Compte rendu AP2 : MediaTek86

SOMMAIRE :

• Contexte :

Je travaille en tant que technicien développeur junior pour l'ESN InfoTech Service 86. InfoTech Services 86 (ITS 86), est une Entreprise de Services Numériques (ESN) 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.

Une entreprise à justement fait récemment appel à InfoTeck dans le cadre de son développement numérique. MediaTek86 gère les médiathèques de la Vienne, et qui a pour rôle de fédérer les prêts de livres, DVD et CD. Sa demande est de développer la médiathèque numérique pour l'ensemble des médiathèques du département.

• Préparatif des outils :

Ce dispositif applicatif sera écrit en C# en utilisant l'IDE Visual Studio de Microsoft. La base de données sera hébergée en local avec MySQL tout le long du développement en utilisant les outils PHPMyAdmin fournit par WampServer.

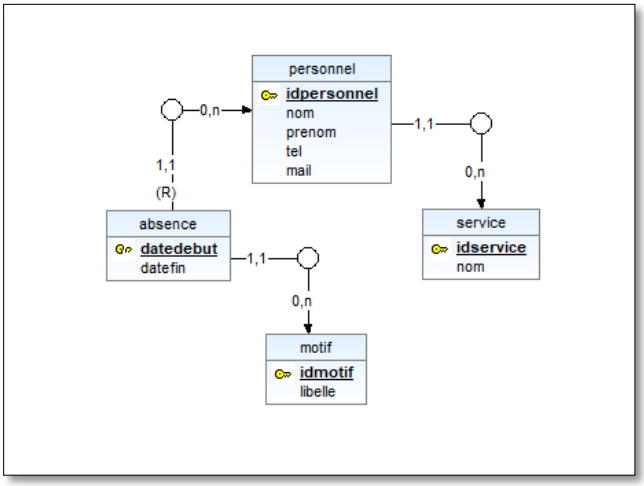
Le projet sera versionné avec un dépôt distant sur GitHub.

Enfin, Le schéma conceptuel de données a été réalisé sous Win'Design et le maquettage sous Pencil.

• Préparation de la base de données :

Le MCD a déjà été réalisé est m'est fourni afin que je mette en place la base de données.

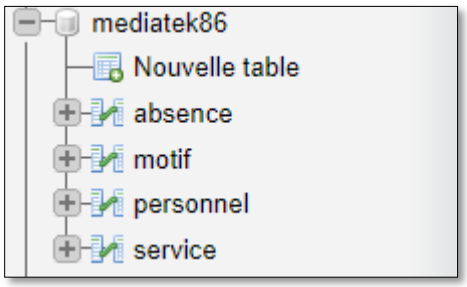
Voici ce schéma :



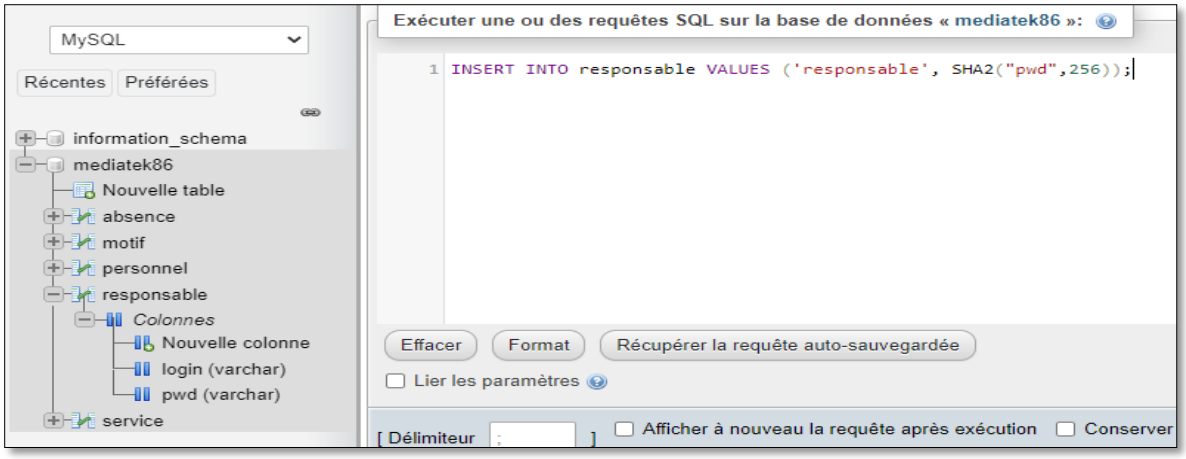
Après avoir généré le model logique, puis le script en MySQL, on obtient le script de création de la base de données. Voici un extrait :

```
DROP DATABASE IF EXISTS MCD AP2;

CREATE DATABASE IF NOT EXISTS MCD AP2;
USE MCD AP2;
# ------------------------------------------------------------
#        TABLE : ABSENCE
# ------------------------------------------------------------

CREATE TABLE IF NOT EXISTS ABSENCE
(
  IDPERSONNEL INTEGER(2) NOT NULL  ,
  DATEDEBUT DATETIME NOT NULL  ,
  IDMOTIF INTEGER(2) NOT NULL  ,
  DATEFIN DATETIME NULL
  , PRIMARY KEY (IDPERSONNEL,DATEDEBUT)
)
 comment = "";

# ------------------------------------------------------------
#        TABLE : MOTIF
```

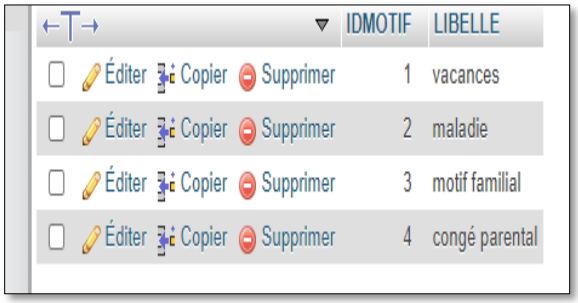
Dans PHPMyAdmin, après avoir exécutée les requêtes SQL du script, on constate que la base de données est créée avec ses tables :



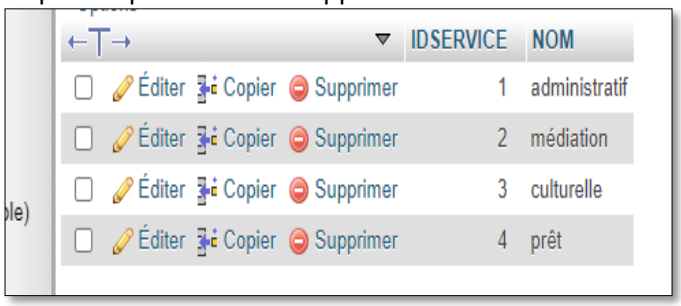
Après avoir créé un compte utilisateur pour gérer cette base de données, je créer une nouvelle table « responsable » ne contenant que deux champs « login » et « pwd ». J'y ajoute un login puis un mot de passe en hachage SHA2 en 256. Ces identifiants de connexion serviront au responsable pour se connecter à l'application. (Le mot de passe «pwd » est temporaire, servant uniquement pendant le développement de l'application)



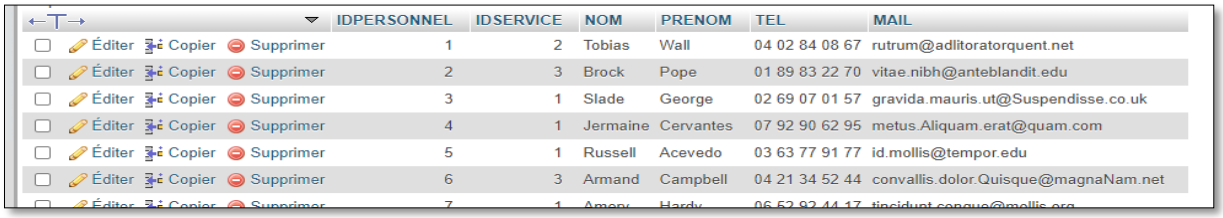
Les tables suivantes sont remplies avec les vrais données sauf pour le personnel et les absences, ces deux tables ont été remplir de données aléatoires pour la phase de développement.



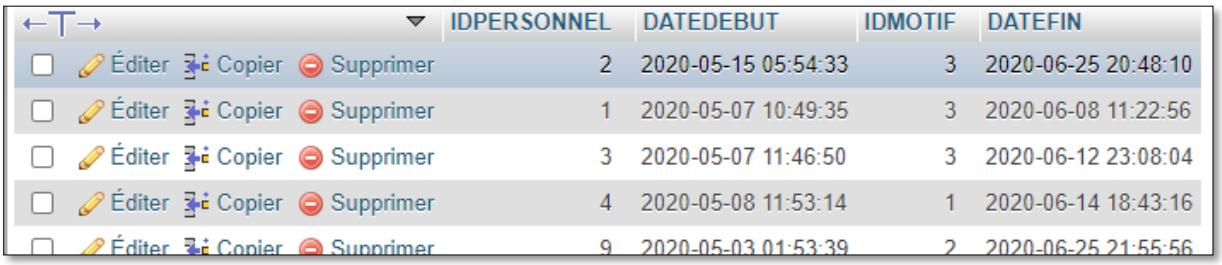
	IDMOTIF	LIBELLE
Éditer Copier Supprimer	1	vacances
Éditer Copier Supprimer	2	maladie
Éditer Copier Supprimer	3	motif familial
Éditer Copier Supprimer	4	congé parental



	IDSERVICE	NOM
Éditer Copier Supprimer	1	administratif
Éditer Copier Supprimer	2	médiation
Éditer Copier Supprimer	3	culturelle
Éditer Copier Supprimer	4	prêt



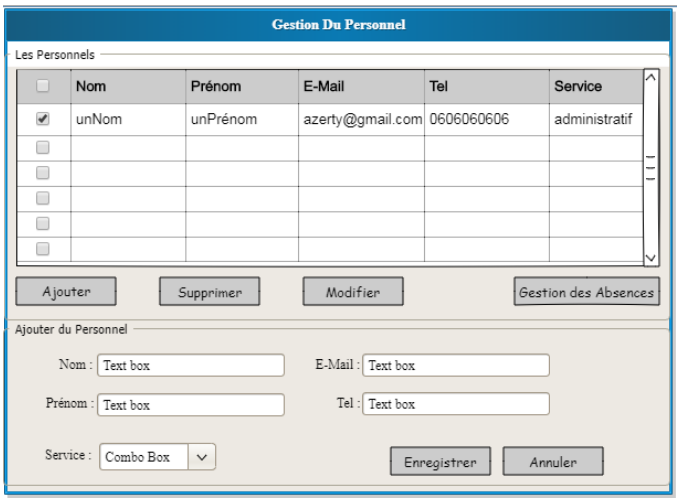
	IDPERSONNEL	IDSERVICE	NOM	PRENOM	TEL	MAIL
Éditer Copier Supprimer	1	2	Tobias	Wall	04 02 84 08 67	rutrum@adlitoratorquent.net
Éditer Copier Supprimer	2	3	Brock	Pope	01 89 83 22 70	vitae.nibh@anteblandit.edu
Éditer Copier Supprimer	3	1	Slade	George	02 69 07 01 57	gravida.mauris.ut@Suspendisse.co.uk
Éditer Copier Supprimer	4	1	Jermaine	Cervantes	07 92 90 62 95	metus.Aliquam.erat@quam.com
Éditer Copier Supprimer	5	1	Russell	Acevedo	03 63 77 91 77	id.mollis@tempor.edu
Éditer Copier Supprimer	6	3	Armand	Campbell	04 21 34 52 44	convallis.dolor.Quisque@magnaNam.net



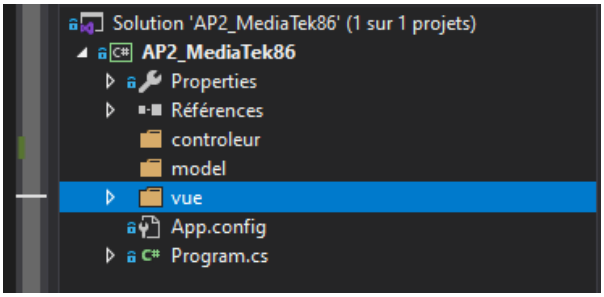
	IDPERSONNEL	DATEDEBUT	IDMOTIF	DATEFIN
Éditer Copier Supprimer	2	2020-05-15 05:54:33	3	2020-06-25 20:48:10
Éditer Copier Supprimer	1	2020-05-07 10:49:35	3	2020-06-08 11:22:56
Éditer Copier Supprimer	3	2020-05-07 11:46:50	3	2020-06-12 23:08:04
Éditer Copier Supprimer	4	2020-05-08 11:53:14	1	2020-06-14 18:43:16
Éditer Copier Supprimer	9	2020-05-03 01:53:39	2	2020-06-25 21:55:56

• Mise en place de l'environnement de développement :

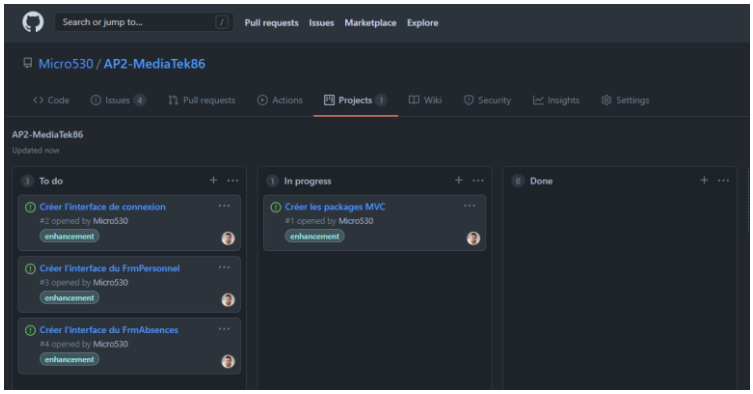
A partir du diagramme de cas d'utilisations et du descriptif des cas d'utilisations, j'ai réalisé tout d'abord avec l'outils Pencil les maquettes des interfaces dont voici un extrait :



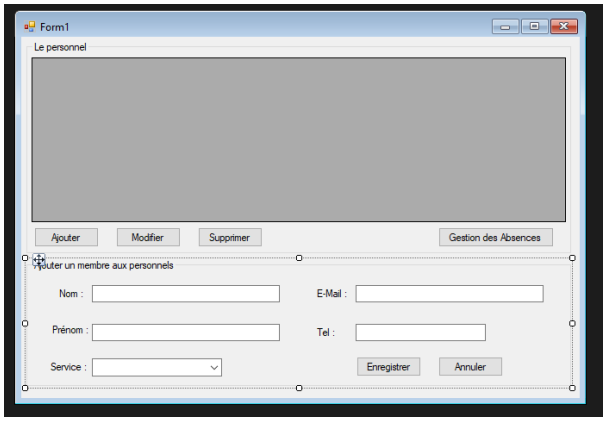
Suite à ça, je créer l'application sous Visual Studio en prenant soins de créer les 3 packages, Model, Vue, Contrôleur de l'architecture MVC :

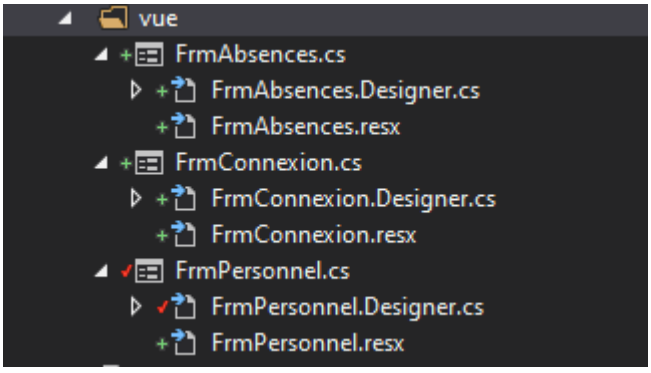


Mise en place du dépôt sur GitHub afin de versionner le projet, avec la mise en place du ToDo afin de gérer les issues :

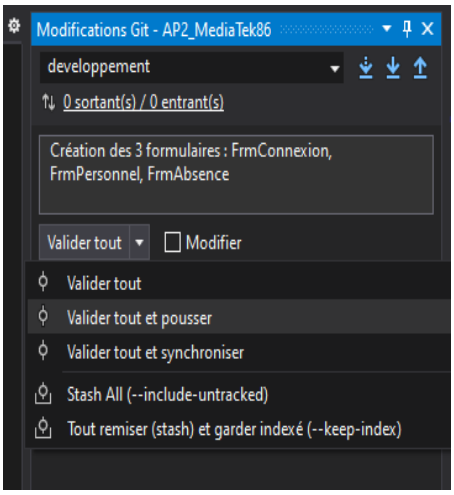


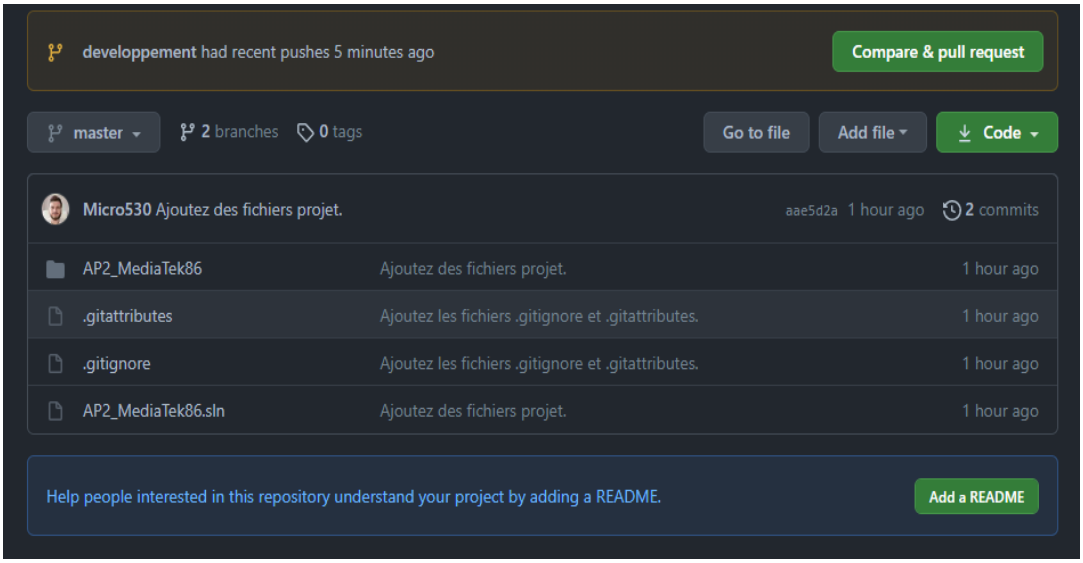
Puis création des formulaires sous Visual Studio :





Après un commit sur le dépôt local, on Push sur le dépôt distant, ce qui envoie instantanément une notification sur GitHub, après vérification je valide le merge, j'en profite pour mettre à jour les issues :

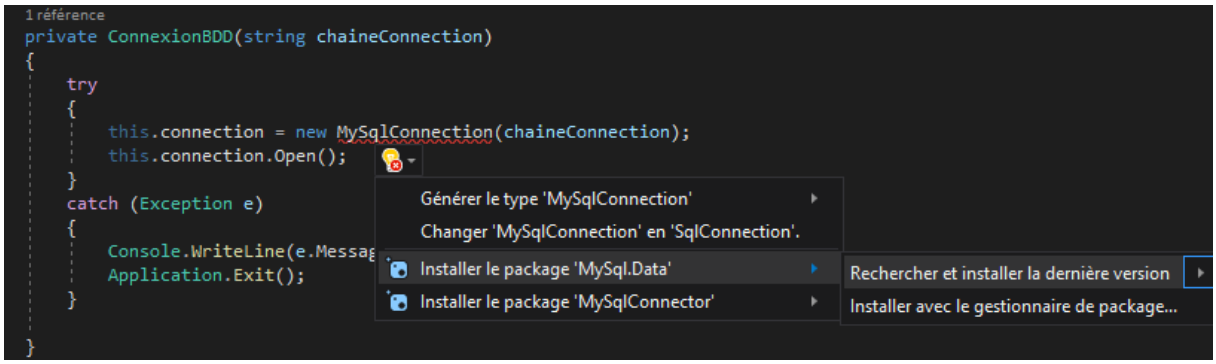




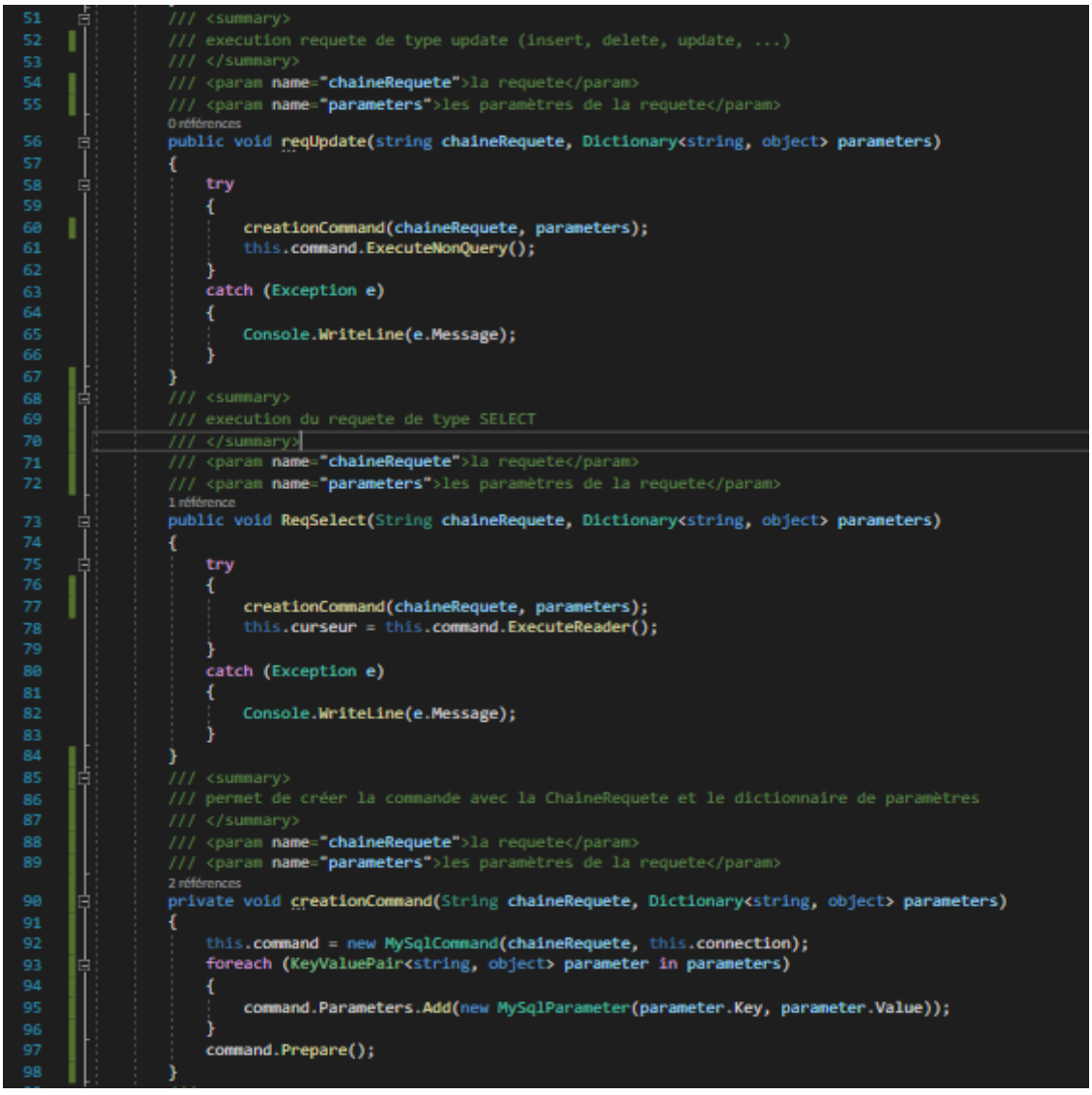
• Création du modèle et des outils de connexion :

Création outils de connexion :

Création du packages connexion. Puis ajout dans ce packages la class ConnexionBDD en Singleton avec son constructeur privé, qui permettra de gérer la connexion à la base de données. Mise à jour également de Visual Studio afin qu'il prenne en charge le packages MySql.data, indispensable pour la connexion à la base de données :



Création des méthodes permettant soit la mise à jour de la base de données (INSERT, UPDATE, DELETE, …), soit une requête d'interrogation (SELECT). Les deux méthodes sont assez identiques, à la différence que reqUpdate ne retourne rien alors ReqSelect envoi dans le curseur (de type MySqlDataReader) command (de type MySqlCommand), command qui contient la requête prête à l'emploi avec la fusion de ChaineRequete et des paramètres permettant ainsi d'éviter les injections SQL. La création de la commande et son exécution, sont placée dans un Try Catch par sécurité.



Création ensuite du package dal contenant la class AccesDonnees qui répondra aux demandes de récupération de données du contrôleur, et qui sera la seule à communiquer avec ConnexionBDD. Pour l'instant elle ne contient que la chaine de connexion, les méthodes seront créées au fil des besoins de l'application :

```
namespace AP2_MediaTek86.dal
{
    0 références
    public class AccesDonnees
    {
        /// <summary>
        /// La chaine de connexion
        /// </summary>
        private static string connectionString = "server=localhost;user id=responsable;password=motdepasse;persistsecurityinfo=True;database=mediatek86";
```

Création des class métier :

On enchaine avec la création des class métiers correspondant aux tables (Personnel, Absences, Motif et Service) de la base de données, à placer dans le package model. Les class métiers on toute la même structure que celle-ci. A noter que seul dans les class Service et Motif ont la méthode ToString a été redéfini. Cette redéfinition permettra une utilisation plus pratique dans les comboBox plus tard.

Les propriétés de chaque class ont également été Encapsulé avec un Get et un Set, cela permettra également une utilisation plus pratique plus tard surtout dans le DataGridView (l'élément qui recevra les listes de Personnels et d'absences)

```
namespace AP2_MediaTek86.model
{
    1 référence
    class Service
    {
        /// <summary>
        /// Propriétés de la table service
        /// </summary>
        private int idService;
        private string libelle;

        0 références
        public int IdService { get => idService; set => idService = value; }
        0 références
        public string Libelle { get => libelle; set => libelle = value; }
        /// <summary>
        /// constructeur de la la class service
        /// </summary>
        /// <param name="idService">l'id unique du service</param>
        /// <param name="libelle">son nom</param>
        0 références
        public Service(int idService, string libelle)
        {
            this.idService = idService;
            this.libelle = libelle;
        }
        /// <summary>
        /// redéfinition de la méthode ToString afin de faciliter son utilisation dans les ComboBox
        /// </summary>
        /// <returns></returns>
        1 référence
        public override string ToString()
        {
            return libelle;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AP2_MediaTek86.model
{
    0 références
    class Absences
    {
        private int idPersonnel;
        private DateTime dateDebut;
        private DateTime dateFin;
        private int idMotif;
        private string motif;
```

Générer le constructeur 'Absences(int, DateTime, DateTime, int, string)'
Générer Equals(object)
Générer Equals et GetHashCode
Encapsuler les champs (et utiliser la propriété)
Encapsuler les champs (mais utiliser toujours le champ)
Remove unused member
Extraire la classe de base...
Ajouter l'attribut 'DebuggerDisplay'
Supprimer ou configurer les problèmes

Lignes 15 à 16
private string motif;

```
public int IdPersonnel { get => idPersonnel; set => idPersonnel = value; }
public DateTime DateDebut { get => dateDebut; set => dateDebut = value; }
public DateTime DateFin { get => dateFin; set => dateFin = value; }
public int IdMotif { get => idMotif; set => idMotif = value; }
public string Motif { get => motif; set => motif = value; }
}
```

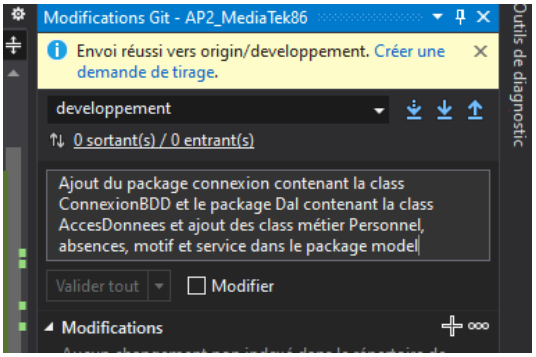
Aperçu des modifications

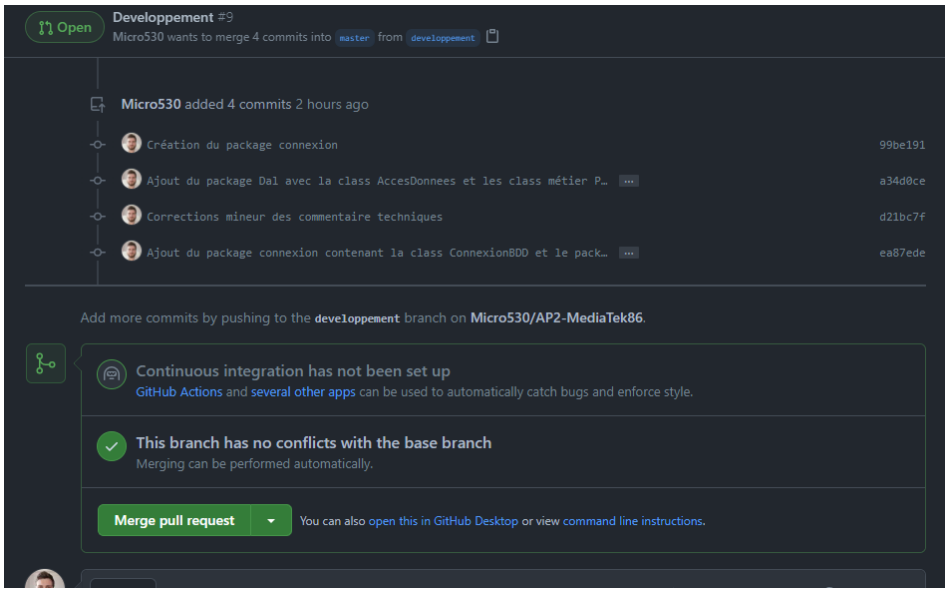
Après correction et ajustement des commentaire normalisé, création de la documentation technique, voici un extrait :

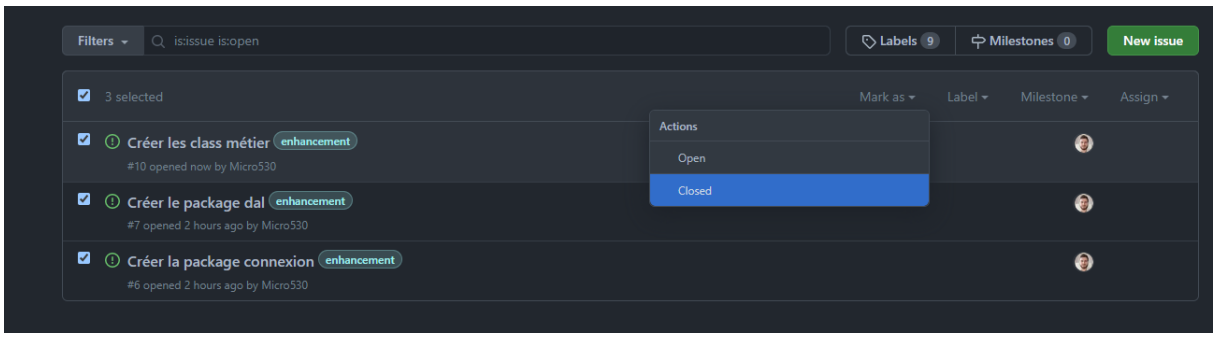
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<doc>
  <assembly>
    <name>AP2_MediaTek86</name>
  </assembly>
  <members>
    <member name="M:AP2_MediaTek86.connexion.ConnexionBDD.#ctor(System.String)">
      <summary> constructeur privé appelé uniquement par GetInstance </summary>
      <param name="chaineConnection">la chaine de connexion à la base de données</param>
    </member>
    <member name="M:AP2_MediaTek86.connexion.ConnexionBDD.getInstance(System.String)">
      <summary> création de l'instance si elle n'est pas deja existante </summary>
      <param name="chaineConnection">la chaine de connexion à la base de données</param>
      <returns>l'instance de la connexion</returns>
    </member>
    <member name="M:AP2_MediaTek86.connexion.ConnexionBDD.reqUpdate(System.String,System.Collections.Generic.Dictionary{System.String,System.Object})">
      <summary> execution requete de type update (insert, delete, update, ...) </summary>
      <param name="chaineRequete">la requete</param>
      <param name="parameters">les paramètres de la requete</param>
```

J’exécute un dernier commit et Push le dépôt local vers le distant. Je vérifie sur GitHub qu’il n’y a pas d’erreur ni de conflit et je valide le merge. J’en profite pour mettre à jour les issues.



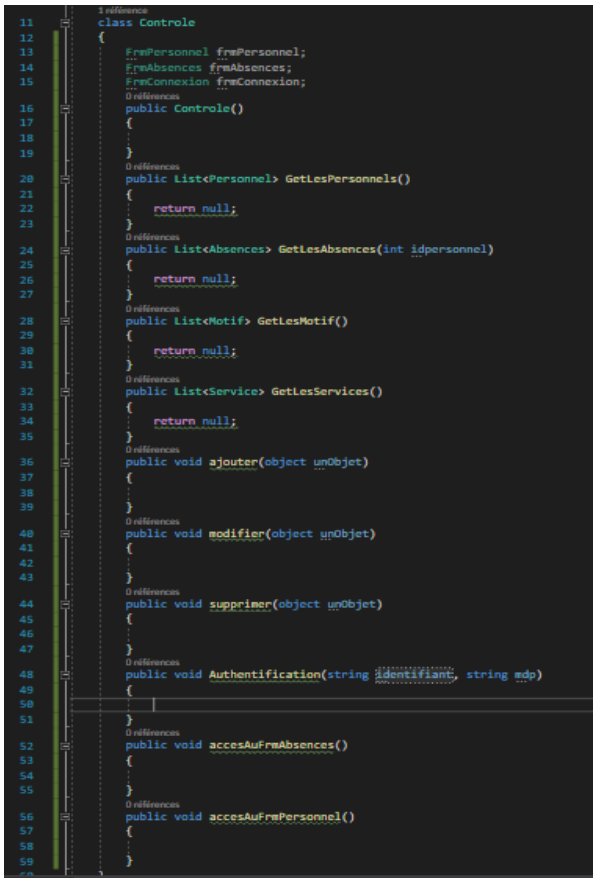




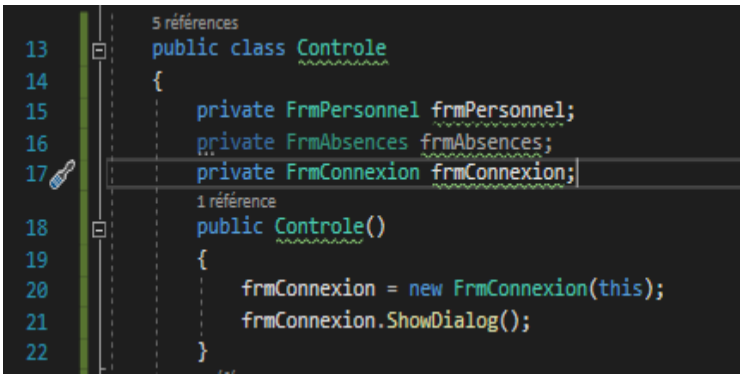
• Développement des fonctionnalités à partir des cas d'utilisation :

Formulaire d'authentification :

Tout d'abord on commence par créer la class Contrôle avec sa structure (image de gauche), les commentaires normalisés seront ajoutés au fil de l'utilisation des méthodes. Deuxième étape, pouvoir démarrer l'application. Après avoir demandé à la class Program de créer un Contrôle dès son exécution, le constructeur de contrôle instancie un nouveau FrmConnexion (image de droite) en envoyant son instance afin que ce formulaire puisse communiquer avec ce contrôle.

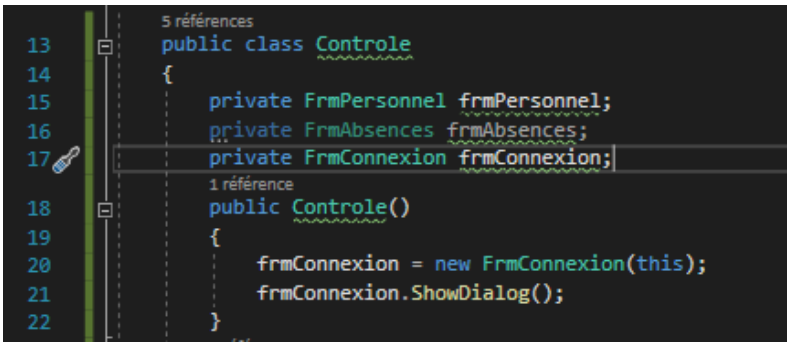


```
1 référence
class Controle
{
    FrmPersonnel frmPersonnel;
    FrmAbsences frmAbsences;
    FrmConnexion frmConnexion;
    0 références
    public Controle()
    {

    }
    0 références
    public List<Personnel> GetLesPersonnels()
    {
        return null;
    }
    0 références
    public List<Absences> GetLesAbsences(int idpersonnel)
    {
        return null;
    }
    0 références
    public List<Motif> GetLesMotif()
    {
        return null;
    }
    0 références
    public List<Service> GetLesServices()
    {
        return null;
    }
    0 références
    public void ajouter(object unObjet)
    {

    }
    0 références
    public void modifier(object unObjet)
    {

    }
    0 références
    public void supprimer(object unObjet)
    {

    }
    0 références
    public void Authentification(string identifiant, string mdp)
    {

    }
    0 références
    public void accesAuFrmAbsences()
    {

    }
    0 références
    public void accesAuFrmPersonnel()
    {

    }
}
```



```
5 références
public class Controle
{
    private FrmPersonnel frmPersonnel;
    private FrmAbsences frmAbsences;
    private FrmConnexion frmConnexion;
    1 référence
    public Controle()
    {
        frmConnexion = new FrmConnexion(this);
        frmConnexion.ShowDialog();
    }
```

Lors du clic sur le bouton de connexion de ce formulaire, le formulaire vérifie que tous les champs ont été saisie puis envoi leurs contenu au contrôle par la méthode Authentification (image 1). Cette méthode Authentification de contrôle va demander à AccesDonnees de lui fournir les informations d'authentification via la méthode Authentification de cette dernière (image 2). Puis AccesDonnees va demander à la class ConnexionBDD d'interroger la base de données et AccesDonnees va retourner les données (image 3). A partir de là contrôle fait la comparaison afin de savoir si les identifiants sont corrects ou non. Arriver à ce point l'application s'ouvre sur le formulaire d'authentification, après saisie correcte des identifiants, le formulaire d'authentification se ferme et celui du personnel s'ouvre.



```
5 références
public class Controle
{
    private FrmPersonnel frmPersonnel;
    private FrmAbsences frmAbsences;
    private FrmConnexion frmConnexion;
    1 référence
    public Controle()
    {
        frmConnexion = new FrmConnexion(this);
        frmConnexion.ShowDialog();
    }
```

```
50      }
        1 référence
51      public void Authentification(string identifiant, string mdp)
52      {
53          mdp = GetStringSha256Hash(mdp);
54          if ((identifiant + "|" + mdp).Equals(AccesDonnees.Authentification()))
55          {
56              frmPersonnel = new FrmPersonnel(this);
57              frmPersonnel.ShowDialog();
58              frmConnexion.Hide();
59          }
60          else
61          {
62              MessageBox.Show("Identifiant ou mot de passe incorrect", "Authentification impossible", MessageBoxButtons.OK, MessageBoxIcon.Warning);
63          }
64
65      }
```

```
            1 référence
18          public static string Authentification()
19          {
20              List<string> lesidentifiants = new List<string>();
21              ConnexionBDD bdd = ConnexionBDD.getInstance(connectionString);
22              Dictionary<string, object> parameters = new Dictionary<string, object>();
23              bdd.ReqSelect("SELECT * from responsable;", parameters);
24              if (bdd.Read())
25              {
26                  return bdd.Field("login").ToString()+ "|" +bdd.Field("pwd").ToString();
27              }
28              else
29              {
30                  return null;
31              }
32          }
33
```

Formulaire Personnel :

Récupération des personnels :

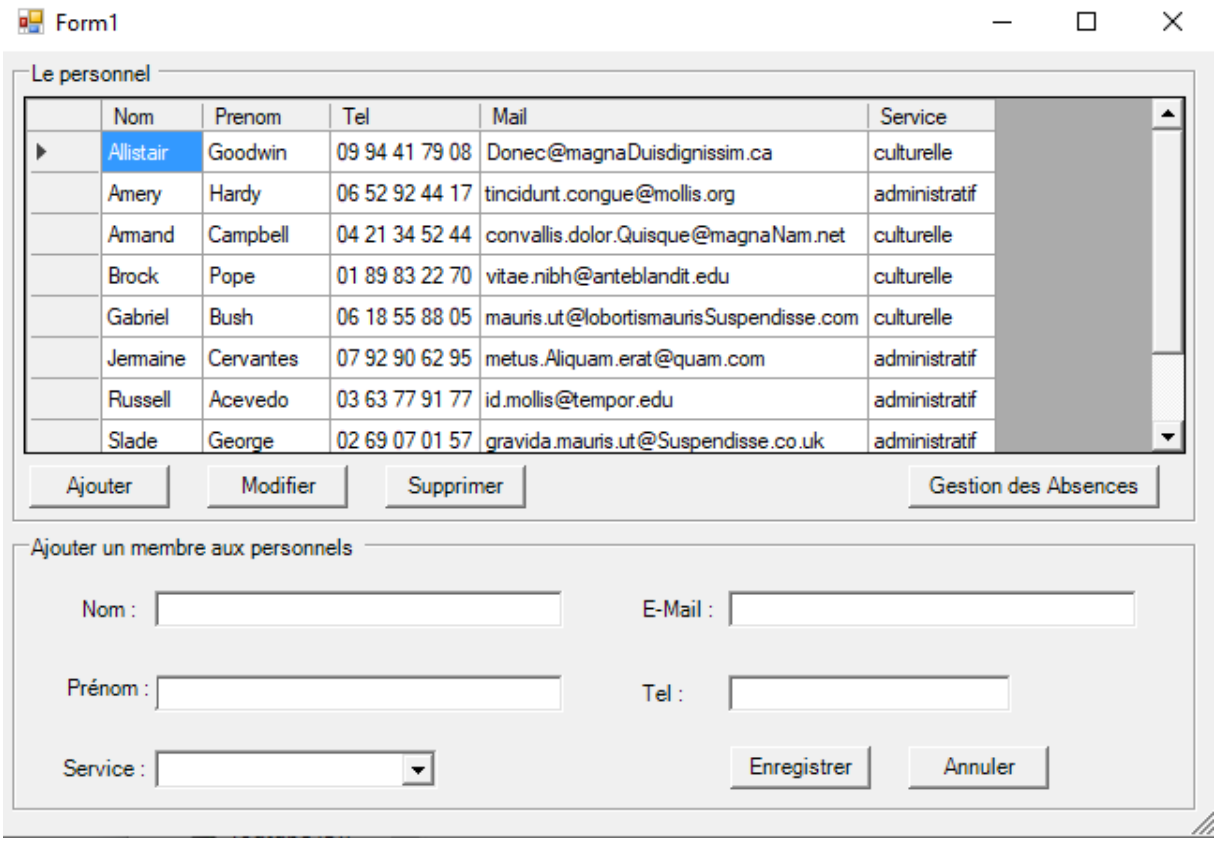
A présent il faut gérer le remplissage du FrmPersonnel après son ouverture. On commence par créer la méthode qui remplira le DataGridView. Cette méthode récupère depuis le contrôleur la liste du personnel qu'elle valorise dans bdsLesPersonnels (de type BindingSource). Ce BindingSource va pouvoir alimenter le DataGridView dgvPersonnel. Notez qu'il n'est pas utile de garder idpersonnel et idservice dans l'affichage, ils sont donc retirés (image 1). Le contrôleur lui ne va que demander puis transmettre les données. Il appel la méthode de même nom depuis la class AccesDonnees (image 2). Le fonctionnement de la méthode d'AccesDonnees est très similaire à celle de l'authentification, à la seule différence qu'ici on remplit une liste est qu'il faut boucler sur le curseur (soit sur chaque ligne que la requête a retournée).

```
23          public FrmPersonnel(Controle controle)
24          {
25              this.controle = controle;
26              InitializeComponent();
27              ResetFormulaire();
28          }
29          /// <summary>
30          /// Permet de mettre ou remettre le formulaire à l'été d'origine
31          /// </summary>
            1 référence
32          private void ResetFormulaire()
33          {
34              RemplirPersonnels();
35          }
36          /// <summary>
37          /// permet de remplire le DataGridView avec le personnel
38          /// </summary>
            1 référence
39          private void RemplirPersonnels()
40          {
41              bdsLesPersonnels.DataSource = controle.GetLesPersonnels();
42              dgvPersonnel.DataSource = bdsLesPersonnels;
43              dgvPersonnel.Columns["idpersonnel"].Visible = false;
44              dgvPersonnel.Columns["idservice"].Visible = false;
45              dgvPersonnel.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
46          }
```

```
26      public List<Personnel> GetLesPersonnels()
27      {
28          return AccesDonnees.GetLesPersonnels();
29      }
```

```
40      /// <summary>
41      /// Gere la récupération de la liste du personnel
42      /// </summary>
43      /// <returns>le liste du personnel</returns>
        1 référence
44      public static List<Personnel> GetLesPersonnels()
45      {
46          List<Personnel> lesPersonnels = new List<Personnel>();
47          ConnexionBDD bdd = ConnexionBDD.getInstance(connectionString);
48          Dictionary<string, object> parameters = new Dictionary<string, object>();
49          bdd.ReqSelect("SELECT idpersonnel, personnel.nom as nompersonnel, prenom, tel, mail, personnel.idservice, service.nom as nomservice" +
50              " from personnel inner join service on personnel.idservice = service.idservice order by nompersonnel, prenom;", parameters);
51          while (bdd.Read())
52          {
53              lesPersonnels.Add(new Personnel((int)bdd.Field("idpersonnel"), (string)bdd.Field("nompersonnel"), (string)bdd.Field("prenom"),
54                  (string)bdd.Field("tel"), (string)bdd.Field("mail"), (int)bdd.Field("idservice"), (string)bdd.Field("nomservice")));
55      }
56          bdd.close();
57          return lesPersonnels;
58      }
```

Ce qui nous donne comme résultat :



Puis on ajoute les différents services dans le Combo de la partie « Ajouter » qui doivent être présent également lors du chargement du formulaire. A noter que cette partie sera caché tant que l'utilisateur n'aura pas cliqué sur le bouton ajouter ou modifier. Le code dans le contrôleur et le code d'AccesDonnees sont étroitement similaire à la récupération du personnel :

```
49      /// <summary>
50      /// permet de remplir le combo des service
51      /// </summary>
        1 référence
52      private void RemplirServices()
53      {
54          bdsLesServices.DataSource = controle.GetLesServices();
55          comboService.DataSource = bdsLesServices;
56      }
        1 référence
```

Evènement Ajouter/Modifier :

On peut ajouter à présent le code derrière l'événement clic sur le bouton ajouter, ce code se contente de rendre visible la zone « Ajouter ». Le code derrière le clic sur le bouton Annuler et celui sur le clic du bouton enregistrer. Etant donnée que le bouton enregistrer est le même que pour la modification, un switch case sur le texte de la zone d'affichage, soit d'ajout, soit de modification est effectué. Les codes étant étroitement similaire, ils ont été effectués en même temps (image 1). Dans le contrôleur, par soucis d'optimisation, il existe une seule méthode Ajouter, que ce soit pour ajouter du personnels ou une absence donc il faut vérifier qu'elle type d'objet est passé en paramètre. Ici par praticité, j'ai incorporé également la gestion de l'ajout d'une absence (image 2). Enfin nous avons dans AccesDonnees la méthode AjoutPersonnel. A noter que l'on utilise pour la première fois le Dictionary permettant de sécuriser les requête SQL en empêchant les injections SQL par les paramètres et que cette fois ci on utilise reqUpdate et plus reqSelect de la class ConnexionBDD (image 3).

```
1 référence
private void btsAjouter_Click(object sender, EventArgs e)
{
    ActiveZoneAjouter();
    zonePers.Enabled = false;
}

1 référence
private void btnZ2Annuler_Click(object sender, EventArgs e)
{
    DesactiveZoneAjouter();
}

1 référence
private void btnZ2Enregistrer_Click(object sender, EventArgs e)
{
    if (!(txtNom.Text.Equals("") || txtPrenom.Text.Equals("") || txtMail.Text.Equals("") || txtTel.Text.Equals("")) || comboService.SelectedIndex.Equals(-1))
    {
        Service unService = (Service)bdsLesServices.List[bdsLesServices.Position];
        switch (zoneAjout.Text)
        {
            case "Ajouter un membre aux personnels":
                controle.Ajouter(new Personnel(bdsLesPersonnels.Count + 1, txtNom.Text, txtPrenom.Text, txtTel.Text, txtMail.Text, unService.IdService, unService.Libelle));
                break;
            case "Modifier un membre du personnel":
                int idPersonnel = (int)dgvPersonnel.SelectedRows[0].Cells["idPersonnel"].Value;
                controle.Modifier(new Personnel(idPersonnel, txtNom.Text, txtPrenom.Text, txtTel.Text, txtMail.Text, unService.IdService, unService.Libelle));
                zonePers.Enabled = true;
                break;
        }
        ResetFormulaire();
    }
    else
    {
        MessageBox.Show("Vous devez entrer des valeurs dans tous les champs de saisie avant de pouvoir enregistrer une nouvelle entrée", "Champ(s) vide(s)", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

```
1 référence
public void Ajouter(object unObjet)
{
    if(unObjet is Personnel)
    {
        AccesDonnees.AjoutPersonnel((Personnel)unObjet);
    }
    else
    {
        AccesDonnees.AjoutAbsences((Absences)unObjet);
    }
}
```

```
/// <summary>
/// Ajoute un personnel
/// </summary>
/// <param name="unePersonne"> la personne à ajouter</param>
1 référence
public static void AjoutPersonnel(Personnel unePersonne)
{
    ConnexionBDD bdd = ConnexionBDD.getInstance(connectionString);
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@nom", unePersonne.Nom);
    parameters.Add("@prenom", unePersonne.Prenom);
    parameters.Add("@tel", unePersonne.Tel);
    parameters.Add("@mail", unePersonne.Mail);
    parameters.Add("@idservice",unePersonne.IdService);
    bdd.reqUpdate("insert into personnel (nom,prenom,tel,mail,idservice) VALUES (@nom, @prenom, @tel, @mail, @idservice);", parameters);
}
```

Au tour à présent de l'événement modifier. Au sein du FrmPersonnel, il ne reste plus qu'à gérer le remplissage des champs avec le personnel sélectionné (image 1). Après quoi il faut ajouter la méthode correspondant dans AccesDonnees (image 2). Cette méthode est très similaire encore une fois aux précédentes.

```
1 référence
private void btnModifier_Click(object sender, EventArgs e)
{
    if(selectionDev("Vous devez selectionner un membre du personel avant de pouvoir le modifier", "Aucun personnel selectionné"))
    {
        ActiveZoneAjouter("Modifier un membre du personnel");
        txtNom.Text = (string)dgvPersonnel.CurrentRow.Cells["nom"].Value;
        txtPrenom.Text = (string)dgvPersonnel.CurrentRow.Cells["prenom"].Value;
        txtMail.Text = (string)dgvPersonnel.CurrentRow.Cells["mail"].Value;
        txtTel.Text = (string)dgvPersonnel.CurrentRow.Cells["tel"].Value;
        comboService.SelectedIndex = comboService.FindStringExact((string)dgvPersonnel.CurrentRow.Cells["service"].Value);
    }
}
/// <summary>
/// permet de vérfier si une ligne st selectionné dans le DataGridView
/// </summary>
/// <param name="msg1"></param>
/// <param name="msg2"></param>
/// <returns></returns>
1 référence
private bool selectionDev(String msg1, String msg2)
{
    if (dgvPersonnel.CurrentRow.Index.Equals(-1))
    {
        MessageBox.Show(msg1, msg2, MessageBoxButtons.OK, MessageBoxIcon.Information);
        return false;
    }
    else
    {
        return true;
    }
}
```

```
1 référence
public static void ModifierPersonnel(Personnel unePersonne)
{
    ConnexionBDD bdd = ConnexionBDD.getInstance(connectionString);
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", unePersonne.IdPersonnel);
    parameters.Add("@nom", unePersonne.Nom);
    parameters.Add("@prenom", unePersonne.Prenom);
    parameters.Add("@tel", unePersonne.Tel);
    parameters.Add("@mail", unePersonne.Mail);
    parameters.Add("@idservice", unePersonne.IdService);
    bdd.reqUpdate("update personnel set nom = @nom, prenom = @prenom, tel = @tel, mail = @mail, idservice = @idservice where idpersonnel = @idpersonnel;", parameters);
    bdd.close();
}
```

Evénement supprimer :

Il suffit de créer l'événement dans FrmPersonnel, vérifier qu'il y a bien une ligne de sélectionnée puis demander une confirmation de suppression (image 1). Puis dans AccesDonnees, la commande SQL est très simple (image 2).

```
private void btnSup_Click(object sender, EventArgs e)
{
    if(selectionDev("Vous devez selectionner un membre du personel avant de pouvoir le supprimer", "Aucun personnel selectionné"))
    {
        Personnel unPersonnel = (Personnel)bdsLesPersonnels.List[bdsLesPersonnels.Position];
        if (MessageBox.Show("Voulez-vous vraiment supprimer " + unPersonnel.Nom + " " + unPersonnel.Prenom + " ?", "Confirmation de suppression",
            MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes )
        {
            controle.Supprimer(unPersonnel);
            ResetFormulaire();
        }
    }
}
```

```
1 référence
public static void SupprimerPersonnel(Personnel unePersonne)
{
    ConnexionBDD bdd = ConnexionBDD.getInstance(connectionString);
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", unePersonne.IdPersonnel);
    bdd.reqUpdate("delete from personnel where idpersonnel = @idpersonnel;", parameters);
}
```

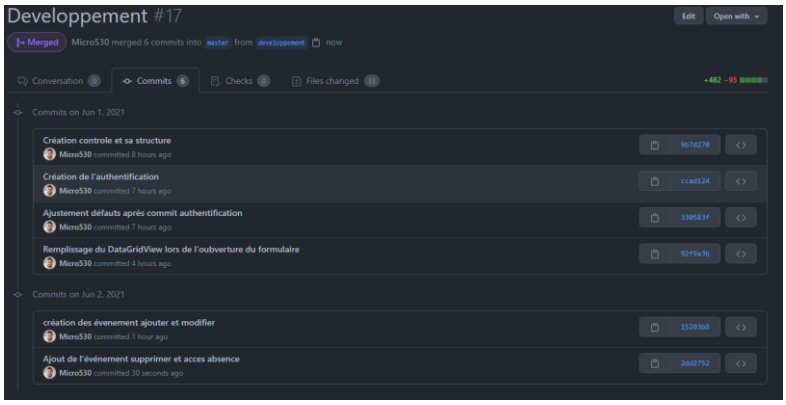
Evénement Gestion des absences :

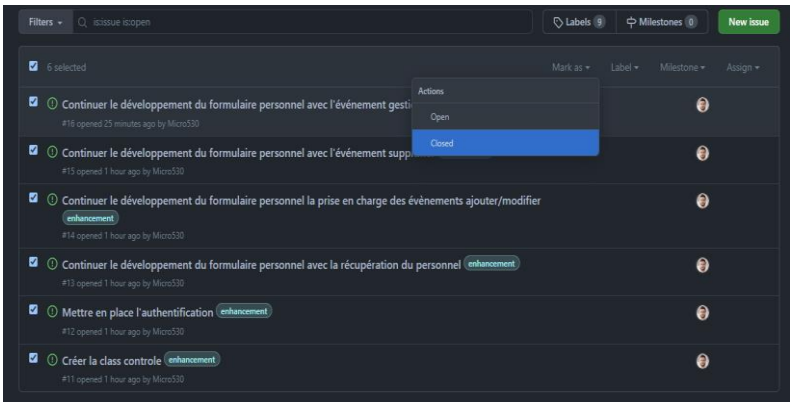
On rajoute simplement l'événement clic sur le bouton, si l'utilisateur à bien sélectionné un personnel, la demande est faite au contrôleur (image 1). Contrôleur qui créer une nouvelle instance de FrmAbsences avec le personnel fournit en paramètre (image 2).

```
1 référence
private void btnAbsence_Click(object sender, EventArgs e)
{
    if (selectionDev("Vous devez selectionner un membre du personel avant de pouvoir gerer ses absences", "Aucun personnel selectionné"))
    {
        controle.AccesAuFrmAbsences((Personnel)bdsLesPersonnels.List[bdsLesPersonnels.Position]);
    }
}
```

```
1 référence
public void AccesAuFrmAbsences(Personnel unPersonnel)
{
    frmPersonnel.Visible = false;
    frmAbsences = new FrmAbsences(this, unPersonnel);
    frmAbsences.ShowDialog();

}
```

A partir d'ici, le FrmPersonnel est terminé, je Push tous les commit vers le dépôt distant et j'en profite pour cocher les issues.





Formulaire Absences :

Remplissage des absences du personnel sélectionné :

Le remplissage des absences dans le DateGridView des absences est identique à celui des personnels, la seule nuance et qu'il faut envoyer jusqu'à AccesDonnees le personnel sélectionné afin de créer la requête SQL.

```
1 référence
public static List<Absences> GetLesAbsences(Personnel unPersonnel)
{
    List<Absences> lesAbsences = new List<Absences>();
    ConnexionBDD bdd = ConnexionBDD.getInstance(connectionString);
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", unPersonnel.IdPersonnel);
    bdd.ReqSelect("SELECT * from absence inner join motif on absence.idmotif = motif.idmotif where idpersonnel = @idpersonnel order by datedebut desc;", parameters);
    while (bdd.Read())
    {
        lesAbsences.Add(new Absences((int)bdd.Field("idpersonnel"), (DateTime)bdd.Field("datedebut"), (DateTime)bdd.Field("datefin"),
            (int)bdd.Field("idmotif"), (string)bdd.Field("libelle")));
    }
    bdd.close();
    return lesAbsences;
}
```

Evénement ajouter/modifier :

La plupart des éléments à mettre dans FrmAbsences sont identique au FrmPersonnel. La partie une il y a le plus de modification est sur l'évènement enregistré. La principe différence ici est dans la gestion des modifications. Contrairement aux personnels, les absences n'ont pas identification propre, pour connaitre une absence, il faut l'id d'un personnel et une date de début. Seulement si on modifie la date de début on ne peut plus mettre à jour cette absence puisque ce sera comme une autre absence. Seule solution, supprimer l'absence modifier et la recréer. C'est précisément ce que la méthode modificationDateCle fait. A noter qu'il y a un contrôle des dates avec compare afin de vérifier que la date de fin n'est pas antérieure à celle de début (image 1). En ce qui concerne le contrôleur, tout a déjà été créé. Sur l'AccesDonnees, la syntaxe est identique, il faut juste changer les champs. Il y a eu également le repérage d'une anomalie dans le contrôle de sélection d'une ligne du DataGridView, si celui-ci était vide (aucunes absences par exemple) ou erreur se produisait. Elle a été corrigée aussi bien sur ce formulaire que sur celui du personnel (image 2).

```
/// <summary>
/// évènement enregistrer un ajout ou une modification
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void btnZ2Enregistrer_Click(object sender, EventArgs e)
{
    if (!(comboMotif.SelectedIndex.Equals(-1) || DateTime.Compare(calendrierDebut.Value, calendrierFin.Value).Equals(1)) )
    {
        Motif unMotif = (Motif)bdsLesMotifs.List[bdsLesMotifs.Position];
        switch (zoneAjout.Text)
        {
            case "Ajouter une absence":
                controle.Ajouter(new Absences(unPersonnel.IdPersonnel, calendrierDebut.Value, calendrierFin.Value, unMotif.IdMotif, unMotif.Libelle));
                break;
            case "Modifier une absence":
                if (calendrierDebut.Value.Equals((DateTime)dgvAbsences.CurrentRow.Cells["datedebut"].Value))
                {
                    DateTime datedebut = (DateTime)dgvAbsences.CurrentRow.Cells["datedebut"].Value;
                    DateTime datedefin = (DateTime)dgvAbsences.CurrentRow.Cells["datefin"].Value;
                    if (MessageBox.Show("Voulez-vous vraiment modifier l'absence du" + datedebut + " au " + datedefin + " ?", "Confirmation de modifiation",
                        MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
                    {
                        controle.Modifier(new Absences(unPersonnel.IdPersonnel, datedebut, calendrierFin.Value, unMotif.IdMotif, unMotif.Libelle));
                    }
                }
                else
                {
                    modificationDateCle();
                }
                break;
                ResetFormulaire();
        }
    }
    else
    {
        MessageBox.Show("Vous n'avez pas sélectionné de motif ou bien les dates ne sont pas cohérente", "Champ(s) vide(s)", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
/// <summary>
/// evenement si la modification change la date de début
/// </summary>
1 référence
private void modificationDateCle()
{
    controle.Supprimer((Absences)bdsLesAbsences.List[bdsLesAbsences.Position]);
    Motif unMotif = (Motif)bdsLesMotifs.List[bdsLesMotifs.Position];
    controle.Ajouter(new Absences(unPersonnel.IdPersonnel, calendrierDebut.Value, calendrierFin.Value, unMotif.IdMotif, unMotif.Libelle));
}
```

```
2 références
private bool selectionDgv(String msg1, String msg2)
{
    if (dgvAbsences.CurrentRow is null || dgvAbsences.CurrentRow.Index.Equals(-1))
    {
        MessageBox.Show(msg1, msg2, MessageBoxButtons.OK, MessageBoxIcon.Information);
        return false;
    }
    else
    {
        return true;
    }
}
```

Évènement Supprimer :

Encore un fois il n'y a que très peu de différences sur la gestion au sein du formulaire. Là où la différence est la plus intéressant, c'est au niveau de AccesDonnees. On remarque que pour correctement identifier une absence, il faut prendre l'idpersonnel et la datedebut dans la requête SQL, conformément aux deux clés primaires de cette table.

```
public static void SupprimerAbsence(Absences uneAbsence)
{
    ConnexionBDD bdd = ConnexionBDD.getInstance(connectionString);
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", uneAbsence.IdPersonnel);
    parameters.Add("@datedebut", uneAbsence.DateDebut);
    bdd.reqUpdate("delete from absence where idpersonnel = @idpersonnel and datedebut = @datedebut;", parameters);
}
```

Le reste des fonctionnalités du formulaire sont identique avec celle du formulaire personnel.

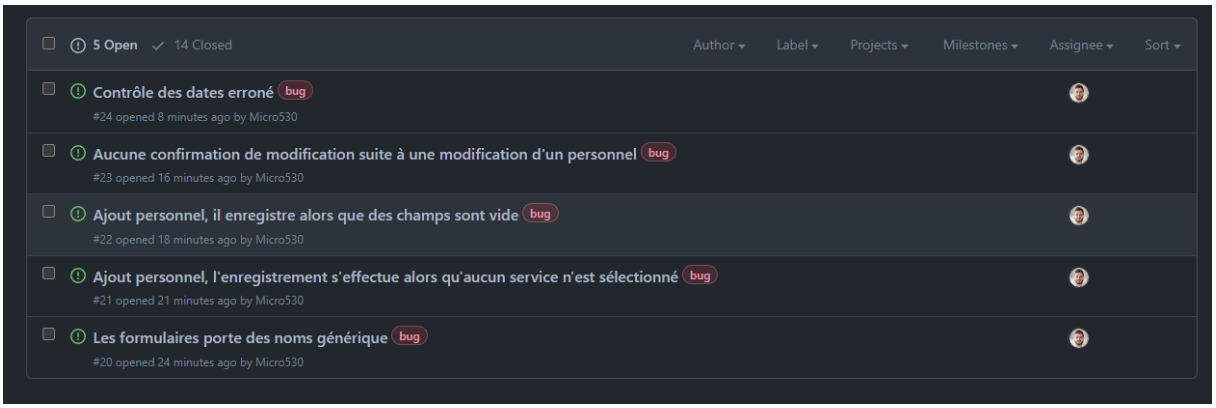
A présent il nous faut pouvoir retourner sur le formulaire personnel, on appel la méthode contrôle correspondant qui s'en charge.

```
public void AccesAuFrmPersonnel()
{
    frmAbsences.Dispose();
    frmPersonnel.Visible = true;
}
```

L'application est maintenant fonctionnelle et va pouvoir passer une batterie de test afin de vérifier les éventuelles bug et disfonctionnements. Mais avant ça, nos modifications vont être Push vers le dépôt distant.

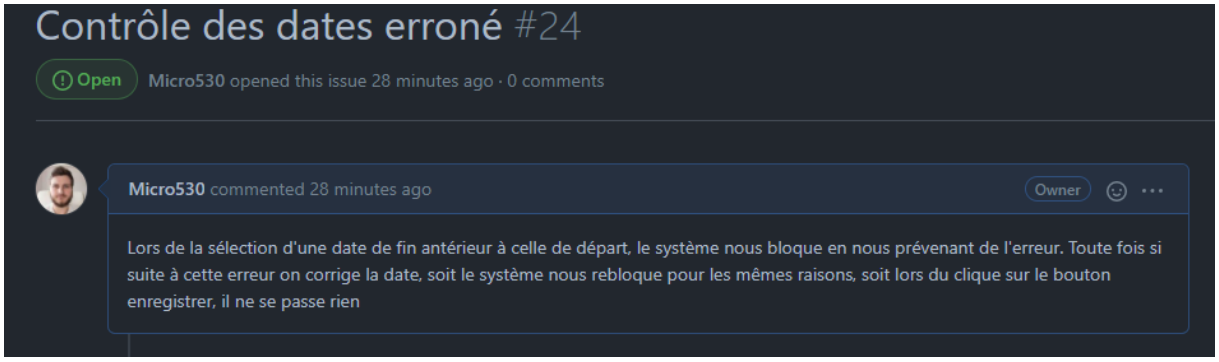
• Bug et disfonctionnements :

Une série de disfonctionnements ont été relevé



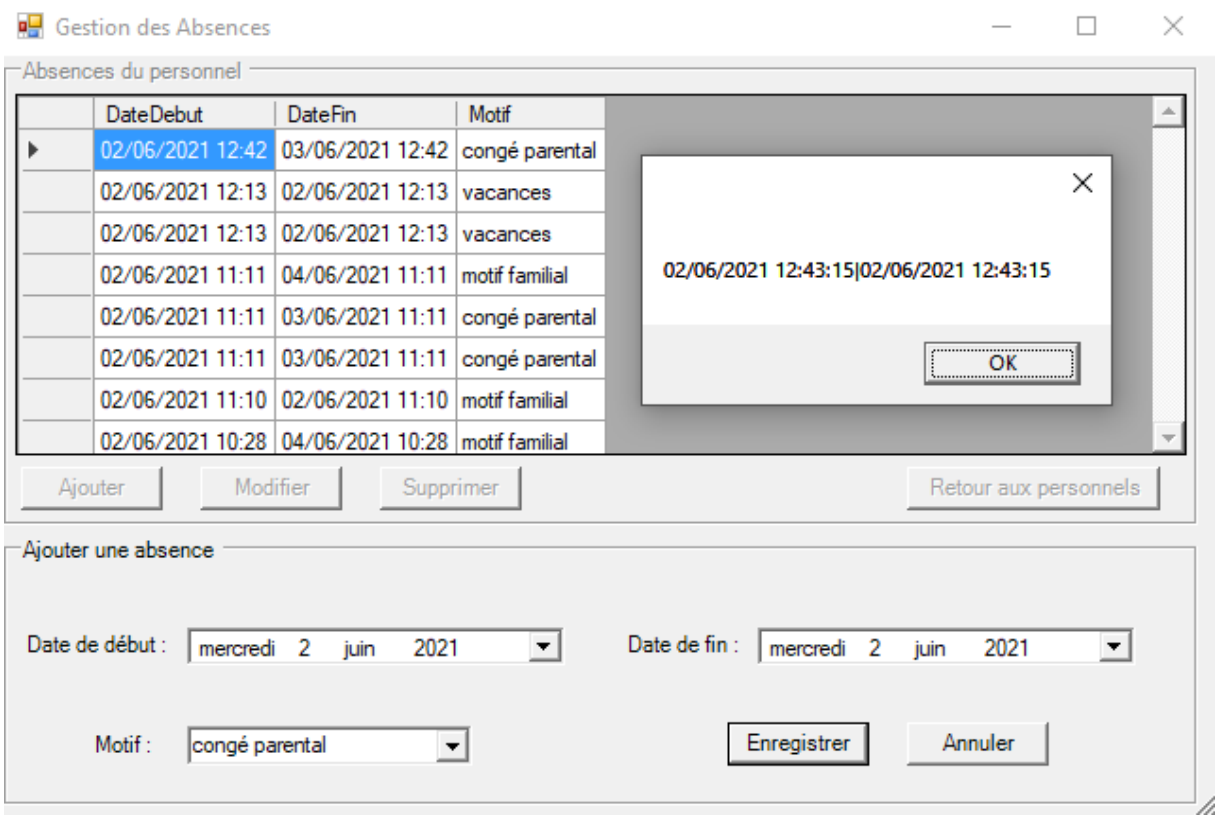
Certain sont juste des oublies comme la demande de confirmation avant la modification, présente sur le formulaire absence mais pas sur le formulaire personnel ou encore les noms des formulaires qui sont des noms génériques. Ils sont rapidement corrigés. Le fait que j'atout se fait alors qu'il manque des champs remplie, est du au fait d'un faute de frappe dans le if qui se charge de vérifier.

Le bug le plus intéressant est celui des dates, voici le détail :



Le fait qu'il ne se passe plus rien suite à l'enregistrement est dû à une erreur d'accolade rapidement corrigé. Toutefois la succession d'évènements où il faut d'abord faire une erreur sur l'ordre des dates puis corriger cette erreur mais en mettant la même date de fin que celle du départ est assez intéressant. Car si vous mettez la date de fin un jour plus tard, le tout fonctionne.

Pour comprendre ce qu'il se passe au niveau des dates, j'utilise un MessageBox comme débug en temps réel pour connaitre le contenu exact des dates. Ce message nous donne la dateDebut à gauche et dateFin à droite, on s'aperçois qu'elles sont strictement identique. Pourtant lors de leurs comparaison le système retourne 1, ce qui signifie que pour le système la deteFin est antérieure à la dateDebut. Honnêtement je n'ai pas d'explication, ma solution va être en cas d'erreur de saisie, après l'avertissement, de remplir dateFin avec la valeur dateDebut. L'erreur est donc corrigée.



L'application est maintenant opérationnelle, après vérification et ajout des commentaires normalisés, je génère la documentation technique et Push la version finale de l'application vers le dépôt distant.

• Bilan :

La solution applicative est terminée et pleinement opérationnelle, chaque cas d'utilisation a été respecté à la lettre. Le fichier contenant la base de données à été exporté et mis à disposition. Vous retrouverez également tous les documents utilisés à disposition (MCD, maquette, code, …)