

Compte-rendu d'activité projet

Simon Lesko

Table des matières

1	Introduction	1
2	Mission	1
3	Etapes de réalisation	2
3.1	Configuration de l'environnement de développement	2
3.2	Maquettage et codage des vues dans le respect de l'architecture MVC	4
3.3	Développement de la partie Modèle et génération d'une documentation technique	5
3.4	Développement et test des fonctionnalités de l'application	6
3.4.1	Connexion d'un responsable	7
3.4.2	Affichage de la liste du personnel	10
3.4.3	Ajout/modification d'un personnel	13
3.4.4	Affichage de la liste des absences d'un personnel	15
3.4.5	Ajout et modification des absences	16
4	Bilan	18

1 Introduction

InfoTech Service 86 est une entreprise de services numériques créée en 2002 et spécialisée dans le domaine du développement de solutions logicielles et des systèmes et réseaux. Elle a remporté un appel d'offre concernant la gestion de plusieurs médiathèques.

2 Mission

En tant que technicien développeur junior pour le compte de la société Infotech 86, je dois réaliser un projet dans le cadre de la mise en place d'un réseau de gestion des médiathèques de la Vienne.

Mediatek86 est une application bureau créée en C# avec Visual Studio permettant la gestion des informations et des absences du personnel d'une

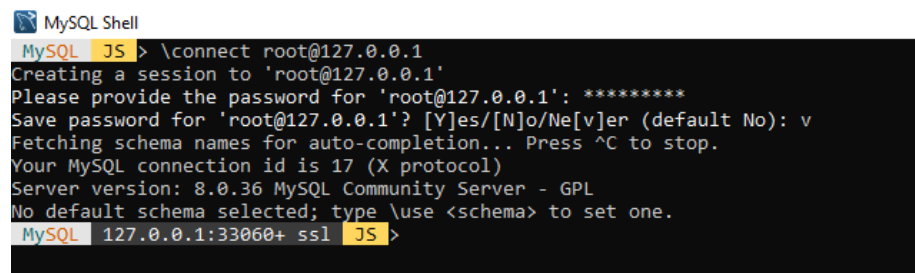
médiathèque par le responsable de la structure sur un poste informatique disposant du système d'exploitation windows.

3 Etapes de réalisation

3.1 Configuration de l'environnement de développement

La première étape de réalisation du projet a été la mise en place de l'environnement de développement de l'application. En effet, la réalisation d'un tel programme nécessite l'installation et la configuration des logiciels présentés ci-dessous :

MySQL : Système de gestion de base de données permettant la sauvegarde des informations du personnel et des absences dans une base de données organisée de façon cohérente. Après installation, un service MySQL est créé en arrière plan, c'est avec lui que l'application doit interagir pour manipuler ou insérer des données. Pour créer la base de données, le shell mysql (command line interface) a été utilisé pour transmettre les instructions au SGBD.



```
MySQL Shell
MySQL JS > \connect root@127.0.0.1
Creating a session to 'root@127.0.0.1'
Please provide the password for 'root@127.0.0.1': *****
Save password for 'root@127.0.0.1'? [Y]es/[N]o/[e]x (default No): y
Fetching schema names for auto-completion... Press ^C to stop.
Your MySQL connection id is 17 (X protocol)
Server version: 8.0.36 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
MySQL 127.0.0.1:33060+ ssl JS >
```

FIG. 1 – 1ère connexion au SGBD en tant que root.

Après connexion en tant que root (superutilisateur) il a fallu créer la base de données 'mediatek86' grâce au code SQL récupéré avec le modèle conceptuel fourni. Une fois créées, les tables de la base de données ont été remplies avec des données pseudo-aléatoires générées sur le site internet <https://www.generatedata.com/>

```

MySQL Shell
MySQL 127.0.0.1:33060+ ssl mediatek86 JS > \use mediatek86
Default schema 'mediatek86' accessible through db.
MySQL 127.0.0.1:33060+ ssl mediatek86 JS > \sql SHOW TABLES;
Fetching global names, object names from 'mediatek86' for auto-completion... Press ^C to stop.
+-----+
| Tables_in_mEDIATEK86 |
+-----+
| absence              |
| motif                |
| personnel             |
| responsable          |
| service               |
+-----+
5 rows in set (0.0007 sec)
MySQL 127.0.0.1:33060+ ssl mediatek86 JS > \sql DESCRIBE personnel;
+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+
| idpersonnel | int       | NO   | PRI | NULL    | auto_increment |
| nom         | varchar(50) | YES  |     | NULL    |              |
| prenom      | varchar(50) | YES  |     | NULL    |              |
| tel         | varchar(15) | YES  |     | NULL    |              |
| mail        | varchar(128) | YES  |     | NULL    |              |
| idservice   | int       | NO   | MUL | NULL    |              |
+-----+
6 rows in set (0.0012 sec)
MySQL 127.0.0.1:33060+ ssl mediatek86 JS >

```

FIG. 2 – Vue des tables créées et description du schéma de la table 'personnel'.

```

MySQL Shell
MySQL 127.0.0.1:33060+ ssl mediatek86 JS > \sql SELECT * FROM personnel;
+-----+
| idpersonnel | nom      | prenom | tel      | mail                                     | idservice |
+-----+
| 1           | Kylan   | Collins | 06 75 03 05 64 | donec@aol.couk                         | 2         |
| 2           | Kenyon  | Flores  | 04 55 89 10 79 | facilisis.suspendisse@aol.com          | 1         |
| 3           | Sage    | Padilla | 07 74 41 45 65 | dui@outlook.edu                        | 1         |
| 4           | Gabriel | Gould   | 05 08 25 16 11 | sit.amet@protonmail.ca                 | 1         |
| 5           | Lacota  | Griffin | 05 61 49 57 45 | eros@icloud.com                        | 3         |
| 6           | Madison | Stark   | 02 85 51 88 18 | pede.praesent@hotmail.net              | 1         |
| 7           | Lawrence | Pollard | 05 33 67 57 56 | vehicula.aliquet@protonmail.couk       | 1         |
| 8           | Zane    | Garrison | 02 69 07 22 83 | rutrum@hotmail.ca                      | 2         |
| 9           | Brynn   | Ellis   | 01 48 72 34 90 | elit@outlook.edu                       | 1         |
| 10          | Jarrod  | Fields  | 04 74 32 02 91 | ornare.libero@aol.ca                   | 1         |
+-----+
10 rows in set (0.0009 sec)
MySQL 127.0.0.1:33060+ ssl mediatek86 JS >

```

FIG. 3 – Vue des données présentent dans la table 'personnel'.

Une fois la base de données 'mediatek86' créée et peuplée, nous avons créé un utilisateur 'dbadmin' avec le mot de passe 'P@\$s\$word2' ayant accès à cette base de données.

Visual Studio : Atelier de Génie Logiciel (AGL) permettant le développement de programmes informatiques dans de nombreux langages dont le C#. Ce logiciel dispose notamment d'un concepteur de vues graphiques facilitant la création des interfaces utilisateur.

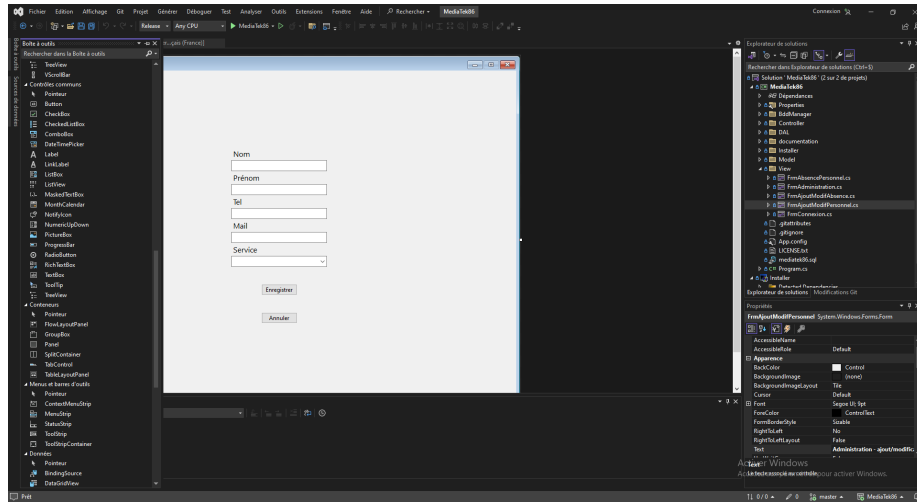


FIG. 4 – Concepteur d'interface graphique Visual Studio

.NET Framework : Boîte à outils logicielle de Microsoft permettant la création d'applications dans de nombreux langages de programmation.

Windows Forms : Boîte à outils logicielle de Microsoft permettant la création d'interfaces graphiques dans une application.

3.2 Maquettage et codage des vues dans le respect de l'architecture MVC

Avant toute chose, il a fallu créer un prototype des fenêtres (vues) de l'application dans le respect du cahier des charges présenté dans le diagramme des cas et des cas utilisateurs. Pour cela, nous avons utilisé le logiciel Pencil.

Une fois le prototype des interfaces graphiques terminé, nous avons conçu les vues à l'aide du concepteur de vues de Visual Studio, en se basant sur les maquettes créées précédemment.

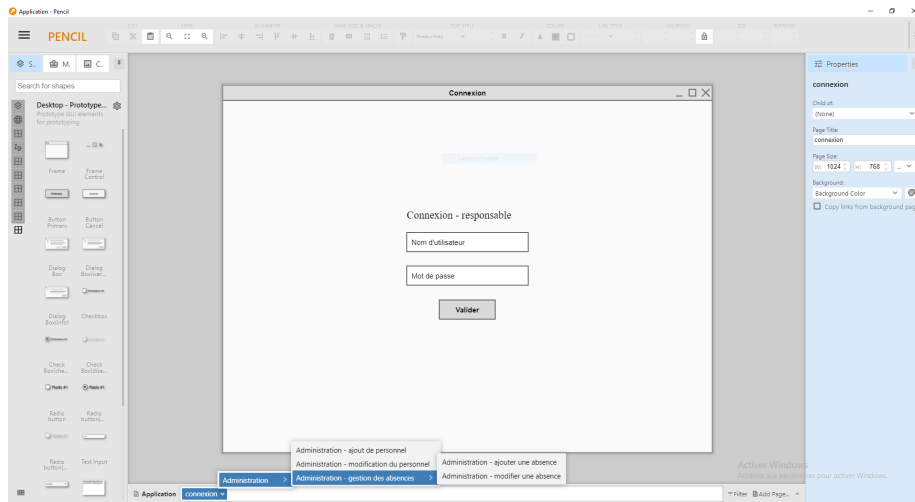


FIG. 5 – *Aperçu de la maquette de l'application dans Pencil*

3.3 Développement de la partie Modèle et génération d'une documentation technique

En partant du schéma de la base de données, nous avons créé la partie métier de l'application. En effet, nous avons identifié les entités nécessaires au fonctionnement de l'application : Le personnel, les absences, les services ainsi que les motifs. Chacune de ces entités sont représentées au niveau de l'application par une classe objet, certaines de ces classes présentent des liens équivalent aux clefs étrangères de la base de données. Voici un extrait de l'objet 'Personnel' :

```

1 namespace MediaTek86.Model
2 {
3     /// <summary>
4     /// Classe metier interne pour memoriser les informations d'un
5     /// </summary>
6     public class Personnel
7     {
8         private int idPersonnel;
9         private String nom;
10        private String prenom;
11        private String tel;
12        private String mail;
13        private Service service;
14        private List<Absence> absences;
15        /// <summary>
16        /// Valorise les proprietes
17        /// </summary>
18        /// <param name="idPersonnel"></param>
19        /// <param name="nom"></param>
20        /// <param name="prenom"></param>
21        /// <param name="tel"></param>

```

```

22      /// <param name="mail"></param>
23      /// <param name="service"></param>
24      /// <param name="absences"></param>
25      public Personnel(int idPersonnel, String nom, String prenom
    , String tel, String mail, Service service, List<Absence>
      absences)
26      {
27          this.idPersonnel = idPersonnel;
28          this.nom = nom;
29          this.prenom = prenom;
30          this.tel = tel;
31          this.mail = mail;
32          this.service = service;
33          this.absences = absences;
34      }
35      ...
36  }
37 }

```

Nous remarquons la présence de commentaires descriptifs selon un format bien spécifique avant chaque classes, constructeurs et méthodes. En effet, ces commentaires permettent la génération automatique d'une documentation technique, au format XML, ou plus généralement, grâce à des extensions ou des outils (doxygen, docfx, Sandcastle), au format HTML, permettant d'être lues à l'aide d'un simple navigateur internet. Dans ce projet, nous avons utilisé le programme Docfx pour générer la documentation des classes au format HTML.

3.4 Développement et test des fonctionnalités de l'application

Une fois les vues et le modèle créé, il est nécessaire de mettre en place des classes d'accès aux données contenues dans la base de données. Pour cela, nous avons créé 3 classes :

- La classe Access dont le rôle est d'initier une nouvelle connexion à la base de données par l'intermédiaire de la classe BddManager permettant d'exécuter des instructions SQL au niveau du SGBD MySQL.
- La classe ResponsableAccess contenant les méthodes nécessaires à la vérification de la validité des identifiants de connexion fournis par l'utilisateur.
- La classe PersonnelAccess contenant les méthodes nécessaires à l'accès et à la modification des données relatives à un personnel (Nom, prénom, liste d'absences etc...)

De manière globale, le développement des fonctionnalités s'est fait selon l'ordre suivant :

- Création des méthodes de gestion événementielles sur les objets graphiques des vues (Clic sur un bouton, chargement de la vue...)

- Création d'un contrôleur pour la vue dont la fonctionnalité à développer dépend. Ce contrôleur sert d'intermédiaire entre la vue et la couche d'accès aux données (DAL).
- Création d'une classe ou de méthodes dans le DAL pour accéder aux données qui nous intéressent : cette partie se charge d'exécuter des instructions SQL au niveau du SGBD puis renvoie les retours des requêtes (s'il y'en a) aux contrôleurs, qui se chargent alors d'initialiser le ou les objets des classes métiers avec les données correspondantes.

3.4.1 Connexion d'un responsable

Pour avoir accès aux vues de gestion du personnel, il est nécessaire de s'authentifier avec les identifiants d'un responsable présents dans la table 'responsable' de la base de données. Il convient de vérifier que le nom d'utilisateur et le mot de passe saisis par l'utilisateur est valide. Pour ce faire, il suffit d'interroger la base de données après avoir hashé le mot de passe saisi par l'utilisateur. En effet, la colonne 'pwd' de la table 'responsable' contient le mot de passe hashé du responsable selon l'algorithme SHA256, si les deux hashes correspondent, cela signifie que le mot de passe saisi (en clair) est le bon. Pour des raisons de sécurité, il est fortement déconseillé de stocker le mot de passe du responsable en clair dans la base, car en cas d'accès non autorisé à celle-ci, le mot de passe risque d'être dérobé.

Méthode événementielle de gestion du clic sur le bouton de connexion

```

1 private void BtnConnect_Click(object sender, EventArgs e)
2 {
3     Console.WriteLine("Connexion");
4     if (TbxLogin.Text.Length > 0 && TbxPwd.Text.Length > 0)
5     {
6         if (controller.controleAuthentification(TbxLogin.Text,
7         TbxPwd.Text))
8         {
9             FrmAdministration frmAdministration = new
10             FrmAdministration();
11             frmAdministration.Show();
12             MessageBox.Show("Connexion reussie.");
13         }
14         else
15         {
16             MessageBox.Show("Erreur, nom d'utilisateur ou mot de
17             passe incorrect.");
18         }
19     }
20     else
21     {
22         MessageBox.Show("Erreur, vous devez saisir tous les champs
23         avant de pouvoir vous connecter.");
24     }
25 }

```

Cette méthode contrôle que les champs du nom d'utilisateur et du mot de passe ont bien été saisis, sinon on affiche un message d'erreur. Si le formulaire a été rempli correctement, on demande au contrôleur de vérifier que les identifiants sont corrects grâce à la méthode `controleAuthentification` de `FrmConnexionController`.

Méthode `controleAuthentification` du contrôleur de la fenêtre de connexion

```

1  /// <summary>
2  /// Controle la validite des identifiants saisis par l'utilisateur
3  /// </summary>
4  /// <param name="login"></param>
5  /// <param name="pass"></param>
6  /// <returns>true si les identifiants sont corrects, false sinon</
7  public Boolean controleAuthentification(String login, String pass)
8  {
9      return responsableAcces.controleAuthentification(login, pass);
10 }
```

Cette méthode délègue la vérification des identifiants à la méthode `controleAuthentification` de la classe `ResponsableAccess` qui s'occupe d'accéder aux données concernant le responsable.

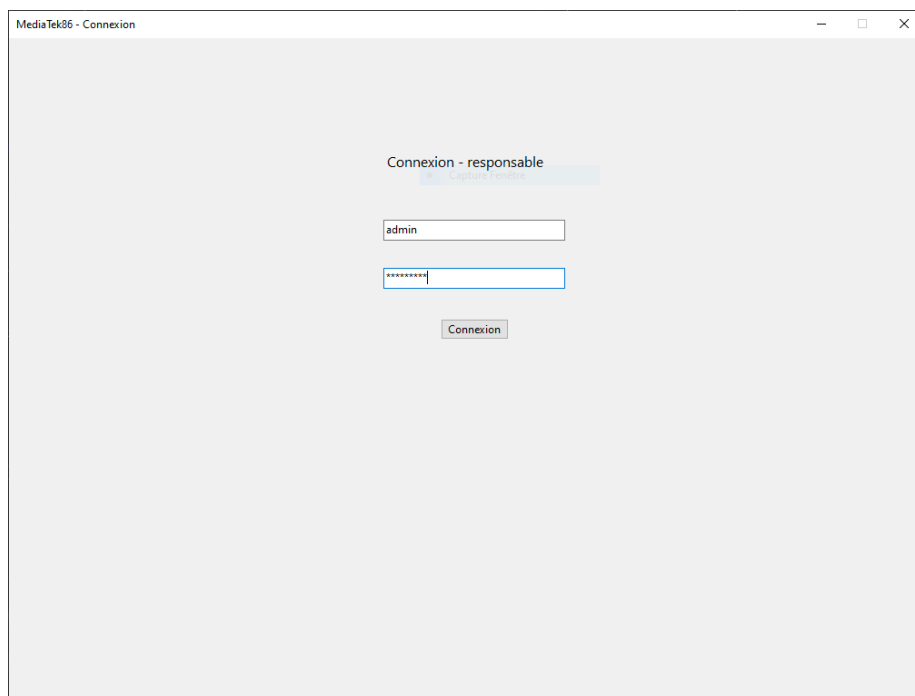
Méthode `controleAuthentification` de la classe `ResponsableAccess`

```

1  public Boolean controleAuthentification(String login, String pass)
2  {
3      if(access.bddManager != null)
4      {
5          String req = "SELECT * FROM responsable WHERE login=@login
6          AND pwd=SHA2(@pwd, 256)";
7          Dictionary<string, object> parameters = new Dictionary<
8          string, object>()
9          {
10             {"@login", login},
11             {"@pwd", pass}
12         };
13         try
14         {
15             List<Object[]> records = access.bddManager.ReqSelect(
16             req, parameters);
17             return records.Count > 0;
18         }
19         catch (Exception ex)
20         {
21             Console.WriteLine(ex.Message);
22             Environment.Exit(3306);
23         }
24     }
25     return false;
26 }
```

Cette méthode s'occupe de vérifier la validité des identifiants fournis en interrogeant la base de données via une requête SQL, dans ce cas, la requête retournera des lignes uniquement si le champ 'login' et 'pwd' correspondent aux paramètres

envoyés à cette méthode. Autrement dit, la requête se charge de vérifier que les identifiants saisis sont corrects, et si c'est le cas, retourne true, sinon false.



MediaTek86 - Connexion

Connexion - responsable

admin

Connexion

FIG. 6 – *Formulaire de connexion*

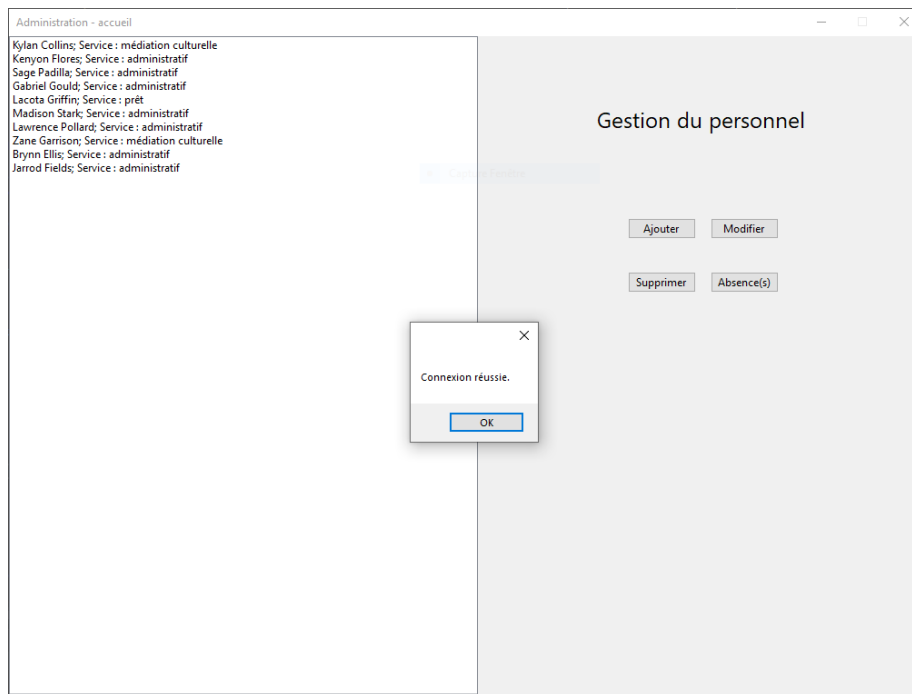


FIG. 7 – *Connexion réussie*

3.4.2 Affichage de la liste du personnel

Pour l’affichage de la liste du personnel, nous avons choisi une `ListBox` dans laquelle apparaît le nom, prénom et le service dont dépend le personnel.

Chargement de la liste du personnel dans la `ListBox` de la vue `FrmAdministration`

```

1 private void FrmAdministration_Load(object sender, EventArgs e)
2 {
3     foreach (Personnel personnel in controller.getLesPersonnels())
4     {
5         LbxPersonnel.Items.Add(personnel.ToString());
6     }
7 }

```

Cette méthode est chargée de récupérer la liste du personnel pour l’afficher dans la `ListBox`. On distingue deux méthodes importantes : `getLesPersonnels` du contrôleur de la fenêtre et la méthode `ToString` de la classe métier `Personnel`.

Méthode `ToString` de la classe `Personnel`

```

1 public override string ToString()
2 {
3     return nom + " " + prenom + "; Service : " + service.getNom();

```

```
4 }
```

Permet d'afficher l'objet Personnel selon le format défini dans cette méthode, qui est une redéfinition de la méthode ToString déjà existante.

Méthode getLesPersonnels du contrôleur de la fenêtre

```
1 public List<Personnel> getLesPersonnels()  
2 {  
3     return personnelAccess.getLesPersonnels();  
4 }
```

Délègue la récupération de la liste du personnel à la méthode getLesPersonnels de PersonnelAccess

Méthode getLesPersonnels de PersonnelAccess

```
1 public List<Personnel> getLesPersonnels()  
2 {  
3     List<Personnel> lesPersonnels = new List<Personnel>();  
4     if(access.bddManager != null)  
5     {  
6         String req = "SELECT * FROM personnel;";  
7         try  
8         {  
9             List<Object[]> lignes = access.bddManager.ReqSelect(req  
10            );  
11            foreach (Object[] l in lignes)  
12            {  
13                int idPersonnel = (int)l[0];  
14                String prenom = (String)l[1];  
15                String nom = (String)l[2];  
16                String tel = (String)l[3];  
17                String mail = (String)l[4];  
18                String _req = "SELECT nom FROM service WHERE  
19                idservice=@idservice";  
20                Dictionary<string, object> parameters = new  
21                Dictionary<string, object>()  
22                {  
23                    { "@idservice", l[5] }  
24                };  
25                try  
26                {  
27                    List<Object[]> rows = access.bddManager.  
28                    ReqSelect(_req, parameters);  
29                    Service service = new Service((int)l[5], rows  
30                    [0].GetValue(0).ToString());  
31                    List<Absence> absences = getPersonnelAbsences(  
32                    idPersonnel);  
33                    Personnel personnel = new Personnel(idPersonnel  
34                    , prenom, nom, tel, mail, service, absences);  
35                    lesPersonnels.Add(personnel);  
36                }  
37                catch (Exception ex)  
38                {  
39                    Console.WriteLine(ex.Message);  
40                    Environment.Exit(3306);  
41                }  
42            }  
43        }  
44    }  
45 }
```

```

34         }
35     }
36 }
37 catch (Exception ex)
38 {
39     Console.WriteLine(ex.Message);
40     Environment.Exit(3306);
41 }
42 }
43 return lesPersonnels;
44 }

```

Cette méthode interroge la base de données en lui demandant de lui retourner toutes les lignes de la table 'personnel' afin de construire une liste d'objet de type `Personnel` qu'il retourne au contrôleur, qui lui même retourne cette liste à la vue. Il est ensuite possible d'ajouter chacun de ces objets dans la `ListBox`, c'est le rôle du `foreach` dans la méthode `FrmAdministration_Load`. Pour chaque `Personnel` dans la liste du personnel, un `Item` est ajouté à la `ListBox` contenant la représentation textuelle de l'instance de l'objet personnel tel que définit dans la méthode `ToString`.

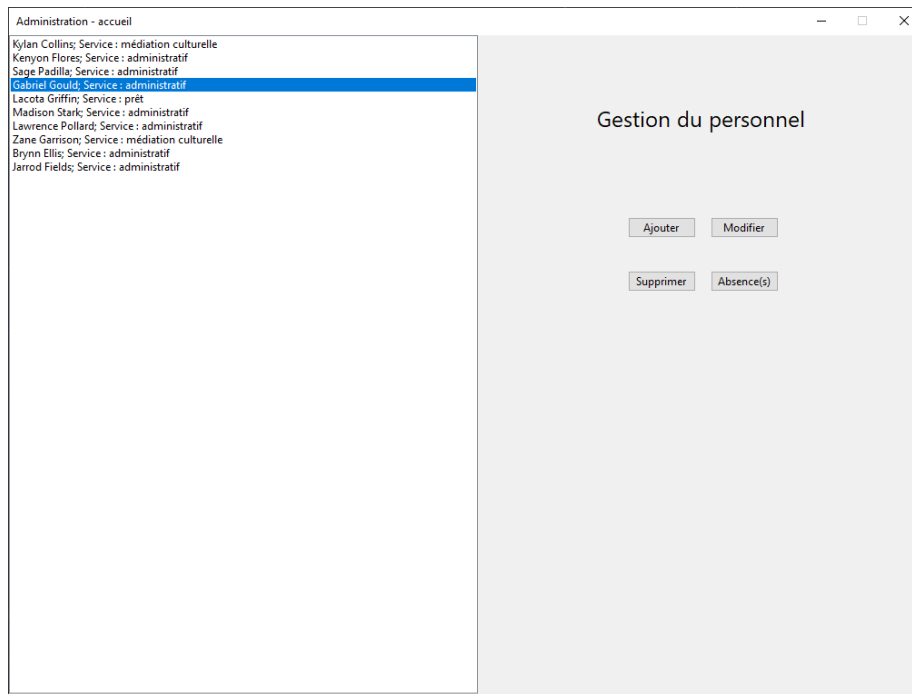


FIG. 8 – Liste du personnel chargée dans la `ListBox`

3.4.3 Ajout/modification d'un personnel

Pour éviter la redondance, nous avons décidé d'utiliser une seule vue pour gérer l'ajout d'un personnel, et la modification des informations d'un personnel déjà existant. En effet, nous distinguons deux cas grâce au constructeur qui attend un personnel en paramètre, si le personnel a une valeur nulle, il s'agit d'un ajout de personnel, sinon de la modification d'un personnel déjà existant. De cette façon, il est possible d'initialiser la fenêtre différemment et de différencier l'action à effectuer selon le cas à gérer (ajout ou modification).

Constructeur de la fenêtre

```
1 public FrmAjoutModifPersonnel(Personnel personnel)
2 {
3     InitializeComponent();
4     this.personnel = personnel;
5     if (this.personnel == null)
6     {
7         this.Text = "Administration - Ajout d'un personnel";
8         LblTitreAjoutModif.Text = "Ajout de personnel";
9         controller = new FrmAjoutModifPersonnelController(null);
10    }
11    else
12    {
13        this.Text = "Administration - Modification d'un personnel ";
14        ;
15        LblTitreAjoutModif.Text = "Modification de " + personnel.
16        getPrenom() + " " + personnel.getNom();
17        TbxNomPersonnel.Text = personnel.getNom();
18        TbxPrenomPersonnel.Text = personnel.getPrenom();
19        TbxTelPersonnel.Text = personnel.getTel();
20        TbxMailPersonnel.Text = personnel.getMail();
21        CbxServicePersonnel.Text = personnel.getService().getNom();
22        controller = new FrmAjoutModifPersonnelController(personnel
23    );
24    };
25    nomServices = controller.getNomServices();
26    CbxServicePersonnel.Items.Clear();
27    foreach (String nom in nomServices)
28    {
29        CbxServicePersonnel.Items.Add(nom);
30    }
31 }
```

Gestion de l'évènement lors du clic sur le bouton de validation

```
1 private void BtnEnregistrerPersonnel_Click(object sender, EventArgs
2     e)
3 {
4     if (personnel == null) //Ajout d'un nouveau personnel
5     {
6         if (TbxPrenomPersonnel.Text.Length > 0 && TbxNomPersonnel.
7         Text.Length > 0 && TbxTelPersonnel.Text.Length > 0 &&
8         TbxMailPersonnel.Text.Length > 0 && CbxServicePersonnel.
9         SelectedIndex != -1)
10        {
11            String nom = TbxNomPersonnel.Text;
```

```

8      String prenom = TbxPrenomPersonnel.Text;
9      String tel = TbxTelPersonnel.Text;
10     String mail = TbxMailPersonnel.Text;
11     int idService = CbxServicePersonnel.SelectedIndex + 1;
12     controller.ajouterPersonnel(nom, prenom, tel, mail,
    idService);
13     FrmAdministration frmAdministration = new
FrmAdministration();
14     this.Close();
15     frmAdministration.Show();
16 }
17 else
18 {
19     MessageBox.Show("Erreur, vous devez saisir toutes les
    informations du formulaire pour ajouter un nouveau personnel.")
    ;
20 }
21 }
22 else //Modification d'un personnel existant
23 {
24     if (TbxPrenomPersonnel.Text.Length > 0 && TbxNomPersonnel.
    Text.Length > 0 && TbxTelPersonnel.Text.Length > 0 &&
    TbxMailPersonnel.Text.Length > 0)
25     {
26         String nom = TbxNomPersonnel.Text;
27         String prenom = TbxPrenomPersonnel.Text;
28         String tel = TbxTelPersonnel.Text;
29         String mail = TbxMailPersonnel.Text;
30         int idService = -1;
31         if (CbxServicePersonnel.SelectedIndex != -1)
32         {
33             idService = CbxServicePersonnel.SelectedIndex + 1;
34         }
35         else
36         {
37             idService = personnel.getService().getIdService();
38         }
39         DialogResult r = MessageBox.Show("Etes-vous sur de
    vouloir modifier les informations de ce personnel ? Cette
    action est irreversible.", null, MessageBoxButtons.YesNo);
40         if (r == DialogResult.Yes)
41         {
42             controller.updatePersonnel(personnel.getIdPersonnel
    (), nom, prenom, tel, mail, idService);
43             FrmAdministration frmAdministration = new
FrmAdministration();
44             this.Close();
45             frmAdministration.Show();
46         }
47     }
48     else
49     {
50         MessageBox.Show("Erreur, vous devez saisir toutes les
    informations du formulaire pour modifier ce personnel.");
51     }
52 }
53 }

```

Administration - Modification d'un personnel

Modification de Stark Madison

Nom
Madison

Prénom
Stark

Tél
02 85 51 88 18

Mail
pede.praesent@hotmail.net

Service
administratif

Enregistrer

Annuler

FIG. 9 – *Modification des informations d'un personnel*

3.4.4 Affichage de la liste des absences d'un personnel

L'affichage de la liste des absence d'un personnel s'effectue après sélection d'un personnel dans la vue FrmAdministration (Liste des personnels).

Constructeur de la vue des absences

```

1 public FrmAbsencePersonnel(Personnel personnel)
2 {
3     InitializeComponent();
4     this.personnel = personnel;
5     LbxAbsencesPersonnel.DataSource = personnel.getAbsences();
6     LblAbsencePersonnel.Text = "Personnel : " + personnel.getPrenom() + " " + personnel.getNom();
7     controller = new FrmAbsencePersonnelController();
8 }

```

Ces absences sont chargées d'un seul coup dans la ListBox 'LbxAbsencePersonnel' grâce à la propriété DataSource et la méthode getAbsences de l'objet Personnel qui retourne la liste des absences pour ce personnel.

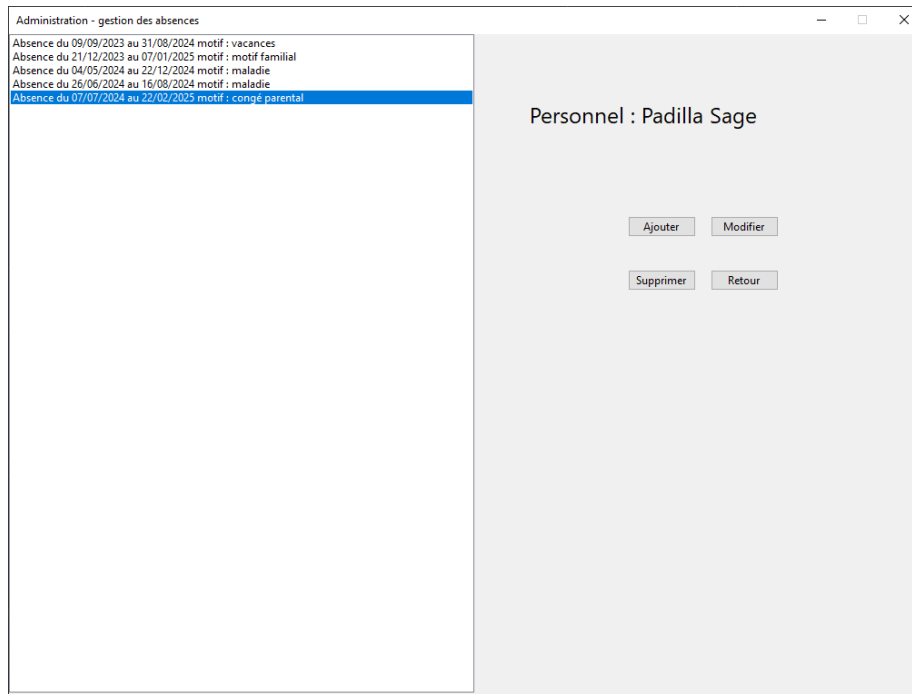


FIG. 10 – Liste des absences du personnel Padilla Sage

3.4.5 Ajout et modification des absences

Toujours pour éviter la redondance dans le code, nous avons choisi d'utiliser une unique vue pour gérer la modification d'une absence existante et l'ajout d'une nouvelle absence, car les formulaires sont les mêmes.

```
1 public FrmAjoutModifAbsence(Personnel personnel, Absence absence)
```

Le constructeur ci-dessus prend deux paramètres, dans le cas où le paramètre absence a une valeur nulle, il s'agit de l'ajout d'une nouvelle absence pour ce personnel, sinon, c'est qu'il s'agit de la modification d'une absence déjà présente. Dans les deux cas, la fenêtre est construite avec les mêmes éléments graphiques, mais dans le cas de la modification d'une absence, le formulaire est pré-rempli avec les informations de l'absence.

L'ajout d'une nouvelle absence se fait grâce à la méthode ajouterAbsencePersonnel de la classe PersonnelAccess, appelée par le contrôleur de la fenêtre

Méthode ajouterAbsencePersonnel de la classe PersonnelAccess

```
1 public void ajouterAbsencePersonnel(Personnel personnel, DateTime
   dateDebut, DateTime dateFin, Motif motif)
2 {
```



```

3      String req = "INSERT INTO absence (idpersonnel, datedebut,
4      datefin, idmotif) VALUES(@idpersonnel, @datedebut, @datefin,
5      @idmotif)";
6      Dictionary<string, object> parameters = new Dictionary<string,
7      object>()
8      {
9          {"@idpersonnel", personnel.getIdPersonnel()},
10         {"@datedebut", dateDebut},
11         {"@datefin", dateFin},
12         {"@idmotif", motif.getIdMotif()}
13     };
14     try
15     {
16         access.bddManager.ReqUpdate(req, parameters);
17     }
18     catch(Exception ex)
19     {
20         Console.WriteLine(ex.Message);
21         Environment.Exit(3306);
22     }

```

Cette méthode est chargée d'insérer une nouvelle absence dans la base de données avec les données saisies par l'utilisateur dans le formulaire d'ajout.

FIG. 11 – *Formulaire d'ajout et de modification d'absences du personnel*

4 Bilan

En conclusion, nous avons créé une application bureau permettant de gérer les informations du personnel de la médiathèque via une interface graphique utilisateur. La partie Modèle de l'application se décline en plusieurs objets : Personnel, Absence, Service et Motif. Ces objets peuvent ensuite être stockés et manipulés dans la base de données MySQL de l'application.