Atelier de professionnalisation

Objectif : Concevoir une application pour la gestion du personnel de plusieurs médiathèques.

Fakher KARCHID

BTS SIO option SLAM

Table des matières

I. Description de la mission

1. Rappel du contexte de la mission

MeditaTek78 est un réseau qui gère les médiathèques de la ville de Vienne et qui a pour rôle de développer les médiathèques numériques de tout le département.

2. Rappel de la mission globale à effectuer.

En tant que technicien développeur junior pour l'ESN InfoTech Services 86, je dois développer une application de bureau qui va permettre de gérer le personnel de chaque médiathèque, leur affectation à un service et leurs absences.

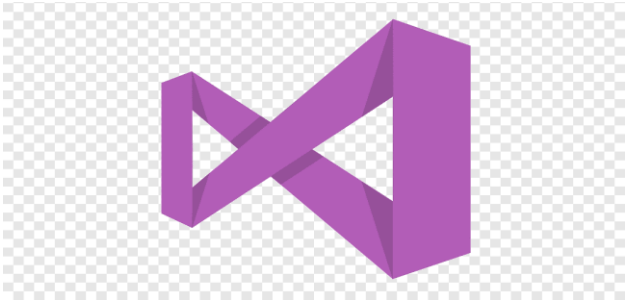
3. Outils de développement

Pour réaliser cette application, j'ai choisi d'utilisé le langage C# et pour la base de données j'ai choisi d'utilisé MySQL.

II. Réalisation des étapes du projet

1. Etape 1 : Conception de l'environnement de travail

La première étape consistait à mettre en place un environnement de travail adapté : j'ai utilisé l'IDE Visual Studio Code pour coder en C# et WampServer pour pouvoir administrer la base de données.



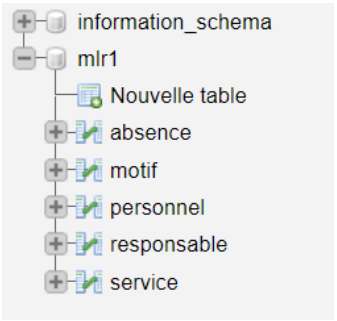
a. Conception de la base de données

La base de données a été configurée avec le script SQL issue du MLD (modèle logique de donnée) et le site www.generatedata.com.

Pour la génération de la table absence, il fallait veiller à mettre des dates de début inférieur à la date de fin et choisir le type mySQL DateTime, comme ci-dessous :

Order	Column Title	Data Type	Examples	Options	Help	Del
1	idpersonnel	Number Range	No examples available.	Between 1 and 13	?	☐
2	datedebut	Date	MySQL datetime	From: 06/02/2020 To: 06/01/2021 Format code: Y-m-d H:i:s	?	☐
3	datefin	Date	MySQL datetime	From: 06/02/2021 To: 06/02/2022 Format code: Y-m-d H:i:s	?	☐
4	idmotif	Number Range	No examples available.	Between 1 and 4	?	☐
Order	Column Title	Data Type	Examples	Options	Help	Del

De plus, il fallait utiliser le type « Number range » pour attribuer aléatoirement des valeurs au clé étrangères. Cependant il fallait corriger certaines valeurs pour assurer l'unicité des clés primaires composées.



L'accès a été sécurisé. Les codes d'accès de l'administrateur sont :

Login : Adminmlr1

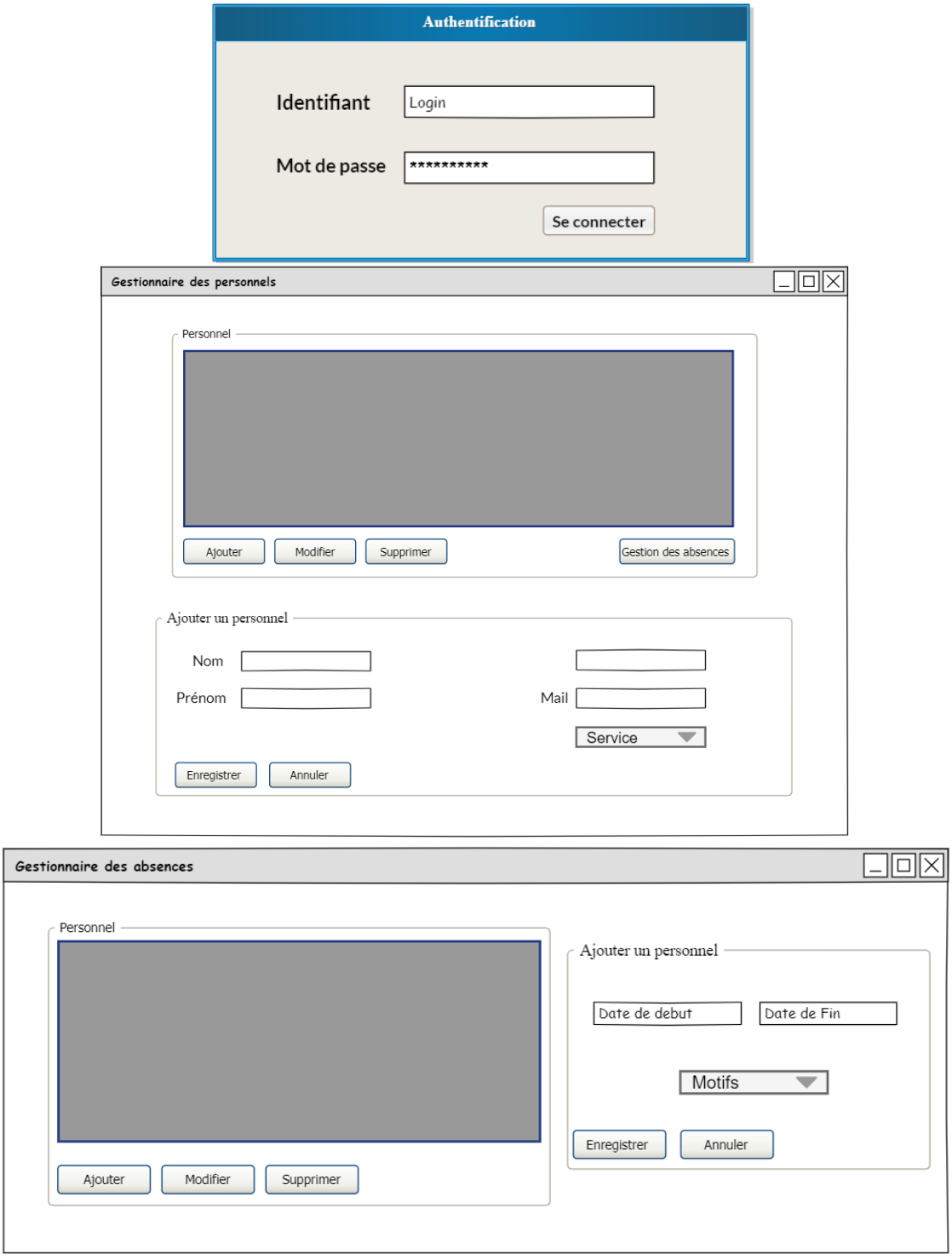
Password : password

Celui-ci possède les droits d'un super utilisateur.

2. Etape 2 : Conception de la vue.

a. Réalisation du prototype visuel.

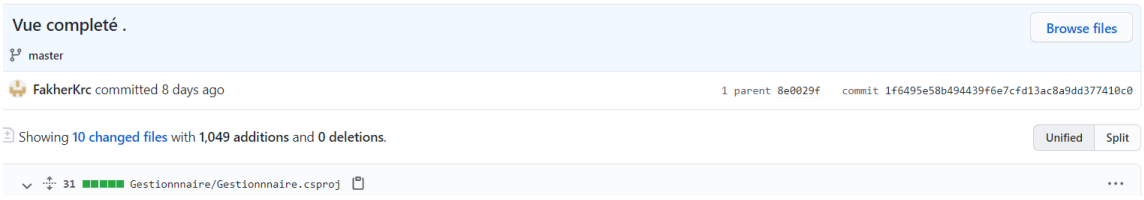
Pour la réalisation de la maquette, j'ai utilisé le logiciel Pencil : j'ai préféré faire une forme pour les deux frames principales (absences et personnels) différentes afin de ne pas perturber les utilisateurs :



b. Réalisation du MVC et sauvegarde

J'ai créé les dossiers pour le MVC (modèle, vue, contrôleur) et j'ai mis en place les composants visuels sur les frames en m'appuyant sur les prototypes que j'ai réalisé précédemment.

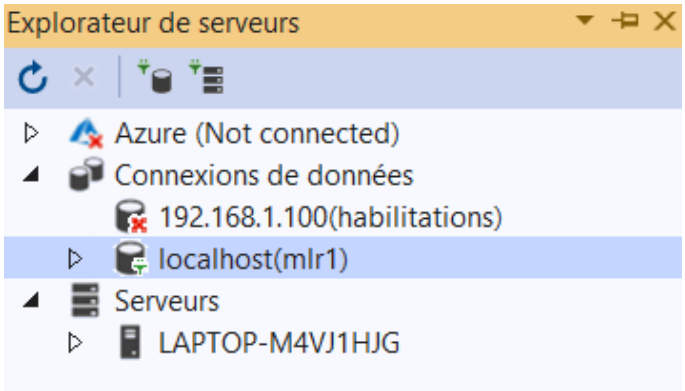
A chaque étape, j'ai sauvegardé les modifications sur mon dépôt GitHub préalablement synchronisé.



3. Etape 3 : Coder le modèle et les outils de connexion, générer la documentation technique

a. Connexion à la base de données

J'ai configuré l'IDE pour qu'il puisse accéder à la base de données même si cela n'était pas nécessaire :



J'ai créé la classe Singleton qui nous permettra de gérer les accès à la base de données en nous permettant de générer qu'une seule instance de la classe en simultanée, la création d'une instance s'effectuera à l'aide de la méthode public « GetInstance() ».

```
public class ConnexionBDD
{

    /// <summary>
    /// Unique instance de la classe
    /// </summary>
    private static ConnexionBDD instance = null;
```

```
    /// <summary>
    /// Constructeur privé pour créer la connexion à la BDD et l'ouvrir
    /// </summary>
    /// <param name="stringConnect">chaine de connexion</param>
    1 référence | FakherKrc, il y a 7 jours | 1 auteur, 1 modification
    private ConnexionBDD(string stringConnect)
    {
        try
        {
            connection = new MySqlConnection(stringConnect);
            connection.Open();
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }

    /// <summary>
    /// Crée une instance unique de la classe
    /// </summary>
    /// <param name="stringConnect">chaine de connexion</param>
    /// <returns>instance unique de la classe</returns>
    11 références | FakherKrc, il y a 7 jours | 1 auteur, 1 modification
    public static ConnexionBDD GetInstance(string stringConnect)
    {
        if (instance is null)
        {
            instance = new ConnexionBDD(stringConnect);
        }
        return instance;
    }
```

Il fallait créer les méthodes :

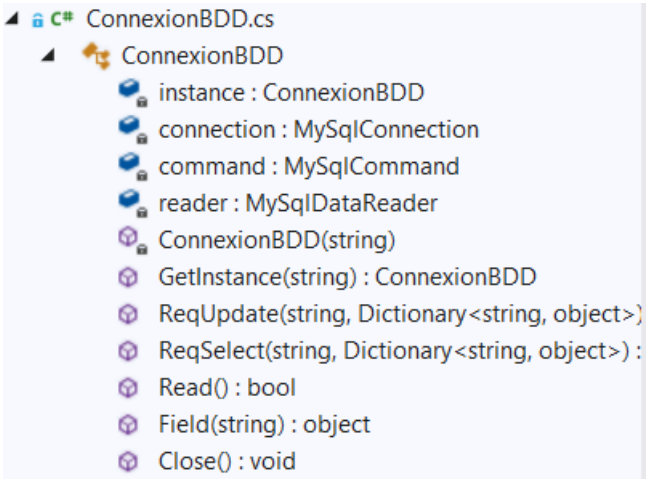
ReqSelect : elle permet de récupérer des données sur la base de données distante.

ReqUpdate : elle permet de d'ajouter de modifier ou de supprimer des données sur la base de données distante.

Read() : elle permet de connaitre l'état du curseur.

Field() : elle nous permet de lire le champs d'un curseur lors de la récupération d'une donnée.

Close : elle permet de fermer le curseur.



Après avoir créé le package Dal, j'ai créé la chaine connexion qui nous permettra d'accéder à la base de données :

```
/// <summary>
/// chaine de connexion à la bdd
/// </summary>
private static string connectionString = "server=localhost;user id=adminmlr1;password=password;database=mlr1;SslMode=none";
```

Ensuite, j'ai créé les 4 classes métiers (Absences, motifs, service et personnel).

Les propriétés de ces différentes classes sont identiques au contenu de la base de données à une exception près : j'ai rajouté dans la classe personnel nom du service pour optimiser l'application.

De plus, pour faciliter l'obtention des valeurs des propriétés des classes métiers et pour rendre le code plus lisible j'ai utilisé des getters comme ci-dessous :

```
4 références | FakherKrc, il y a 7 jours | 1 auteur, 1 modification
public int IdPersonnel { get => idpersonnel; }
4 références | FakherKrc, il y a 7 jours | 1 auteur, 1 modification
public string Nom { get => nom; }
4 références | FakherKrc, il y a 7 jours | 1 auteur, 1 modification
public string Prenom { get => prenom; }
3 références | FakherKrc, il y a 7 jours | 1 auteur, 1 modification
public string Tel { get => tel; }
3 références | FakherKrc, il y a 7 jours | 1 auteur, 1 modification
public string Mail { get => mail; }
2 références | FakherKrc, il y a 7 jours | 1 auteur, 1 modification
public int IdService { get => idservice; }
1 référence | FakherKrc, il y a 7 jours | 1 auteur, 1 modification
public string Service { get => service; }
```

Les setters n'étaient pas utiles et la modification des données s'effectuait dans la base de données distante.

b. Génération de la documentation technique

J'ai généré la documentation technique au format XML en veillant à mettre exclusivement des commentaires utiles.

4. Etape 4 : Coder les fonctionnalités de l'application à partir des cas d'utilisation

a. Codage de la classe AccesDonnees.

Dans un premier temps, j'ai préféré coder la classe AccesDonnees, car je savais quels étaient les données qu'ils faillaient récupérer ou modifier.

De plus en commençant par cette classe, je pouvais tester plus rapidement les fonctionnalités de l'application.

J'ai codé la methode GetLesPersonnels(), GetLesAbsences , GetLesMotifs, GetLesServices de la même façon à quelques exceptions près :

```
/// <summary>
/// Récupère et retourne les absences d'un personnel provenant de la BDD
/// </summary>
/// <returns>liste des absences d'un personnel</returns>
/// <param name="idpersonnel"></param>
1 référence | FakherKrc, Il y a 23 heures | 1 auteur, 4 modifications
public static List<Absence> GetLesAbsences(int idpersonnel)
{
    List<Absence> lesAbsences = new List<Absence>();
    string req = "SELECT datedebut , datefin, idmotif, motif.libelle as motif ";
    req += "FROM absence JOIN motif using(idmotif) ";
    req += " WHERE idpersonnel = @idpersonnel;";
    ConnexionBDD curs = ConnexionBDD.GetInstance(connectionString);
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", idpersonnel);
    curs.ReqSelect(req, parameters);
    while (curs.Read())
    {
        Absence absence = new Absence(idpersonnel, (DateTime)curs.Field("datedebut"), (DateTime)curs.Field("datefin"),
            (int)curs.Field("idmotif"), (string)curs.Field("motif"));
        lesAbsences.Add(absence);
    }
    curs.Close();
    return lesAbsences;
}
```

Grâce à l'identifiant du personnel on récupère la liste des absences d'un personnel.

MySQL DateTime est compatible avec la classe DateTime du langage C# ce qui a permis de faciliter la récupération et la modification d'une date.

Les méthodes pour supprimer, ajouter ou modifier sont similaires, cependant elles utilisent la classe ReqUpdate.

Le contrôle de l'authentification se fait de la manière suivante :

```
/// <summary>
/// Controle si l'utillisateur a le droit de se connecter (login, pwd)
/// </summary>
/// <param name="login"></param>
/// <param name="pwd"></param>
/// <returns></returns>
1 référence | FakherKrc, il y a 5 jours | 1 auteur, 2 modifications
public static Boolean ControleAuthentification(string login, string pwd)
{
    string req = "select * from responsable";
    req += " where login=@login and pwd=SHA2(@pwd, 256);";
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@login", login);
    parameters.Add("@pwd", pwd);
    ConnexionBDD curs = ConnexionBDD.GetInstance(connectionString);
    curs.ReqSelect(req, parameters);
    if (curs.Read())
    {
        curs.Close();
        return true;
    }
    else
    {
        curs.Close();
        return false;
    }
}
```

On récupère les données de la table responsable que l'on compare aux données entrées par l'utilisateur : si on arrive à lire une ligne au moins, alors cela veut dire que les identifiants sont corrects.

b. Codage de la classe contrôle :

Une fois la classe AccesDonnées conçu, il était facile de coder une grande partie de la classe contrôle puisque la vue doit passer par le contrôleur pour accéder aux méthodes de la classe AccesDonnees.

De plus, l'instanciation des 3 frames est déléguée à la classe contrôle : l'instanciation de la Frame d'authentification s'effectue lors de la construction de la classe contrôle.

```
public Controle()
{

    frmAuthentification = new FrmAuthentification(this);
    frmAuthentification.ShowDialog();
}
```

c. Codage de la vue

J'ai codé la vue en m'appuyant sur le diagramme de cas d'utilisation

La Frame Authentification :

```
        /// <summary>
        /// Controle l'authentification au clique de la souris
        /// Si champs incorrect : affiche erreur. P0
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        1 référence | FakherKrc, Il y a 22 heures | 1 auteur, 3 modifications
        private void btnSeconnecter_Click(object sender, EventArgs e)
        {
            if (!txtIdentifiant.Text.Equals("") && !txtMdp.Text.Equals(""))
            {
                if (!controle.ControleAuthentification(txtIdentifiant.Text,  txtMdp.Text))
                {
                    MessageBox.Show("Authentification incorrecte ou vous n'êtes pas admin", "Alerte");
                    txtIdentifiant.Text = "";

                    txtMdp.Text = "";
                    txtIdentifiant.Focus();
                }
            }
            else
            {
                MessageBox.Show("Tous les champs doivent être remplis.", "Information");
            }
        }
    }
}
```

Lorsque l'utilisateur clique sur le bouton, le programme vérifie si les champs ont tous été remplis, si ce n'est pas le cas un message d'erreur s'affiche. Puis il vérifie si les identifiants correspondent bien, si ce n'est pas le cas il renvoie un message d'erreur et efface les données entrées. Si c'est le cas, la méthode appelée instancie la Frame de gestion du personnel.

a) La Frame de gestion du personnel

Lors de l'instanciation de la frame, on récupère l'instance de contrôle et on initialise la frame de la façon suivante :

```
/// <summary>
/// Remplie la fenêtre de la liste des personnels et des services.
/// </summary>
1 référence | FakherKrc, Il y a 1 jour | 1 auteur, 2 modifications
public void Init()
{
    RemplirListePersonnel();
    RemplirListeService();
    grbPersonnel.Enabled = false;
}
```

On remplit la DataGriedView et la ComboBox en récupérant la liste des personnels et des services dans la base de données distante.

Lorsque que l'on clique sur le bouton Ajouter, on peut ajouter un personnel, si on clique sur le bouton enregistrer alors l'évènement btnEnregistrer est appelé :

```
private void btnEnregPersonnel_Click(object sender, EventArgs e)
{
    if (!txtNom.Text.Equals("") && !txtPrenom.Text.Equals("") && !txtTel.Text.Equals("") && !txtMail.Text.Equals("") && cboService.SelectedIndex != -1)
    {
        Service service = (Service)bdgServices.List[bdgServices.Position];
        int idpersonnel = 0;
        if (enCoursDeModif)
        {
            idpersonnel = (int)dgvPersonnels.SelectedRows[0].Cells["idpersonnel"].Value;
        }
        Personnel personnel = new Personnel(idpersonnel, txtNom.Text, txtPrenom.Text, txtTel.Text, txtMail.Text, service.Idservice, service.Nom);
        if (enCoursDeModif)
        {
            if (MessageBox.Show("Voulez-vous vraiment Enregistrer la modification d'un personnel ?", "Confirmation",
                MessageBoxButtons.YesNo) == DialogResult.Yes)
            {
                controle.UpdatePersonnel(personnel);
                enCoursDeModif = false;
                grbPersonnel.Text = "ajouter un personnel";
            }
            else
            {
                Annuler();
            }

        }
        else
        {
            if(MessageBox.Show("Voulez-vous vraiment Enregistrer l'ajout d'un personnel?",
                "Confirmation", MessageBoxButtons.YesNo) == DialogResult.Yes)
            {
                controle.AddPersonnel(personnel);
            }
            else
            {
                Annuler();
```

Si tous les champs n'ont pas été rempli cela renvoie une erreur, sinon une demande de confirmation est envoyé à l'utilisateur, si l'utilisateur refuse la méthode Annuler() est appelé :

```
3 références | FakherKrc, Il y a 1 jour | 1 auteur, 1 modification
private void Annuler()
{
    ViderPersonnel();
    grbPersonnel.Enabled = true;
    enCoursDeModif = false;
    grbPersonnel.Text = "ajouter un développeur";
    btnAjouter.Enabled = true;
    btnModifier.Enabled = true;
    grbPersonnel.Enabled = false;
}
```

Lorsque l'utilisateur clique sur le bouton modifier le programme est similaire sauf que l'identifiant du personnel est récupéré préalablement pour permettre de modifier un personnel dans la base de données distante.

La méthode évènementielle sur le bouton supprimer envoie une confirmation à l'utilisateur, si l'utilisateur répond oui, le personnel est supprimé de la base de données distante et la DataGriedView est actualisée.

La Frame Absence est similaire à la Frame Personnel à quelques exceptions près.

III. Bilan générale.

Pour concevoir cette application, j'ai utilisé le langage C# et la base de données mySQL. J'ai sauvegardé mon travail à chaque étape, en commençant d'abord par créer la classe de connexion à la BDD, puis j'ai codé les classes métiers et la classe qui nous permet de récupérer et de modifier des données importantes pour l'application. Puis j'ai codé la classe contrôle et l'interaction entre l'utilisateur et l'application. Les commentaires insérés dans les codes m'ont permis de générer la documentation technique au format XML