
            else
            {
                sortedList = lesCommandesDocument.OrderByDescending(o => o.Date).ToList();
                orderLivreASC = true;
            }
            break;
        case "Montant":
            if (orderLivreASC == true)
            {
                sortedList = lesCommandesDocument.OrderBy(o => o.Montant).ToList();
                orderLivreASC = false;
            }
            else
            {
                sortedList = lesCommandesDocument.OrderByDescending(o => o.Montant).ToList();
                orderLivreASC = true;
            }
            break;
        case "NbExemplaire":
            if (orderLivreASC == true)
            {
                sortedList = lesCommandesDocument.OrderBy(o => o.NbExemplaire).ToList();
                orderLivreASC = false;
            }
            else
            {
                sortedList = lesCommandesDocument.OrderByDescending(o => o.NbExemplaire).ToList();
                orderLivreASC = true;
            }
            break;

    }
```

Tâche 2 : les commandes de revues

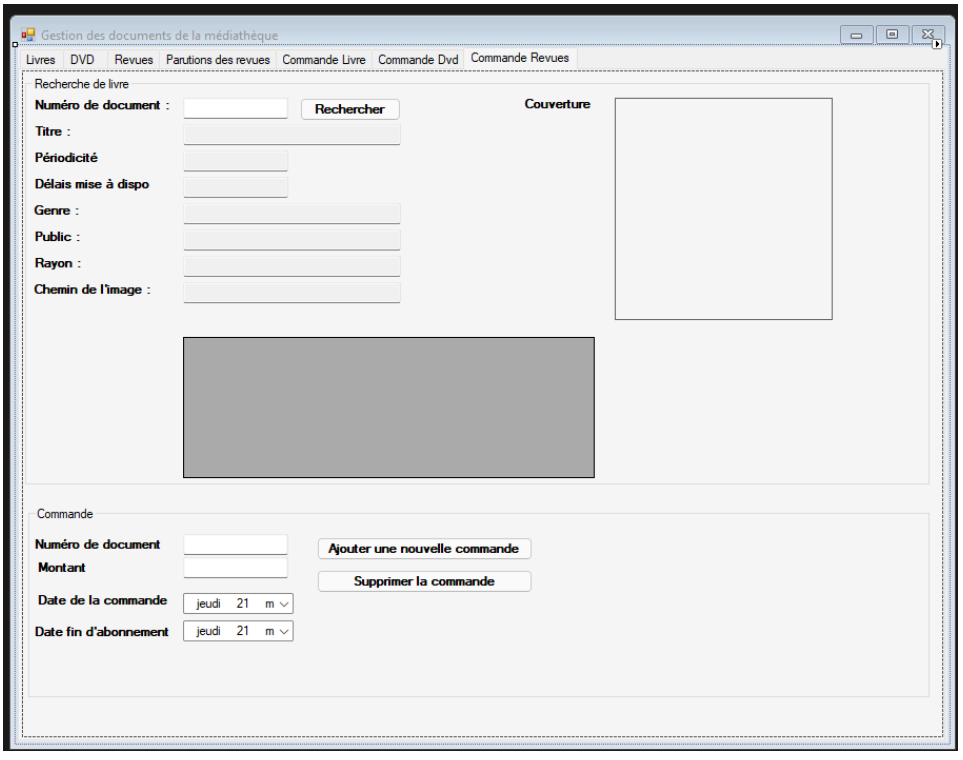
Gérer les commandes de revues #2

Open jnyazbek/mediatekdocuments Public

jnyazbek opened this issue now

Tâche 2 : gérer les commandes de revues (4h)
Créer un onglet (ou une fenêtre) pour gérer les commandes de revues : une commande représente un nouvel abonnement ou le renouvellement d'un abonnement. Dans le cas d'un nouvel abonnement, la revue sera préalablement créée dans l'onglet Revues. Donc, dans l'onglet des commandes de revues, il n'y a pas de distinction entre un nouvel abonnement et un renouvellement.
La charte graphique doit correspondre à l'existant.
Dans toutes les listes, permettre le tri sur les colonnes.
Permettre la sélection d'une revue par son numéro, afficher les informations de la revue ainsi que la liste des commandes (abonnements), triée par date (ordre inverse de la chronologie). La liste doit comporter les informations suivantes : date de la commande, montant et date de fin d'abonnement.
Créer un groupbox qui permet de saisir les informations d'une nouvelle commande (nouvel abonnement ou renouvellement, le principe est identique) et de l'enregistrer.
Permettre de supprimer une commande de revue uniquement si aucun exemplaire n'est rattaché (donc, en vérifiant la date de l'exemplaire, comprise entre la date de la commande et la date de fin d'abonnement). Pour cela, créer et utiliser la méthode 'ParutionDansAbonnement' qui reçoit en paramètre 3 dates (date commande, date fin abonnement, date parution) et qui retourne vrai si la date de parution est entre les 2 autres dates. Créer le test unitaire sur cette méthode.
Toutes les sécurités seront mises en place pour éviter des erreurs de manipulation.
Créer une méthode qui permet d'obtenir la liste des revues dont l'abonnement se termine dans moins de 30 jours. Dès l'ouverture de l'application, ouvrir une petite fenêtre d'alerte rappelant la liste de ces revues (titre et date de fin abonnement) triée sur la date dans l'ordre chronologique.

On commence par créer l'interface utilisateur avec windows forms :



Cet onglet permet la recherche de revue afin de visualiser les informations qui la concernent ainsi que les commandes qui lui sont attribuées.
Le cadre commande permet également de créer une commande ou d'en supprimer dans le cas où aucun exemplaire n'y est attaché.

Les commandes de revue sont différentes des commandes de documents (DVD et livre) : il nous faut donc créer la classe Abonnement qui héritera aussi de commande et ajouter la table dans la base de données.

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	id	varchar(5)	utf8mb4_0900_ai_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
2	dateFinAbonnement	datetime			Oui	*NULL*			Modifier Supprimer Plus
3	idRevue	varchar(10)	utf8mb4_0900_ai_ci		Non	*Aucun(e)*			Modifier Supprimer Plus

```
38 références
public class Abonnement : Commande
{
    /// <summary>
    /// Récupère ou définit la date de fin de l'abonnement
    /// </summary>
    10 références
    public DateTime DateFinAbonnement { get; set; }

    /// <summary>
    /// Récupère ou définit l'identifiant de la revue correspondant à l'abonnement
    /// </summary>
    6 références
    public string IdRevue { get; set; }

    /// <summary>
    /// Récupère ou définit le titre de la revue correspondant à l'abonnement
    /// </summary>
    0 références
    public string Titre { get; set; }

    /// <summary>
    /// Initialisation d'un nouvel objet Abonnement
    /// </summary>
    /// <param name="id">Id de l'abonnement</param>
    /// <param name="dateCommande">Date de la commande de l'abonnement</param>
    /// <param name="montant">Montant de l'abonnement</param>
    /// <param name="dateFinAbonnement">Date de fin de l'abonnement</param>
    /// <param name="idRevue">Id de la revue concernée par l'abonnement</param>

    2 références
    public Abonnement(string id, DateTime dateCommande, double montant, DateTime dateFinAbonnement, string idRevue) :
        base(id, dateCommande, montant,null,null)
    {
        this.DateFinAbonnement = dateFinAbonnement;
        this.IdRevue = idRevue;

    }
}
```

1. **Recherche et affichage des informations et des commandes**

En double-cliquant sur le bouton “Recherche”, on crée une écoute sur l’évènement sur ce bouton. On ajoute donc dans ce delegate la méthode suivante qui sera appelée à chaque fois que l'événement sera déclenché :

```
/// <summary>
/// recherche et affiche les information sur la revue séléctionnée
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
2 références
private void buttonRechercheRevue_Click(object sender, EventArgs e)
{
    string numdoc = textBoxNumRevue.Text;
    Console.WriteLine("button recherche num dvd " + numdoc);
    if (numdoc != null)
    {
        lesCommandesRevues = controller.GetCommandesRevue(numdoc);
        foreach (Abonnement commande in lesCommandesRevues)
        {
            Console.WriteLine("ID: " + commande.Id  + ", ID Revue: " + commande.IdRevue
        }

        RemplirCommandesRevueListe(lesCommandesRevues);
    }


    if (!textBoxNumRevue.Text.Equals(""))
    {
        Revue revue = lesRevues.Find(x => x.Id.Equals(numdoc));
        if (revue != null)
        {
            AfficheRevueCommandeInfos(revue);
        }
        else
        {
            MessageBox.Show("numéro introuvable");
        }
    }
}
```

Si l'identifiant de revue entré est valide, une requête est envoyée à la BDD par l'API afin de récupérer les commandes associées à la revue ainsi que les informations sur la revue de la même façon que pour les commandes de livres :

```
/// <summary>
/// affiche les commandes correspondant a la revue selectionnée
/// </summary>
/// <param name="lesCommandesRevue"></param>
3 références
private void RemplirCommandesRevueListe(List<Abonnement> lesCommandesRevue)
{
    if (lesCommandesRevue != null)
    {
        bindingSourceCommandesRevue.DataSource = lesCommandesRevue;
        dataGridViewRevue.DataSource = bindingSourceCommandesRevue;
        foreach (Abonnement commande in lesCommandesRevue)
        {
            Console.WriteLine(" remplircommandeslivres liste id " + commande.Id + "date " + commande.Date
        }

        dataGridViewRevue.Columns["id"].Visible = false;
        dataGridViewRevue.Columns["idRevue"].Visible = false;
        dataGridViewRevue.Columns["titre"].Visible = false;
        dataGridViewRevue.Columns["suivi"].Visible = false;
        dataGridViewRevue.Columns["LibelleSuivi"].Visible = false;
        dataGridViewRevue.Columns["Libelle"].Visible = false;
        dataGridViewRevue.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
        dataGridViewRevue.Columns["date"].DisplayIndex = 0;
        dataGridViewRevue.Columns["montant"].DisplayIndex = 1;
        dataGridViewRevue.Columns[4].HeaderCell.Value = "Date de commande";
        dataGridViewRevue.Columns[0].HeaderCell.Value = "Date de fin d'abonnement";



    }
    else
    {
        bindingSourceCommandesRevue.DataSource = null;
    }
}
```

```
/// <summary>
/// remplis les champs d'information sur la revue
/// </summary>
/// <param name="revue"></param>
1 référence
private void AfficheRevueCommandeInfos(Revue revue)
{
    textBoxTitreRevue.Text = revue.Titre;
    textBoxPeriodeRevue.Text = revue.Periodicite;
    textBoxDelaisRevue.Text = revue.DelaiMiseADispo.ToString();
    textBoxGenreRevue.Text = revue.Genre;
    textBoxPublicRevue.Text = revue.Public;
    textBoxRayonRevue.Text = revue.Rayon;



}
```

La deuxième partie permet la création et la suppression de commandes de revue.
On remplit les informations dans les champs prévus à cet effet, puis on clique sur le bouton “ajouter une commande” qui déclenche la méthode :

```
1 référence
private void buttonAddCommandeRevue_Click(object sender, EventArgs e)
{
    if (!string.IsNullOrEmpty(textBoxNumRevueAdd.Text) &&
      !string.IsNullOrEmpty(textBoxMontantRevueAdd.Text) &&
      dateTimePickerCommandeRevue.Value < dateTimePickerFinAbo.Value)

    {
        bool revueExsite = false;
        string numrevue = null;
        foreach (Revue revue in lesRevues)
        {
            if (textBoxNumRevueAdd.Text == revue.Id)
            {
                revueExsite = true;
                numrevue = revue.Id;
                break;
            }
        }
        if (revueExsite == true)
        {
            Console.WriteLine("clik validé xxxxxxxxxxxxxxxxxxx");
            Abonnement commande = CreateCommandeRevue();

            controller.AddCommandeRevue(commande);

            textBoxNumRevue.Text = numrevue;
            buttonRechercheRevue_Click(buttonRechercheRevue, new EventArgs());
        }
        else
        {
            MessageBox.Show("Revue non exsitante.");
        }

    }
    else
    {
        MessageBox.Show("Veuillez remplir toutes les informations sur la commande.");

    }
}
```

Si les informations sont valides, on créé la commande de revue à l'aide de la méthode suivante :

```
/// <summary>
/// créée un abonnement à une revue
/// </summary>
/// <returns></returns>
1 référence
private Abonnement CreateCommandeRevue()
{
    string numcommande = GetLastNumCommandes();
    string numlivre = textBoxNumRevueAdd.Text;
    double montant = Convert.ToDouble(textBoxMontantRevueAdd.Text);
    DateTime date = dateTimePickerCommandeRevue.Value;
    DateTime dateFinAbo = dateTimePickerFinAbo.Value;
    if (numcommande != "" && numlivre != "" && date != null && dateFinAbo != null && montant != null)
    {
        Abonnement commande = new Abonnement(numcommande,date,montant,dateFinAbo,numlivre);

        return commande;
    }
    else
    {
        return null;
    }

}
```

Puis, on appelle la AddCommande Revue dans le controller qui lui-même fait appel à la la méthode suivante dans Access :

```
1 référence
public bool AddCommandeRevue(Abonnement commande)
{

    try
    {
        var aboData = new
        {
            id = commande.Id,
            dateFinAbonnement = commande.DateFinAbonnement.ToString("yyyy-MM-dd HH:mm:ss"),
            idRevue = commande.IdRevue

        };
        // Création de l'objet à envoyer
        var commandeData = new
        {
            id = commande.Id,
            dateCommande = commande.Date.ToString("yyyy-MM-dd HH:mm:ss"), // Assurez-vous du format de date attendu par l'API
            montant = commande.Montant,

        };



        string jsonCreerAbo = JsonConvert.SerializeObject(aboData);
        // Sérialisation de l'objet en JSON
        string jsonCreerCommande = JsonConvert.SerializeObject(commandeData);
        Console.WriteLine("jsonCreerCommandeDocument: " + jsonCreerAbo);



        // Envoi de la requête POST
        List<Commande> listeCommande = TraitementRecup<Commande>(POST, uriApi + "commande", jsonCreerCommande);
        List<Abonnement> listeCommandesRevue = TraitementRecup<Abonnement>(POST, uriApi + "abonnement", jsonCreerAbo);

        // Vérifiez ici le succès de l'opération (peut-être en vérifiant la taille de la liste ou d'autres indicateurs)
        return (listeCommandesRevue != null && listeCommande
            .Count > 0);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Erreur lors de l'ajout d'une commande document: " + ex.Message);
        return false;
    }
}
```

On utilise les informations de l'abonnement, en en faisant un json qui va constituer le corps de la requête POST envoyée à l'API.

2. Création et suppression d'abonnement

La création d'abonnement se fait de la même façon que pour les commandes de document.
Cependant,pour la suppression, il est impossible de supprimer un abonnement auquel est attaché au moins un exemplaire.

ParutionDansAbonnement permet de vérifier si un exemplaire est paru sur la période d'un abonnement.
Tandis-que Exemplairecheck prend un Abonnement en paramètre et utilise la méthode précédente pour déterminer si au moins un exemplaire est rattaché à l'abonnement.

```
public bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFinAbonnement, DateTime dateParution)
{
    return (DateTime.Compare(dateCommande, dateParution) < 0 && DateTime.Compare(dateParution, dateFinAbonnement) < 0);
}
/// <summary>
/// vérifie si l'abonnement est en lien avec des exemplaires dans la bdd
/// </summary>
/// <param name="abonnement"></param>
/// <returns></returns>
1 référence
private bool ExemplaireCheck(Abonnement abonnement)
{
    List<Exemplaire> lesExemplairesAbonnement = controller.GetExemplairesRevue(abonnement.IdRevue);
     bool datedeparution = false;
    foreach (Exemplaire exemplaire in lesExemplairesAbonnement.Where(exemplaires => ParutionDansAbonnement(abonnement.Date, abonnement.DateFinAbonnement, exemplaires.DateAchat)))
    {
        datedeparution = true;

    }
    return !datedeparution;
}
```

Si aucun exemplaire n'est associé alors la bool canDelete = true est la suppression peut se faire.

```
1 référence
private void buttonDeleteCommandeRevue_Click(object sender, EventArgs e)
{

    foreach (Abonnement commande in lesCommandesRevues)
    {
        Console.WriteLine("seleted commande id is " + selectedCommandeDocument + " and this command id is " + commande.Id);
        if (commande.Id == selectedCommandeDocument)
        {
            bool canDelete = ExemplaireCheck(commande);

            if (canDelete)
            {
                Console.WriteLine("Delete commande numùber " + commande.Id);

                controller.DeleteCommandeRevue(commande);
                lesCommandesRevues.Remove(commande);
                bindingSourceCommandesRevue.DataSource = lesCommandesRevues;
                bindingSourceCommandesRevue.ResetBindings(false);

                dataGridViewRevue.DataSource = bindingSourceCommandesRevue;
                selectedCommandeDocument = null;

            }
            else
            {
                Console.WriteLine("Impossible de supprimer " + commande.Id);
                MessageBox.Show("Impossible de supprimer un abonnement avec des exemplaires");
            }


            break;
        }
    }
}
```

Enfin, il nous faut mettre en place un système prévenant l'utilisateur des abonnements touchant à leur fin dans les 30 jours. On utilise alors :

```
1 référence
public List<Abonnement> abonnementsEnFin()
{
    List<Abonnement> lesAbos = controller.GetAllCommandesRevue();

    // Filtrer les abonnements dont la date de fin est à moins de 30 jours
    List<Abonnement> abonnementsProchesDeFin = lesAbos.Where(abonnement =>
    abonnement.DateFinAbonnement <= DateTime.Now.AddDays(30) && abonnement.DateFinAbonnement >= DateTime.Now).ToList();
    return abonnementsProchesDeFin;
}
/// <summary>
/// Ouvre une pop up avertissement
/// </summary>
1 référence
public void DisplayFinAbo()
{
    List<Abonnement> lesAbosEnFin = abonnementsEnFin();
    List<Abonnement> lesAbosEnFinTri = lesAbosEnFin.OrderBy(a => a.DateFinAbonnement).ToList();
    List<(string Titre, DateTime DateFinAbonnement)> titresEtDates = new List<(string, DateTime)>();

    foreach (Abonnement abonnement in lesAbosEnFinTri)
    {
        foreach (Revue revue in lesRevues)
        {
            if (revue.Id == abonnement.IdRevue)
            {
                titresEtDates.Add((revue.Titre, abonnement.DateFinAbonnement));
            }
        }
    }

    StringBuilder message = new StringBuilder();
    message.AppendLine("Les abonnements suivants arrivent à échéance :");
    foreach (var (Titre, DateFinAbonnement) in titresEtDates)
    {
        message.AppendLine($"{Titre} - Fin le {DateFinAbonnement:dd/MM/yyyy}");
    }

    // Afficher le message dans une MessageBox
    MessageBox.Show(message.ToString(), "Alerte Abonnements en Fin", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

Mission 2 : mettre en place des authentifications

Mettre en place des authentifications #3

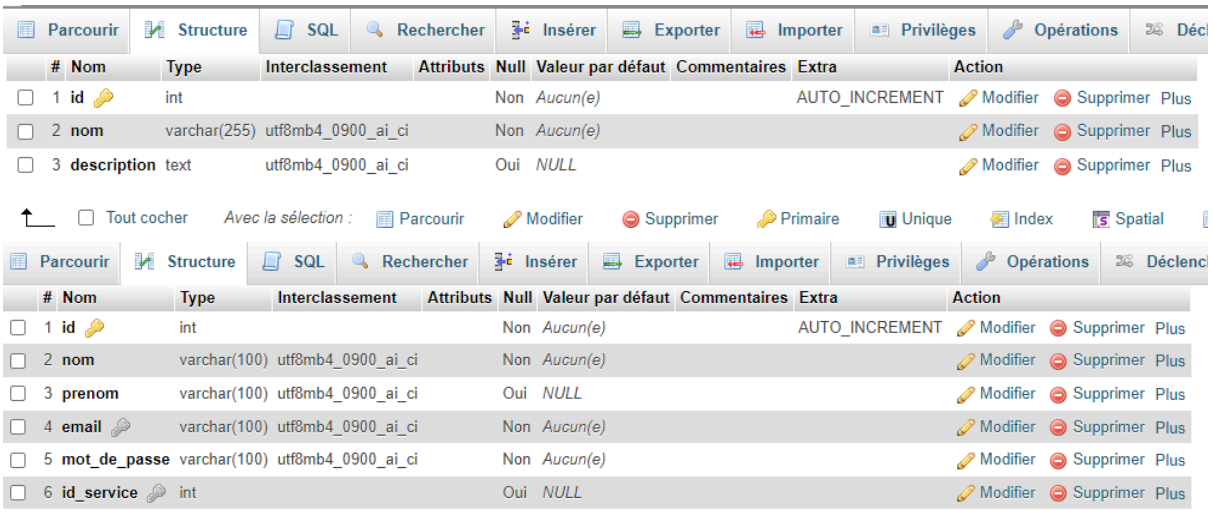
Open jnyazbek/mediatekdocuments Public

jnyazbek opened this issue now

Dans la base de données, ajouter une table Utilisateur et une table Service, sachant que chaque utilisateur ne fait partie que d'un service.
Pour réaliser les tests, remplir les tables d'exemples.
Ajouter une première fenêtre d'authentification. Faire en sorte que l'application démarre sur cette fenêtre.
Suivant le type de personne authentifiée, empêcher certains accès en rendant invisibles ou inactifs certains onglets ou objets graphiques.
Dans le cas du service Culture qui n'a accès à rien, afficher un message précisant que les droits ne sont pas suffisants pour accéder à cette application, puis fermer l'application.
Faire en sorte que l'alerte de fin d'abonnement n'apparaisse que pour les personnes concernées (qui gèrent les commandes).

Temps estimé : 4h
Temps réel : 8h

On commence par ajouter les tables utilisateurs et service dans la BDD car nous en aurons besoin par la suite pour l'authentification est les droits d'utilisation.



Parcourir | Structure | SQL | Rechercher | Insérer | Exporter | Importer | Privilèges | Opérations | Déc

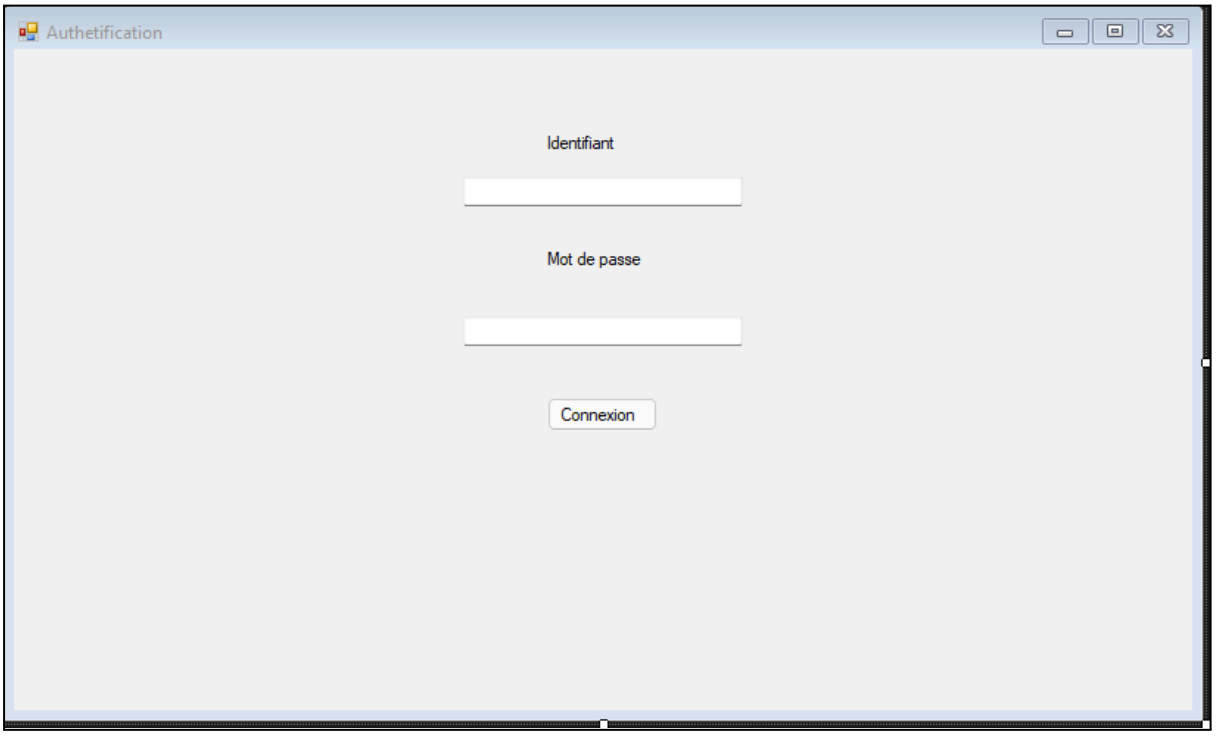
#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	id	int			Non	*Aucun(e)*		AUTO_INCREMENT	Modifier Supprimer Plus
2	nom	varchar(255)	utf8mb4_0900_ai_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
3	description	text	utf8mb4_0900_ai_ci		Oui	*NULL*			Modifier Supprimer Plus

Tout cocher *Avec la sélection :* Parcourir Modifier Supprimer Primaire Unique Index Spatial

Parcourir | Structure | SQL | Rechercher | Insérer | Exporter | Importer | Privilèges | Opérations | Déclenc

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	id	int			Non	*Aucun(e)*		AUTO_INCREMENT	Modifier Supprimer Plus
2	nom	varchar(100)	utf8mb4_0900_ai_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
3	prenom	varchar(100)	utf8mb4_0900_ai_ci		Oui	*NULL*			Modifier Supprimer Plus
4	email	varchar(100)	utf8mb4_0900_ai_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
5	mot_de_passe	varchar(100)	utf8mb4_0900_ai_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
6	id_service	int			Oui	*NULL*			Modifier Supprimer Plus

Il nous faut à présent créer la fenêtre d'authentification :



Nous devons également faire en sorte que ce soit cette fenêtre qui s'ouvre à l'ouverture du programme, il nous faut donc modifier la classe Program.cs :

```
static class Program
{
    /// <summary>
    /// Point d'entrée principal de l'application.
    /// </summary>
    [STAThread]
    0 références
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        // First, show the login form
        FormAuth loginForm = new FormAuth();
        DialogResult result = loginForm.ShowDialog();

        // Check the result
        if (result == DialogResult.OK)
        {
            Console.WriteLine(" main id is " + loginForm.ServiceId);
            // If login was successful, use the ServiceId from FormAuth
            Application.Run(new FrmMediatek(loginForm.ServiceId));
        }
        else
        {
            Application.Exit(); // Exit the application if login was not successful or was canceled
        }
    }
}
```

On crée un instance de FormAuth et on conditionne le lancement de l'application au succès de l'authentification et à l'obtention de l'ID de service.

On ajoute la méthode suivante au bouton pour effectuer une requête est vérifier que l'email entré existe et correspond bien au mot de passe :

```
1 référence
private void buttonConnexion_Click(object sender, EventArgs e)
{
    List<Utilisateur> users = controller.GetAllUtilisateurs();
    Utilisateur utilisateur = null;
    foreach(Utilisateur user in users)
    {
        Console.WriteLine("user " + user.Email + "comaperd to " + textBoxId.Text + "mdp "+ user.Mot_de_passe);
        if(user.Email == textBoxId.Text)
        {
            utilisateur = user;
        }
    }

    if(utilisateur != null)
    {
        Console.WriteLine("mdp " + textBoxMdp.Text + "compared to "+ utilisateur.Mot_de_passe);
        if (utilisateur.Mot_de_passe == textBoxMdp.Text)
        {
            this.ServiceId = utilisateur.Id_service; // Set the ServiceId based on the authenticated user
            Console.WriteLine("formauth the service id is  " + utilisateur.Id_service);
            this.DialogResult = DialogResult.OK; // Indicate success
            this.Close(); // Close the form


        }
        else
        {
            MessageBox.Show("mot de passe incorrect.");
        }
    }
    else
    {
        MessageBox.Show("Nom d'utilisateur inexistant.");
    }


}
```

Une fois l'authentification réussie, on ouvre la FrmMedeitak et on passe l'ID de service dans son constructeur :

```
1 référence
internal FrmMediatek(int service)
{
    InitializeComponent();
    this.controller = new FrmMediatekController();
    this.service = service;
    AuthorizationCheck();
    RemplirComboBoxSuivi();

}
```

Ce constructeur appelle la fonction suivante qui ajoute ou enlève des onglets en fonction du service auquel appartient l'utilisateur :

```
/// <summary>
/// vérifie les autorisations de l'utilisateur et adapte les onglets
/// </summary>
1 référence
public void  AuthorizationCheck()
{
    switch (service)
    {
        case 1:

            break;
        case 2:
            foreach(TabPage tab in frmOnglets.TabPages)
            {
                Console.WriteLine("tab found name is " + tab.Name);

            }
            frmOnglets.TabPages.Remove(tabPage1);
            frmOnglets.TabPages.Remove(tabCommandeDVD);
            frmOnglets.TabPages.Remove(tabCommandesRevue);
            groupBox7.Enabled = false;
            break;
        case 3:
            MessageBox.Show("Vous n'avez aucune autorisation");

            Environment.Exit(0);
            break;
    }
}
```

Mission 3 : assurer la sécurité, la qualité et intégrer des logs

Temps estimé : 8h
Temps réel : 12h

a) Tâche 1 : corriger les problèmes de sécurité

Corriger les problèmes de sécurité #4

Open — jnyazbek/mediatekdocuments — Public

jnyazbek opened this issue now

Tâche 1 : corriger des problèmes de sécurité (3h)
Problème n°1 :

Actuellement, l'accès à l'API se fait en authentification basique, avec le couple "login:pwd" en dur dans le code de l'application (dans le constructeur de la classe Access). Le but est de sécuriser cette information.

Problème n°2 :

Si, pour accéder à l'API directement dans un navigateur, on donne juste l'adresse de l'API sans mettre de paramètres :

http://localhost//rest_mediatekdocuments/

on obtient la liste des fichiers contenus dans le dossier de l'API. Le but est d'avoir un retour d'erreur.

Afin de résoudre le problème 1, on crée une méthode dans la classe ApiRest :

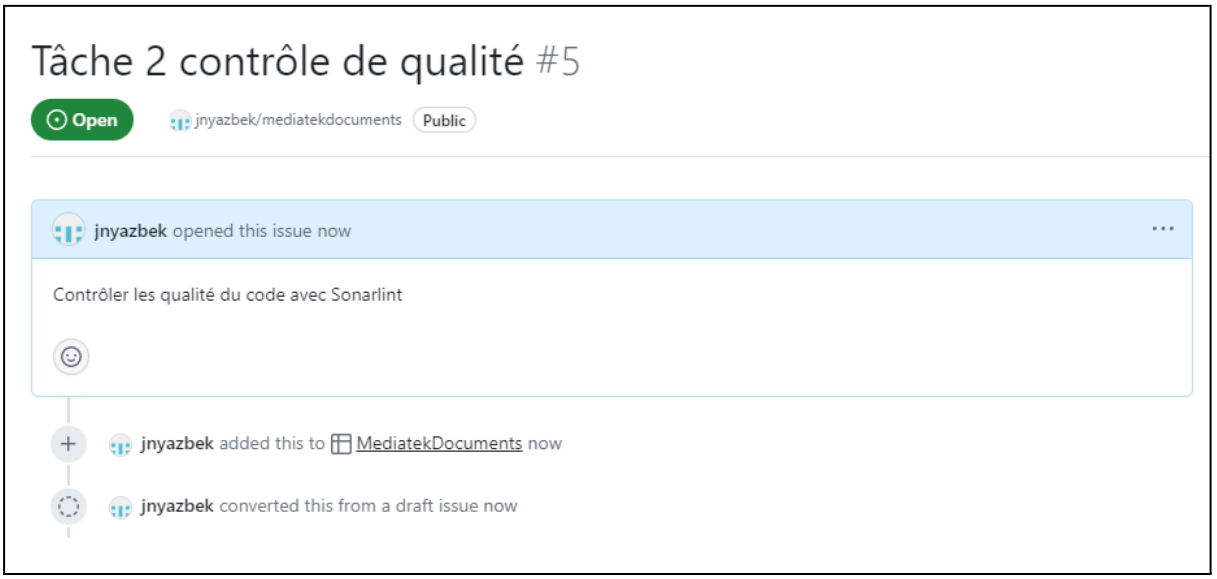
```
1 référence
private ApiRest(String uriApi, String authenticationString = "")
{
    httpClient = new HttpClient() { BaseAddress = new Uri(uriApi) };
    // prise en compte dans l'url de l'authentificaiton (basic authorization), si elle n'est pas vide
    if (!String.IsNullOrEmpty(authenticationString))
    {
        String base64EncodedAuthenticationString = Convert.ToBase64String(System.Text.ASCIIEncoding.ASCII.GetBytes(authenticationString));
        httpClient.DefaultRequestHeaders.Add("Authorization", "Basic " + base64EncodedAuthenticationString);
        Console.WriteLine("instance apirest initialisée "+ authenticationString );
    }
    else
    {
        Console.WriteLine("connectionstring null or empty " + authenticationString);
    }
}
```

Les connectionstring contenant l'utilisateur et le mot de passe d'accès à l'API est alors paramétré dans App.config :

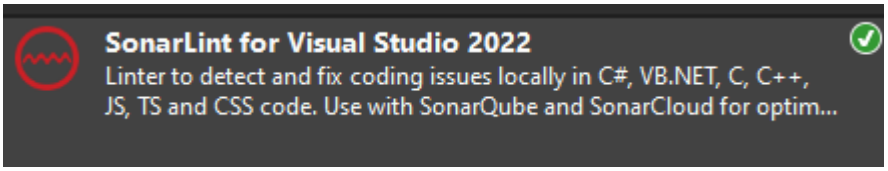
```
<connectionStrings>
    <add name="MediaTekDocuments.Properties.Settings.mediatek86ConnectionString" connectionString="
</connectionStrings>
```

Afin de résoudre le problème 2, il nous faut modifier le fichier htacces pour prendre en compte une route vide et en tenir compte dans le fichier mediatekdocuments.php. On y ajoute donc la règle :
RewriteRule ^$ mediatekdocuments.php?table=routevide [L,QSA]

b) Tâche 2 : contrôler la qualité

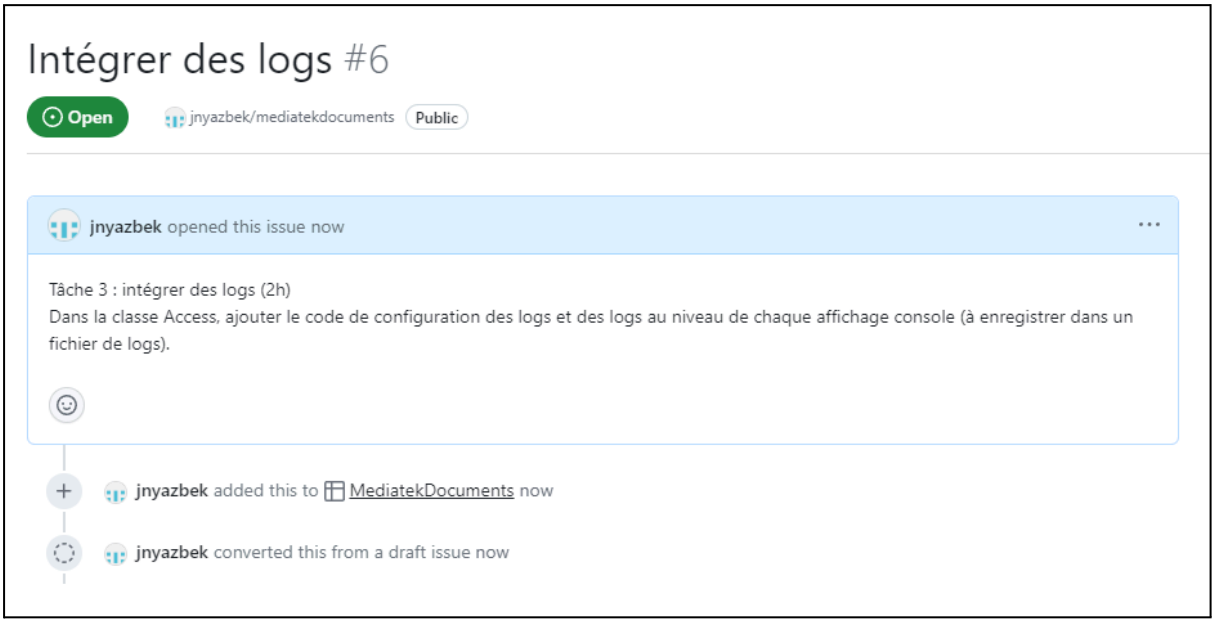


On utilise l'extension :



Qui permet d'analyser le code afin de le nettoyer et de résoudre les problèmes tel que la répétition des string ou la prévision des cas par défaut dans les switch.

c) Tâche 3 : intégration des logs



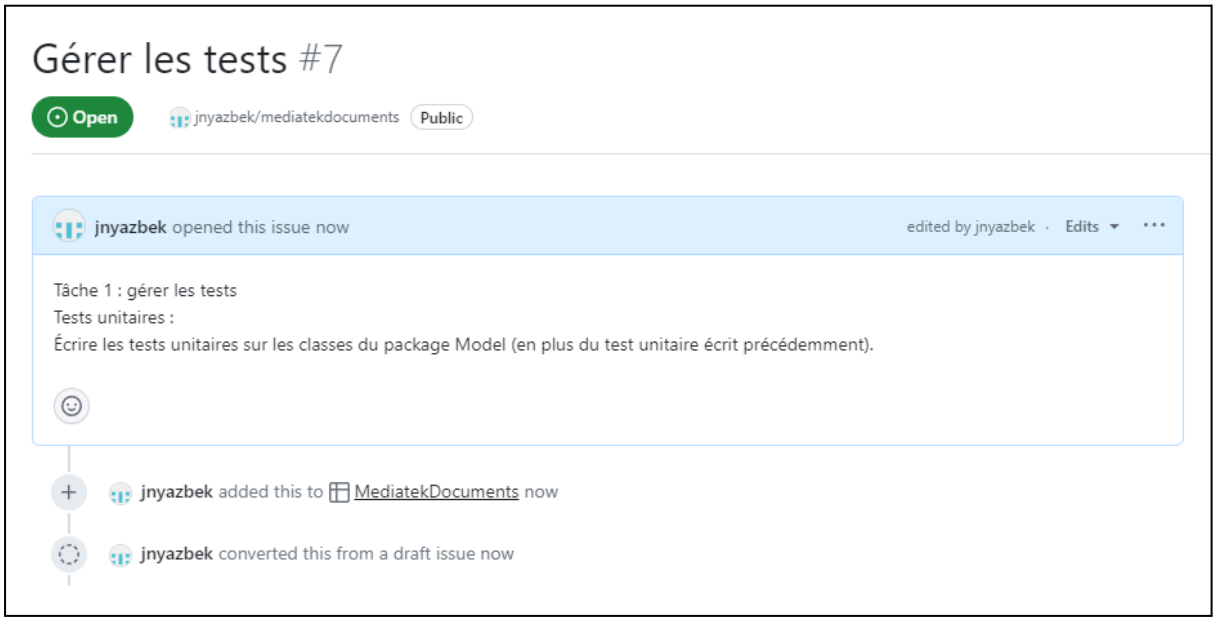
Il nous suffit d'ajouter les package nécessaire et de modifier notre méthode dans Access : using Serilog;
using Serilog.Formatting.Compact;
using Serilog.Formatting.Json;
et la méthode :

```
private Access()
{
    String authenticationString;
    try
    {
        // Récupération de la chaîne de connexion à partir du nom
         authenticationString = GetConnectionStringByName(connectionName);


        // Configuration du logging avec Serilog
        Log.Logger = new LoggerConfiguration()
            .MinimumLevel.Information()
            .WriteTo.Console()
            .WriteTo.File(new JsonFormatter(), "logs/log.txt", rollingInterval: RollingInterval.Day)
            .CreateLogger();

        // Initialisation de l'API avec la chaîne d'authentification
        api = ApiRest.GetInstance(uriApi, authenticationString);
    }
    catch (Exception e)
    {
        // Log de l'erreur fatale et arrêt de l'application
        Log.Fatal("Access.Access catch connectionString={0} erreur={1}", connectionName, e.Message);
        Environment.Exit(0);
    }

}
```

Mission 4 : tester et documenter

Temps estimé : 8h
Temps réel : 12h

a) Tâche 1 : gérer les tests



On va ajouter à notre solution un nouveau projet de test en utilisant MSTest.
On crée ensuite une classe de test par classe dans modèle.
Voici un exemple de test :

```
namespace MediaTekDocumentsTest
{
    [TestClass]
    0 références
    public class CommandeDocumentTests
    {
        [TestMethod]
        0 références
        public void CommandeDocument_Test()
        {
            // Arrange
            string id = "0008";
            DateTime dateCommande = DateTime.Now;
            double montant = 150.0;
            int nbExemplaire = 3;
            string idLivreDvd = "0008";
            string idSuivi = "0008";
            string libelle = "Livrée";

            // Act
            CommandeDocument commandeDocument = new CommandeDocument(id, dateCommande, montant, nbExemplaire, idLivreDvd, idSuivi, libelle);

            // Assert
            Assert.AreEqual(id, commandeDocument.Id, "Id doit être = à la valeur attendue");
            Assert.AreEqual(dateCommande, commandeDocument.Date, "DateCommande doit être = à la valeur attendue");
            Assert.AreEqual(montant, commandeDocument.Montant, "Montantdoit être = à la valeur attendue");
            Assert.AreEqual(nbExemplaire, commandeDocument.NbExemplaire, "NbExemplaire doit être = à la valeur attendue");
            Assert.AreEqual(idLivreDvd, commandeDocument.IdLivreDvd, "IdLivreDvd doit être = à la valeur attendue");
            Assert.AreEqual(idSuivi, commandeDocument.Suivi.Id, "IdSuivi doit être = à la valeur attendue");
            Assert.AreEqual(libelle, commandeDocument.LibelleSuivi, "LibelleSuivi doit être = à la valeur attendue");
        }
    }
}
```

Une fois tous les tests rédigés, il nous suffit de cliquer sur : test→ exécuter tous les tests dans la barre d'outils.

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
Vérifier que l'ID et le libellé sont correctement assignés lors de la création d'un objet Catégorie.	Créer un objet Categorie avec Id = "0008" et Libelle = "Fiction".	Id de l'objet doit être "0008" et Libelle doit être "Fiction".	OK
Contrôler la méthode ToString() de la classe Categorie pour voir si elle retourne le libellé.	Créer un objet de type Categorie avec le libellé "Fiction". Appeler la méthode ToString() sur cet objet.	La chaîne de caractères "Fiction".	OK
Vérifier que les propriétés d'une CommandeDocument sont correctement assignées lors de la création de l'objet.	Créer un objet CommandeDocument avec les valeurs spécifiées pour id, dateCommande, montant, nbExemplaire, idLivreDvd, idSuivi, et libelle.	Chaque propriété de l'objet doit correspondre à la valeur avec laquelle elle a été initialisée.	OK

Vérifier que les propriétés d'une Commande sont correctement assignées lors de la création de l'objet.	Créer un objet Commande avec les valeurs spécifiées pour id, date, montant, idSuivi, et libelle.	Chaque propriété de l'objet doit correspondre à la valeur avec laquelle elle a été initialisée, et l'objet Suivi associé ne doit pas être null.	OK
Vérifier le libellé de suivi quand l'objet Suivi existe.	Créer un objet Commande avec un objet Suivi valide et récupérer le libelle de suivi.	Le libelle de suivi doit correspondre à la valeur attendue ("En cours").	OK
Vérifier le libellé de suivi retourne une chaîne vide si Suivi est null.	Créer un objet Commande sans objet Suivi (ou en le mettant explicitement à null) et récupérer le libelle de suivi.	Le libelle de suivi doit être une chaîne vide.	OK
Vérifier que les propriétés d'un Document sont correctement assignées lors de la création de l'objet.	Créer un objet Document avec les valeurs spécifiées pour id, titre, image, idGenre, genre, idPublic, lePublic, idRayon, et rayon.	Chaque propriété de l'objet doit correspondre à la valeur avec laquelle elle a été initialisée.	OK

Vérifier que les propriétés spécifiques et héritées d'un DVD sont correctement assignées lors de la création de l'objet.	Créer un objet DVD avec les valeurs spécifiées pour id, titre, image, duree, realisateur, synopsis, idGenre, genre, idPublic, lePublic, idRayon, et rayon.	Chaque propriété de l'objet doit correspondre à la valeur avec laquelle elle a été initialisée, y compris les propriétés héritées et spécifiques à un DVD.	OK
Vérifier que les propriétés Id et Libelle d'un Etat sont correctement assignées lors de la création de l'objet.	Créer un objet Etat avec les valeurs spécifiées pour expectedId et expectedLibelle.	Id de l'objet doit être "0001" et Libelle doit être "Neuf".	OK
Vérifier que toutes les propriétés d'un Exemplaire sont correctement assignées lors de la création de l'objet.	Créer un objet Exemplaire avec les valeurs spécifiées pour expectedNumero, expectedDateAchat, expectedPhoto, expectedIdEtat, et expectedIdDocument.	Chaque propriété de l'objet doit correspondre à la valeur avec laquelle elle a été initialisée, y compris le numéro, la date d'achat, la photo, l'ID de l'état, et l'ID du document.	OK

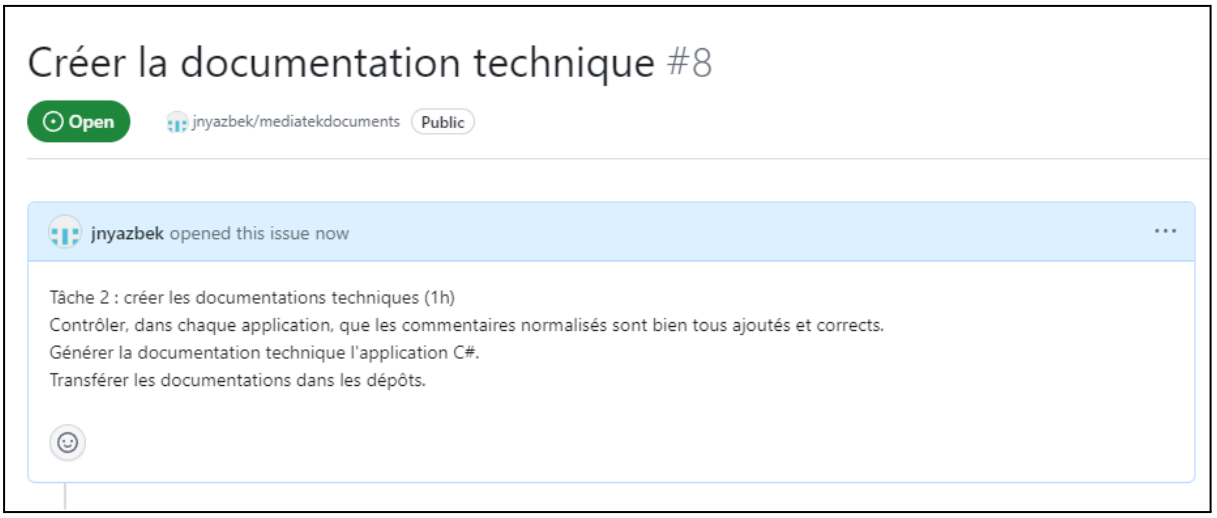
Vérifier que les propriétés Id et Libelle d'un Genre sont correctement assignées lors de la création de l'objet.	Créer un objet Genre avec les valeurs spécifiées pour expectedId et expectedLibelle.	Id de l'objet doit être "0003" et Libelle doit être "Science-Fiction".	OK
Vérifier que les propriétés spécifiques et héritées d'un Livre (comme exemple de Livre DVD) sont correctement assignées lors de la création de l'objet.	Créer un objet Livre avec les valeurs spécifiées pour expectedId, expectedTitre, expectedImage, expectedIsbn, expectedAuteur, expectedCollection, expectedIdGenre, expectedGenre, expectedIdPublic, expectedPublic, expectedIdRayon, et expectedRayon.	Chaque propriété de l'objet, incluant celles héritées de Document et les spécifiques à Livre, doit correspondre à la valeur avec laquelle elle a été initialisée.	OK
Vérifier que toutes les propriétés d'un Livre sont correctement assignées lors de la création de l'objet.	Créer un objet Livre avec les valeurs spécifiées pour expectedId, expectedTitre, expectedImage, expectedIsbn, expectedAuteur, expectedCollection, expectedIdGenre, expectedGenre, expectedIdPublic,	Chaque propriété de l'objet, incluant l'ISBN, l'auteur, et la collection, en plus des propriétés héritées de Document, doit correspondre à la valeur avec	OK

	expectedPublic, expectedIdRayon, et expectedRayon.	laquelle elle a été initialisée.	
Chaque propriété de l'objet, incluant l'ISBN, l'auteur, et la collection, en plus des propriétés héritées de Document, doit correspondre à la valeur avec laquelle elle a été initialisée.	Créer un objet Public avec les valeurs spécifiées pour expectedId et expectedLibelle.	Id de l'objet doit être "00017" et Libelle doit être "Jeunesse".	OK
Contrôler que la méthode ToString() de la classe Public retourne correctement le Libelle.	Créer un objet Public avec un Libelle spécifié, puis appeler la méthode ToString() sur cet objet.	La méthode doit retourner la chaîne de caractères correspondant au Libelle fourni, dans ce cas, "Adultes".	OK
Vérifier que les propriétés Id et Libelle d'un Rayon sont correctement assignées lors de la création de l'objet.	Créer un objet Rayon avec les valeurs spécifiées pour expectedId et expectedLibelle.	Id de l'objet doit être "0004" et Libelle doit être "Littérature".	OK

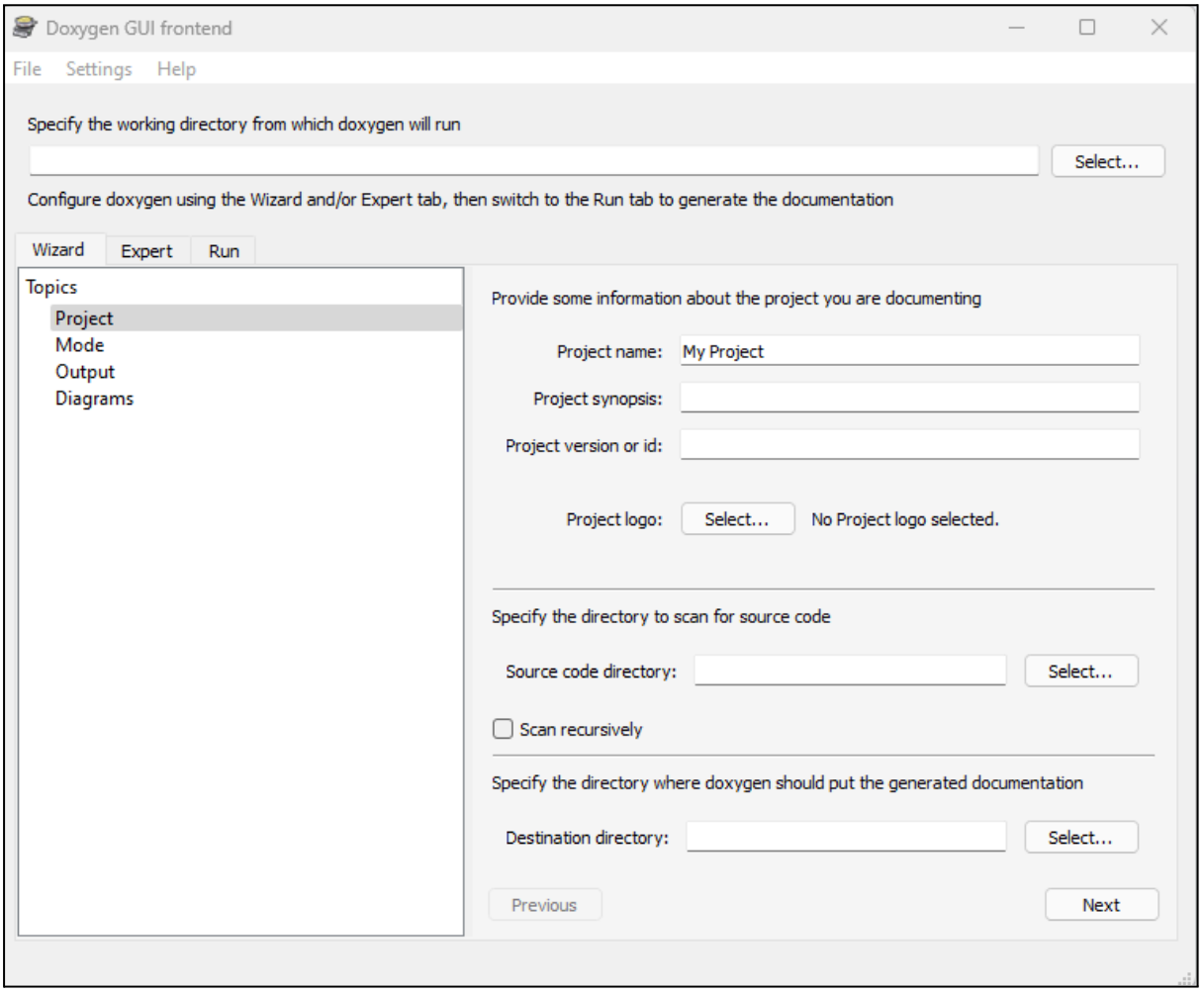
Contrôler que la méthode ToString() de la classe Rayon retourne correctement le Libelle.	Créer un objet Rayon avec un Libelle spécifié, puis appeler la méthode ToString() sur cet objet.	La méthode doit retourner la chaîne de caractères correspondant au Libelle fourni, dans ce cas, "Science-Fiction".	OK
Vérifier que toutes les propriétés d'une Revue, incluant les spécifiques, sont correctement assignées lors de la création de l'objet.	Créer un objet Revue avec les valeurs spécifiées pour expectedId, expectedTitre, expectedImage, expectedIdGenre, expectedGenre, expectedIdPublic, expectedPublic, expectedIdRayon, expectedRayon, expectedPeriodicite, et expectedDelaiMiseADispo.	Chaque propriété de l'objet, y compris la périodicité et le délai de mise à disposition, doit correspondre à la valeur avec laquelle elle a été initialisée.	OK
Vérifier que les propriétés Id, Nom, et Description d'un Service sont correctement assignées.	Initialiser un objet Service puis assigner directement les valeurs expectedId, expectedNom, et expectedDescription.	Id de l'objet doit être 1, Nom doit être "Informatique", et Description doit être "Service chargé de la gestion des	OK

		systèmes informatiques.".	
Vérifier que les propriétés d'un Utilisateur sont correctement assignées lors de l'initialisation et peuvent être mises à jour correctement.	Créer un objet Utilisateur avec les valeurs initiales pour Id, Nom, Prenom, Email, Mot_de_passe, et Id_service. Vérifier ces valeurs, puis mettre à jour Nom et Id_service avec de nouvelles valeurs et vérifier à nouveau.	Les propriétés initiales doivent correspondre aux valeurs fournies, et les propriétés mises à jour (Nom et Id_service) doivent refléter les nouvelles valeurs assignées.	OK

b) Tâche 2 : gérer la documentation technique



Afin de créer la documentation technique il nous faut télécharger le logiciel Doxygen :

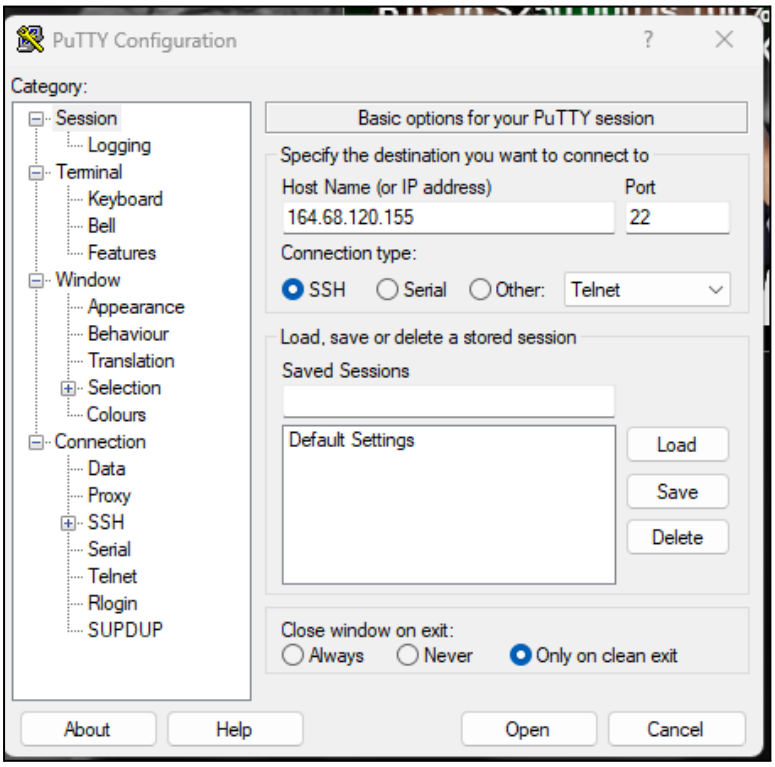


Il faut ensuite remplir le chemin vers le dossier de mediatekdocument dans le champ source code directory, puis lancer la génération de la documentation. Les commentaires sur les classes et les méthodes seront ainsi indexées pour créer une documentation en html.

Mission 5 : déployer et gérer les sauvegardes de données

a) Tâche 1 : déployer le projet

Afin de déployer l'API, j'ai choisi d'utiliser mon VPS Contabo. Afin d'interagir avec cette VM on utilise putty :



On entre l'adresse ip de la vm puis on clique sur open. Après s'être identifié, on installe Apache et MySql.

On met d'abord les paquets à jour:
sudo apt update
sudo apt upgrade -y

Installation d'Apache:
sudo apt install apache2 -y
sudo systemctl status apache2

Installation MySql :
sudo apt install mysql-server -y
sudo mysql_secure_installation

On exporte la base de données jusqu'alors hébergée en local afin d'obtenir un script .sql.

On transfert ce fichier à la VM en utilisant le powershell :
scp -r C:\Users\MonUser\Documents\rest_mediatekdocument root@164.68.120.155:/root
On lance MySql sur la VM : mysql -u root -p

On crée la base de donnée mediatek86 :
CREATE DATABASE mediatek86;
EXIT;
mysql -u root -p mediatek86 < /root/mediatek86.sql

Maintenant que la BDD est mise en place, on pense bien a vérifier que les identifiants / mots de passe dans l'API sont les mêmes qu'en local. On évitera d'accéder à la BDD avec l'utilisateur root car cela peut causer des erreurs.

On va maintenant ajouter l'API sur la VM.

J'ouvre mon terminal powershell et j'utilise la commande suivante pour transférer mon API dans la VM :
scp -r C:\Users\MonUser\Documents\rest_mediatekdocument root@164.68.120.155:/var/www/html/

Cela permet le transfert de mes fichiers vers la VM.

On accorde les permission d'accès, lecture et écriture au dossier
sudo chown -R www-data:www-data /var/www/html/rest_mediatekdocument

Il ne nous reste plus qu'à tester l'API avec Postman avec une requête telle que celle-ci :
GET http://164.68.120.155/rest_mediatekdocuments/utilisateur/1

On obtient le résultat suivant : {
  "code": 200,
  "message": "OK",
  "result": [
    {
      "id": "1",
      "nom": "Doe",
      "prenom": "John",
      "email": "john.doe@example.com",
      "mot_de_passe": "password123",
      "id_service": "1"
    }
  ]
}

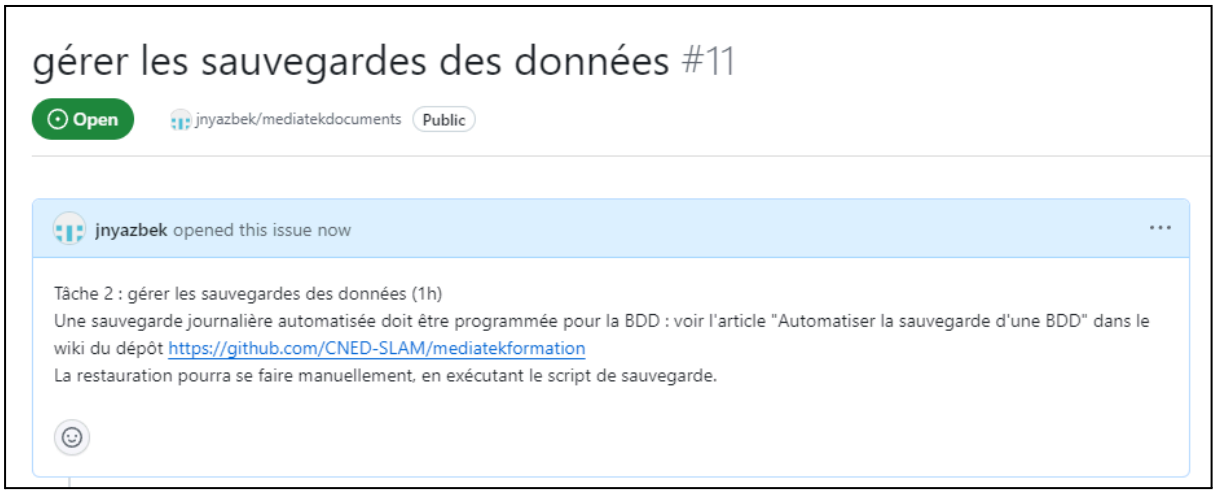
Le déploiement est opérationnel !

Il ne manque plus qu'a changer l'url API dans l'application C#...

```
namespace MediaTekDocuments.dal
{
    /// <summary>
    /// Classe d'accès aux données
    /// </summary>
    8 références
    public class Access
    {
        /// <summary>
        /// adresse de l'API//"http://localhost/rest_mediatekdocuments/";//
        /// </summary>
        private static readonly string uriApi = "http://164.68.120.155/rest_mediatekdocuments/";
```

Et à créer l'installateur en publiant le projet : clic droit sur le projet mediatekdocument puis en cliquant sur publier.

b) Tâche 2 : gérer les sauvegardes de données



Il ne nous reste qu'à automatiser la sauvegarde de la BDD.
On créé pour cela un script backup8mediatek86.sh :

```
#!/bin/bash

# Définition des variables
DB_NAME="mediatek86"
DB_USER="root"
DB_PASSWORD="*******"
BACKUP_DIR="/var/www/html/rest_mediatekdocuments"
DATE=$(date +"%Y-%m-%d")
```

Commande de sauvegarde
mysqldump -u $DB_USER -p$DB_PASSWORD $DB_NAME > $BACKUP_DIR/${DB_NAME}_$DATE.sql

Optionnel : Compression de la sauvegarde
gzip $BACKUP_DIR/${DB_NAME}_$DATE.sql

Puis de set up une tache cron

Pour cela on ouvre l'éditeur de cron avec : crontab -e
et on ajoute la ligne suivant au fichier
0 2 * * * /var/www/html/rest_mediatekdocuments/backup_mediatek86.sh

Enfin on rend le fichier exécutable avec
chmod +x /var/www/html/rest_mediatekdocuments/backup_mediatek86.sh,
afin de finaliser le projet.

Conclusion

Nous avons ainsi fait évoluer notre application C#, elle peut désormais être installée sur n'importe quel poste compatible.
L'APIi est déployée et les utilisateurs disposant des accès pourront consulter les documents, gérer leurs commandes et suivis en fonction de leur service.