Schoukroun Alexandre
BTS SIO – SLAM

Atelier de professionnalisation 3

Table des matières

1 Rappel du contexte : 1
2 Rappel des missions : 1
3 Outils utilisés : 2
4 Mission 0 : 3
5 Mission 2 : 3
6 Mission 4 : 9
7 Mission 5 : 11
8 Mission 6 : 12
9 Bilan final : 12
10 Compétences officielles couvertes : 12

1 Rappel du contexte :

L'ESN InfoTech Services 86 (ITS 86) vient de remporter le marché pour différentes interventions au sein du réseau MediaTek86, dont certaines dans le domaine du développement d'application. En tant que technicien développeur junior pour ITS 86, mon rôle a été de faire évoluer une application de bureau (C#) exploitant une base de données relationnelle MySQL, permettant de consulter le catalogue de MediaTek86, ainsi que de gérer les commandes. Il s'agit d'un réseau qui gère les médiathèques de la Vienne, ayant pour rôle de fédérer les prêts de livres, DVD et CD, et de développer la médiathèque numérique englobant l'entièreté des médiathèques du département.

2 Rappel des missions :

Cet atelier peut être réalisé seul ou en groupe. S'il est réalisé seul, les mission 1 et 3 sont optionnelles. **Ayant travaillé seul et n'ayant pas eu suffisamment de temps, je n'ai donc pas réalisé ces deux missions.**

- La mission 0 consiste simplement à préparer l'environnement de travail avant de réellement commencer l'atelier.
- La mission 2 consiste à coder toute la gestion des commandes.
  La mission 4 consiste à coder la partie authentification, avec différents niveaux d'accès.
- La mission 5 consiste à contrôler la qualité du code, ajouter des tests, générer la documentation technique, déployer la BDD en ligne et créer puis tester l'installeur.

-
- La mission 6 consiste à créer une courte vidéo de démonstration de l'application.
- La mission bilan, qui consiste à déployer la BDD en ligne, rédiger le présent compte-rendu et créer une page dans mon portfolio.

3 Outils utilisés :

Logiciels et web :

- Microsoft Visual Studio 2019 version 16.11.11
- Wampserver version 3.2.3
- phpMyAdmin version 5.0.2
- Apache version 2.4.46
- MySQL version 5.7.31
- GitHub

Extensions Visual Studio 2019 :

- Microsoft Visual Studio Installer Projects version 1.0.2
- SonarLint for Visual Studio 2019 version 5.5.0.43817
- SLAF version 5.5.0.43817
- Visual Studio Spell Checker (VS2017 and VS2019) version 2022.1.3.0
- Extended XML Doc Comments Completion Provider version 2019.10.27.0
- Github Extension for Visual Studio version 2.11.106.19330
- MySQL for Visual Studio version 1.2.9
- Visual Studio IntelliCode version 2.2.1462.13379
- NuGetRecommender [Préversion] version 1.600
- Visual Studio Rich Navigation [Préversion] version 0.1.272.30069

NuGet packages Visual Studio 2019 :

- Serilog version 2.10.0
- Serilog.Sinks.Console version 4.0.1
- Serilog.Sinks.EventLog version 3.1.0
- Serilog.Sinks.File version 5.0.0
- System.Buffers version 4.5.1

- BouncyCastle version 1.8.5
- Google.Protobuf version 3.14.0

4 Mission 0 :

Avant de passer à la mission 2 de l'atelier, il m'a d'abord fallu préparer l'environnement de travail.

- J'ai vérifié que WampServer était bien installé sur mon PC.
- J'ai vérifié que l'IDE Visual Studio 2019 était bien installé et à jour sur mon PC.
- Ensuite, j'ai récupéré le dossier documentaire qui présente l'application actuelle et les demandes à traiter dans les mission suivantes.
- Puis j'ai récupéré le code source du projet sous GitHub, avant de dézipper sont contenu dans un dossier créé au préalable et servant de dépôt local. J'ai ensuite renommé le dossier dézippé en « Mediatek86 ».
- J'ai récupéré le script de la base données que j'ai ensuite importé dans phpMyAdmin, SGBD MySQL.
- Une fois la BDD installée et le projet lancé sous VS, j'ai d'abord vérifier que l'application de base fonctionnait bien (par rapport à ce qui était indiqué dans le dossier documentaire).
- J'ai bien pris le temps d'analyser le code existant de l'application afin de pouvoir respecter sa structure et m'inspirer de ce qui a déjà été fait pour réaliser mon travail.
  Dans mon compte GitHub, j'ai créé le dépôt « mediatek86 » et fais le lien avec le projet sous VS. J'ai également fait un commit et push, phase 0.

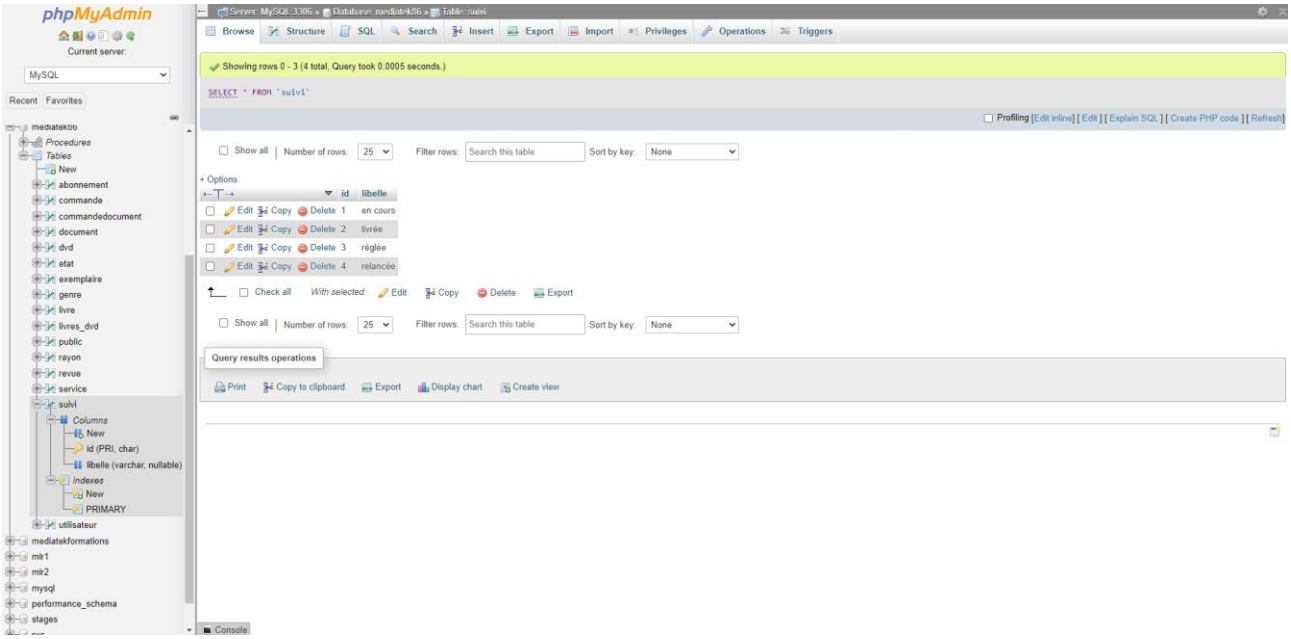
Bilan : tous les objectifs ont été atteints.

- Dossier documentaire récupéré : OK
- Application installée en local et projet configuré sous Visual Studio : OK
- BDD installée en local dans phpMyAdmin au format MySQL : OK
- Application opérationnelle : OK
- Application reliée à mon dépôt distant sur GitHub : OK

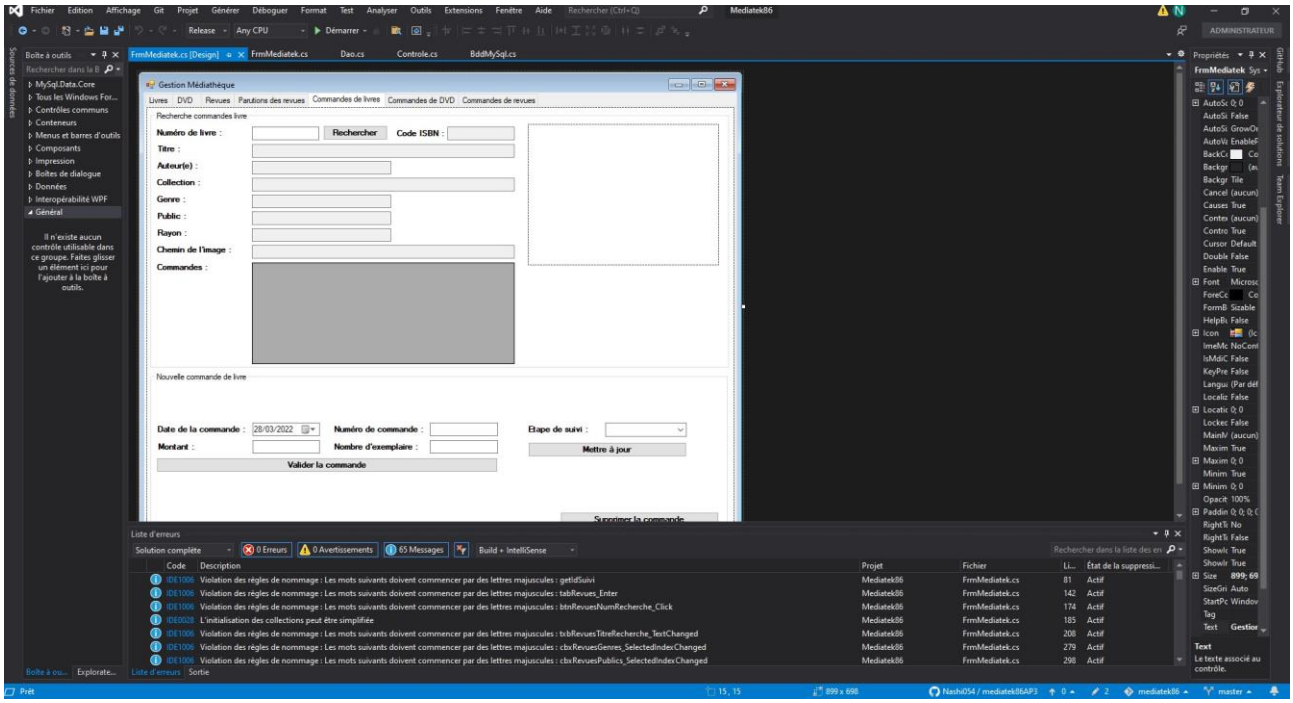
5 Mission 2 :

La mission 2 consiste à coder les fonctionnalités de gestion des commandes.

- La première phase a été de créer la table « suivi » dans la BDD. Elle contient les différentes étapes de suivi d'une commande de livre ou de DVD. Je l'ai donc rempli avec quatre étapes : « en cours », « livrée », « réglée » et « relancée ». J'ai ensuite fait le lien avec la table « commandecocument ».



- La deuxième phase a été de coder la gestion des commandes de livre.
- J'ai d'abord commencé par réaliser la partie graphique sur le modèle de l'onglet « Parutions de revues », en créant un nouvel onglet dans lequel j'y ai placé deux groupBox : « Recherche commandes livres » et « Nouvelle commande de livre ».
- La partie « Recherche commandes livres » fonctionne de la même manière que l'affichage des informations détaillées de l'onglet « Livres », mais avec en plus un tableau permettant l'affichage des commandes enregistrées pour le livre recherché.
- La partie « Nouvelle commande de livre » comporte un formulaire de création d'une commande, ou affichant les informations de la commande actuellement sélectionnée dans le tableau. Cette partie permet également de gérer les étapes de suivi et de supprimer une commande.



- Pour la partie code, tout ce qui concerne l'affichage des informations détaillées du livre sélectionné (par son numéro) a été grandement repris du code de l'onglet « Livres », en changeant les noms des méthodes et en modifiant leur contenu pour l'adapter au éléments graphiques de l'onglet « Commandes de livres ».
- J'ai créé les classes métiers « Commande », « CommandeDocument » et « Suivi ».
- La méthode d'affichage des commande fait appel (via le contrôleur) à la méthode « GetCommandesLivresDvd »de la classe « Dao », comprenant une requête « select ».

```
/// <summary>
/// Retourne toutes les commandes de livre ou de DVD à partir de la BDD
/// </summary>
/// <param name="idDocument"></param>
/// <returns>Collection d'objets CommandeDocument</returns>
1 référence | NashiO54, il y a 3 jours | 1 auteur, 1 modification
public static List<CommandeDocument> GetCommandesLivreDvd(string idDocument)
{
    List<CommandeDocument> lesCommandes = new List<CommandeDocument>();
    string req = "Select d.id, c.dateCommande as 'date commande', c.montant, c.idSuivi, s.libelle as suivi, ";
    req += "d.nbExemplaire as 'nombre exemplaire', d.idLivreDvd ";
    req += "from commandedocument d join commande c on d.id=c.id ";
    req += "join suivi s on s.id=c.idSuivi ";
    req += "where d.idLivreDvd = @id ";
    req += "order by 'date commande' DESC ";
    Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", idDocument}
        };

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

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

    return lesCommandes;
}
```

- • • • • • • • • • • • • • • • •

  •

  Plusieurs méthodes ont été créées pour gérer la création de commande, la mise à jour du suivi, ainsi que la suppression.
- Pour la création de commande, le code consiste à récupéré les information entrées dans le formulaire et les envoyer en tant que paramètre à deux requêtes d'insertion (une pour « commande » et une pour « commandedocument »), elles même écritent dans la méthode « CreerCommandeLivreDvd » de la classe « Dao ».
- Concernant la mise à jour du suivi, j'ai écrit une méthode de remplissage du combo en fonction de l'étape actuelle de suivi de la commande. Ainsi, lorsque la commande vient d'être créée, elle passe directement à l'étape « en cours ». Le combo affiche alors les autres étapes possibles, sauf « réglée », car une commande ne peut être réglée avant d'être livrée. Lorsque l'étape est mise en « livrée », la seule option disponible dans le combo est « réglée », car il n'est pas possible de revenir à une étape précédente. Une fois passée à « réglée », le combo est vidé.

•

```
/// <summary>
/// Rempli un des 3 combo (genre, public, rayon)
/// </summary>
/// <param name="lesSuivis"></param>
/// <param name="bdg"></param>
/// <param name="cbx"></param>
4 références | 0 modification | 0 auteur, 0 modification
public void RemplirComboSuivi(List<Suivi> lesSuivis, BindingSource bdg, ComboBox cbx, string suivi)
{
    switch (suivi)
    {
        case "en cours":
            lesSuivis.RemoveAt(2);
            break;
        case "livrée":
            lesSuivis.RemoveAt(3);
            lesSuivis.RemoveAt(0);
            break;
        case "réglée":
            lesSuivis.Clear();
            break;
        case "relancée":
            lesSuivis.RemoveAt(3);
            lesSuivis.RemoveAt(2);
            lesSuivis.RemoveAt(0);
            break;
    }
    bdg.DataSource = lesSuivis;
    cbx.DataSource = bdg;
    if (cbx.Items.Count > 0)
    {
        cbx.SelectedIndex = 0;
    }
}
```

- Lorsque l'on clique sur le bouton « Mettre à jour », l'information sélectionnée dans le combo est envoyée en tant que paramètre d'une requête « update », elle-même écrite dans la méthode « UpdateCommande » de la classe « Dao ».
- Lorsque l'on clique sur le bouton « Supprimer la commande », deux requêtes « delete » (une pour « commande » et une pour « commandedocument ») sont envoyées avec, en paramètre, l'id de la commande sélectionnée dans le tableau. Ces requêtes se trouvent dans la méthode « DeleteCommandeLivreDvd » dans la classe « Dao ». Cependant, l'appel à cette méthode n'est effectué que si le suivi de la commande sélectionnée est à « en cours » ou à « relancée ». Il n'est donc pas possible de supprimer une commande livrée ou réglée.
- Dans phpMyAdmin, j'ai écrit un trigger (après « update » dans « commande ») qui créer un nombre de tuples, dans « exemplaire », égal à la valeur du champ « nbExemplaire » de la commande sélectionnée, lorsque le suivi passe à « livrée ». Ces exemplaires sont numérotés de manière séquentielle par rapport au livre concerné.
- J'ai effectué un commit et push de la phase 1 vers mon dépôt distant sur GitHub.

- Pour la partie « Commandes de DVD », exactement le même principe a été suivi, en copiant ce qui a été fait pour la partie « Commandes de livres » puis en renommant les méthode et en adaptant leur contenu aux noms des objets graphiques de cet onglet.

- Les méthodes « Dao » permettant de créer, mettre à jour le suivi et supprimer une commandes de DVD sont les mêmes que pour les livres, car les commandes de DVD sont également enregistrées dans la table « commandedocument ».
- J'ai effectué un commit et push de la phase 2 vers mon dépôt distant sur GitHub.

- Pour la partie « Commandes de revues » une grosse partie a été reprise des deux onglets précédents, mais avec quelques différences.
- Une commande de revue correspond à un abonnement (ou un renouvellement). Elle ne dispose donc pas d'un suivi et n'est pas concernée par un nombre d'exemplaire. En revanche, une date de fin d'abonnement doit être précisée. Aucun exemplaire n'est créé par le trigger codé précédemment, puisque les exemplaires sont rajoutés manuellement depuis l'onglet « Parutions des revues ».
- J'ai donc créé une classe métier « Abonnement » et je n'ai pas codé de partie de mise à jour de suivi.
- Ensuite, j'ai codé les méthodes « CreerCommandeRevue » et « DeleteCommandeRevue » dans « Dao », permettant d'exécuter les requêtes d'insertion et de suppression propres aux abonnements.
- J'ai codé les conditions de suppression de telle sorte qu'un abonnement ne puisse être supprimé que dans l'éventualité où aucun exemplaire de la revue ne serait apparue durant l'abonnement. Pour ce faire, j'ai notamment codé la méthode « ParutionDansAbonnement » dans la classe « Controle » (pour pouvoir créer un test par la suite), qui vérifie si la date de parution d'un exemplaire est comprise entre la date de commande et la date de fin d'abonnement.

```
/// <summary>
/// Retourne vrai si la date de parution est comprise entre la date de commande
/// et la date de fin d'abonnement
/// </summary>
/// <param name="dateCommande"></param>
/// <param name="dateFinAbonnement"></param>
/// <param name="dateParution"></param>
/// <returns>True si dateParution entre dateCommande et dateFinAbonnement</returns>
3 références | Nashi054, Il y a 1 jour | 1 auteur, 1 modification
public bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFinAbonnement, DateTime dateParution)
{
    if ((dateParution >= dateCommande) && (dateParution < dateFinAbonnement))
    {
        return true;
    }
    return false;
}
```

- J'ai veillé à ce que des mauvaises manipulations ne puissent pas arriver (lettres au lieu de chiffres pour le nombre d'exemplaire par exemple) et qu'un message d'erreur s'affiche.
- J'ai également codé un trigger dans « commandedocument » et un autre dans « abonnement » pour respecter la contrainte de partition de l'entité « Commande ».
- Enfin, j'ai codé une procédure stockée dans la BDD qui permet d'obtenir la liste des revues dont l'abonnement se termine dans moins de 30 jours. J'ai également codé, dans l'application, l'affichage d'une fenêtre avec le tableau contenant la liste des fins approchantes d'abonnement. Cet affichage s'effectue à l'ouverture de l'application, avant l'affichage de la fenêtre principale. Par ailleurs, il m'aura fallu créer une classe métier FinAbonnement, ainsi que la méthode « GetAllFinAbonnement » dans « Dao » pour gérer l'affichage de la liste. Cette méthode envoie une requête d'appel de la procédure stockée « finAbonnement » créée précédemment.
- J'ai effectué un commit et push de la phase 3 vers mon dépôt distant sur GitHub.

```
/// <summary>
/// Retourne tous les abonnements qui arrivent à leur terme
/// </summary>
/// <returns>Collection d'objets FinAbonnement</returns>
1 référence | Nashi054, il y a 2 jours | 1 auteur, 1 modification
public static List<FinAbonnement> GetAllFinAbonnement()
{
    List<FinAbonnement> lesFinAbonnements = new List<FinAbonnement>();
    string req = "Call finAbonnement()";

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

    while (curs.Read())
    {
        FinAbonnement finAbonnement = new FinAbonnement((string)curs.Field("titre"), (DateTime)curs.Field("dateFinAbonnement"));
        lesFinAbonnements.Add(finAbonnement);
    }
    curs.Close();
    return lesFinAbonnements;
}
```

Bilan : presque tous les objectifs ont été atteints

- Gestion des commandes de livres : OK
- Gestion des commandes de DVD : OK
- Gestion des commandes de revues : OK
- Présentation similaire à l'onglet « Parutions de revues » : OK
- Tri sur colonnes dans toutes les listes : non atteint

- Trigger de contrainte de partition de commandes : OK
- Affichage des abonnement arrivant à leur terme : OK

6 Mission 4 :

La mission 4 consiste à mettre en place des authentifications et à limiter l'accès à certaines fonctionnalités de l'application, suivant le niveau d'habilitation de l'utilisateur connecté.

- J'ai commencé par ajouter la table « service » dans la base de données. Je l'ai ensuite remplie avec les 4 services et habilitations : « administrateur », « administratif », « prêts » et « culture ».
- Ensuite, j'ai ajouté la table « utilisateur » dans la BDD, en la liant à la table précédente. Je l'ai rempli avec 4 utilisateurs, chacun ayant un rôle différent.
- Voici la liste des identifiants créés et leur niveau d'habilitation :
  - login : JRoger, mot de passe : JRogerpwd, service : administrateur
  - login : EDupont, mot de passe : EDupontpwd, service : administratif
  - login : ABoulanger, mot de passe : ABoulangerpwd, service : prêts
  - login : FGregoire, mot de passe : FGregoirepwd, service : culture

- J'ai créé une fenêtre d'authentification, en modifiant le code du contrôleur, de telle sorte qu'il l'ouvre en premier.
- Toujours dans le contrôleur, j'ai codé la méthode « Authentification » qui gère les fenêtres à afficher en fonction du niveau d'habilitation. Ainsi, pour les membres « administrateur » et « administratif », la fenêtre d'avertissement de fins d'abonnement s'affiche après l'authentification, puis la fenêtre principale de l'application. Pour les membres « prêts », seule la fenêtre principale s'ouvre après authentification. Pour les membres « culture », aucune des fenêtres ne s'affiche et l'application s'arrête.

```
/// <summary>
/// Gère les affichages dans l'application en fonction
/// du service auquel appartient l'utilisateur
/// </summary>
/// <param name="service"></param>
2 références | NashiO54, il y a 2 jours | 1 auteur, 1 modification
public void Authentification(int service)
{
    if (service == 1)
    {
        FrmFinAbonnement frmFinAbonnement = new FrmFinAbonnement(this);
        frmFinAbonnement.ShowDialog();
        FrmMediatek frmMediatek = new FrmMediatek(this);
        frmMediatek.ShowDialog();
    }
    else if (service == 2)
    {
        servicePrets = true;
        FrmMediatek frmMediatek = new FrmMediatek(this);
        frmMediatek.ShowDialog();
    }
}

/// <summary>
/// getter sur servicePrets
/// </summary>
/// <returns>True si l'utilisateur fait partie du service Prêts</returns>
1 référence | NashiO54, il y a 2 jours | 1 auteur, 1 modification
public bool getServicePrets()
{
    return this.servicePrets;
}
```

- Il m'a ensuite fallu modifier le code de la fenêtre d'authentification, afin d'afficher un popup pour les utilisateurs « culture » leur indiquant qu'ils n'ont pas le niveau d'habilitation nécessaire pour accéder à l'application
- Puis j'ai modifié le code de la fenêtre principale, afin d'y ajouter des conditions sur l'activation des groupBox des onglets « Parutions des revues », « Commandes de livres », « Commandes de DVD » et « Commandes de revues » de telle sorte qu'ils soient désactivé si l'utilisateur connecté est du service « prêts ».
- J'ai également codé la méthode « GetAllUtilisateur » dans « Dao » contenant la requête permettant de vérifier que le login et le mot de passe entrés dans le formulaire d'authentification correspondent à un utilisateur enregistré dans la BDD.
- J'ai effectué un commit et push de la phase 4 vers mon dépôt distant sur GitHub.

```
/// <summary>
/// Retourne l'utilisateur dont le login et le pwd ont été passé en paramètre de la BDD
/// </summary>
/// <param name="login"></param>
/// <param name="pwd"></param>
/// <returns>Liste d'objet Utilisateur</returns>
1 référence | Nashi054, il y a 2 jours | 1 auteur, 1 modification
public static List<Utilisateur> GetUtilisateur(string login, string pwd)
{
    List<Utilisateur> lesUtilisateurs = new List<Utilisateur>();
    string req = "Select * from utilisateur where login = @login and pwd = @pwd";
    Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@login", login},
            { "@pwd", pwd}
        };

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

    while (curs.Read())
    {
        Utilisateur utilisateur = new Utilisateur((string)curs.Field("id"), login, pwd, (string)curs.Field("idService"));
        lesUtilisateurs.Add(utilisateur);
    }

    curs.Close();
    return lesUtilisateurs;
}
```

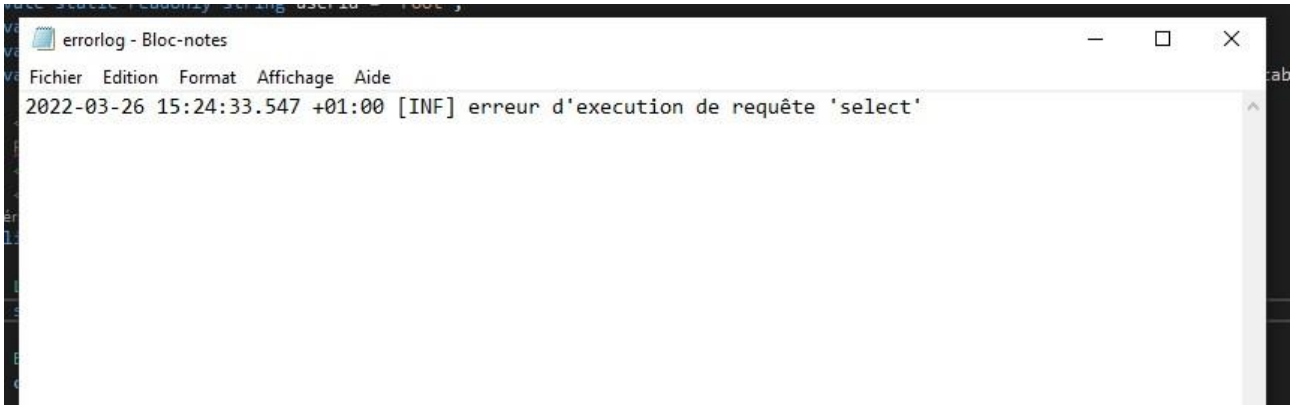
Bilan : tous les objectifs ont été atteints

- Authentification au démarrage de l'application : OK
- Accès limité en fonction de l'habilitation : OK

7 Mission 5 :

La mission 5 consiste à contrôler la qualité du code, ajouter des tests, générer la documentation technique, déployer la BDD en ligne et créer puis tester l'installeur.

- J'ai donc commencé par nettoyer le code, notamment à l'aide de SonarLint, ainsi que le contrôleur de code de l'IDE.
- J'ai créé quelques tests unitaires sur certaines méthodes.
- J'ai ajouté des logs dans les catch qui contiennent des affichages consoles, en faisant en sorte qu'ils soient enregistrés dans un fichier texte (disponible dans le dossier bin/Debug/logs, fichier errorlog.txt)



- J'ai généré la documentation technique de l'application complète .J'ai utilisé DocFx Help pour se faire.
- Après avoir déployé la BDD en ligne, j'ai modifié le server, le user id, le password et la database du connection string afin que l'application se connecte automatiquement au serveur Azure Database pour MySQL.
- Enfin, j'ai créé un installeur dans Visual Studio, j'ai modifié le nom de l'auteur pour y mettre le miens, ainsi que le nom du manufacturer afin d'y mettre le nom du projet « Mediatek86 ». Après test, l'installation s'est bien déroulée et l'application fonctionne très bien.
- Le fichier d'installation Mediatek86Setup.msi est disponible sur mon portfolio

Bilan : presque tous les objectifs ont été atteints

- Code nettoyé : OK
- Tests créés : OK
- Logs ajoutés : OK
- Documentation technique générée : OK, mais fichier « index.html » bugué
- BDD déployée en ligne: OK
- L'installeur créé : OK

8 Mission 6 :

La mission 6 consiste à créer une courte vidéo (10min) de démonstration des fonctionnalités de l'application.

- J'ai filmé la vidéo avec le logiciel vidéo de Windows La vidéo est disponible sur mon portfolio

Bilan final :

En conclusion, la quasi-totalité des objectifs ont été atteints. Cet atelier m'aura surtout permis de m'entraîner à coder en C# une application de bureau, gérer une BDD locale et la déployer en ligne, mais aussi de m'entraîner à nettoyer du code, créer des tests unitaires, mettre en place des logs, générer de la documentation technique et créer un installeur.

9 Compétences officielles couvertes :

- Concevoir et développer une solution applicative
- Assurer la maintenance corrective ou évolutive d'une solution applicative
- Gérer les données