Boudier Nathan

ATELIER DE PROFESSIONNALISATION 2

COMPTE RENDU D'ACTIVITÉ

Sommaire

1- Contexte

MediaTek86 est un réseau qui gère les médiathèques de la Vienne, et qui a pour rôle de fédérer les prêts de livres, DVD et CD et de développer la médiathèque numérique pour l'ensemble des médiathèques du département.
L'ESN InfoTech Services 86 qui vient de remporter le marché pour différentes interventions au sein du réseau MediaTek86, dont certaines dans le domaine du développement d'application.

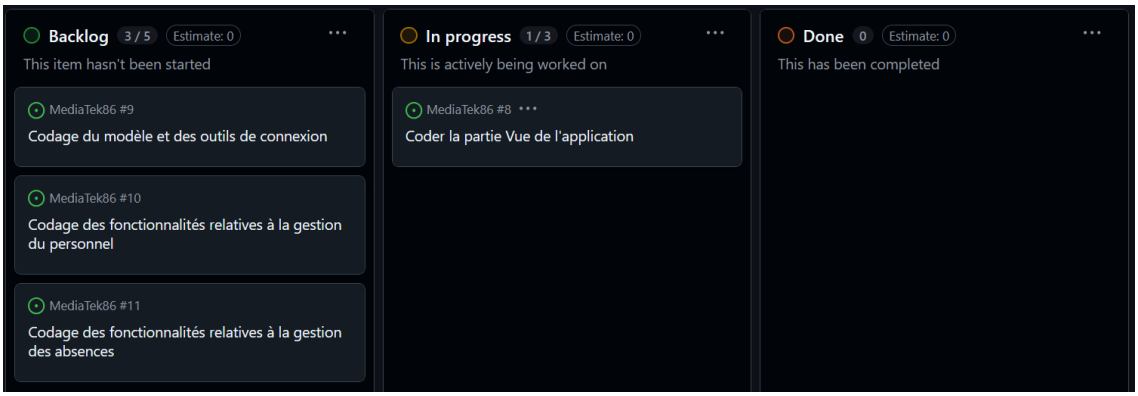
2- Mission

L'objectif est de développer une application de bureau qui va permettre de gérer le personnel de chaque médiathèque, leur affectation à un service et leurs absences. Cette application doit pouvoir permettre d'ajouter ou de supprimer un membre du personnel, ou de modifier ses informations. Elle doit aussi permettre d'ajouter ou de supprimer des absences, ou enfin de modifier des absences existantes.

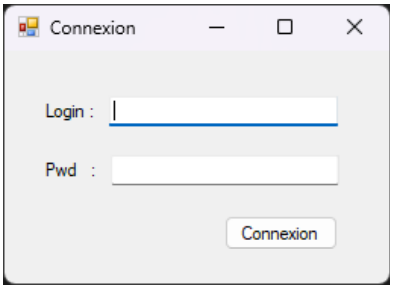
3- Présentation des étapes

3.1- Codage de la partie Vue

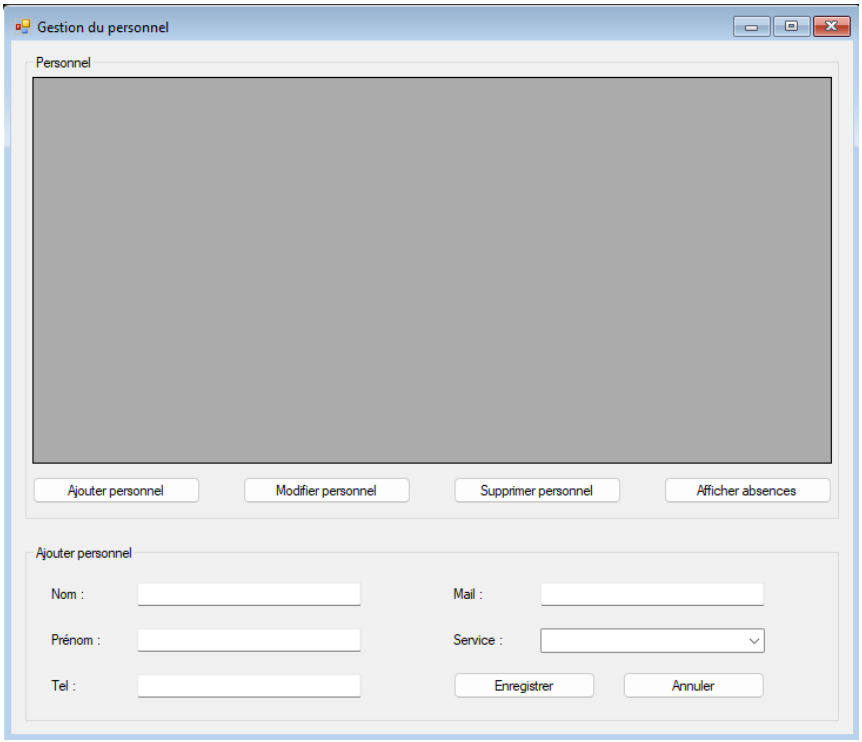
Dans un premier temps le but est de coder le visuel des interfaces.



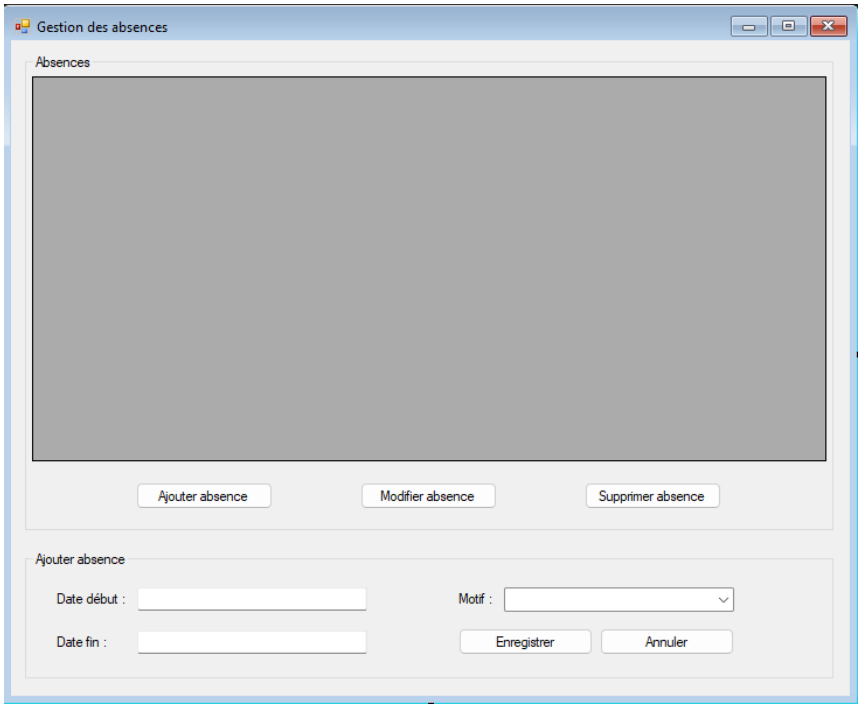
On commence donc par le formulaire de connexion qui permet d'entrer un login et un mot de passe puis de se connecter :



Il y a ensuite le formulaire de gestion du personnel, qui affiche les informations sur les membres du personnel puis qui permet d'ajouter ou supprimer un membre, ou modifier les informations d'un membre existant :



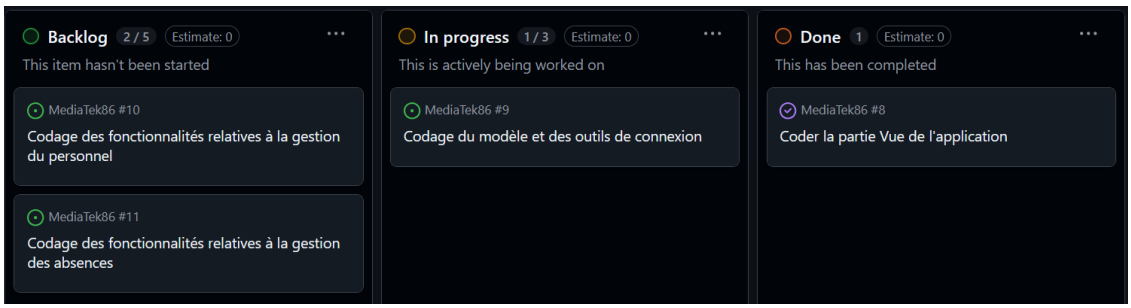
Enfin il y a le formulaire de gestion des absences, qui affiche les absence de la personne sélectionnée et permet d'ajouter ou supprimer une absence, ainsi que de modifier une absence existante :



Les interfaces sont désormais prêtes, on peut commencer à coder les fonctionnalités de l'application.

3.2- Préparation du modèle et des outils de connexion

Dans cette partie on va créer les différentes classes du modèle, ainsi que les classes qui vont gérer la connexion à la base de données.



On commence par créer la classe **BddManager** qui gère la connexion à la base de données :

```
public class BddManager
{
      (…)
      //constructeur pour créer la connexion à la BDD et l'ouvrir
      private BddManager(string stringConnect)
      {
            connection = new MySqlConnection(stringConnect);
            connection.Open();
      }
      //création d'une seule instance de la classe
      public static BddManager GetInstance(string stringConnect)
```

```
	{
		if (instance == null)
		{
			instance = new BddManager(stringConnect);
		}
		return instance;
	(…)
}
```

On crée ensuite la classe **Access** qui va servir d'intermédiaire entre **BddManager** et les autres classes.

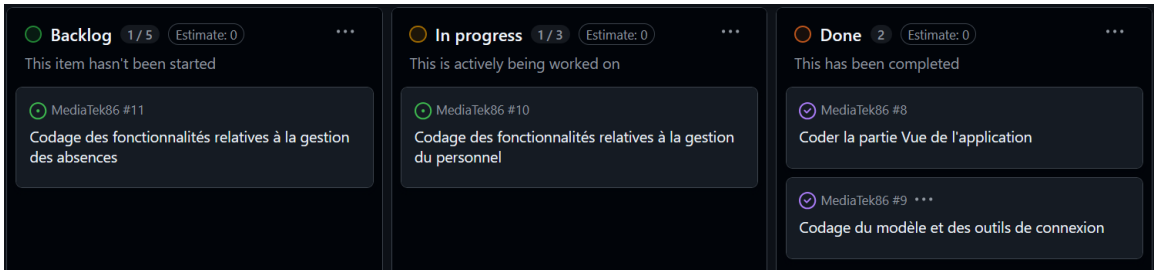
```
public class Access
{
	//chaîne de connexion à la BDD
	private static readonly string connectionString =
"Server=localhost;Port=3308;Database=mediatek;Uid=mediatek;Pwd=RespMediaTek86;";

	//instance unique de la classe
	private static Access instance = null;

	//getter sur l'objet d'accès aux données
	public BddManager Manager { get; }

	//création unique de l'objet de type BddManager
	private Access()
	{
		try
		{
			Manager = BddManager.GetInstance(connectionString) ;
		}
		catch (Exception)
		{
			Environment.Exit(0) ;
		}
	}

	//création d'une seule instance de la classe
	public static Access GetInstance()
	{
		if (instance == null)
		{
			instance = new Access() ;
		}
		return instance ;
	}
}
```

Enfin, on crée les classes du modèle : **Absence**, **Motif**, **Personnel**, **Responsable**, **Service**. Ces classe seront complétées dans les étapes suivantes.

Les outils de connexion sont désormais fonctionnels et les classes du modèle sont créées, on peut maintenant passer à la suite.

3.3- Codage de la partie gestion du personnel

Dans cette partie, nous allons coder les différentes fonctionnalités relatives à la gestion du personnel.



On commence par créer dans le package **controller** les classes **FrmConnexionController** et **FrmGestionPersonelController**. Ces classes vont servir à communiquer les demandes de la vue au package dal.

Dans ce package dal, on crée la classe **PersonnelAccess**, qui va permettre d'exécuter des requêtes SQL, comme par exemple cette méthode qui retourne la liste des informations sur le personnel :

```
public List<Personnel> GetLePersonnel()
{
      List<Personnel> lePersonnel = new List<Personnel>();
      if (access.Manager != null)
      {
            string req = "select p.idpersonnel as idpersonnel, p.nom as nom,
            p.prenom as prenom, p.tel as tel, p.mail as mail, s.idservice as
            idservice, s.nom as service ";
            req += "from personnel p join service s on (p.idservice =
            s.idservice) ";
            req += "order by nom, prenom;";
            try
            {
                  List<Object[]> records = access.Manager.ReqSelect(req);
                  if (records != null)
                  {
                        foreach (Object[] record in records)
                        {
                              Service service = new Service((int)record[5],
                              (string)record[6]);
                              Personnel personnel = new Personnel((int)record[0],
                              (string)record[1], (string)record[2],
                              (string)record[3], (string)record[4], service);
                              lePersonnel.Add(personnel);
                        }
                  }
            }
            catch (Exception e)
            {
                  Console.WriteLine(e.Message);
                  Environment.Exit(0);
            }
      }
      return lePersonnel ;
}
```

On crée aussi la classe **ServiceAccess** avec une méthode similaire qui retourne les informations sur les services.

On complète ensuite la classe **Personnel** :

```
public Personnel(int idpersonnel, string nom, string prenom, string tel, string
mail, Service service)
{
      this.Idpersonnel = idpersonnel;
      this.Nom = nom;
      this.Prenom = prenom;
      this.Tel = tel;
      this.Mail = mail;
      this.Service = service;
}
public int Idpersonnel { get; }
public string Nom { get; set; }
public string Prenom { get; set; }
public string Tel { get; set; }
public string Mail { get; set; }
public Service Service { get; set; }
```

Puis on complète de la même manière les classes **Service** et **Responsable**.

Dans la classe **FrmConnexion**, on ajoute la méthode suivante pour contrôler si l'utilisateur est bien responsable :

```
private void btnConnexion_Click(object sender, EventArgs e)
{
      String nom = txtLogin.Text;
      String pwd = txtPwd.Text;
      if (String.IsNullOrEmpty(nom) || String.IsNullOrEmpty(pwd))
      {
            MessageBox.Show("Tous les champs doivent être remplis.",
            "Information");
      }
      else
      {
            Responsable responsable = new Responsable(nom, pwd);
            if (controller.ControleConnexion(responsable))
            {
                  FrmGestionPersonnel frm = new FrmGestionPersonnel();
                  frm.ShowDialog();
            }
            else
            {
                  MessageBox.Show("Authentification incorrecte ou vous n'êtes pas
                  responsable", "Alerte");
            }
      }
}
```

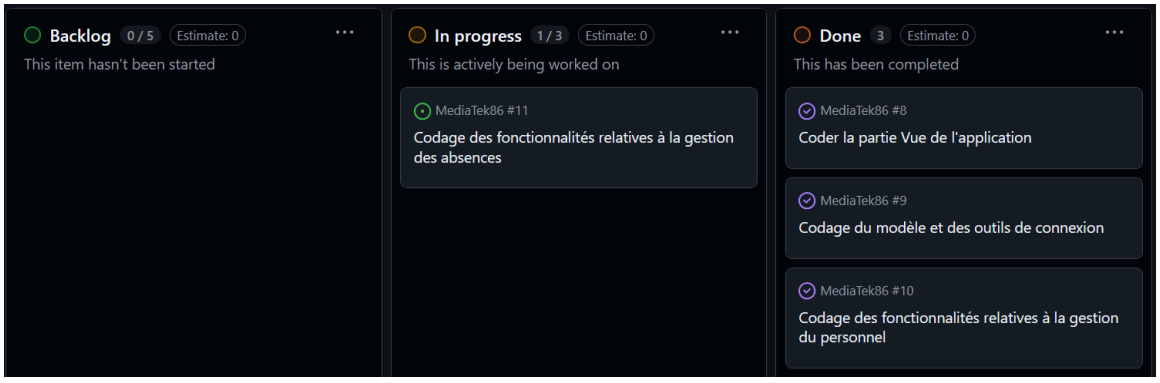
On complète enfin la classe **FrmGestionPersonnel**, avec notamment la méthode suivante qui permet d'enregistrer l'ajout ou la modification d'un membre du personnel dans la base de données :

```
private void btnEnregistrer_Click(object sender, EventArgs e)
{
      if (!txtNom.Text.Equals("") && !txtPrenom.Text.Equals("") && !
      txtTel.Text.Equals("") && !txtMail.Text.Equals("") &&
      cmbService.SelectedIndex != -1)
      {
            Service service = (Service)bdgServices.List[bdgServices.Position];
            if (enCoursDeModifPersonnel)
            {
                  Personnel personnel =
                  (Personnel)bdgPersonnel.List[bdgPersonnel.Position];
                  personnel.Nom = txtNom.Text;
                  personnel.Prenom = txtPrenom.Text;
                  personnel.Tel = txtTel.Text;
                  personnel.Mail = txtMail.Text;
                  personnel.Service = service;
                  controller.UpdatePersonnel(personnel);
            }
            else
            {
                  Personnel personnel = new Personnel(0, txtNom.Text,
                  txtPrenom.Text, txtTel.Text, txtMail.Text, service);
                  controller.AddPersonnel(personnel);
            }
            RemplirListePersonnel();
            EnCoursModifPersonnel(false);
            grpAjouterPerso.Enabled = false;
      }
      else
      {
            MessageBox.Show("Tous les champs doivent être remplis.",
            "Information");
      }
}
```

Toutes les fonctionnalités liées à la gestion du personnel sont désormais fonctionnelles. On peut donc s'occuper de la partie absences.

3.4- Codage de la partie gestion des absences

Dans cette partie, il est temps de coder les fonctionnalités du formulaire de gestion des absences.



On commence par créer et compléter la classe **FrmAbsencesController** qui va servir de relai entre la vue et le package dal.

On crée dans le package dal les classes **AbsenceAccess** et **MotifAccess**. Voici par exemple la méthode de AbsenceAccess qui sert à ajouter une absence dans la base de données :

```
public void AddAbsence(Absence absence)
{
      if (access.Manager != null)
      {
            string req = "insert into absence(idpersonnel, datedebut, datefin,
            idmotif) ";
            req += "values (@idpersonnel, @datedebut, @datefin, @idmotif);";
            Dictionary<string, object> parameters = new Dictionary<string,
            object>();
            parameters.Add("@idpersonnel", absence.Idpersonnel);
            parameters.Add("@datedebut", absence.Datedebut);
            parameters.Add("@datefin", absence.Datefin);
            parameters.Add("@idmotif", absence.Motif.Idmotif);
            try
            {
                  access.Manager.ReqUpdate(req, parameters);
            }
            catch (Exception e)
            {
                  Console.WriteLine(e.Message);
                  Environment.Exit(0);
            }
      }
}
```

On complète ensuite la classe **Absence** :

```
public Absence(int idpersonnel, DateTime datedebut, DateTime datefin, Motif
motif)
{
      this.Idpersonnel = idpersonnel;
      this.Datedebut = datedebut;
      this.Datefin = datefin;
      this.Motif = motif;
}

public int Idpersonnel { get; }
public DateTime Datedebut { get; set; }
public DateTime Datefin { get; set; }
public Motif Motif { get; set; }
```

On complète de la même façon la classe **Motif**.

Enfin, on complète la classe FrmAbsences avec par exemple la méthode suivante pour remplir la liste des absences :

```
private void RemplirListeAbsences(int id)
{
      unId = id;
      List<Absence> lesAbsences = controller.GetLesAbsences(id);
      bdgAbsences.DataSource = lesAbsences;
      dgvAbsences.DataSource = bdgAbsences;
      dgvAbsences.Columns["idpersonnel"].Visible = false;
      dgvAbsences.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
}
```

L'application est désormais complète et toutes les fonctionnalités demandées sont implémentées.

4- Bilan

Le développement de l'application de gestion du personnel et des absences pour le réseau MediaTek86 a été mené à bien avec succès. Cette application permet désormais de gérer efficacement le personnel des médiathèques affiliées, leurs affectations aux différents services, ainsi que leurs absences.

4.1- Réalisation des Objectifs

L'objectif principal de ce projet était de créer une application de bureau complète et fonctionnelle permettant d'ajouter, de supprimer et de modifier les informations relatives au personnel et à leurs absences. Grâce à une planification rigoureuse et une exécution méthodique, nous avons pu atteindre cet objectif.

- **Interface Utilisateur** : Les interfaces utilisateur pour la connexion, la gestion du personnel et la gestion des absences ont été conçues et implémentées avec soin pour offrir une expérience utilisateur intuitive et efficace.
- **Fonctionnalités** : Toutes les fonctionnalités demandées ont été développées et testées. L'application permet désormais d'ajouter, de supprimer et de modifier les membres du personnel ainsi que leurs absences.
- **Connexion à la Base de Données** : Les outils de connexion à la base de données ont été développés avec succès, permettant une interaction fluide et sécurisée avec les données.

4.2- Défis Rencontrés

Comme pour tout projet de développement logiciel, nous avons rencontré plusieurs défis :

- **Complexité de la Gestion des Données** : La gestion des données relatives au personnel et à leurs absences a nécessité une attention particulière pour garantir l'intégrité et la cohérence des données.
- **Sécurité** : Assurer la sécurité des informations de connexion et des données sensibles a été une priorité, nécessitant l'implémentation de mesures de sécurité comme le hachage des mots de passe.
- **Intégration des Différentes Parties** : L'intégration des différentes parties de l'application, notamment l'interface utilisateur et la logique métier, a demandé une attention minutieuse.

4.3- Perspectives d'Amélioration

Bien que l'application soit désormais opérationnelle, il existe toujours des possibilités d'amélioration et d'extension :

- **Amélioration de l'Interface Utilisateur** : Continuer à affiner l'interface utilisateur pour améliorer l'expérience utilisateur et ajouter des fonctionnalités supplémentaires pour faciliter la gestion du personnel.
- **Optimisation des Performances** : Optimiser les performances de l'application, notamment en ce qui concerne les requêtes à la base de données, pour garantir une utilisation fluide même avec un grand nombre de données.

4.4- Conclusion

Le projet de développement de l'application de gestion du personnel et des absences pour MediaTek86 a été une réussite. Grâce à une planification rigoureuse et une exécution méthodique, nous avons pu livrer une application complète et fonctionnelle qui répond aux besoins du réseau MediaTek86.