

Compte Rendu d'Activité - Projet MediaTek86 Gestion du Personnel

Table des matières

Introduction :.....	2
Début du projet :.....	3
Dessiner les interfaces et structurer l'application en MVC et coder le visuel des interfaces :.....	3
Coder le modèle et les outils de connexion, générer la documentation technique :.....	5
Coder les fonctionnalités de l'application :.....	6
Connexion :.....	6
Voir la liste du personnel :.....	6
Ajout d'un nouveau personnel :.....	7
Modification d'un personnel existant :.....	7
Suppression d'un personnel :.....	8
Affichage des absences d'un personnel :.....	8
Ajout, Modification et Suppression d'absences :.....	9
Bilan final :.....	10

Introduction :

Ce document est un compte rendu d'activité du Projet MediaTek86 réalisé dans le contexte du cours d'Atelier de Professionnalisation de l'option SLAM

Rappel du Contexte :

Le réseau **MediaTek86**, qui fédère l'ensemble des médiathèques du département de la Vienne à pour but d'harmoniser et de centraliser la gestion des prêts de documents (livres, DVD, CD) et de développer une offre de médiathèque numérique accessible à toutes les structures du réseau.

Rappel de la Mission Globale :

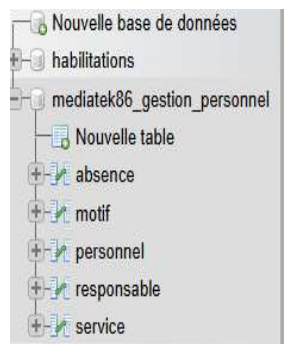
Pour y parvenir infotech Services 86 ma confié la mission de développer une application de bureau destinée au service administratif du réseau Mediatek86. Cette application a pour but principal de faciliter et de centraliser la gestion du personnel affecté à chaque médiathèque. Plus spécifiquement, elle doit permettre de gérer les informations des employés, leur affectation à un service particulier (administratif, médiation culturelle, prêt) ainsi que le suivi de leurs absences (vacances, maladie, etc.).

L'application doit être développée en langage C# avec le framework .NET pour Windows Forms.

Début du projet :

Pour commencer j'ai du créer la base de données dans phpMyAdmin que j'ai appelé mediatek86_gestion_personnel. J'ai ensuite exécuté le script SQL fourni pour créer les tables service, motif, personnel, et absence. J'ai ensuite ajouté la table responsable avec pour identifiant admin_mediatek et comme mot de passe P@\$wOrd123 qui sera hashé en SHA256. J'ai ensuite rempli les tables de référence motif (vacances, maladie, etc.) et service (administratif, médiation culturelle, prêt) avec les données fournies.

Pour les tables personnel et absence, j'ai utilisé l'outil en ligne generatedata.com pour générer des scripts d'insertion SQL. 10 ont été générés pour personnel et 50 pour absence.

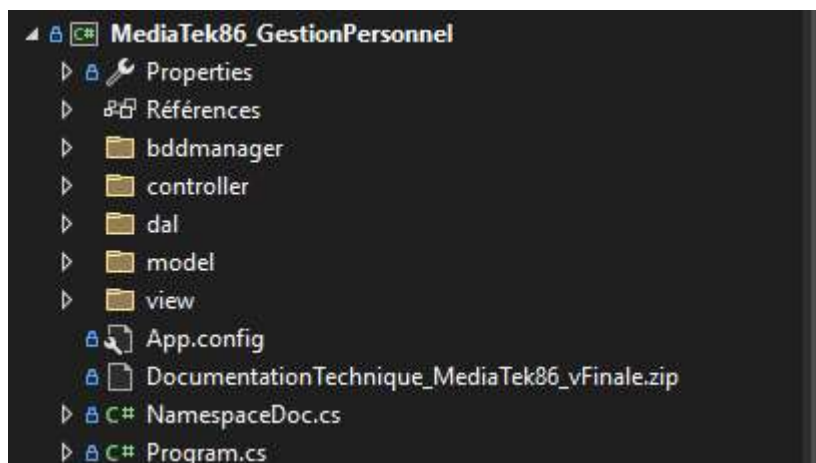


The screenshot shows the phpMyAdmin interface for a database named 'mediatek86_gestion_personnel'. On the left, a tree view shows the database structure with tables: absence, motif, personnel, responsable, and service. The main area displays a table overview for these five tables.

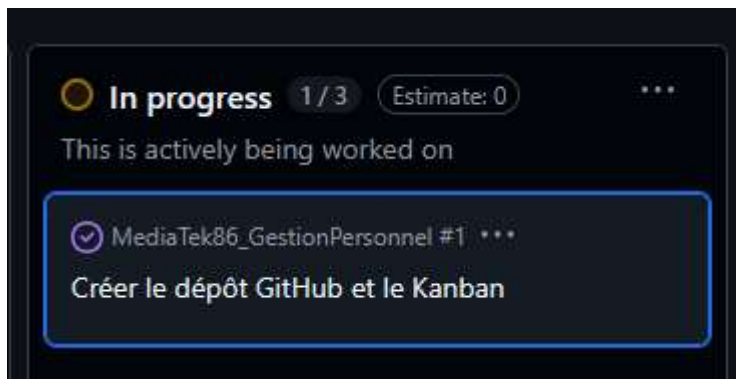
Table	Action	Lignes	Type	Interclassement	Taille	Perte
absence	Parcourir Structure Rechercher Insérer Vider Supprimer	50	MyISAM	utf8mb4_0900_ai_ci	4,0 kio	57 o
motif	Parcourir Structure Rechercher Insérer Vider Supprimer	4	MyISAM	utf8mb4_0900_ai_ci	2,1 kio	-
personnel	Parcourir Structure Rechercher Insérer Vider Supprimer	11	MyISAM	utf8mb4_0900_ai_ci	3,8 kio	84 o
responsable	Parcourir Structure Rechercher Insérer Vider Supprimer	1	MyISAM	utf8mb4_0900_ai_ci	1,1 kio	-
service	Parcourir Structure Rechercher Insérer Vider Supprimer	3	MyISAM	utf8mb4_0900_ai_ci	2,1 kio	-
5 tables	Somme	69	MyISAM	utf8mb4_0900_ai_ci	13,0 kio	141 o

Dessiner les interfaces et structurer l'application en MVC et coder le visuel des interfaces :

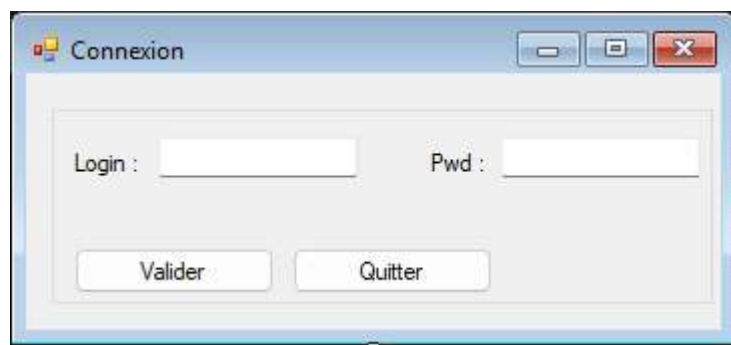
J'ai ensuite créé un nouveau projet en C# sur Visual Studio que j'ai nommé MediaTek86_GestionPersonnel et j'ai créé les dossiers pour organiser le code selon le modèle MVC.



J'ai ensuite créer un dépôt distant sur github ainsi qu'un tableau Kanban pour suivre l'avancement du projet.



Les formulaire d'interfaces graphiques ont ensuite été crée et le code des boutons généré.



Exemple avec le fenêtre de connexion.

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

Premièrement, il a fallu ajouter le package `MySQL.Data` qui gère l'accès à la base de données MySQL. J'ai ensuite adapté la classe `BddManager.cs` provenant du projet Habilitation.

Dans dal j'ai ajouté `Acess.cs` qui sert à centraliser la gestion de la chaîne de connexion et permet de se connecter avec l'utilisateur qui a accès à la base de données dans `phpMyAdmin`.

```
/// <summary>
/// chaîne de connexion à la bdd
/// </summary>
private static readonly string connectionString = "server=localhost;user id=mediatek_user;password=Monsupermotdepasse;database=mediatek86_gestion_personnel";
```

J'ai ensuite créer les classes métiers `Personnel.cs`, `Service.cs`, `Motif.cs`, `Absence.cs` dans le dossier

```
namespace MediaTek86_GestionPersonnel.bddmanager
{
    /// <summary>
    /// Singleton : connexion à la base de données et exécution des requêtes
    /// </summary>
    9 références
    public class BddManager
    {
        /// <summary>
        /// instance unique de la classe
        /// </summary>
        private static BddManager instance = null;
        /// <summary>
        /// objet de connexion à la BDD à partir d'une chaîne de connexion
        /// </summary>
        private readonly MySqlConnection connection;

        /// <summary>
        /// Constructeur pour créer la connexion à la BDD et l'ouvrir
        /// </summary>
        /// <param name="stringConnect">chaîne de connexion</param>
        1 référence
        private BddManager(string stringConnect)
        {
            connection = new MySqlConnection(stringConnect);
            connection.Open();
        }
    }
}
```

model.. Chaque classe contient les propriétés correspondant aux colonnes des tables de la base, ainsi que des constructeurs et des commentaires XML pour la documentation.

Pour générer la documentation technique j'ai utilisé Sandcastle en lui fournissant le fichier xml créer par visual studio. Pour éviter les erreurs de commentaires de résumé manquants au niveau des namespaces j'ai créé des classes `NamespaceDoc.cs` vides dans chaque espaces.

La documentation technique à ensuite été ajouté sous format .zip au github du projet.

Coder les fonctionnalités de l'application :

Les fonctionnalités à implémenter sont les suivantes :

- Pouvoir se connecter à l'application.
- Pouvoir voir la liste actuelle du personnel
- Pouvoir ajouter/modifier/supprimer du personnel
- Pouvoir voir les absences de chaque membre du personnel
- Et pouvoir ajouter/modifier/supprimer les absences

Connexion :

Premièrement, pour la connexion c'est le formulaire FrmConnexion du dossier vue qui collecte le login et le mot de passe de l'utilisateur. Lorsque l'on clique sur Valider les informations sont transmises à FrmConnexionController qui fait ensuite appel à la méthode VerifierIdentifiants de la classe ResponsableAccess. Cette méthode interroge la table responsable de la base de données MySQL en comparant le login fourni et le hash SHA256 du mot de passe saisi avec l'enregistrement existant. En cas de réussite FrmConnexion est masquée, et la fenêtre principale FrmGestionPersonnel

```
public bool VerifierIdentifiants(string login, string pwd)
{
    string req = "SELECT COUNT(*) FROM responsable WHERE login = @login AND pwd = SHA2(@pwd, 256);";
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@login", login);
    parameters.Add("@pwd", pwd);

    try
    {
        List<object[]> result = bddManager.ReqSelect(req, parameters);
        if (result != null && result.Count > 0)
        {
            long count = (long)result[0][0];
            return count > 0;
        }
    }
    catch (System.Exception ex)
    {
        System.Diagnostics.Debug.WriteLine("Erreur lors de la vérification des identifiants : " + ex.Message);
    }
    return false;
}
```

est affichée.

Méthode VerifierIdentifiant

Voir la liste du personnel :

Au chargement de FrmGestionPersonnel son constructeur initialise FrmGestionPersonnelController ce qui appelle la méthode ChargerPersonnel(). Cette méthode ChargerPersonnel() fait appel à la méthode GetLePersonnel() du FrmGestionPersonnelController. Le contrôleur, à son tour, sollicite la méthode GetAllPersonnel() de la classe PersonnelAccess

```

1 référence
public List<Personnel> GetAllPersonnel()
{
    List<Personnel> lesPersonnels = new List<Personnel>();
    // Requête SQL
    string req = "SELECT p.idpersonnel, p.nom, p.prenom, p.tel, p.mail, p.idservice, s.nom AS service_nom ";
    req += "FROM personnel p JOIN service s ON p.idservice = s.idservice ORDER BY p.nom, p.prenom;";

    try
    {
        List<object[]> records = bddManager.ReqSelect(req);
        foreach (object[] record in records)
        {
            Personnel personnel = new Personnel(
                (int)record[0], // idpersonnel
                (string)record[1], // nom
                (string)record[2], // prenom
                (string)record[3], // tel
                (string)record[4], // mail
                (int)record[5], // idservice
                (string)record[6] // service_nom
            );
            lesPersonnels.Add(personnel);
        }
    }
}

```

Cette méthode exécute une requête SQL qui récupère tous les enregistrements de la table personnel, en effectuant une jointure avec la table service pour obtenir le nom du service associé à chaque employé. Ces données sont ensuite envoyées dans un dataGridView pour être affichées.

Ajout d'un nouveau personnel :

Depuis la fenêtre FrmGestionPersonnel, un clic sur le bouton "Ajouter Personnel" déclenche l'ouverture de la fenêtre FrmAjoutModifPersonnel. L'utilisateur remplit les champs (nom, prénom, téléphone, mail) et sélectionne un service. Lors du clic sur le bouton "Enregistrer" de FrmAjoutModifPersonnel, ensuite, le contrôleur transmet cet objet Personnel à la méthode AddPersonnel() de la classe PersonnelAccess. Cette dernière construit et exécute une requête SQL INSERT INTO pour ajouter le nouveau personnel dans la table personnel.

```

public bool AddPersonnel(Personnel personnel)
{
    string req = "INSERT INTO personnel (nom, prenom, tel, mail, idservice) ";
    req += "VALUES (@nom, @prenom, @tel, @mail, @idservice);";

    Dictionary<string, object> parameters = new Dictionary<string, object>();
    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);
}

```

Modification d'un personnel existant :

Pour modifier un personnel, l'utilisateur sélectionne d'abord une ligne dans le DataGridView de FrmGestionPersonnel, puis clique sur le bouton "Modifier Personnel". L'objet Personnel correspondant est récupéré depuis le BindingSource. La fenêtre FrmAjoutModifPersonnel est alors

ouverte, cette fois en lui passant l'objet Personnel à modifier ce qui préremplit les cases. Une fois les modifications souhaitées effectuées on appuie sur "Enregistrer" ce qui met à jour le personnel modifié et l'affiche ensuite dans le dataGridView.

```
public bool 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("@nom", personnel.Nom);
    parameters.Add("@prenom", personnel.Prenom);
    parameters.Add("@tel", personnel.Tel);
    parameters.Add("@mail", personnel.Mail);
    parameters.Add("@idservice", personnel.IdService);
    parameters.Add("@idpersonnel", personnel.IdPersonnel);
}
```

Suppression d'un personnel :

Après avoir sélectionné un personnel dans le DataGridView de FrmGestionPersonnel, l'utilisateur clique sur "Supprimer Personnel". La méthode SupprimerPersonnel() du FrmGestionPersonnelController est appelée, en lui passant l'IdPersonnel de l'employé sélectionné. Le contrôleur transmet cet ID à la méthode DeletePersonnel() de PersonnelAccess. Pour ne pas qu'un personnel soit supprimé et que ces absences restent dans la base de données la méthode DeletePersonnel() procède en deux étapes. Elle exécute d'abord une requête DELETE FROM absence WHERE idpersonnel = @idpersonnel pour supprimer toutes les absences liées à ce personnel, puis une requête DELETE FROM personnel WHERE idpersonnel = @idpersonnel pour supprimer le personnel lui-même.

```
public bool DeletePersonnel(int idPersonnel)
{
    //Supprimer les absences liées à ce personnel
    string reqDeleteAbsences = "DELETE FROM absence WHERE idpersonnel = @idpersonnel;";
    Dictionary<string, object> parametersAbsences = new Dictionary<string, object>();
    parametersAbsences.Add("@idpersonnel", idPersonnel);

    //Supprimer le personnel
    string reqDeletePersonnel = "DELETE FROM personnel WHERE idpersonnel = @idpersonnel;";
    Dictionary<string, object> parametersPersonnel = new Dictionary<string, object>();
    parametersPersonnel.Add("@idpersonnel", idPersonnel);
}
```

Le dataGridView est ensuite refresh et s'affiche avec un personnel en moins .

Affichage des absences d'un personnel :

Depuis FrmGestionPersonnel, après avoir sélectionné un employé, un clic sur "Gérer les absences" ouvre la fenêtre FrmGestionAbsences. Le constructeur de cette fenêtre reçoit l'objet Personnel sélectionné. Le titre de la fenêtre est mis à jour pour indiquer de quel personnel il s'agit, et son nom

est affiché dans un Label. Le constructeur de FrmGestionAbsences initialise ensuite son propre contrôleur, FrmGestionAbsencesController. La méthode ChargerAbsences() du formulaire est appelée, laquelle demande au contrôleur la liste des absences pour l'IdPersonnel concerné via la méthode GetLesAbsences(). FrmGestionAbsencesController fait appel à AbsenceAccess.GetAbsences(), qui exécute une requête SELECT sur la table absence (avec une jointure sur motif pour obtenir le libellé) pour le personnel spécifié, triant les résultats par date de début décroissante. Cette liste est ensuite ajoutée au dataGridView pour afficher les absences à l'utilisateur.

```
public List<Absence> GetAbsences(int idPersonnel)
{
    List<Absence> lesAbsences = new List<Absence>();
    string req = "SELECT a.idpersonnel, a.datedebut, a.datefin, a.idmotif, m.libelle AS motif_libelle ";
    req += "FROM absence a JOIN motif m ON a.idmotif = m.idmotif ";
    req += "WHERE a.idpersonnel = @idpersonnel ORDER BY a.datedebut DESC;";

    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", idPersonnel);
}
```

Ajout, Modification et Suppression d'absences :

Le processus pour ajouter, modifier et supprimer une absence à partir de FrmGestionAbsences suit une logique très similaire à celle de la gestion du personnel, mais en utilisant FrmAjoutModifAbsence et les méthodes correspondantes dans FrmGestionAbsencesController et AbsenceAccess.

Pour ajouter une absence, Le bouton "Ajouter" sur FrmGestionAbsences ouvre FrmAjoutModifAbsence . Après saisie et validation des dates et du motif, controller.AjouterAbsence() est appelé, qui utilise AbsenceAccess.AddAbsence() pour l'insertion en base.

```
public bool AddAbsence(Absence absence)
{
    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); // Doit être au format YYYY-MM-DD pour la BDD
    parameters.Add("@datefin", absence.DateFin); // Pareil
    parameters.Add("@idmotif", absence.IdMotif);
}
```

Pour modifier une absence, la sélection d'une absence dans dgvAbsencesPersonnel et un clic sur "Modifier" ouvre FrmAjoutModifAbsence en mode modification, en lui passant l'objet Absence à éditer avec les champs préremplis. Lors de l'enregistrement, controller.ModifierAbsence() est appelé, transmettant l'objet Absence mis à jour et la date de début originale de l'absence. AbsenceAccess.UpdateAbsence() met à jour l'enregistrement en base en utilisant l'idpersonnel et la dateDebutOriginale dans sa clause WHERE.

```

public bool UpdateAbsence(Absence absenceModifiee, DateTime dateDebutOriginale)
{
    if (absenceModifiee.DateFin < absenceModifiee.DateDebut)
    {
        System.Diagnostics.Debug.WriteLine("Erreur dans UpdateAbsence : Date de fin antérieure à la date de début.");
        return false;
    }

    string req = "UPDATE absence SET datefin = @datefin, idmotif = @idmotif, datedebut = @datedebut ";
    req += "WHERE idpersonnel = @idpersonnel AND datedebut = @dateDebutOriginale;";

    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@datefin", absenceModifiee.DateFin.Date);
    parameters.Add("@idmotif", absenceModifiee.IdMotif);
    parameters.Add("@datedebut", absenceModifiee.DateDebut.Date);
    parameters.Add("@idpersonnel", absenceModifiee.IdPersonnel);
    parameters.Add("@dateDebutOriginale", dateDebutOriginale.Date);
}

```

Enfin, pour supprimer une absence controller.SupprimerAbsence() est appelé avec l'idpersonnel et la dateDebut de l'absence à supprimer. AbsenceAccess.DeleteAbsence() et exécute la requête DELETE.

```

public bool DeleteAbsence(int idPersonnel, DateTime dateDebut)
{
    string req = "DELETE FROM absence WHERE idpersonnel = @idpersonnel AND datedebut = @datedebut;";

    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@idpersonnel", idPersonnel);
    parameters.Add("@datedebut", dateDebut);
}

```

Pour ces trois opérations, j'ai dû standardiser les dates avec l'utilisation de .Date (pour ne considérer que la partie jour et mettre l'heure à 00:00:00).

Bilan final :

L'objectif principal de la mission était de développer une application C# Windows Forms permettant la gestion du personnel, de leurs affectations et de leurs absences, tout en respectant une architecture de type MVC. Je considère que ces objectifs ont été atteints.

- Toutes les fonctionnalités spécifiées dans les cas d'utilisation ont été codées et testées.
- L'application a été structurée en suivant une approche inspirée du MVC, avec une séparation claire des responsabilités.
- Les interfaces ont été conçues pour être fonctionnelles et intuitives.
- Une documentation technique a été générée à partir des commentaires XML du code, et une documentation utilisateur sous forme de vidéo de démonstration a été produite.

Ce projet a été l'occasion de mettre en pratique et de renforcer de nombreuses compétences tel que la programmation objet en C#, la maîtrise de visual studio, de github, la génération de documentation XML, l'interaction avec une base de données MySQL etc.