



### **Compte Rendu d'Activité**

#### **Développement d'une application de gestion du personnel et des absences**

*(Architecture MVC, Base de données MySQL, Interface utilisateur Windows Forms)*

Projet réalisé dans le cadre de l'Atelier de professionnalisation en 1<sup>ère</sup> année de BTS SIO option SLAM

# Sommaire

<b>1.Introduction :</b>	<b>5</b>
<b>1.1. Contexte :</b>	<b>5</b>
<b>1.2 Présentation de la mission globale</b>	<b>5</b>
<b>2. Etapes du projet</b>	<b>6</b>
<b>2.1 Étape 1 - Préparer l'environnement de travail et créer la base de données</b>	<b>6</b>
Objectifs de l'étape	6
Mise en place de l'environnement de travail	6
Création de la base de données	6
Création de l'utilisateur d'accès à la base de données	7
Création et remplissage de la table 'responsable'	8
Remplissage des tables 'motif' et 'service'	8
Pour la table 'service', je la remplit des services suivants : vacances, administratif, médiation culturelle, prêt. Son contenu ne sera pas modifié dans l'application	9
INSERT INTO service (idservice, nom) VALUES	9
(1, 'administratif'),	9
(2, 'médiation culturelle'),	9
(3, 'prêt');	9
Remplissage des tables 'personnel' et 'absence'	9
Bilan de l'étape	11
<b>2.2 Étape 2 - dessiner les interfaces, structurer l'application en MVC, créer un dépôt, coder le visuel</b>	<b>11</b>
Objectif de l'étape	11
Analyse documentaire	12
Maquettage des interfaces	12
Mise en place de la structure MVC	13
Dépôt GitHub et Kanban	14
Développement du visuel des interfaces (Vue)	14
Bilan de l'étape	15
<b>2.3 Étape 3 - coder le modèle et les outils de connexion, générer la documentation technique</b>	<b>16</b>

Objectifs de l'étape .....	16
Configuration de l'environnement de développement .....	16
Classe BddManager (connexion, requêtes).....	16
Classe Access (chaîne de connexion, Singleton).....	19
Création des classes métiers (model) .....	20
Documentation technique .....	20
Dépôt GitHub et Kanban .....	21
Bilan de l'étape .....	21
<b>2.4 Étape 4 - coder les fonctionnalités de l'application à partir des cas d'utilisation .....</b>	<b>21</b>
Objectifs de l'étape .....	21
Développement des classes métiers (model) .....	22
Création et développement des autres classes dal .....	23
Création et développement du contrôleur principal (FrmMediaTek86Controller) ..	25
Ajout des fonctionnalités de gestion des personnels (affichage, ajout, modification, suppression) .....	27
Ajout des fonctionnalités de gestion des absences (affichage, ajout, modification, suppression) .....	35
Vérification anti-doublon sur ajout/modification de personnel (nom, prenom, service).....	43
Authentification (login sécurisé, vue dédiée) .....	45
Bilan de l'étape .....	48
<b>2.5 Étape 5 – Création d'une documentation utilisateur en vidéo .....</b>	<b>48</b>
<b>2.6 Étape 6 – Déploiement, compte rendu final et intégration au portfolio .....</b>	<b>48</b>
Objectif de l'étape .....	49
Création de l'installateur sous VisualStudio .....	49
Script SQL complet de la base de données.....	49
Rédaction du compte rendu d'activité.....	49
Création de la page dans le portfolio .....	50
Bilan de l'étape .....	50
<b>3. Bilan général du projet .....</b>	<b>50</b>
<b>3.1 Résumé des objectifs atteints à chaque étape .....</b>	<b>51</b>
Étape 1 : Préparation de l'environnement et création de la base de données.....	51

Étape 2 : Conception des interfaces, structure MVC et dépôt GitHub .....	51
Étape 3 : Implémentation du modèle et accès aux données .....	51
Étape 4 : Développement des fonctionnalités métiers .....	51
Étape 5 : Création de la documentation utilisateur vidéo .....	52
Étape 6 : Déploiement, compte rendu final et intégration au portfolio .....	52
<b>4. Conclusion générale .....</b>	<b>53</b>

# 1.Introduction :

## 1.1. Contexte :

Dans le cadre de cet exercice, je simule le rôle d'une technicienne développeur junior travaillant pour InfoTech Services 86 (ITS 86), une Entreprise de Services Numériques (ESN) basée dans la Vienne.

L'entreprise ITS 86 est spécialisée dans le développement d'applications (web, mobile, bureau), l'infogérance, l'hébergement web, et la gestion de parcs informatiques. Forte de ses 32 collaborateurs, elle répond régulièrement à des marchés publics ou à des demandes de TPE/PME locales.

Dans ce contexte simulé, ITS 86 a remporté un appel d'offres lancé par le réseau MediaTek86, un organisme qui regroupe l'ensemble des médiathèques du département de la Vienne. MediaTek86 a pour rôle de :

- centraliser les prêts de livres, CD et DVD,
- développer une médiathèque numérique départementale,
- accompagner les médiathèques dans leur modernisation numérique.

L'un des besoins exprimés dans le marché remporté par ITS 86 concerne le développement d'une application interne permettant de gérer les ressources humaines de MediaTek86.

## 1.2 Présentation de la mission globale

Ma mission, en tant que technicienne développeur junior en simulation, est de concevoir et développer une application de bureau monoposte, destinée au service administratif de MediaTek86. Cette application sera installée sur un seul ordinateur et ne nécessitera pas de connexion réseau.

Les fonctionnalités attendues sont les suivantes :

- Gérer le personnel de chaque médiathèque (création, modification, suppression),
- Gérer leur affectation à un service (ex. : administratif, prêt, médiation culturelle),
- Gérer les absences du personnel (création, modification, suppression).

L'objectif est de fournir une solution logicielle fonctionnelle, claire et fiable, adaptée à un usage administratif local.

## 2. Etapes du projet

### 2.1 Étape 1 - Préparer l'environnement de travail et créer la base de données

#### Objectifs de l'étape

- Installer un environnement de développement complet (outils de conception, développement et gestion de base de données).
- Créer une base de données relationnelle à partir d'un schéma conceptuel.
- Alimenter la base de données avec des données de test.
- Sécuriser l'accès à la base de données.

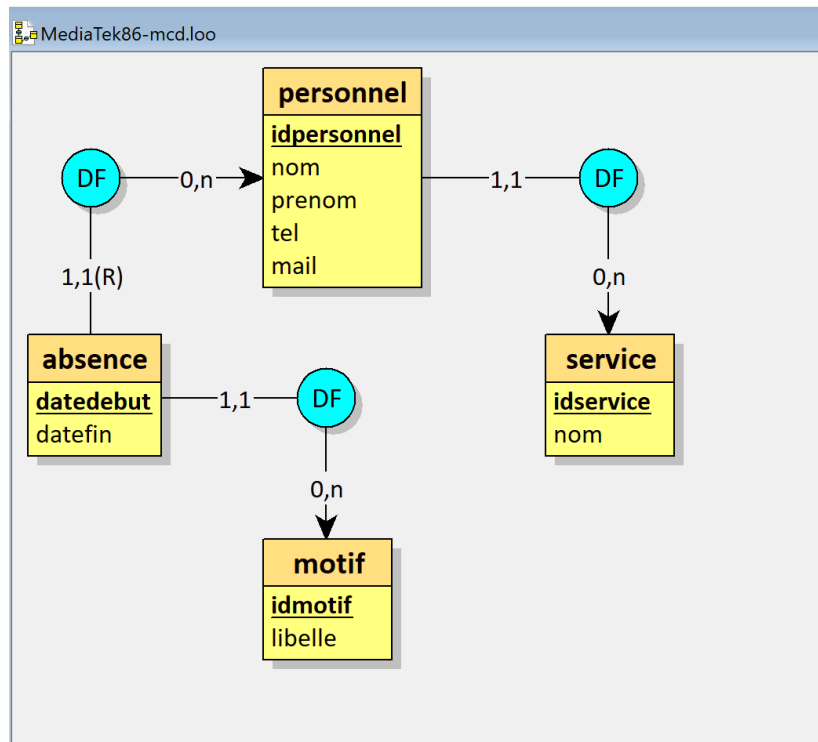
#### Mise en place de l'environnement de travail

##### Outils installés :

- **WampServer** pour gérer la base de données localement.
- **Visual Studio 2022 Entreprise** pour le développement de l'application.
- **Looping** pour visualiser et manipuler le schéma conceptuel de données.

#### Création de la base de données

J'ai récupéré le schéma conceptuel ci-dessous.



Puis via Looping, j'ai g n r  le script SQL au format MySQL correspondant. Je l'ai ensuite ex cut  via phpMyAdmin (inclus dans WampServer) pour g n rer la base de donn e que j'ai nomm  « mediatek86 ».

phpMyAdmin

Serveur courant : MySQL

R centes Pr f r es

Nouvelle base de donn es

- cfi
- chocolatein
- gestionprod
- habillations
- mediatek86
  - Nouvelle table
  - absence
  - motif
  - personnel
  - responsable
  - service
- test
- wordpress

Structure SQL Rechercher Requete Exporter Importer Op rations Privil ges Proc dures stock es  v nements Plus

Filtres

Contenant le mot :

Table	Action	Lignes	Type	Interclassement	Taille	Perte
absence	Parcourir Structure Rechercher Ins�rer Vider Supprimer	62	MyISAM	utf8mb4_unicode_ci	4,3 kio	114 o
motif	Parcourir Structure Rechercher Ins�rer Vider Supprimer	4	MyISAM	utf8mb4_unicode_ci	2,1 kio	-
personnel	Parcourir Structure Rechercher Ins�rer Vider Supprimer	15	MyISAM	utf8mb4_unicode_ci	4,1 kio	124 o
responsable	Parcourir Structure Rechercher Ins�rer Vider Supprimer	1	MyISAM	utf8mb4_unicode_ci	1,1 kio	-
service	Parcourir Structure Rechercher Ins�rer Vider Supprimer	3	MyISAM	utf8mb4_unicode_ci	2,1 kio	-
<b>5 tables</b>	<b>Somme</b>	<b>85</b>	<b>MyISAM</b>	<b>utf8mb4_unicode_ci</b>	<b>13,6 kio</b>	<b>238 o</b>

Tout cocher / V rifier les tables non optimis es Avec la s lection :

Imprimer Dictionnaire de donn es

Cr er une nouvelle table

Nom de table : Nombre de colonnes : 4 Cr er

Cr ation de l'utilisateur d'acc s   la base de donn es

Toujours sur phpMyAdmin, j'ai créé un utilisateur avec des droits limités à cette base uniquement :

```
CREATE USER 'mediatek86'@'%' IDENTIFIED BY 'mediatheque86Vienne';
```

```
GRANT USAGE ON *.* TO 'mediatek86'@'%';
```

```
GRANT ALL PRIVILEGES ON `mediatek86`.* TO 'mediatek86'@'%';
```

<input type="checkbox"/> mediatek86	%	spécifique à cette base de données	ALL PRIVILEGES	Non	<a href="#">Éditer les privilèges</a>	<a href="#">Exporter</a>
-------------------------------------	---	------------------------------------	----------------	-----	---------------------------------------	--------------------------

## Création et remplissage de la table 'responsable'

J'ai ajouté dans la base de données, une table 'responsable' sans identifiant qui ne contiendra qu'une ligne pour mémoriser le login et le mot de passe du responsable afin qu'il puisse se connecter à la future application. Cette table va contenir 2 champs : 'login' et 'pwd' de type varchar longueur 64.

J'ai alimenté la base de données avec un login et un pwd chiffré en utilisant la fonction SHA2 : SHA2("le pwd", 256) pour le hashage.

## Remplissage des tables 'motif' et 'service'

Pour la table 'motif', je la remplit des motifs suivants : vacances, maladie, motif familial, congé parental. Son contenu ne sera pas modifié dans l'application.

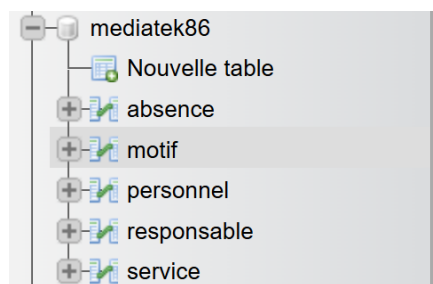
```
INSERT INTO motif (idmotif, libelle) VALUES
```

```
(1, 'vacances'),
```

```
(2, 'maladie'),
```

```
(3, 'motif familial'),
```

```
(4, 'congé parental');
```

						
	<input type="checkbox"/>	Éditer	Copier	Supprimer	1	vacances
	<input type="checkbox"/>	Éditer	Copier	Supprimer	2	maladie
	<input type="checkbox"/>	Éditer	Copier	Supprimer	3	motif familial
	<input type="checkbox"/>	Éditer	Copier	Supprimer	4	congé parental



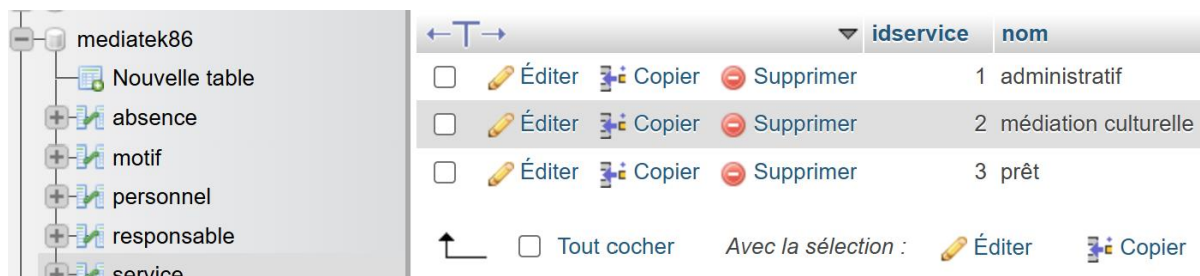
Pour la table 'service', je la remplie des services suivants : vacances, administratif, médiation culturelle, prêt. Son contenu ne sera pas modifié dans l'application.

```
INSERT INTO service (idservice, nom) VALUES
```

```
(1, 'administratif'),
```

```
(2, 'médiation culturelle'),
```

```
(3, 'prêt');
```



	idservice	nom
<input type="checkbox"/> Éditer Copier Supprimer	1	administratif
<input type="checkbox"/> Éditer Copier Supprimer	2	médiation culturelle
<input type="checkbox"/> Éditer Copier Supprimer	3	prêt

↑ ☐ Tout cocher Avec la sélection : Éditer Copier

### Remplissage des tables 'personnel' et 'absence'

Pour ces 2 tables, j'ai utilisé le site <https://www.generatedata.com/> pour générer des données de test c'est-à-dire différents exemples de personnels avec un id, un nom, un prénom, un numéro de téléphone et un mail et différents exemples d'absences avec un idpersonnel, une date de début, une date de fin et un idmotif.

RécentesPréférées

Nouvelle base de données

cfi

chocolatein

gestionprod

habilitations

mediatek86

Nouvelle table

absence

motif

personnel

responsable

service

test

wordpress

SELECT \* FROM `personnel`

Profilage [ Éditer en ligne ] [ Éditer ] [ Expliquer SQL ] [ Créer le code source PHP ] [ Actualiser ]

Tout afficherNombre de lignes : 25Filtrer les lignes: Chercher dans cette tableTrier par clé : Aucun(e)

Options supplémentaires

		idpersonnel	nom	prenom	tel	mail	idservice	
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	1	Velez	Jade	08 08 36 47 07	velez_jade747@yahoo.com	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	2	Mcbride	Maxime	03 79 04 97 84	maxime-mcbride9293@hotmail.fr	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	4	Matthews	Aurélie	09 33 38 71 70	m_aurlie@yahoo.fr	3
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	24	Pierui	Pierre	02 05 56 25 52	pierui pierre@gmail.com	1
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	6	Walls	Valerie	02 77 76 57 39	y-walls@hotmail.com	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	7	Haynes	Justine	06 08 11 58 88	haynes_justine2495@gmail.com	3
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	8	Riddle	Jeanne	04 37 24 94 89	jeanne-riddle9254@gmail.com	3
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	9	Gilbert	Clara	03 62 71 15 13	gilbert-clara@yahoo.com	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	10	Wade	Marion	07 12 55 20 59	marion_wade@yahoo.com	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	11	Mayo	Camille	03 94 72 67 18	m_camille@outlook.com	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	12	Gilbert	Manon	01 90 13 66 68	manon_gilbert989@orange.com	3
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	13	Brian	Morgane	03 65 14 22 11	morganebrian7370@outlook.com	1
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	14	Madden	Emma	02 12 91 36 86	m.emma136@yahoo.fr	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	15	Fowler	Sarah	03 97 88 46 78	s_fowler@yahoo.com	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	16	PAILLET	Dylan	0545522636	pailletdylan550@yahoo.fr	1

Tout cocherAvec la sélection : Éditer Copier Supprimer Exporter

Serveur courant : MySQL

RécentesPréférées

Nouvelle base de données

cfi

chocolatein

gestionprod

habilitations

mediatek86

Nouvelle table

absence

motif

personnel

responsable

service

test

wordpress

Options supplémentaires

		idpersonnel	datedebut	datefin	idmotif	
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	8	2025-03-27 06:13:45	2025-05-13 15:16:44	3
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	7	2024-09-27 07:52:06	2024-09-29 21:24:54	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	11	2024-12-05 23:52:29	2024-12-25 22:18:47	3
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	5	2025-02-14 07:55:29	2025-03-01 23:10:52	1
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	13	2025-03-05 15:05:43	2025-04-05 21:07:38	3
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	2	2024-05-22 10:15:27	2024-07-14 06:54:11	4
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	12	2025-02-07 23:14:06	2025-02-23 04:47:54	1
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	7	2024-11-07 05:59:21	2024-11-10 07:21:19	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	15	2024-10-06 23:27:30	2024-11-13 15:49:23	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	9	2024-08-23 18:36:11	2024-10-12 14:01:22	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	10	2024-08-29 09:49:47	2024-10-05 06:25:25	1
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	3	2024-10-17 08:30:46	2024-11-05 13:43:18	4
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	7	2025-03-26 06:39:39	2025-06-03 04:02:32	1
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	4	2025-04-25 01:02:22	2025-10-23 07:52:05	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	1	2024-11-19 20:46:11	2024-11-26 17:23:42	3
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	8	2024-05-05 21:43:48	2024-06-02 01:51:30	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	2	2024-06-08 11:20:35	2024-07-22 15:28:30	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	5	2024-12-17 07:29:59	2024-12-18 00:15:20	4
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	10	2024-12-04 13:36:54	2024-12-12 06:40:14	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	3	2025-01-17 12:03:16	2025-02-01 18:18:37	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	8	2024-08-29 01:29:45	2024-08-31 22:02:28	2
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	12	2024-07-19 21:11:55	2024-07-25 03:20:51	3
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	6	2024-07-23 21:14:18	2024-09-20 01:51:06	3
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	10	2025-05-27 15:08:18	2025-05-28 15:08:18	3
<input type="checkbox"/>	Éditer	<input type="checkbox"/>	9	2024-06-12 15:26:03	2024-08-04 10:54:36	2

## Bilan de l'étape

L'ensemble des objectifs fixés pour cette première étape a été atteint avec succès.

Tout d'abord, l'environnement de développement a été correctement installé et configuré. Les trois outils requis – WampServer, Visual Studio 2022 Entreprise et Looping – ont été installés sur le poste de travail, testés et sont pleinement opérationnels. Cela garantit des conditions de travail optimales pour la suite du projet, tant pour la conception que pour le développement et la gestion de la base de données.

Ensuite, la base de données relationnelle a été créée à partir d'un script SQL conforme au schéma conceptuel fourni. Cette base, nommée mediatek86, a été générée dans MySQL via phpMyAdmin, et son contenu respecte les structures attendues.

Un utilisateur spécifique a été créé pour accéder à cette base, avec des droits d'accès limités afin de renforcer la sécurité.

La table responsable, servant à la connexion de l'administrateur dans l'application, a été ajoutée manuellement. Elle a été alimentée avec un login et un mot de passe chiffré à l'aide de la fonction SHA2, assurant une première couche de sécurité des données sensibles.

Les tables motif et service ont été peuplées avec les données statiques demandées, conformes au contexte de l'application. Ces données ne sont pas modifiées par la suite, mais sont essentielles à son fonctionnement.

Enfin, des jeux de données de test ont été insérés dans les tables personnel et absence . Ces données ont été générées automatiquement via un outil en ligne, dans le but de simuler un contexte réaliste et de permettre les tests à venir.

Cette étape a permis de poser les fondations techniques du projet. L'environnement est stable, la base de données est fonctionnelle, sécurisée et alimentée. Toutes les conditions sont réunies pour aborder les prochaines phases de développement.

## 2.2 Étape 2 - dessiner les interfaces, structurer l'application en MVC, créer un dépôt, coder le visuel

### Objectif de l'étape

Cette étape vise à :

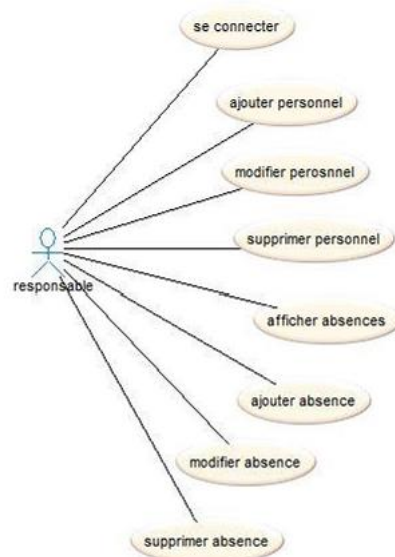
- Concevoir les interfaces utilisateur à partir des besoins fonctionnels exprimés.
- Structurer l'application en respectant l'architecture **MVC** (Modèle – Vue – Contrôleur).

- Mettre en place un **dépôt GitHub** pour assurer la gestion de version et la traçabilité du projet.
- Développer la **partie visuelle des interfaces** sans y intégrer la logique métier pour l'instant.

## Analyse documentaire

J'ai commencé par analyser le diagramme de cas d'utilisation et le descriptif fonctionnel fournis dans le dossier documentaire. Cette analyse m'a permis de mieux comprendre les fonctionnalités attendues, les rôles des utilisateurs, ainsi que les différentes interactions avec l'application.

### Diagramme de cas d'utilisation



Remarque : tous les cas d'utilisation ne sont accessibles qu'après s'être connecté.

## Maquettage des interfaces

À l'aide de l'outil de maquettage Pencil (ou autre, selon ce que tu as utilisé), j'ai conçu les interfaces principales de l'application, en m'appuyant sur les cas d'utilisation étudiés. Chaque écran a été conçu en respectant l'ergonomie et la logique de navigation.

Voici l'interface de connexion à l'application :

The screenshot shows a window titled "FrmAuthentification". Inside the window, there is a light beige rectangular area containing the following elements:

- A label "Login :" followed by a text input field.
- A label "Mot de passe :" followed by a text input field.
- A button labeled "Se connecter" positioned below the password field.

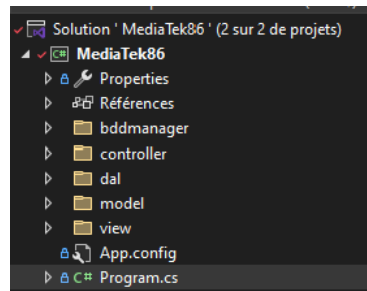
L'interface principale de l'application MediaTek86 :

The screenshot shows the main interface of the application, titled "FrmMediaTek86". It is divided into several sections:

- Les personnels**: A large grey rectangular area representing a list of personnel. Below it are four buttons: "Ajouter", "Modifier", "Supprimer", and "Absences".
- Ajouter un personnel**: A form with input fields for "Nom", "Prénom", "Mail", and "Tel", and a dropdown menu for "Service". At the bottom are "Enregistrer" and "Annuler" buttons.
- Les absences**: A large grey rectangular area representing a list of absences. Below it are three buttons: "Ajouter", "Modifier", and "Supprimer".
- Ajouter une absence**: A form with input fields for "Date début" (containing "20/05/2025 17:39"), "Date fin" (containing "21/05/2025 08:00"), and "Motif" (with a dropdown arrow). At the bottom are "Enregistrer" and "Annuler" buttons.

## Mise en place de la structure MVC

J'ai ensuite créé une nouvelle application dans l'IDE Visual Studio 2022, en respectant l'architecture MVC : package model, package view, package controller.



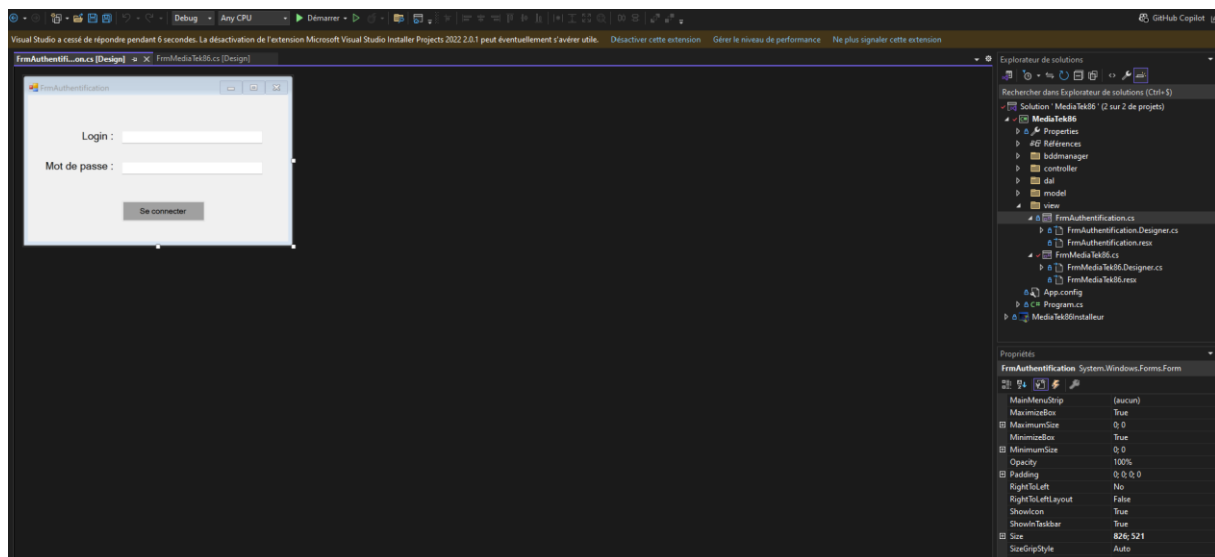
## Dépôt GitHub et Kanban

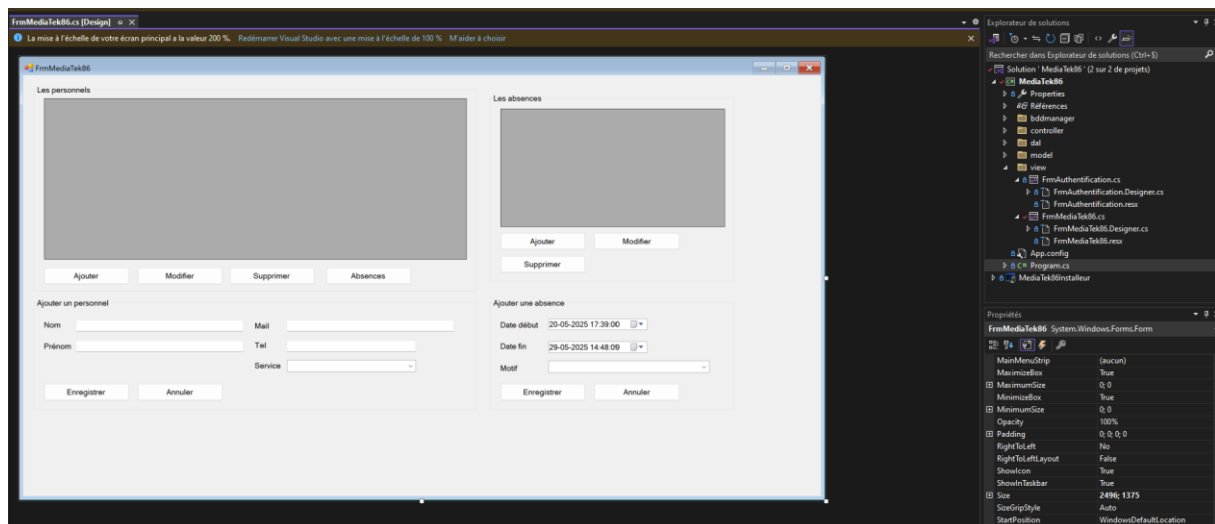
J'ai créé un dépôt distant GitHub pour versionner le projet. Une première sauvegarde a été réalisée dès la création du squelette du projet.

J'ai également créé un Kanban GitHub Project, dans lequel j'ai répertorié toutes les tâches à réaliser sous forme d'issues, regroupées dans des colonnes : *À faire*, *En cours*, *Terminée*. À chaque progression, j'ai mis à jour le statut de chaque tâche.

## Développement du visuel des interfaces (Vue)

Dans cette partie, je me suis uniquement occupée de la partie visuelle des interfaces, sans y intégrer la logique métier. J'ai créé 2 frames : une pour l'authentification de l'utilisateur (FrmAuthentification) et une autre pour la gestion du personnel (FrmMediaTek86). Ces interfaces ont été déplacé dans le package view.





J'ai également modifié Program.cs pour que l'interface apparaisse en première :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MediaTek86
{
    /// <summary>
    /// Application de gestion du personnel d'une médiathèque
    /// </summary>
    0 références | leaso, il y a 7 jours | 1 auteur, 1 modification
    internal class NamespaceDoc
    {
    }

    0 références | leaso, il y a 2 jours | 1 auteur, 4 modifications
    internal static class Program
    {
        /// <summary>
        /// Point d'entrée principal de l'application.
        /// </summary>
        [STAThread]
        0 références | leaso, il y a 2 jours | 1 auteur, 4 modifications
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new view.FrmAuthentication());
        }
    }
}
```

Puis j'ai fait une sauvegarde du travail sur le dépôt distant et mis à jour le kanban.

**Lien vers le dépôt GitHub :** [Phase 2 : Création de l'interface de connexion \(FrmAuthentication\) ... · leasourmail/MediaTek86@a73bb65](https://github.com/leasourmail/MediaTek86@73bb65)

## Bilan de l'étape

Tous les objectifs de cette étape ont été atteints :

- Les interfaces ont été dessinées à l'aide d'un outil de maquettage, en cohérence avec les besoins exprimés dans les cas d'utilisation.
- L'application a été créée et structurée selon le modèle MVC, facilitant la séparation des responsabilités et la maintenabilité du code.
- Un dépôt GitHub a été mis en place, avec un suivi via le système de project Kanban.
- La partie visuelle (Vue) des interfaces a été créé.

Cette étape constitue une base solide pour intégrer prochainement la logique métier et les traitements côté contrôleur et modèle.

## 2.3 Étape 3 - coder le modèle et les outils de connexion, générer la documentation technique

### Objectifs de l'étape

Cette étape avait pour but de :

- Mettre en œuvre la couche **modèle** de l'application (architecture MVC).
- Créer les **outils de connexion** à la base de données.
- Structurer les packages techniques (**bddmanager**, **dal**, **model**).
- Générer une **documentation technique** à l'aide de Visual Studio et SandCastle.
- Versionner l'avancement via **GitHub**, en maintenant le Kanban à jour.

### Configuration de l'environnement de développement

Avant de démarrer le codage, l'IDE Visual Studio a été configuré pour permettre l'accès à la base de données MySQL.

### Classe BddManager (connexion, requêtes)

J'ai créé un package bddmanager dans lequel j'ai intégré la classe 'BddManager' qui permet d'accéder à la base de données MySQL et d'exécuter les requêtes. Avec cette architecture, on isole la connexion (bddmanager) par rapport au reste de l'application.

Cette classe est construite selon le patron de conception Singleton qui permet de gérer une connexion unique à la base de données pendant l'exécution de l'application.



La classe BddManager a pour fonctions principales :

- Ouvrir une connexion unique à la base de données MySQL (singleton).
- Exécuter des requêtes de lecture (type SELECT) et de modification (INSERT, UPDATE, DELETE).
- Gérer les paramètres SQL de manière sécurisée avec MySqlParameter, limitant les risques d'injection SQL.
- Centraliser la préparation et l'exécution des commandes via MySqlCommand.

La classe BddManager intègre plusieurs fonctionnalités essentielles pour assurer la communication efficace et sécurisée avec la base de données MySQL.

La classe BddManager remplit trois fonctions principales :

- Connexion à la base de données : Une seule instance de connexion est créée grâce au design pattern Singleton. La méthode GetInstance prend une chaîne de connexion MySQL et établit la connexion une seule fois, assurant stabilité et performance.
- Exécution de requêtes de mise à jour : La méthode ReqUpdate permet d'exécuter des requêtes SQL de type INSERT, UPDATE ou DELETE. Les paramètres SQL sont gérés via un dictionnaire, garantissant la sécurité contre les injections.
- Exécution de requêtes de lecture : La méthode ReqSelect exécute les requêtes SELECT et retourne les résultats sous forme d'une liste de tableaux d'objets, chaque tableau représentant une ligne du résultat.

Voici un aperçu du constructeur privé qui initialise la connexion à la base :

```
/// <summary>
/// Constructeur pour créer la connexion à la BDD et l'ouvrir
/// </summary>
/// <param name="stringConnect">chaîne de connexion</param>
1 référence | leaso, il y a 7 jours | 1 auteur, 1 modification
private BddManager(string stringConnect)
{
    connection = new MySqlConnection(stringConnect);
    connection.Open();
}
```

L'appel à cette classe se fait via la méthode GetInstance, qui applique le pattern Singleton :

```

/// <summary>
/// Création d'une seule instance de la classe
/// </summary>
/// <param name="stringConnect">chaîne de connexion</param>
/// <returns>instance unique de la classe</returns>
1 référence | leaso, il y a 7 jours | 1 auteur, 1 modification
public static BddManager GetInstance(string stringConnect)
{
    if (instance == null)
    {
        instance = new BddManager(stringConnect);
    }
    return instance;
}

```

Pour exécuter des requêtes avec ou sans paramètres, deux méthodes distinctes ont été créées :

```

/// <summary>
/// Exécution d'une requête autre que "select"
/// </summary>
/// <param name="stringQuery">requête autre que select</param>
/// <param name="parameters">dictionnaire contenant les paramètres</param>
6 références | leaso, il y a 7 jours | 1 auteur, 1 modification
public void ReqUpdate(string stringQuery, Dictionary<string, object> parameters = null)
{
    MySqlCommand command = new MySqlCommand(stringQuery, connection);
    if (!(parameters is null))
    {
        foreach (KeyValuePair<string, object> parameter in parameters)
        {
            command.Parameters.Add(new MySqlParameter(parameter.Key, parameter.Value));
        }
    }
    command.Prepare();
    command.ExecuteNonQuery();
}

```

```

/// <summary>
/// Execution d'une requête de type "select"
/// </summary>
/// <param name="stringQuery">requête select</param>
/// <param name="parameters">dictionnaire contenant les paramètres</param>
/// <returns>liste de tableaux d'objets contenant les valeurs des colonnes</returns>
5 références | leaso, il y a 4 jours | 1 auteur, 2 modifications
public List<Object[]> ReqSelect(string stringQuery, Dictionary<string, object> parameters = null)
{
    MySqlCommand command = new MySqlCommand(stringQuery, connection);
    if (!(parameters is null))
    {
        foreach (KeyValuePair<string, object> parameter in parameters)
        {
            command.Parameters.Add(new MySqlParameter(parameter.Key, parameter.Value));
        }
    }
    try
    {
        if (parameters != null && parameters.Count > 0)
        {
            command.Prepare();
        }

        MySqlDataReader reader = command.ExecuteReader();
        int nbCols = reader.FieldCount;
        List<Object[]> records = new List<Object[]>();
        while (reader.Read())
        {
            Object[] attributs = new Object[nbCols];
            reader.GetValues(attributs);
            records.Add(attributs);
        }
        reader.Close();
        return records;
    }
    catch (Exception ex)
    {
        Console.WriteLine("Erreur dans ReqSelect : " + ex.Message);
        throw;
    }
}

```

## Classe Access (chaîne de connexion, Singleton)

Ensuite, j'ai créé le package dal (Data Access Layer) qui répond aux demandes du paquetage 'controller' et exploite 'bddmanager' en lui demandant d'exécuter des requêtes. Ainsi, le controller ne sait pas d'où viennent les données. Le paquetage dal fait l'intermédiaire en préparant des requêtes sql.

A ce stade, j'ai créé une première classe clé : Access. Cette classe est conçue comme un Singleton, garantissant une seule instance active durant toute l'exécution de l'application. A l'instanciation, elle initialise la connexion à la base de donnée via la classe 'BddManager' en lui transmettant une chaîne de connexion MySQL qui utilise la propriété connectionString définissant l'hôte, l'utilisateur, le mot de passe et le nom de la base de données. Et, en cas d'échec de la connexion, l'application se ferme immédiatement d'où le try/catch.

Une propriété publique Manager a été créée pour permettre d'accéder aux méthodes ReqSelect et ReqUpdate de BddManager. Cette classe va permettre aux autres classes de dal d'accéder facilement à la base via cette instance unique. Ainsi, elles vont récupérer toujours la même connexion unique à la BDD.

Cette architecture permet donc aux classes du package dal de réutiliser une seule et même connexion sécurisée, tout en respectant le principe de séparation des responsabilités (le contrôleur ne s'occupe pas directement de la logique d'accès aux données).

```

namespace MediaTek86.dal
{
    /// <summary>
    /// Singleton : classe d'accès à BddManager
    /// </summary>
    14 références | leaso, il y a 7 jours | 1 auteur, 2 modifications
    public class Access
    {
        /// <summary>
        /// chaîne de connexion à la bdd
        /// </summary>
        private static readonly string connectionString = "server=localhost;user id=mediatek86;password=mediatheque86Vienne;database=mediatek86;";
        /// <summary>
        /// instance unique de la classe
        /// </summary>
        private static Access instance = null;
        /// <summary>
        /// Getter sur l'objet d'accès aux données
        /// </summary>
        22 références | leaso, il y a 7 jours | 1 auteur, 1 modification
        public BddManager Manager { get; }

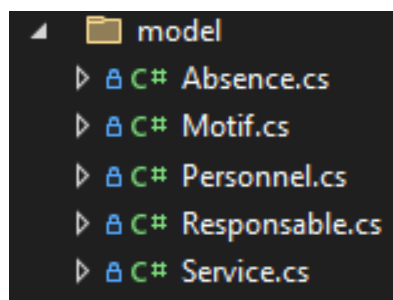
        /// <summary>
        /// Création unique de l'objet de type BddManager
        /// Arrête le programme si l'accès à la BDD a échoué
        /// </summary>
        1 référence | leaso, il y a 7 jours | 1 auteur, 1 modification
        private Access()
        {
            try
            {
                Manager = BddManager.GetInstance(connectionString);
            }
            catch (Exception)
            {
                Environment.Exit(0);
            }
        }

        /// <summary>
        /// Création d'une seule instance de la classe
        /// </summary>
        /// <returns></returns>
        5 références | leaso, il y a 7 jours | 1 auteur, 1 modification
        public static Access GetInstance()
        {
            if (instance == null)
            {
                instance = new Access();
            }
            return instance;
        }
    }
}

```

## Création des classes métiers (model)

Dans le package model, j'ai créé les classes métiers correspondants aux tables de la base de données : Responsable, Personnel, Absence, Motif, Service.



## Documentation technique

J'ai procédé à la génération automatique de documentation XML dans Visual Studio.

Puis j'ai utilisé le logiciel SandCastle pour obtenir une version HTML de la documentation technique.

Je les ai ensuite zippé dans un dossier, intégrer au projet et pousser la nouvelle version vers GitHub.

## Dépôt GitHub et Kanban

Tous les fichiers modifiés ont été commités. Le dépôt distant a été mis à jour :

[Phase 3 : ajout du model, des outils de connexion et de la documentat... · leasourmail/MediaTek86@c327d8c](#)

Le Kanban GitHub Project a également été mis à jour.

## Bilan de l'étape

Tous les objectifs prévus pour cette étape ont été atteints :

- IDE configuré pour MySQL : la connexion à la base a été testée avec succès depuis Visual Studio.
- Package bddmanager entièrement développé :  
J'ai implémenté une classe BddManager complète en Singleton, permettant :
  - o la gestion de la connexion à la base via une chaîne MySQL,
  - o l'exécution de requêtes SELECT (ReqSelect) avec récupération des résultats,
  - o l'exécution de requêtes de mise à jour (ReqUpdate) avec paramètres.
- Package dal structuré avec la classe Access : Toutes les classes métiers correspondant aux tables de la base ont été créées pour faciliter la manipulation des données.
- Documentation technique générée : XML + HTML zippée puis intégrée dans le dépôt vers GitHub.
- Dépôt distant et Kanban à jour

L'accès aux données est fiable, structuré, réutilisable, et la documentation est conforme. On peut passer au codage des fonctionnalités de l'application.

## 2.4 Étape 4 - coder les fonctionnalités de l'application à partir des cas d'utilisation

### Objectifs de l'étape

L'objectif principal de cette étape est de développer l'ensemble des fonctionnalités de l'application, en s'appuyant sur les cas d'utilisation définis. Chaque fonctionnalité doit être implémentée dans le respect du modèle MVC, testée, commentée et intégrée au dépôt distant. Il s'agit aussi de mettre à jour la documentation technique pour garantir la lisibilité et la maintenabilité du code.

### Développement des classes métiers (model)

Ce package représente les tables de BDD. Elles contiennent chacune les propriétés privées correspondant aux champs de leur table dans la BDD. Chaque classe est construite avec un constructeur qui valorise toutes ses propriétés. Les propriétés disposent de getters et de setters pour permettre l'accès contrôlé aux données, tout en assurant la modification uniquement là où c'est nécessaire. Par exemple, certaines propriétés comme les identifiants (Idpersonnel, Idmotif, etc.) sont en lecture seule pour préserver l'intégrité des données.

Dans les classes Service et Motif, j'ai également redéfini la méthode ToString() afin de retourner uniquement le libellé du motif ou le nom du service. Cela permet un affichage clair et direct dans les interfaces graphiques, notamment dans les menus déroulants ou les listes, où seul le nom est utile à l'utilisateur final.

Exemple :

```

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

namespace MediaTek86.model
{
    15 références | leaso, il y a 5 jours | 1 auteur, 2 modifications
    public class Service
    {
        5 références | leaso, il y a 5 jours | 1 auteur, 1 modification
        public int Idservice { get; }
        3 références | leaso, il y a 5 jours | 1 auteur, 1 modification
        public string Nom { get; }

        ///<summary>
        ///Valorise les propriétés
        ///<param name="idservice"></param>
        ///<param name="nom"></param>
        /// </summary>
        2 références | leaso, il y a 5 jours | 1 auteur, 1 modification
        public Service(int idservice, string nom)
        {
            this.Idservice = idservice;
            this.Nom = nom;
        }

        ///<summary>
        ///Définit l'information à afficher (juste le nom)
        /// </summary>
        /// <returns>nom du service</returns>

        0 références | leaso, il y a 5 jours | 1 auteur, 1 modification
        public override string ToString()
        {
            return this.Nom;
        }
    }
}

```

## Création et développement des autres classes dal

Comme dit précédemment, ce package fait le lien entre les classes métiers du model et la couche bddmanager, tout en masquant les détails d'accès à la base de données au reste de l'application.

Chaque classe métier possède désormais sa classe dédiée dans dal pour accéder aux données associées :

- PersonnelAccess
- AbsenceAccess
- ServiceAccess
- MotifAccess

La classe ResponsableAccess, bien que créée, n'est pas encore utilisée car elle interviendra plus tard, au moment de gérer l'authentification.

Toutes les classes d'accès sont construites sur le même modèle :

- Elles utilisent l'unique instance de la classe Access (singleton).
- Elles appellent les méthodes ReqSelect de BddManager pour interroger la base.
- Elles transforment les résultats des requêtes SQL en objets métier (ex. Personnel, Motif...).

```

namespace MediaTek86.dal
{
    /// <summary>
    /// Classe permettant de gérer les demandes concernant les absences
    /// </summary>
    3 références | leaso, il y a 4 jours | 1 auteur, 3 modifications
    public class AbsenceAccess
    {
        ///<summary>
        ///Instance unique de l'accès aux données
        /// </summary>
        private readonly Access access = null;

        ///<summary>
        ///Constructeur pour créer l'accès aux données
        /// </summary>
        1 référence | leaso, il y a 5 jours | 1 auteur, 1 modification
        public AbsenceAccess()
        {
            access = Access.GetInstance();
        }
    }
}

```

Ensuite, chacune a sa fonction propre :

- **GetLesPersonnels() dans PersonnelAccess** : récupère tous les personnels avec leur service associé, triés par nom et prénom.
- **GetLesAbsences dans AbsenceAccess** : récupère toutes les absences du personnel sélectionné, triées de la plus récente à la plus ancienne.
- **GetLesServices dans ServiceAccess** : retourne la liste complète des services pour notamment alimenter les menus déroulants dans l'IHM.
- **GetLesMotifs dans MotifAccess** : retourne tous les motifs d'absence existants pour notamment alimenter les menus déroulants dans l'IHM.

Dans chaque méthode, on utilise une List<T> (ex. List<Personnel>, List<Motif>, etc.) car :

- la base peut renvoyer plusieurs enregistrements (ex. plusieurs absences, services, motifs ou personnels) ;
- on souhaite les stocker en mémoire pour pouvoir ensuite les parcourir, ou les afficher dans l'interface ;
- la collection List<T> permet un accès souple et performant à ces objets avec des méthodes qui nous seront utiles quand on s'occupera des fonctionnalités d'ajout, suppression, et modification.



## Exemple : PersonnelAccess

```
///<summary>
///Récupère et retourne les personnels
///</summary>
///<returns>Liste des personnels</returns>
1 référence | lasso il y a 4 jours | 1 auteur, 2 modifications
public List<Personnel> GetLesPersonnels()
{
    List<Personnel> lesPersonnels = 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);
                    lesPersonnels.Add(personnel);
                }
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Environment.Exit(0);
        }
    }
    return lesPersonnels;
}
```

À ce stade, seules les fonctionnalités de lecture sont opérationnelles. Les fonctionnalités de création, suppression et modification d'un personnel ou d'une absence ne sont pas encore implémentées, elles seront ajoutées dans les étapes suivantes.

## Création et développement du contrôleur principal (FrmMediaTek86Controller)

Dans le respect du modèle MVC, la vue (interface graphique) ne doit pas interagir directement avec la base de données. C'est le rôle du contrôleur de faire l'intermédiaire entre la vue et la couche d'accès aux données (dal).

Pour cela, j'ai créé une nouvelle classe FrmMediaTek86Controller dans le package controller. Cette classe :

- reçoit les demandes de la vue (par exemple, afficher la liste du personnel ou des absences),
- appelle les méthodes appropriées du dal,
- retourne les données métiers prêtes à être affichées.

Il possède 4 objets privés : personnelAccess, serviceAccess, absenceAccess et motifAccess qui encapsulent les appels vers la base.

Dans son constructeur, ces objets sont instanciés une seule fois, en appelant leurs classes respectives.

```
/// <summary>
/// Contrôleur de FrmMediaTek86
/// </summary>
3 références | leaso, il y a 4 jours | 1 auteur, 4 modifications
public class FrmMediaTek86Controller
{
    /// <summary>
    /// objet d'accès aux opérations possibles sur Personnel
    /// </summary>
    private readonly PersonnelAccess personnelAccess;
    /// <summary>
    /// objet d'accès aux opérations possible sur Service
    /// </summary>
    private readonly ServiceAccess serviceAccess;

    ///<summary>
    ///objet d'accès aux opérations possibles sur Absence
    /// </summary>
    private readonly AbsenceAccess absenceAccess;

    ///<summary>
    ///objet d'accès aux opérations possible sur Motif
    /// </summary>
    private readonly MotifAccess motifAccess;

    /// <summary>
    /// Récupère les acces aux données
    /// </summary>
    1 référence | leaso, il y a 4 jours | 1 auteur, 2 modifications
    public FrmMediaTek86Controller()
    {
        personnelAccess = new PersonnelAccess();
        serviceAccess = new ServiceAccess();
        absenceAccess = new AbsenceAccess();
        motifAccess = new MotifAccess();
    }
}
```

Il expose ensuite 4 méthodes :

- GetLesPersonnels() : retourne la liste complète des personnels ;
- GetLesServices() : retourne la liste complète des services ;
- GetLesAbsences() : retourne la liste complète des absences ;
- GetLesMotifs() : retourne la liste complète des motifs.

```

    /// <summary>
    /// Récupère et retourne les infos des personnels
    /// </summary>
    /// <returns>liste des personnels</returns>
    2 références | leaso, il y a 5 jours | 1 auteur, 1 modification
    public List<Personnel> GetLesPersonnels()
    {
        return personnelAccess.GetLesPersonnels();
    }

    /// <summary>
    /// Récupère et retourne les infos des services
    /// </summary>
    /// <returns>liste des services</returns>
    1 référence | leaso, il y a 5 jours | 1 auteur, 1 modification
    public List<Service> GetLesServices()
    {
        return serviceAccess.GetLesServices();
    }

    ///<summary>
    ///Récupère et retourne les infos des absences
    /// </summary>
    2 références | leaso, il y a 4 jours | 1 auteur, 1 modification
    public List<Absence> GetLesAbsences(string nom, string prenom)
    {
        return absenceAccess.GetLesAbsences(nom, prenom);
    }

    ///<summary>
    ///Récupère et retourne les infos des motifs
    /// </summary>
    1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
    public List<Motif> GetLesMotifs()
    {
        return motifAccess.GetLesMotifs();
    }

```

J'ai sauvegarder toutes ces modifications sur mon **dépôt GitHub** :

[Phase 4 : Codage des classes du package model et des classes d'accès ... · leasourmail/MediaTek86@05cbf68](#)

## Ajout des fonctionnalités de gestion des personnels (affichage, ajout, modification, suppression)

Après avoir mis en place la récupération de la liste des personnels, j'ai intégré à la classe PersonnelAccess (dans le package dal) les trois nouvelles méthodes permettant de gérer dynamiquement les données dans la base :

- Méthode AddPersonnel(Personnel personnel) : Cette méthode permet d'ajouter un nouveau personnel dans la base. Elle crée une requête INSERT INTO avec des paramètres, et insère les données saisies dans les champs appropriés (nom, prenom, tel, mail, service).

```

    ///<summary>
    ///Demande d'ajouter d'un personnel
    /// </summary>
    /// <param name="personnel">objet personnel à ajouter</param>
    1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
    public void AddPersonnel (Personnel personnel)
    {
        if (access.Manager != null)
        {
            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.Service.Idservice);
            try
            {
                access.Manager.ReqUpdate(req, parameters);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Environment.Exit(0);
            }
        }
    }
}

```

- Méthode UpdatePersonnel(Personnel personnel): Elle sert à modifier un personnel existant, par exemple si un numéro de téléphone ou un service change. La requête SQL est un UPDATE avec les paramètres liés à l'objet Personnel passé en argument.

```

    ///<summary>
    ///Demande de modification d'un personnel
    /// </summary>
    /// <param name="personnel">object à modifier</param>
    1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
    public void UpdatePersonnel(Personnel personnel)
    {
        if (access.Manager != null)
        {
            string req = "update personnel set nom = @nom, prenom = @prenom, tel = @tel, mail = @mail, idservice = @idservice ";
            req += "where idpersonnel = @idpersonnel";
            Dictionary<string, Object> parameters = new Dictionary<string, object>();
            parameters.Add("@idpersonnel", personnel.Idpersonnel);
            parameters.Add("@nom", personnel.Nom);
            parameters.Add("@prenom", personnel.Prenom);
            parameters.Add("@tel", personnel.Tel);
            parameters.Add("@mail", personnel.Mail);
            parameters.Add("@idservice", personnel.Service.Idservice);
            try
            {
                access.Manager.ReqUpdate(req, parameters);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Environment.Exit(0);
            }
        }
    }
}

```

- Méthode DelPersonnel(Personnel personnel) : Elle supprime un personnel de la base via une requête DELETE. Seul l'identifiant du personnel est nécessaire ici.

```

    ///<summary>
    ///Demande de suppression d'un personnel
    /// </summary>
    /// <param name="personnel">objet personnel à supprimer</param>
    1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
    public void DelPersonnel(Personnel personnel)
    {
        if (access.Manager != null)
        {
            string req = "delete from personnel where idpersonnel = @idpersonnel;";
            Dictionary<string, object> parameters = new Dictionary<string, object>();
            parameters.Add("@idpersonnel", personnel.Idpersonnel);
            try
            {
                access.Manager.ReqUpdate(req, parameters);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Environment.Exit(0);
            }
        }
    }
}

```

Comme vu précédemment, le contrôleur agissant comme un intermédiaire entre la vue et la couche dal dans une architecture MVC, j'ai mis à jour la classe FrmmediaTek86Controller pour exposer les nouvelles méthodes de gestion du personnel ajoutées dans PersonnelAccess. Les méthodes ajoutées sont :

- AddPersonnel(Personnel personnel) : permet à la vue de demander l'ajout d'un nouveau personnel.
- UpdatePersonnel(Personnel personnel) : permet de modifier les données d'un personnel existant.
- DelPersonnel(Personnel personnel) : permet de supprimer un personnel.

```

    ///<summary>
    ///Demande de suppression d'un personnel
    /// </summary>
    /// <param name="personnel">objet personnel à supprimer</param>
    1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
    public void DelPersonnel(Personnel personnel)
    {
        personnelAccess.DelPersonnel(personnel);
    }

    ///<summary>
    ///Demande d'ajout d'un personnel
    /// </summary>
    /// <param name="personnel">objet personnel à ajouter</param>
    1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
    public void AddPersonnel(Personnel personnel)
    {
        personnelAccess.AddPersonnel(personnel);
    }

    ///<summary>
    ///Demande de modification d'un personnel
    /// </summary>
    /// <param name="personnel">objet personnel à modifier</param>
    1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
    public void UpdatePersonnel(Personnel personnel)
    {
        personnelAccess.UpdatePersonnel(personnel);
    }
}

```

Dans une architecture MVC, la vue est responsable de l'affichage des données et de la gestion des interactions utilisateur. J'ai donc développé l'interface FrmMediaTek86 afin qu'elle respecte les cas d'utilisation tout en déléguant les traitements métier au contrôleur.

La classe instancie un contrôleur (FrmMediaTek86Controller) pour communiquer avec la couche dal via ce dernier.

Elle utilise des BindingSource pour lier dynamiquement les données des personnels et services aux composants visuels (DataGridView et ComboBox).

On crée un booléen enCoursDeModifPersonnel qui permet de savoir si l'utilisateur est en train de modifier un personnel ou d'en ajouter un nouveau et donc selon le contexte, d'adapter l'interface.

```
/// <summary>
/// Modification d'affichage suivant si on est en cours de modif ou d'ajout d'un personnel
/// </summary>
/// <param name="modif"></param>
4 références | leaso, il y a 5 jours | 1 auteur, 1 modification
private void EnCoursDeModifPersonnel(Boolean modif)
{
    enCoursDeModifPersonnel = modif;
    grbLesPersonnels.Enabled = !modif;
    if (modif)
    {
        grbPersonnel.Text = "Modifier un personnel";
    }
    else
    {
        grbPersonnel.Text = "Ajouter un personnel";
        txtNom.Text = "";
        txtPrenom.Text = "";
        txtTel.Text = "";
        txtMail.Text = "";
    }
}
```

Au démarrage de l'interface, il faut remplir le DataGridView des personnels (en ne faisant pas apparaître l'idpersonnel) et le comboBox des services, on appelle donc les méthodes RemplirListePersonnels() et RemplirListeServices(). On désactive aussi les champs de saisie du groupe grbPersonnel.

```

/// <summary>
/// Initialisations : Création du controleur et remplissage des listes
/// </summary>
1 référence | leaso, il y a 5 jours | 1 auteur, 1 modification
private void Init()
{
    controller = new FrmMediaTek86Controller();
    RemplirListePersonnels();
    RemplirListeServices();
    EnCoursDeModifPersonnel(false);
}

```

Ensuite on code les autres fonctionnalités de sorte que quand l'utilisateur appuie sur « Ajouter », « Modifier » ou « suppression », l'action se lance :

#### ➔ Ajout d'un personnel :

L'utilisateur clique sur **“Ajouter”** → btnDemandeAjoutPers\_Click\_1() active les champs de saisie.

```

/// <summary>
/// Demande d'ajout d'un personnel
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | leaso, il y a 5 jours | 1 auteur, 1 modification
private void btnDemandeAjoutPers_Click_1(object sender, EventArgs e)
{
    grbPersonnel.Enabled = true;
}

```

Une fois le formulaire rempli, il clique sur **“Enregistrer”** → btnEnregPers\_Click\_1() :

- On vérifie que tous les champs sont remplis,
- On appelle la méthode controller.AddPersonnel() pour l'insérer dans la base,
- On actualise l'affichage et désactive le formulaire.

```

/// <summary>
/// Demande d'enregistrement de l'ajout ou de la modification d'un développeur
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | leaso, il y a 2 jours | 1 auteur, 3 modifications
private void btnEnregPers_Click_1(object sender, EventArgs e)
{
    if (!txtNom.Text.Equals("") && !txtPrenom.Text.Equals("") && !txtTel.Text.Equals("") && !txtMail.Text.Equals("") && cboService.SelectedIndex != -1)
    {
        string nom = txtNom.Text;
        string prenom = txtPrenom.Text;
        string tel = txtTel.Text;
        string mail = txtMail.Text;
        Service service = (Service)bdgServices.List[bdgServices.Position];

        // Déterminer le personnel à exclure en cas de modification
        Personnel personnelExclu = enCoursDeModifPersonnel ? (Personnel)dgvPersonnels.SelectedRows[0].DataBoundItem : null;

        // Vérifie s'il existe déjà un personnel avec le même nom, prénom et service (sans tenir compte de celui qu'on modifie)
        if (PersonnelDejaExistant(nom, prenom, service, personnelExclu))
        {
            MessageBox.Show("Un personnel avec ce nom, prénom et service existe déjà.", "Doublon détecté");
            return;
        }

        if (enCoursDeModifPersonnel)
        {
            // Confirmation d'enregistrement
            DialogResult result = MessageBox.Show("Voulez-vous enregistrer ces modifications ?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

            if (result != DialogResult.Yes)
            {
                return; // L'utilisateur a cliqué sur Non et on annule l'enregistrement
            }

            Personnel personnel = (Personnel)dgvPersonnels.SelectedRows[0].DataBoundItem;
            personnel.Nom = nom;
            personnel.Prenom = prenom;
            personnel.Tel = tel;
            personnel.Mail = mail;
            personnel.Service = service;
            controller.UpdatePersonnel(personnel);
        }
        else
        {
            Personnel personnel = new Personnel(0, txtNom.Text, txtPrenom.Text, txtTel.Text, txtMail.Text, service);
            controller.AddPersonnel(personnel);
        }

        RemplirListePersonnels();
        enCoursDeModifPersonnel(false);
        grbPersonnel.Enabled = false;
    }
    else
    {
        MessageBox.Show("Tous les champs doivent être remplis", "Information");
    }
}

```

## ➔ Modification d'un personnel

L'utilisateur sélectionne un personnel et clique sur **“Modifier”** → btnDemandeModifPers\_Click\_1() :

- On active les champs de saisie,
- On préremplit les champs avec les données du personnel sélectionné,
- On passe en mode modification (enCoursDeModifPersonnel = true).



```
/// <summary>
/// Demande de modification d'un personnel
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>

1 référence | leaso, il y a 5 jours | 1 auteur, 1 modification
private void btnDemandeModifPers_Click_1(object sender, EventArgs e)
{
    if (dgvPersonnels.SelectedRows.Count > 0)
    {
        grbPersonnel.Enabled = true;
        EnCoursDeModifPersonnel(true);
        Personnel personnel = (Personnel)dgvPersonnels.SelectedRows[0].DataBoundItem;
        txtNom.Text = personnel.Nom;
        txtPrenom.Text = personnel.Prenom;
        txtTel.Text = personnel.Tel;
        txtMail.Text = personnel.Mail;
        cboService.SelectedIndex = cboService.FindStringExact(personnel.Service.Nom);
    }
    else
    {
        MessageBox.Show("Une ligne doit être sélectionnée.", "Information");
    }
}
```

Lors de l'enregistrement :

- Les modifications sont appliquées à l'objet existant,
- controller.UpdatePersonnel() est appelé,
- La liste est rechargée.

### ➔ Suppression d'un personnel

L'utilisateur sélectionne un personnel et clique sur **“Supprimer”** → btnDemandeSupprPers\_Click() :

- une confirmation est affichée,
- en cas de validation, la suppression est exécutée via controller.DelPersonnel(),
- la liste est actualisée.

```

/// <summary>
/// Demande de suppression d'un personnel
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | leaso, il y a 4 jours | auteur, 1 modification
private void btnDemandeSupprPers_Click(object sender, EventArgs e)
{
    if (dgvPersonnels.SelectedRows.Count > 0)
    {
        Personnel personnel = (Personnel)dgvPersonnels.SelectedRows[0].DataBoundItem;
        if (MessageBox.Show("Voulez-vous vraiment supprimer " + personnel.Nom + " " + personnel.Prenom + " ?", "Confirmation de suppression", MessageBoxButtons.YesNo) == DialogResult.Yes)
        {
            controller.DelPersonnel(personnel);
            RemplirListePersonnels();
        }
    }
    else
    {
        MessageBox.Show("Une ligne doit être sélectionnée.", "Information");
    }
}

```

## ➔ Annulation d'une opération

Si l'utilisateur clique sur **"Annuler"**, la méthode btnAnnulPers\_Click\_1() réinitialise les champs, annule le mode modification et désactive la saisie.

```

/// <summary>
/// Annule la demande d'ajout ou de modification d'un personnel
/// </summary>
/// Vide les zones de saisie du personnel
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | leaso, il y a 4 jours | auteur, 2 modifications
private void btnAnnulPers_Click_1(object sender, EventArgs e)
{
    if (MessageBox.Show("Voulez-vous vraiment annuler ?", "Confirmation", MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
        EnCoursDeModifPersonnel(false);
        grbPersonnel.Enabled = false;
    }
}

```

Après le développement complet de la chaîne vue → contrôleur → DAL → base de données pour la gestion des personnels, j'ai réalisé une série de tests manuels afin de m'assurer que chaque fonctionnalité d'ajout, de modification, de suppression des personnels fonctionne conformément aux cas d'utilisation définis dans le dossier documentaire.

## Fonctionnalités testées :

- **Ajout d'un personnel :**
  - Champ obligatoire non rempli → message bloquant.
  - Données valides et confirmation → ajout correct à la base.
- **Modification d'un personnel :**
  - Sélection correcte du personnel.
  - Pré-remplissage des champs.

- Si données valides et confirmation -> Enregistrement des modifications avec mise à jour visible immédiate.
- **Suppression d'un personnel :**
  - Sélection d'une ligne.
  - Affichage d'un message de confirmation.
  - Suppression effective après validation.

Chaque fonctionnalité a été testée dans différentes conditions (valeurs manquantes, doublons, annulation d'action), et toutes les actions ont été exécutées correctement sans erreurs bloquantes. Les données dans la base ont bien été mises à jour en conséquence.

J'ai sauvegardé les nouvelles modifications sur **GitHub** :

[Phase 5 : Implémentation de la gestion des personnels · leasourmail/MediaTek86@2f3f79a](#)

### Ajout des fonctionnalités de gestion des absences (affichage, ajout, modification, suppression)

Après avoir mis en place la récupération de la liste des absences, j'ai intégré à la classe `AbsenceAccess` (dans le package `dal`) les trois nouvelles méthodes permettant de gérer dynamiquement les données dans la base :

- Méthode `AddAbsence(Absence absence)` : Cette méthode permet d'ajouter une nouvelle absence dans la base. Elle crée une requête `INSERT INTO` avec des paramètres, et insère les données saisies dans les champs appropriés (`datedebut`, `datefin`, `motif`).

```

///<summary>
///Demande d'ajout une absence
/// </summary>
/// <param name="absence">objet absence à ajouter</param>
1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
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);
        }
    }
}

```

- Méthode UpdateAbsence(Absence absence) : Elle sert à modifier une absence existante, par exemple si la date de début ou de fin n'est pas bonne ou le motif a changé. La requête SQL est un UPDATE avec les paramètres liés à l'objet Absence passé en argument.

```

///<summary>
///Demande de modification d'une absence (avec date de début modifiable)
/// </summary>
/// <param name="ancienneAbsence">objet ancienne absence</param>
/// <param name="nouvelleAbsence">objet nouvelle absence</param>
1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
public void UpdateAbsence (Absence ancienneAbsence, Absence nouvelleAbsence)
{
    if (access.Manager != null)
    {
        string req = "update absence set datedebut = @nouvelleDatedebut, datefin = @Datefin, idmotif = @idmotif ";
        req += "where idpersonnel = @idpersonnel AND datedebut = @ancienneDatedebut";
        Dictionary<string, object> parameters = new Dictionary<string, object>();
        parameters.Add("@idpersonnel", ancienneAbsence.Idpersonnel);
        parameters.Add("@ancienneDatedebut", ancienneAbsence.Datedebut);
        parameters.Add("@nouvelleDatedebut", nouvelleAbsence.Datedebut);
        parameters.Add("@datefin", nouvelleAbsence.Datefin);
        parameters.Add("@idmotif", nouvelleAbsence.Motif.Idmotif);
        try
        {
            access.Manager.ReqUpdate(req, parameters);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Environment.Exit(0);
        }
    }
}

```

- Méthode DelAbsence(Absence absence) : Elle supprime l'absence sélectionnée de la base via une requête DELETE.

```

///<summary>
///Demande de suppression d'une absence
///</summary>
///<param name="absence">objet absence à supprimer</param>
1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
public void DelAbsence(Absence absence)
{
    if (access.Manager != null)
    {
        string req = "delete from absence where idpersonnel = @idpersonnel AND datedebut = @datedebut AND datefin = @datefin AND idmotif = @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);
        }
    }
}

```

Comme vu précédemment, le contrôleur agissant comme un intermédiaire entre la vue et la couche dal dans une architecture MVC, j'ai mis à jour la classe FrmmediaTek86Controller pour exposer les nouvelles méthodes de gestion d'une absence ajoutées dans AbsenceAccess. Les méthodes ajoutées sont :

- AddAbsence(Absence absence) : permet à la vue de demander l'ajout d'une nouvelle absence.
- UpdateAbsence(Absence absence) : permet de modifier les données d'une absence existante.
- DelAbsence (Absence absence) : permet de supprimer une absence.

```

///<summary>
///Demande de suppression d'une absence
///</summary>
///<param name="absence">objet absence à supprimer</param>
1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
public void DelAbsence(Absence absence)
{
    absenceAccess.DelAbsence(absence);
}

///<summary>
///Demande d'ajout une absence
///</summary>
///<param name="absence">objet absence à ajouter</param>
1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
public void AddAbsence (Absence absence)
{
    absenceAccess.AddAbsence(absence);
}

///<summary>
///Demande de modifier une absence
///</summary>
///<param name="ancienneAbsence">objet ancienne absence</param>
///<param name="nouvelleAbsence">objet nouvelle absence</param>
1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
public void UpdateAbsence (Absence ancienneAbsence, Absence nouvelleAbsence )
{
    absenceAccess.UpdateAbsence(ancienneAbsence, nouvelleAbsence);
}

```

Dans une architecture MVC, la vue est responsable de l'affichage des données et de la gestion des interactions utilisateur. J'ai donc continué à développer l'interface FrmMediaTek86 afin qu'elle intègre les nouvelles fonctionnalités des boutons ajouter, supprimer et modifier une absence et qu'elle permette leurs affiches dans le DataGridView du GroupBox 'grbLesAbsences'.

Sur la même logique que pour la gestion des personnels, cette classe utilise des BindingSource pour lier dynamiquement les données des absences et motifs aux composants visuels (DataGridView et ComboBox).

On crée un booléen enCoursDeModifAbsence qui permet de savoir si l'utilisateur est en train de modifier une absence ou d'en ajouter une nouvelle et donc selon le contexte, d'adapter l'interface.

```
/// <summary>
/// Modification d'affichage suivant si on est en cours de modif ou d'ajout d'une absence
/// </summary>
/// <param name="modif"></param>
4 références | leaso, il y a 4 jours | 1 auteur, 1 modification
private void EnCoursDeModifAbsence(Boolean modif)
{
    enCoursDeModifAbsence = modif;
    grbLesAbsences.Enabled = !modif;
    if (modif)
    {
        grbAbsence.Text = "Modifier une absence";
    }
    else
    {
        grbAbsence.Text = "Ajouter une absence";
        dtpDateDebut.Value = DateTime.Now;
        dtpDateFin.Value = DateTime.Now;
    }
}
```

Lorsque l'utilisateur va cliquer sur le bouton 'Absences', il faut que le DataGridView se remplissent avec les absences du personnel sélectionné (en ne faisant pas apparaitre l'idpersonnel) et que le comboBox des motifs se remplissent également. On appelle donc les méthodes RemplirListeAbsences() et RemplirListeMotifs(). On désactive aussi les champs du groupe grbAbsence.

```

/// <summary>
/// Demande de gérer les absences
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
private void btnDemandeAbsences_Click(object sender, EventArgs e)
{
    grbLesAbsences.Enabled = true;
    RemplirListeAbsences();
    dgvAbsences.Enabled = true;
    EnCoursDeModifAbsence(false);
}

/// <summary>
/// Afficher les absences
/// </summary>
3 références | leaso, il y a 4 jours | 1 auteur, 1 modification
private void RemplirListeAbsences()
{
    if (dgvPersonnels.SelectedRows.Count == 0)
    {
        MessageBox.Show("Veuillez sélectionner un personnel d'abord.", "Information");
        return;
    }

    Personnel personnel = (Personnel)dgvPersonnels.SelectedRows[0].DataBoundItem;
    List<Absence> lesAbsences = controller.GetLesAbsences(personnel.Nom, personnel.Prenom);
    bdgAbsences.DataSource = lesAbsences;
    dgvAbsences.DataSource = bdgAbsences;
    dgvAbsences.Columns["idpersonnel"].Visible = false;
    dgvAbsences.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
    dgvAbsences.Columns["Idpersonnel"].Visible = false;
}

```

Ensuite on code les autres fonctionnalités de sorte que quand l'utilisateur appuie sur « Ajouter », « Modifier » ou « suppression », l'action se lance :

### ➔ Ajout d'une absence :

L'utilisateur clique sur **“Ajouter”** → btnDemandeAjoutAbs\_Click() active les champs de saisie.

```

/// <summary>
/// Demande d'ajouter une absence
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
private void btnDemandeAjoutAbs_Click(object sender, EventArgs e)
{
    RemplirListeMotifs();
    grbAbsence.Enabled = true;
}

```

Une fois le formulaire rempli, il clique sur **“Enregistrer”** → btnEnregAbs\_Click() :

- On vérifie que la date de fin est postérieure à la date de début sinon un message bloquant s’affiche.
- On vérifie que tous les champs sont remplis sinon un message bloquant s’affiche.
- On vérifie également qu’il n’existe pas déjà une absence sur cette période (en excluant celle qu’on veut rajouter du test). Pour cela, on a dû créer une méthode `AbsenceDejaExistante()` qui compare la période saisie avec les absences déjà enregistrées pour ce personnel.
- On appelle la méthode `controller.AddAbsence()` pour l’insérer dans la base,
- On actualise l’affichage et désactive le formulaire.

```

/// <summary>
/// Demande d'enregistrement de l'ajout ou de la modification d'un développeur
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | 1 leuzy, il y a 2 jours | 1 auteur, 2 modifications
private void btnEnregAbs_Click(object sender, EventArgs e)
{
    if (dtpDateDebut.Value <= dtpDateFin.Value)
    {
        if (cboMotif.SelectedIndex != -1 && cboMotif.SelectedItem != null)
        {
            Motif motif = (Motif)cboMotif.SelectedItem;
            Personnel personnel = (Personnel)dgvPersonnels.SelectedRows[0].DataBoundItem;
            Absence nouvelleAbsence = new Absence(personnel.Idpersonnel, dtpDateDebut.Value, dtpDateFin.Value, motif);

            // Vérifie conflit
            if (AbsenceDejaExistante(personnel, dtpDateDebut.Value, dtpDateFin.Value))
            {
                MessageBox.Show("Une absence existe déjà pendant cette période.", "Information");
                return;
            }

            if (enCoursDeModifAbsence && ancienneAbsence != null)
            {
                // Confirmation d'enregistrement
                DialogResult result = MessageBox.Show("Voulez-vous enregistrer ces modifications ?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

                if (result != DialogResult.Yes)
                {
                    return; // L'utilisateur a cliqué sur Non et on annule l'enregistrement
                }

                controller.UpdateAbsence(ancienneAbsence, nouvelleAbsence);
            }
            else
            {
                controller.AddAbsence(nouvelleAbsence);
            }

            RemplirListeAbsences();
            EnCoursDeModifAbsence(false);
            ancienneAbsence = null;
        }
        else
        {
            MessageBox.Show("Tous les champs doivent être remplis", "Information");
        }
        grbAbsence.Enabled = false;
    }
    else
    {
        MessageBox.Show("La date de fin doit être postérieure à la date de début", "Information");
    }
}

```

## ➔ Modification d’une absence

L'utilisateur sélectionne une absence et clique sur **“Modifier”** ➔ `btnDemandeModifAbs_Click()` :

- On active les champs de saisie,
- On préremplit les champs avec les données de l’absence sélectionnée,
- On passe en mode modification (`enCoursDeModifAbsence = true`).



```

/// <summary>
/// Demande de modifier une absence
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
private void btnDemandeModifAbs_Click(object sender, EventArgs e)
{
    if (dgvAbsences.SelectedRows.Count > 0)
    {
        RemplirListeMotifs();
        grbAbsence.Enabled = true;
        EnCoursDeModifAbsence(true);
        ancienneAbsence = (Absence)dgvAbsences.SelectedRows[0].DataBoundItem;

        //Remplissage des champs du formulaire avec les valeurs existantes
        dtpDateDebut.Value = ancienneAbsence.Datedebut;
        dtpDateFin.Value = ancienneAbsence.Datefin;
        cboMotif.SelectedIndex = cboMotif.FindStringExact(ancienneAbsence.Motif.Libelle);
    }
    else
    {
        MessageBox.Show("Une ligne doit être sélectionnée.", "Information");
    }
}

```

Lors de l'enregistrement :

- On vérifie que la date de fin est toujours postérieure à la date de début sinon un message bloquant s'affiche.
- On vérifie que tous les champs sont remplis sinon un message bloquant s'affiche.
- On vérifie également qu'il n'existe pas déjà une absence sur cette période (en excluant celle qu'on modifie du test). Pour cela, on utilise, comme lors de l'ajout d'une absence, la méthode AbsenceDejaExistante() qui compare la période saisie avec les absences déjà enregistrées pour ce personnel.
- Les modifications sont appliquées à l'objet existant,
- controller.UpdateAbsence() est appelé,
- La liste est rechargée.

### ➔ Suppression d'une absence

L'utilisateur sélectionne une absence et clique sur **“Supprimer”** → btnDemandeSupprAbs\_Click () :

- une confirmation est affichée,
- en cas de validation, la suppression est exécutée via controller.DelAbsence(),

- la liste est actualisée.

```

/// <summary>
/// Demande de supprimer une absence
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
/// 1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
private void btnDemandeSupprAbs_Click(object sender, EventArgs e)
{
    if (dgvAbsences.SelectedRows.Count > 0)
    {
        Absence absence = (Absence)dgvAbsences.SelectedRows[0].DataBoundItem;
        if (MessageBox.Show("Voulez-vous vraiment supprimer l'absence du " + dtpDateDebut.Value + " au " + dtpDateFin.Value + " ?", "Confirmation de suppression", MessageBoxButtons.YesNo) == DialogResult.Yes)
        {
            controller.DeleteAbsence(absence);
            ReafficherListeAbsences();
        }
    }
    else
    {
        MessageBox.Show("Une ligne doit être sélectionnée.", "Information");
    }
}

```

## ➔ Annulation d'une opération

Si l'utilisateur clique sur **“Annuler”**, la méthode btnAnnulAbs\_Click() réinitialise les champs, annule le mode modification et désactive la saisie.

```

/// <summary>
/// Annule la demande d'ajout ou de modification d'une absence
/// Vide les zones de saisie de l'absence
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
/// 1 référence | leaso, il y a 4 jours | 1 auteur, 1 modification
private void btnAnnulAbs_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Voulez-vous vraiment annuler ?", "Confirmation", MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
        EnCoursDeModifAbsence(false);
        grbAbsence.Enabled = false;
    }
}

```

Après le développement complet de la chaîne vue → contrôleur → DAL → base de données pour la gestion des absences, j'ai réalisé une série de tests manuels afin de m'assurer que l'affichage des absences et que chaque fonctionnalité d'ajout, de modification, de suppression des absences fonctionnent conformément aux cas d'utilisation définis dans le dossier documentaire.

## Fonctionnalités testées :

- **Affichage des absences d'un personnel présélectionné :**
  - Si pas de personnel sélectionnée -> message bloquant
  - Sélection valide -> Affichage de la liste des absences du personnel concernés
- **Ajout d'une absence :**

- Champ obligatoire non rempli → message bloquant.
- Données valides et confirmation → ajout correct à la base.
- Doublon (chevauchement de période) → message d'erreur.
- **Modification d'une absence :**
  - Sélection correcte de l'absence.
  - Pré-remplissage des champs.
  - Si modification entraîne chevauchement d'une absence déjà existante -> message bloquant.
  - Si données valide et si confirmation -> enregistrement des modifications avec mise à jour visible immédiate.
- **Suppression d'une absence :**
  - Sélection d'une ligne.
  - Affichage d'un message de confirmation.
  - Suppression effective après validation.

Chaque fonctionnalité a été testée dans différentes conditions (valeurs manquantes, chevauchement de date, annulation d'action), et toutes les actions ont été exécutées correctement sans erreurs bloquantes. Les données dans la base ont bien été mises à jour en conséquence.

### Vérification anti-doublon sur ajout/modification de personnel (nom, prenom, service)

Cette vérification n'a pas été expressément demandée par le cahier des charges mais en découle logiquement. En effet, il faut rendre impossible l'ajout ou la modification d'un personnel avec le même nom, prénom et service qu'un personnel existant. Je rajoute service dans les paramètres car il doit être possible qu'un même personnel appartienne à 2 services.

Je crée donc un booléen `PersonnelDejaExistant` avec les paramètres `nom`, `prenom`, `service`. Ce boolean ne tient pas compte du personnel sélectionné si on est dans le cadre d'une modification.

```

/// <summary>
/// Vérifie si un personnel avec le même nom, prénom et service existe déjà (sans tenir compte du personnel sélectionné si modification)
/// </summary>
/// <param name="nom"></param>
/// <param name="prenom"></param>
/// <param name="service"></param>
/// <param name="personnelModifie"></param>
/// <returns></returns>
1 référence | l'asso, il y a 4 jours | 1 auteur, 1 modification
private bool PersonnelDejaExistant(string nom, string prenom, Service service, Personnel personnelModifie = null)
{
    List<Personnel> personnels = controller.GetLesPersonnels();

    foreach (Personnel p in personnels)
    {
        // Ne pas comparer avec lui-même en cas de modification
        if (personnelModifie != null && p.Idpersonnel == personnelModifie.Idpersonnel)
            continue;

        if (p.Nom.Equals(nom, StringComparison.InvariantCultureIgnoreCase) &&
            p.Prenom.Equals(prenom, StringComparison.InvariantCultureIgnoreCase) &&
            p.Service.Idservice == service.Idservice)
        {
            return true;
        }
    }

    return false;
}

```

Je fais ensuite appel à cette méthode lorsque j'enregistre mon ajout ou ma modification d'absence.

```

/// <summary>
/// Demande d'enregistrement de l'ajout ou de la modification d'un développeur
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | l'asso, il y a 2 jours | 1 auteur, 3 modifications
private void btnEnregPers_Click_1(object sender, EventArgs e)
{
    if (!txtNom.Text.Equals("") && !txtPrenom.Text.Equals("") && !txtTel.Text.Equals("") && !txtMail.Text.Equals("") && cboService.SelectedIndex != -1)
    {
        string nom = txtNom.Text;
        string prenom = txtPrenom.Text;
        string tel = txtTel.Text;
        string mail = txtMail.Text;
        Service service = (Service)bdgServices.List[bdgServices.Position];

        // Déterminer le personnel à exclure en cas de modification
        Personnel personnelExclu = enCoursDeModifPersonnel ? (Personnel)dgvPersonnels.SelectedRows[0].DataBoundItem : null;

        // Vérifie s'il existe déjà un personnel avec le même nom, prénom et service (sans tenir compte de celui qu'on modifie)
        if (PersonnelDejaExistant(nom, prenom, service, personnelExclu))
        {
            MessageBox.Show("Un personnel avec ce nom, prénom et service existe déjà.", "Doublon détecté");
            return;
        }

        if (enCoursDeModifPersonnel)
        {
            // Confirmation d'enregistrement
            DialogResult result = MessageBox.Show("Voulez-vous enregistrer ces modifications ?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

            if (result != DialogResult.Yes)
            {
                return; // L'utilisateur a cliqué sur Non et on annule l'enregistrement
            }

            Personnel personnel = (Personnel)dgvPersonnels.SelectedRows[0].DataBoundItem;
            personnel.Nom = nom;
            personnel.Prenom = prenom;
            personnel.Tel = tel;
            personnel.Mail = mail;
            personnel.Service = service;
            controller.UpdatePersonnel(personnel);
        }
        else
        {
            Personnel personnel = new Personnel(0, txtNom.Text, txtPrenom.Text, txtTel.Text, txtMail.Text, service);
            controller.AddPersonnel(personnel);
        }

        RemplirListePersonnels();
        enCoursDeModifPersonnel(false);
        grbPersonnel.Enabled = false;
    }
    else
    {
        MessageBox.Show("Tous les champs doivent être remplis", "Information");
    }
}

```

J'ai ensuite testé de rajouter ou modifier un personnel en rajoutant le même nom, prénom et service qu'un personnel déjà existant et j'ai bien un message bloquant à l'enregistrement. J'ai également vérifié que ça ne prenait pas en compte le personnel en cours de modification.

J'ai sauvegardé toutes ces nouvelles modifications sur **GitHub** :

[Phase 6 : Gestion des absences + validation anti-doublon absence et p... · leasourmail/MediaTek86@f1305f7](#)

### Authentification (login sécurisé, vue dédiée)

Après avoir terminé les fonctionnalités principales de gestion des personnels, j'ai mis en place le système d'authentification afin de sécuriser l'accès à l'application. Cette partie a été développée dans le respect du modèle MVC, avec trois composants distincts :

- ResponsableAccess dans le package dal
- FrmAuthentificationController dans le package controller
- FrmAuthentification dans le package view

**À noter :** La classe Responsable, utilisée pour encapsuler le login et le mot de passe saisis, avait déjà été codée précédemment dans le package model. Elle contient deux propriétés (Login, Pwd) définies via le constructeur, et sert ici de support aux échanges de données.

Dans la classe ResponsableAccess (dal), j'ai créé la méthode ControleAuthentification(Responsable responsable), qui vérifie dans la base de données si le login et le mot de passe (chiffré avec SHA2) correspondent à une entrée valide.

- La requête utilise un dictionnaire de paramètres pour sécuriser les données.
- Si une ligne est trouvée, la méthode retourne true, sinon false.

```

/// <summary>
/// Controle si l'utilisateur a le droit de se connecter (login, pwd)
/// </summary>
/// <param name="responsable">Objet Responsable avec login et mot de passe</param>
/// <returns>vrai si login et pwd est correct</returns>
1 référence | leaso, il y a 3 jours | 1 auteur, 1 modification
public Boolean ControleAuthentification(Responsable responsable)
{
    if (access.Manager != null)
    {
        string req = "select * from responsable ";
        req += "where login = @login AND pwd = SHA2(@pwd, 256)";
        Dictionary<string, object> parameters = new Dictionary<string, object>();
        parameters.Add("@login", responsable.Login);
        parameters.Add("@pwd", responsable.Pwd);
        try
        {
            List<Object[]> records = access.Manager.ReqSelect(req, parameters);
            if (records != null)
            {
                return (records.Count > 0);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Environment.Exit(0);
        }
    }
    return false;
}

```

J'ai ensuite créé le contrôleur de cette frame : FrmAuthentificationController. Ce contrôleur interagit avec la couche dal. Il instancie un objet ResponsableAccess et expose la méthode ControleAuthentification, qui est appelée par la vue pour valider les informations de connexion.

```

/// <summary>
/// Vérifie l'authentification
/// </summary>
/// <param name="responsable">objet contenant les informations de connexion</param>
/// <returns> vrai si les informations de connexion sont correctes</returns>
1 référence | leaso, il y a 3 jours | 1 auteur, 1 modification
public Boolean ControleAuthentification(Responsable responsable)
{
    return responsableAccess.ControleAuthentification(responsable);
}

```

Je me suis par la suite concentrée sur la classe FrmAuthentification (view).

La classe FrmAuthentification représente l'interface de connexion de l'application :

- L'utilisateur saisit son login et son mot de passe.
- Un clic sur le bouton "Connexion" déclenche les vérifications suivantes :
  - Si un champ est vide, un message d'information est affiché.
  - Si les identifiants sont incorrects, un message d'erreur empêche l'accès.
  - Si les identifiants sont valides, la fenêtre principale (FrmMediaTek86) est lancée.

```
/// <summary>
/// Demande au controleur de controler l'authentification
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | leaso, il y a 3 jours | 1 auteur, 1 modification
private void BtnConnect_Click(object sender, EventArgs e)
{
    String login = txtLogin.Text;
    String pwd = txtPwd.Text;
    if (String.IsNullOrEmpty(login) || String.IsNullOrEmpty(pwd))
    {
        MessageBox.Show("Tous les champs doivent être remplis.", "Information");
    }
    else
    {
        Responsable responsable = new Responsable(login, pwd);
        if (controller.ControleAuthentification(responsable))
        {
            FrmMediaTek86 frm = new FrmMediaTek86();
            frm.ShowDialog();
        }
        else
        {
            MessageBox.Show("Authentification incorrecte ou vous n'êtes pas autorisé à vous connecter", "Alerte");
        }
    }
}
```

Enfin, j'ai testé plusieurs scénarios pour m'assurer du bon fonctionnement :

- Affichage automatique de la fenêtre d'authentification au lancement.
- Messages d'erreur si un champ est vide ou si les identifiants sont incorrects.
- Accès autorisé uniquement lorsque les identifiants sont valides.

Toutes ces modifications ont été enregistrés sur le **dépôt distant GitHub**, le **Kanban** et **la documentation technique** ont été mis à jour également.

[Phase 7 : Code de l'authentification + améliorations diverses · leasourmail/MediaTek86@0c3991b](#)

[Phase 8 : Mise à jour de la documentation technique · leasourmail/MediaTek86@4ac9080](#)

## Bilan de l'étape

L'étape 4 a été une phase essentielle du projet, car elle a permis de rendre l'application pleinement fonctionnelle, conformément aux cas d'utilisation définis dans le dossier documentaire. Toutes les fonctionnalités principales ont été développées dans le respect de l'architecture MVC, garantissant une structure claire, modulaire et maintenable.

J'ai commencé par mettre en place les classes du modèle (model) représentant les entités de la base de données. Ensuite, j'ai développé les classes d'accès aux données dans le package dal, qui assurent les échanges avec la base via la classe BddManager. Ces classes permettent aujourd'hui de gérer les personnels, les services, les motifs et les absences.

Le contrôleur FrmMediaTek86Controller a été enrichi progressivement pour relayer toutes les opérations nécessaires à la vue, comme l'ajout, la modification ou la suppression d'un personnel.

Côté interface utilisateur (FrmMediaTek86), toutes les fonctionnalités prévues ont été implémentées avec une gestion claire des événements, un affichage dynamique des données et des contrôles de cohérence (champs obligatoires, doublons, confirmation avant suppression...).

Enfin, j'ai développé un système complet d'authentification, avec sa propre vue (FrmAuthentification), son contrôleur et sa classe d'accès aux données. J'ai vérifié qu'il fonctionnait correctement à l'ouverture de l'application, qu'il empêchait les connexions non valides, et qu'il redirigeait l'utilisateur vers la fenêtre principale en cas de succès.

Toutes ces fonctionnalités ont été testées et les résultats obtenus sont conformes aux attentes fonctionnelles des cas d'utilisation fournis.

## 2.5 Étape 5 – Création d'une documentation utilisateur en vidéo

Cette étape avait pour objectif de produire une documentation utilisateur sous forme de vidéo. Elle permet de présenter l'application de manière concrète, en illustrant les principales fonctionnalités à travers une démonstration filmée et commentée. L'objectif est de rendre l'outil compréhensible et accessible à tout utilisateur, même sans connaissance technique.

Chaque action est expliquée oralement pendant la démonstration, afin de guider l'utilisateur pas à pas. Le ton est clair, pédagogique et accessible.

## 2.6 Étape 6 – Déploiement, compte rendu final et intégration au portfolio



## Objectif de l'étape

L'objectif de cette dernière étape est de finaliser le projet en le rendant prêt à être déployé et partagé. Il s'agit de produire tous les livrables attendus pour :

- permettre l'installation de l'application sur un autre poste,
- fournir un compte rendu structuré et documenté,
- valoriser le projet via une page dédiée dans le portfolio.

## Création de l'installateur sous VisualStudio

Un fichier exécutable d'installation a été généré à l'aide des outils fournis par Visual Studio. Cet installateur permet de déployer facilement l'application sur un autre poste sans avoir besoin de recompiler le projet.

Les fichiers nécessaires sont regroupés dans un dossier ou un setup, prêt à être partagé ou utilisé.

## Script SQL complet de la base de données

Un script SQL complet a été généré. Il contient :

- la création des tables nécessaires à l'application (personnel, service, absence, motif, responsable, etc.),
- les insertions de données de test,
- la création d'un utilisateur avec les droits d'accès à la base (login / pwd) rajoutée manuellement,
- l'utilisation de la fonction SHA2() pour chiffrer le mot de passe du responsable.

Ce script peut être exécuté directement sur un serveur MySQL pour initialiser l'environnement.

Sauvegarde sur le dépôt distant **GitHub** :

[Phase 9 : Ajout vidéo documentation utilisateur + création d'un insta... · leasourmail/MediaTek86@dcf75ff](#)

## Rédaction du compte rendu d'activité

Il s'agit du document PDF que vous lisez.

Le document met en valeur à la fois le travail technique, l'organisation méthodologique, et la capacité à documenter un projet.

### Création de la page dans le portfolio

Une page dédiée à la mission a été ajoutée dans mon portfolio personnel. Elle contient :

- une présentation résumée du contexte et des objectifs,
- le lien vers le dépôt GitHub du projet, avec l'historique complet des sauvegardes,
- le lien vers le compte rendu d'activité au format PDF,
- l'intégration de la vidéo de démonstration des fonctionnalités.

Cette page permet de présenter le projet de manière professionnelle à un futur recruteur ou évaluateur, tout en fournissant toutes les ressources nécessaires pour comprendre, tester ou installer l'application.

Lien de mon portfolio : <https://leasourmail.wixsite.com/my-site-2>

### Bilan de l'étape

À la fin de cette étape :

- l'application peut être déployée facilement sur un autre poste,
- la base de données est initialisable via un script SQL complet,
- un compte rendu détaillé, illustré et structuré est disponible,
- le projet est visiblement valorisé dans le portfolio, avec tous les liens nécessaires accessibles en un clic.

## 3. Bilan général du projet

Le projet s'est déroulé sur plusieurs étapes progressives, chacune apportant une nouvelle dimension à la construction de l'application. Au fil de ces étapes, j'ai pu mettre en œuvre des compétences techniques, méthodologiques et organisationnelles, tout en respectant les principes de l'architecture **MVC**.

## 3.1 Résumé des objectifs atteints à chaque étape

### Étape 1 : Préparation de l'environnement et création de la base de données

- Mise en place de l'environnement de développement (Visual Studio, Wampserver, Looping).
- Création et initialisation d'une base de données MySQL à partir d'un script SQL fourni.
- Insertion de données de test cohérentes pour valider les traitements futurs.
- Création d'un utilisateur sécurisé avec un mot de passe chiffré (SHA2).
- Objectifs atteints : environnement prêt, base de données fonctionnelle et testée.

### Étape 2 : Conception des interfaces, structure MVC et dépôt GitHub

- Réalisation de maquettes d'interfaces avec Pencil.
- Création du projet Visual Studio structuré en MVC (model, dal, controller, view).
- Création du dépôt distant GitHub, initialisation du Kanban (issues par tâche).
- Début du développement visuel (Vue).
- Objectifs atteints : structure claire, interfaces visuelles prêtes, versionnage démarré.

### Étape 3 : Implémentation du modèle et accès aux données

- Développement des classes BddManager et Access pour gérer la connexion à la base.
- Création des classes métiers dans model.
- Génération de la documentation XML technique, versionnée dans le dépôt.
- Objectifs atteints : base technique solide, architecture prête pour les traitements métiers.

### Étape 4 : Développement des fonctionnalités métiers

- Création des classes d'accès aux données (PersonnelAccess, AbsenceAccess, etc.).

- Mise en place du contrôleur FrmMediaTek86Controller.
- Développement complet de la vue FrmMediaTek86 avec gestion des événements (ajout, modification, suppression).
- Mise en œuvre de l'authentification (ResponsableAccess, FrmAuthentificationController, FrmAuthentification).
- Tests fonctionnels réalisés pour valider les cas d'utilisation.
- Objectifs atteints : application fonctionnelle, cas d'utilisation respectés, interface réactive et fiable.

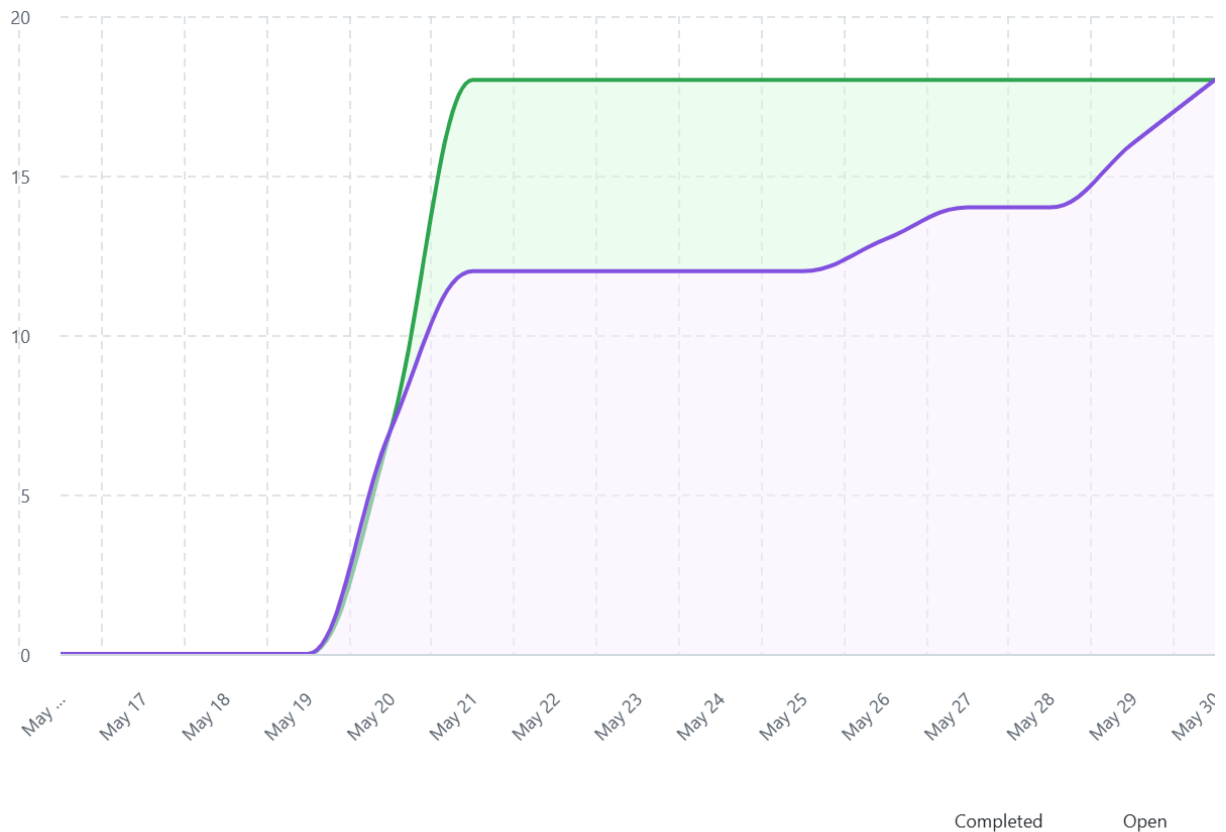
### Étape 5 : Création de la documentation utilisateur vidéo

- Enregistrement d'une vidéo commentée de 5 minutes présentant l'application.
- Démonstration fluide et claire de toutes les fonctionnalités principales.
- Objectif atteint : documentation accessible pour tout utilisateur, complémentaire au livrable technique.

### Étape 6 : Déploiement, compte rendu final et intégration au portfolio

- Création d'un installeur de l'application.
- Génération du script complet de la base de données.
- Rédaction complète du compte rendu d'activité au format PDF, structuré avec sommaire et bilans.
- Création d'une page de portfolio regroupant :
  - Une présentation du projet.
  - Le lien GitHub.
  - Le compte rendu PDF.
  - La vidéo de démonstration intégrée.
- Objectifs atteints : projet déployable, documenté, et valorisé professionnellement.

**Voici l'évolution du Kanban :**



## 4. Conclusion générale

Ce projet m'a permis de mener à bien une application complète, depuis la préparation de l'environnement jusqu'au déploiement, en passant par le développement, les tests et la documentation.

Il m'a offert une expérience concrète et structurante autour des bonnes pratiques de développement, de la gestion de projet (avec GitHub et Kanban), et du respect de l'architecture MVC.

Je suis désormais capable de concevoir, développer, sécuriser et livrer une application professionnelle, tout en la documentant pour les utilisateurs comme pour les développeurs.