

Compte rendu

Sommaire

CONTEXTE	2
MISSION	2
DESCRIPTION DE L'ACTIVITÉ.....	3
ETAPE 1 : Préparation de l'environnement de travail et création de la BDD	3
ETAPE 2 : Dessiner les interfaces, structurer l'application en MVC, créer un dépôt, coder le visuel ..	4
ETAPE 3 : Coder le modèle et les outils de connexion, générer la documentation technique	7
ETAPE 4 : Coder les fonctionnalités de l'application.....	10

CONTEXTE

InfoTech Services 86, est une Entreprise de Services Numériques spécialisée dans le développement informatique (applications de bureau, web, mobile), l'hébergement de site web, l'infogérance, la gestion de parc informatique et l'ingénierie système et réseau. Elle répond régulièrement à des appels d'offres en tant que société d'infogérance et prestataire de services informatiques. ITS 86 est avant tout une équipe composée de 32 collaborateurs, administratifs, ingénieurs et techniciens. Les activités d'ITS 86 sont organisées autour de deux pôles : Développement et Systèmes et réseau.

Parmi les marchés gagnés récemment par ITS 86, on trouve celui de la gestion du parc informatique du réseau des médiathèques de la Vienne, MediaTek86. Les médiathèques mettent à la disposition de leurs adhérents des documents sur divers supports (papier, cédérom, livre numérique, DVD, etc.)

MISSION

Travaillant en tant que technicien développeur junior pour la société InfoTech Services 86 qui a remporté le marché pour différentes intervention au sein du réseau MediaTek86, il m'a été confié le développement d'une application de bureau qui permet de gérer le personnel de chaque médiathèque, leur affectation à un service et leurs absences.

L'application à été créer en C# et la gestion de la base de donnée est gérer avec le SGBDR MySQL.

DESCRIPTION DE L'ACTIVITÉ

ETAPE 1 : Préparation de l'environnement de travail et création de la BDD

Pour commencer, j'ai décidé d'utiliser Wampserver qui contient l'application phpMyAdmin afin de pouvoir gérer ma base de données avec le Système de Gestion de Base de Données MySQL. Ayant choisis le C # comme langage de programmation, j'ai décidé d'utiliser l'IDE Visual Studio 2019. Pour le logiciel de modélisation, j'avais à ma disposition le logiciel Win'design.

Ayant à ma disposition le schéma conceptuel de données je l'ai donc ouvert avec Win'design afin de générer le script SQL. J'ai ensuite été sur phpMyAdmin afin d'y créer la base de données sous MySQL que j'ai nommé "mediatek86" puis j'ai importé le script SQL.

J'ai créé un utilisateur (login/pwd) afin que seul ce utilisateur puisse avoir accès à tous les privilèges sur cette base de données. Je l'ai ensuite ajouté dans une nouvelle table "responsable" avec deux champs "login" et "pwd" qui servira pour la connexion à l'application.

Pour finir cette étape, j'ai alimenté la base de données :

- la table "responsable" contient le login et pwd chiffré avec la fonction SHA2("pwd", 256),
- la table "motif" contient les motifs suivants : vacances, maladie, motif familial, congé parental,
- la table "service" contient les services suivants : administratif, médiation culturelle, prêt,
- les tables "personnel" et "absence" ont été remplies aléatoirement à l'aide du site "<https://www.generatedata.com/>"

Bilan de cette étape :

A la fin de cette étape mon environnement de travail est prêt et la base de données est créée et prête à l'emploi.

ETAPE 2 : Dessiner les interfaces, structurer l'application en MVC, créer un dépôt, coder le visuel

J'avais à ma disposition le dossier documentaire qui contenait le diagramme de cas d'utilisation ainsi que le descriptif de chaque cas d'utilisation. A l'aide de la description des cas d'utilisation, j'ai pu dessiner les interfaces sur le logiciel Pencil.

Exemple de dessin de certaines interfaces :

Interface contenant la gestion de la liste du personnel :

Nom	Prenom	Téléphone	Mail
Moralles	Garrison	0516691213	dolon@Morbiquis.ca
Phillips	Odette	0253244541	eu@tellusAeneanegestas.ca
Larsen	Chaim	0848120702	turpis.Aliquam.adipiscing@Maecenasmalesuada.net
Colley	Gray	0434513006	in.faucibus.orci@sitametdapibus.org
Coleman	Rhea	0993503553	vitae@maurisaliquameu.edu

Nom :

Service :

Prénom :

Tél :

Mail :

Interface pour modifier un personnel :

Modification du personnel :

Nom :

Service :

Prénom :

Tél :

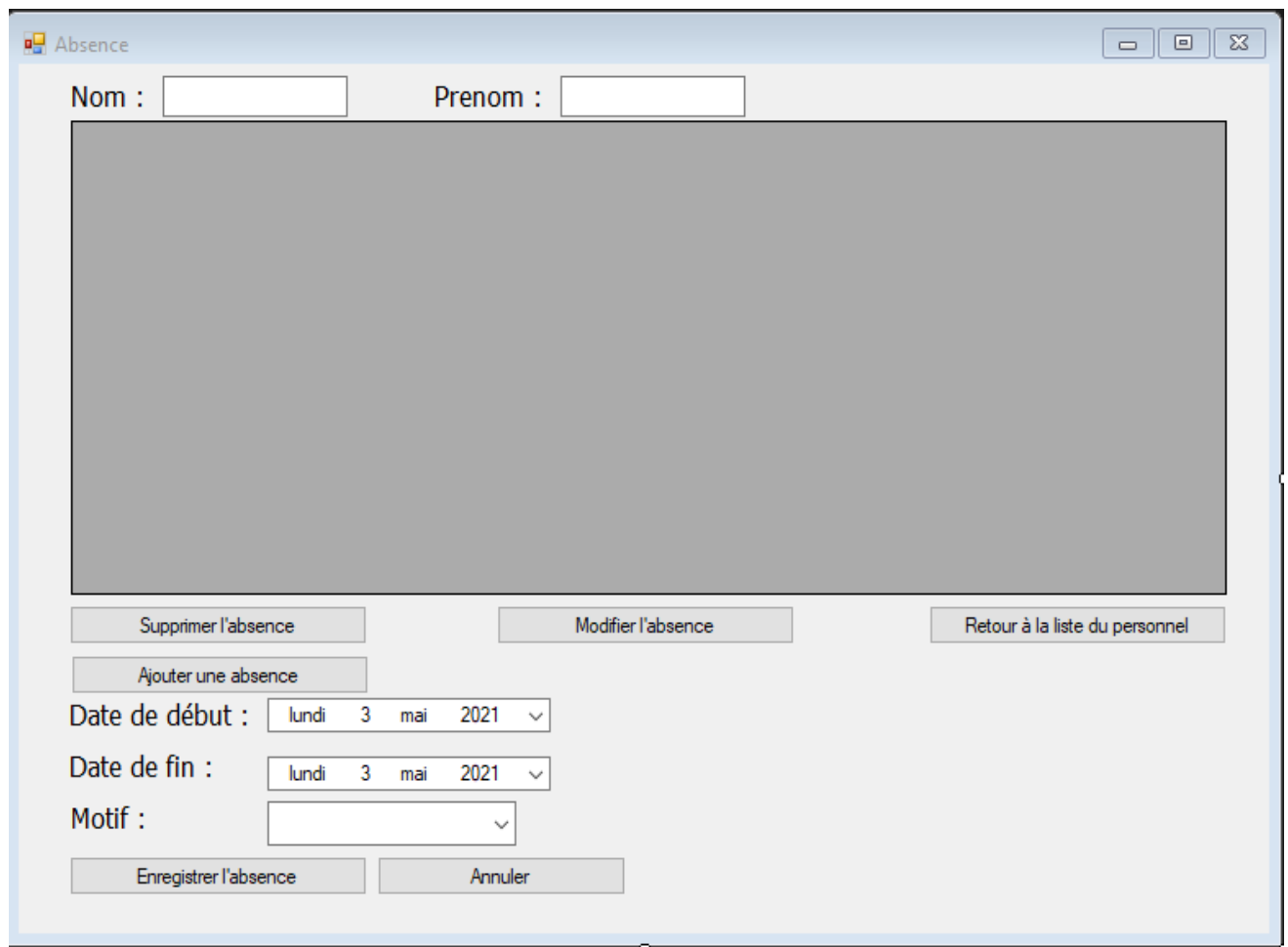
Mail :

Après la création de toutes les interfaces sur Pencil, j'ai créé une nouvelle application sur Visual Studio 2019 puis j'ai créé les dossiers "modèle", "vue" et "contrôle" afin de programmer en respectant la méthode MVC.

J'ai ensuite créé un dépôt distant sur GitHub nommé "MediaTek86" afin d'y faire une première sauvegarde de mon projet. Puis pour compléter cette étape, j'ai codé le visuel de toutes les interfaces du projet que j'ai rangé dans le dossier "vue".

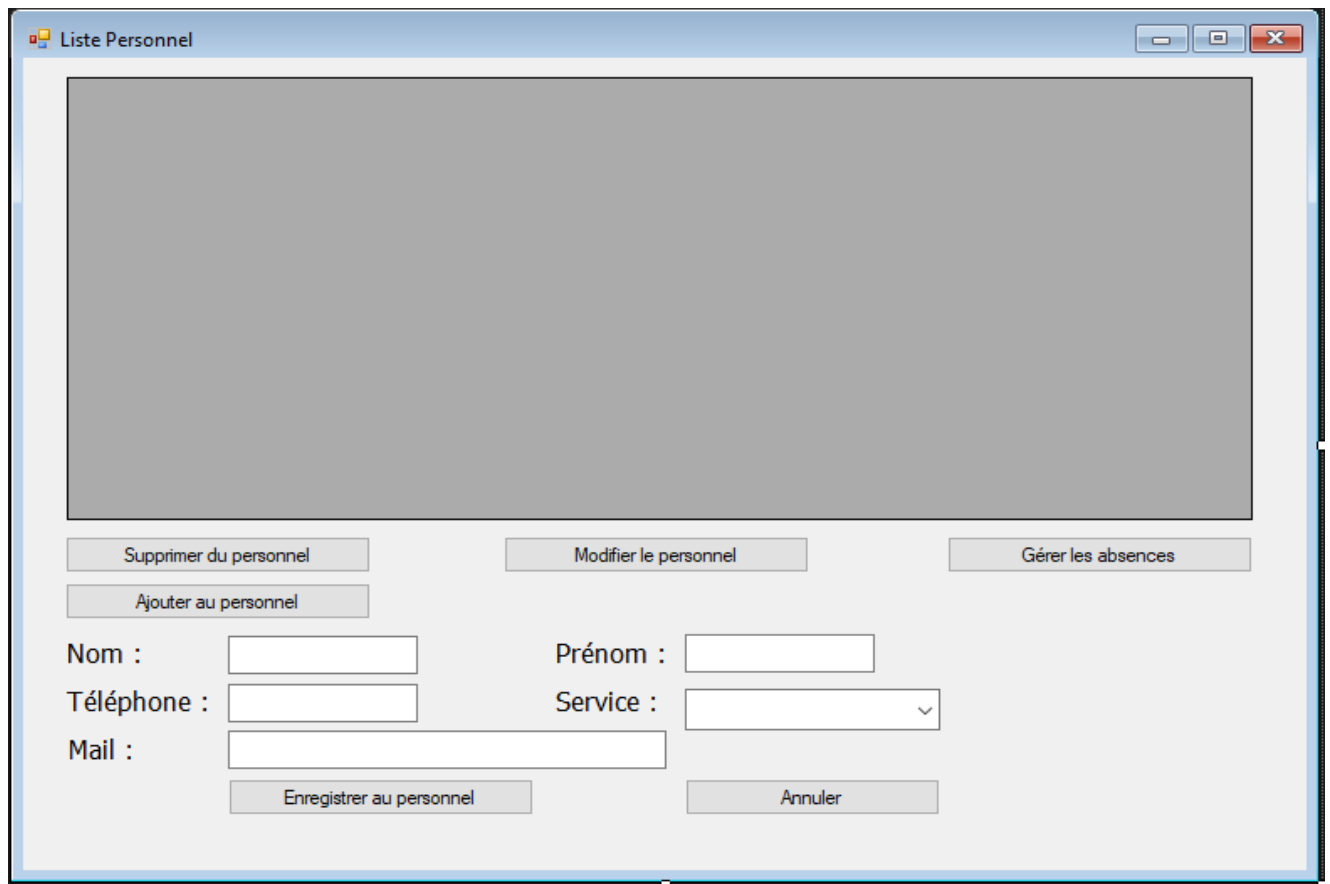
Exemple de visuel de certaines interfaces :

Interface contenant la gestion des absences :



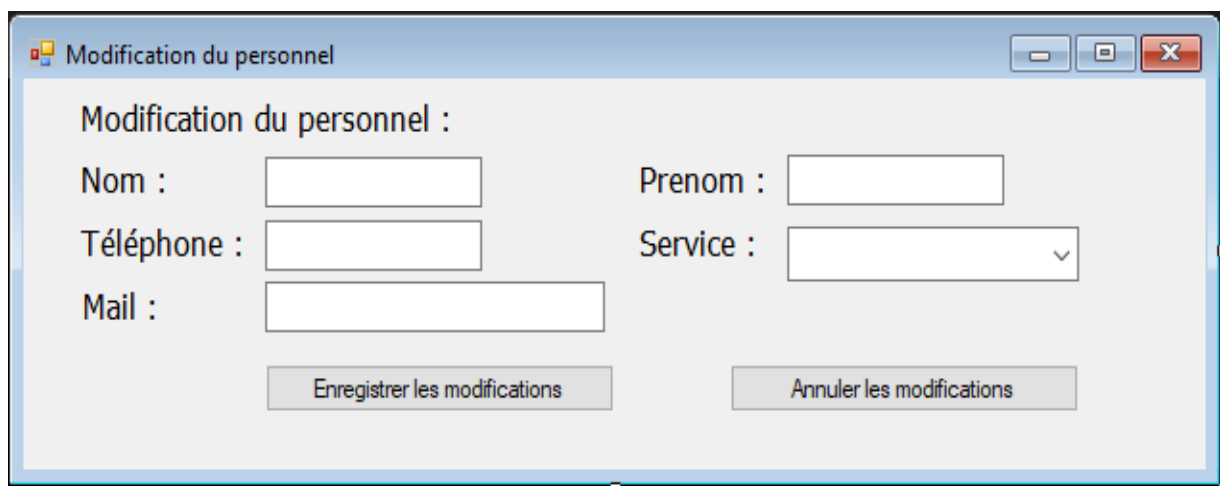
The screenshot shows a web application window titled "Absence". At the top, there are two input fields for "Nom :" and "Prenom :". Below these is a large, empty rectangular area, likely a placeholder for a list or table. At the bottom, there are several buttons: "Supprimer l'absence", "Modifier l'absence", "Retour à la liste du personnel", "Ajouter une absence", "Enregistrer l'absence", and "Annuler". Additionally, there are two date pickers for "Date de début :" and "Date de fin :", both showing "lundi 3 mai 2021", and a dropdown menu for "Motif :".

Interface contenant la gestion de la liste du personnel :



The screenshot shows a window titled "Liste Personnel". It features a large, empty rectangular area for displaying a list of personnel. Below this area, there are four buttons: "Supprimer du personnel", "Ajouter au personnel", "Modifier le personnel", and "Gérer les absences". At the bottom, there is a form for adding new personnel with fields for "Nom", "Prénom", "Téléphone", "Service" (a dropdown menu), and "Mail". There are also "Enregistrer au personnel" and "Annuler" buttons at the bottom of the form.

Interface pour modifier un personnel:



The screenshot shows a window titled "Modification du personnel". It contains a form for editing personnel details with fields for "Nom", "Prenom", "Téléphone", "Service" (a dropdown menu), and "Mail". At the bottom of the form, there are two buttons: "Enregistrer les modifications" and "Annuler les modifications".

Bilan de cette étape :

A la fin de cette étape les interfaces sont dessinées dans l'outil Pencil et l'application est créée avec le visuel des interfaces codé. Et le travail effectué est enregistré sur GitHub.

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

Tout d'abord, j'ai configuré Visual Studio pour avoir accès à la base de données. J'ai ensuite créé un package "connexion" avec une classe singleton nommé "ConnexionBDD" qui contient le code pour créer la connexion à la base de données ainsi que toutes les méthodes nécessaires pour gérer l'accès aux données.

Constructeur de la classe qui gère la connexion à la BDD:

```
/// <summary>
/// Constructeur privé pour créer la connexion à la BDD et l'ouvrir
/// </summary>
/// <param name="stringConnect">chaîne de connexion</param>
1 référence
private ConnexionBDD(string stringConnect)
{
    try
    {
        connection = new MySqlConnection(stringConnect);
        connection.Open();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        Application.Exit();
    }
}
```

Méthode pour exécuter les requêtes autre que "select":

```
/// <summary>
/// Exécution d'une requête autre que "select"
/// </summary>
/// <param name="stringQuery">requête autre que select</param>
/// <param name="parameters">dictionnaire contenant les paramètres</param>
6 références
public void ReqUpdate(string stringQuery, Dictionary<string, object> parameters)
{
    try
    {
        command = new MySqlCommand(stringQuery, connection);
        if (!(parameters is null))
        {
            foreach (KeyValuePair<string, object> parameter in parameters)
            {
                command.Parameters.Add(new MySqlParameter(parameter.Key, parameter.Value));
            }
        }
        command.Prepare();
        command.ExecuteNonQuery();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Méthode pour exécuter les requêtes de type "select":

```
/// <summary>
/// Exécute une requête type "select" et valorise le curseur
/// </summary>
/// <param name="stringQuery">requête select</param>
5 références
public void ReqSelect(string stringQuery, Dictionary<string, object> parameters)
{
    try
    {
        command = new MySqlCommand(stringQuery, connection);
        if (!(parameters is null))
        {
            foreach (KeyValuePair<string, object> parameter in parameters)
            {
                command.Parameters.Add(new MySqlParameter(parameter.Key, parameter.Value));
            }
        }
        command.Prepare();
        reader = command.ExecuteReader();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Méthode pour la gestion du curseur :

```
/// <summary>
/// Tente de lire la ligne suivante du curseur
/// </summary>
/// <returns>false si fin de curseur atteinte</returns>
5 références
public bool Read()
{
    if (reader is null)
    {
        return false;
    }
    try
    {
        return reader.Read();
    }
    catch
    {
        return false;
    }
}
```


Méthode pour lire le contenu d'un champ :

```
/// <summary>
/// Retourne le contenu d'un champ dont le nom est passé en paramètre
/// </summary>
/// <param name="nameField">nom du champ</param>
/// <returns>valeur du champ</returns>
18 références
public object Field(string nameField)
{
    if (reader is null)
    {
        return null;
    }
    try
    {
        return reader[nameField];
    }
    catch
    {
        return null;
    }
}
```

Après mettre occuper du package "connexion", j'ai créé un package "dal" avec la classe "AccesDonnees" qui va servir à répondre aux demande du contrôleur.

Pour cette étape, dans cette classe, j'ai seulement déclaré la propriété "connectionString" qui contient la chaîne de connexion à la base de données.

```
/// <summary>
/// chaîne de connexion à la bdd
/// </summary>
private static string connectionString = "server=localhost;user id=admin;password=WHIwAcI3DiYBRv0;database=mediatek86;SslMode=none";
```

Ensuite dans le package "modele", j'ai créé les classes métiers correspondant aux tables de la base de données. Puis pour finir j'ai générer la documentation technique et j'ai sauvegardé le travail réalisé sur cette étape sur le dépôt distant.

Bilan de cette étape : A la fin de cette étape, le package "connexion" est créé et contient la classe singleton "ConnexionBDD", le package "dal" est créé avec la classe "AccesDonnees" qui contiendra les requêtes et le package "modele" est créé avec toutes les classes métiers correspondant aux tables de la base de données.

ETAPE 4 : Coder les fonctionnalités de l'application

Se connecter :

Pour la première fonctionnalité, je me suis chargé de la fenêtre de connexion. En cliquant sur le bouton « Se connecter », le programme appelle la fonction « btnSeConnecter_Click » qui vérifie si le login et le mot de passe sont remplis et si c'est le cas, il fait appel à la fonction « ControleAuthentification » dans la classe « Contrôle ». Cette fonction fait ensuite appel à la fonction « ControleAuthentification » de la classe « AccesDonnees » qui exécute la requête qui vérifie si le login et le mot de passe existe dans la base de donnée et ouvre ensuite la fenêtre qui affiche la liste du personnel.

```
/// <summary>
/// Permet de se connecter à l'application
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void btnSeConnecter_Click(object sender, EventArgs e)
{
    if (!txtLogin.Text.Equals("") && !txtMotDePasse.Text.Equals(""))
    {
        if (!controle.ControleAuthentification(txtLogin.Text, txtMotDePasse.Text))
        {
            MessageBox.Show("Authentification incorrecte ou vous n'êtes pas admin", "Alerte");
            txtLogin.Text = "";
            txtMotDePasse.Text = "";
            txtLogin.Focus();
        }
    }
    else
    {
        MessageBox.Show("Tous les champs doivent être remplis.", "Information");
    }
}
```

```
/// <summary>
/// Demande la vérification de l'authentification
/// Si correct, alors ouvre la fenêtre principale
/// </summary>
/// <param name="login"></param>
/// <param name="pwd"></param>
/// <returns></returns>
1 référence
public Boolean ControleAuthentification(string login, string pwd)
{
    if (AccesDonnees.ControleAuthentification(login, pwd))
    {
        frmSeConnecter.Hide();
        frmListePersonnel = new frmListePersonnel(this);
        frmListePersonnel.ShowDialog();
        return true;
    }
    else
    {
        return false;
    }
}
```

```
/// <summary>
/// Demande la vérification de l'authentification
/// Si correct, alors ouvre la fenêtre principale
/// </summary>
/// <param name="login"></param>
/// <param name="pwd"></param>
/// <returns></returns>
1 référence
public Boolean ControleAuthentification(string login, string pwd)
{
    if (AccesDonnees.ControleAuthentification(login, pwd))
    {
        frmSeConnecter.Hide();
        frmListePersonnel = new frmListePersonnel(this);
        frmListePersonnel.ShowDialog();
        return true;
    }
    else
    {
        return false;
    }
}
```

Ajouter un personnel :

Pour ajouter un personnel, il faut cliquer sur le bouton « Ajouter un personnel ». Le clic sur ce bouton fait apparaître les zones de saisies grâce à la fonction « GerezZoneTexte ». Une fois les zones saisies il faut cliquer sur le bouton « Enregistrer au personnel ». La fonction sur ce bouton vérifie si les zones de saisies sont bien remplies puis appelle la fonction « AddPersonnel » de la classe « Contrôle ». Cette fonction récupère en paramètre les informations nécessaires pour créer le personnel et fait appel à la fonction « AddPersonnel » de la classe « AccesDonnees » qui exécute la requête d'ajout du personnel.

```
/// <summary>
/// Enregistre et ajoute un personnel à la liste
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void btnEnregistrerPersonnel_Click(object sender, EventArgs e)
{
    if (!txtNom.Text.Equals("") && !txtPrenom.Text.Equals("") && !txtTel.Text.Equals("") && !txtMail.Text.Equals("") && cbbService.SelectedIndex != -1)
    {
        controle.AddPersonnel(txtNom.Text, txtPrenom.Text, txtTel.Text, txtMail.Text);
        RemplirListePersonnel();
        ViderZoneTexte();
    }
    else
    {
        MessageBox.Show("Tous les champs doivent être remplis.", "Information");
    }
}
```

```
/// <summary>
/// Demande d'ajout d'un personnel
/// </summary>
/// <param name="nom"></param>
/// <param name="prenom"></param>
/// <param name="tel"></param>
/// <param name="mail"></param>
1 référence
public void AddPersonnel(string nom, string prenom, string tel, string mail)
{
    Service service = (Service)frmListePersonnel.bdgService.List[frmListePersonnel.bdgService.Position];
    int idpersonnel = 0;
    Personnel personnel = new Personnel(idpersonnel, nom, prenom, tel, mail, service.Idservice, service.Nom);
    AccesDonnees.AddPersonnel(personnel);
}
```

Supprimer un personnel :

Pour la suppression d'un personnel, en cliquant sur le bouton « Supprimer un personnel », il fait appel à la fonction « DelPersonnel » de la classe « Contrôle » qui regarde la ligne sélectionné dans la liste et exécute ensuite deux requêtes. La première, efface toutes les absences liées à ce personnel et la deuxième efface le personnel.

```

/// <summary>
/// Demande de suppression d'un personnel
/// </summary>
1 référence
public void DelPersonnel()
{
    Personnel personnel = (Personnel)frmListePersonnel.bdgPersonnel.List[frmListePersonnel.bdgPersonnel.Position];
    if (MessageBox.Show("Voulez-vous vraiment supprimer " + personnel.Nom + " " + personnel.Prenom + " ?", "Confirmation de suppression", MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
        AccesDonnees.DelAllAbsence(personnel);
        AccesDonnees.DelPersonnel(personnel);
    }
}

```

```

/// <summary>
/// Suppression d'un personnel
/// </summary>
/// <param name="personnel">objet developpeur à supprimer</param>
1 référence
public static void DelPersonnel(Personnel personnel)
{
    string req = "delete from personnel where idpersonnel = @idpersonnel;";
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", personnel.Idpersonnel);
    ConnexionBDD conn = ConnexionBDD.GetInstance(connectionString);
    conn.ReqUpdate(req, parameters);
}

```

```

/// <summary>
/// Supprime la totalité des absences d'un personnel
/// </summary>
/// <param name="personnel"></param>
1 référence
public static void DelAllAbsence(Personnel personnel)
{
    string req = "delete from absence where idpersonnel = @idpersonnel;";
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", personnel.Idpersonnel);
    ConnexionBDD conn = ConnexionBDD.GetInstance(connectionString);
    conn.ReqUpdate(req, parameters);
}

```

Modifier un personnel :

Il faut cliquer sur le bouton « Modification d'un personnel » pour accéder à la fenêtre de modification. Sur cette nouvelle fenêtre une fois les modifications effectué, en cliquant sur le bouton « Enregistrer les modifications » le programme appelle la fonction « EnrUpdateAbsence » de la classe « Contrôle » qui récupère toutes les informations mise à jours et appelle la fonction « EnrUpdateAbsence » de la classe « AccesDonnees » qui exécute la requête de modifications et retourne ensuite sur la fenêtre qui contient la liste du personnel.

```

/// <summary>
/// Enregistre les modifications d'un personnel
/// </summary>
/// <param name="nom"></param>
/// <param name="prenom"></param>
/// <param name="tel"></param>
/// <param name="mail"></param>
1 référence
public void EnrUpdatePersonnel(string nom, string prenom, string tel, string mail)
{
    Service service = (Service)frmModificationPersonnel.bdgService.List[frmModificationPersonnel.bdgService.Position];
    Personnel personnel = new Personnel(frmModificationPersonnel.GetIdPersonnel(), nom, prenom, tel, mail, service.Idservice, service.Nom);
    AccesDonnees.UpdatePersonnel(personnel);
    frmModificationPersonnel.Hide();
    frmListePersonnel = new frmListePersonnel(this);
    frmListePersonnel.ShowDialog();
}

```

```

/// <summary>
/// Modification d'un personnel
/// </summary>
/// <param name="personnel"></param>
1 référence
public static void UpdatePersonnel(Personnel personnel)
{
    string req = "update personnel set nom = @nom, prenom = @prenom, tel = @tel, mail = @mail, idservice = @idservice ";
    req += "where idpersonnel = @idpersonnel;";
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", personnel.Idpersonnel);
    parameters.Add("@nom", personnel.Nom);
    parameters.Add("@prenom", personnel.Prenom);
    parameters.Add("@tel", personnel.Tel);
    parameters.Add("@mail", personnel.Mail);
    parameters.Add("@idservice", personnel.Idservice);
    ConnexionBDD conn = ConnexionBDD.GetInstance(connectionString);
    conn.ReqUpdate(req, parameters);
}

```

Afficher les absences :

Pour afficher les absences d'un personnel il faut le sélectionner et cliquer sur « Gérer les absences » qui va faire appel à la fonction « DemGererAbsence » de la classe « Contrôle » qui récupère les informations du personnel qui a été sélectionné et qui va ensuite remplir la liste des absences grâce à la requête.

```
/// <summary>
/// Demande pour accéder aux absences d'un personnel
/// </summary>
1 référence
public void DemGererAbsence()
{
    Personnel personnel = (Personnel)frmListePersonnel.bdgPersonnel.List[frmListePersonnel.bdgPersonnel.Position];
    savePersonnel = personnel;
    frmGererAbsence = new frmGererAbsence(this);
    frmGererAbsence.RemplirListeAbsence(personnel);
    frmGererAbsence.SetNom(personnel.Nom);
    frmGererAbsence.SetPrenom(personnel.Prenom);
    frmGererAbsence.SetIdPersonnel(personnel.Idpersonnel);
    Console.WriteLine("Nom :" + personnel.Nom);
    Console.WriteLine("Prenom :" + personnel.Prenom);
    frmGererAbsence.ShowDialog();
}
```

```
/// <summary>
/// Récupère et retourne les absences d'un personnel provenant de la BDD
/// </summary>
/// <param name="personnel"></param>
/// <returns></returns>
1 référence
public static List<Absence> GetLesAbsences(Personnel personnel)
{
    List<Absence> lesAbsences = new List<Absence>();
    string req = "select p.idpersonnel, p.nom as nom, p.prenom as prenom, a.datedebut as datedebut, a.datefin as datefin, m.libelle as motif, m.idmotif as idmotif ";
    req += "from personnel p JOIN absence a ON p.idpersonnel = a.idpersonnel JOIN motif m ON a.idmotif = m.idmotif ";
    req += "where p.idpersonnel = @idpersonnel ";
    req += "order by datedebut desc";
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", personnel.Idpersonnel);
    ConnexionBDD curs = ConnexionBDD.GetInstance(connectionString);
    curs.ReqSelect(req, parameters);
    while (curs.Read())
    {
        Absence absence = new Absence((int)curs.Field("idpersonnel"), (string)curs.Field("nom"), (string)curs.Field("prenom"), (DateTime)curs.Field("datedebut"),
            (DateTime)curs.Field("datefin"), (int)curs.Field("idmotif"), (string)curs.Field("motif"));
        lesAbsences.Add(absence);
    }
    curs.Close();
    return lesAbsences;
}
```

En ce qui concerne les trois derniers cas d'utilisations, c'est-à-dire l'ajout, la modification et la suppression d'une absence, ils sont programmés de la même manière que pour un personnel.

Gère la modification d'une absence :

```
/// <summary>
/// Demande pour accéder à la modification d'une absence
/// </summary>
1 référence
public void DemUpdateAbsence()
{
    // Ferme la fenêtre active
    frmGererAbsence.Hide();
    frmModificationAbsence = new frmModificationAbsence(this);
    Absence absence = (Absence)frmGererAbsence.bdgAbsence.List[frmGererAbsence.bdgAbsence.Position];
    frmModificationAbsence.SetIdPersonnel(absence.Idpersonnel);
    frmModificationAbsence.SetNom(absence.Nom);
    frmModificationAbsence.SetPrenom(absence.Prenom);
    frmModificationAbsence.SetDateDebut(absence.Date_de_debut);
    frmModificationAbsence.SetDateFin(absence.Date_de_fin);
    frmModificationAbsence.SetIdMotif(absence.Idmotif);
    frmModificationAbsence.SetMotif(absence.Motif);
    // Ouvre la fenêtre frmModificationPersonnel
    frmModificationAbsence.ShowDialog();
}
```

Requête de la modification d'une absence :

```
/// <summary>
/// Modification d'une absence
/// </summary>
/// <param name="absence"></param>
/// <param name="newdatedebut"></param>
/// <param name="newdatefin"></param>
1 référence
public static void UpdateAbsence(Absence absence, DateTime newdatedebut, DateTime newdatefin)
{
    string req = "update absence set datedebut = @newdatedebut, datefin = @newdatefin, idmotif = @idmotif ";
    req += "where idpersonnel = @idpersonnel and datedebut = @datedebut and datefin = @datefin;";
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", absence.Idpersonnel);
    parameters.Add("@newdatedebut", newdatedebut);
    parameters.Add("@newdatefin", newdatefin);
    parameters.Add("@datedebut", absence.Date_de_debut);
    parameters.Add("@datefin", absence.Date_de_fin);
    parameters.Add("@idmotif", absence.Idmotif);
    ConnexionBDD conn = ConnexionBDD.GetInstance(connectionString);
    conn.ReqUpdate(req, parameters);
}
```

Bilan final :

J'obtiens au final une application complète coder en C# avec tous les cas d'utilisations fonctionnelle. Chaque fonction est expliqué avec des commentaires et la documentation technique est créer. Le programme complet est sauvegardé sur GitHub.