

COMPTE RENDU D'ACTIVITE

Application de gestion du personnel

**C#
SQL**

Par Aurélie Demange

Mai 2025

Table des matières

1	Contexte.....	3
2	Mission globale.....	3
3	Présentation des étapes.....	4
3.1	Étape 1 : préparation de l'environnement de développement.....	4
3.2	Étape 2 : création et alimentation de la base de données.....	4
3.3	Étape 3 : représentation des interfaces répondant aux besoins sur Pencil.....	7
3.4	Étape 4 : création du projet sous Visual Studio 2022 et du dépôt distant.....	9
3.5	Étape 5 : codage de la partie Vue de l'application (uniquement le visuel).....	10
3.6	Étape 6 : constitution des packages dal et bddmanager, codage de la classe Bddmanager..	12
3.7	Étape 7 : conception des classes métiers dans le package model.....	14
3.8	Étape 8 : génération de la première documentation technique.....	15
3.9	Étape 9 : développement de l'ensemble des fonctionnalités de l'application.....	17
	A) Se connecter.....	17
	B) Ajouter un personnel	17
	C) Supprimer un personnel	17
	D) Modifier un personnel	17
	E) Afficher les absences.....	17
	F) Ajouter une absence.....	17
	G) Supprimer une absence.....	17
	H) Modifier une absence.....	17
	I) Retour à la gestion du personnel et déconnexion.....	17
3.10	Étape 10 : ajout/correction des commentaires normalisés et mise à jour de la documentation technique.....	18
3.11	Étape 11 : production de la documentation utilisateur sous format vidéo.....	18
3.12	Étape 12 : création d'un installateur.....	18
3.13	Étape 13 : générer un script complet de la base de données.....	18
3.14	Étape 14 : rédaction du compte-rendu d'activité et de la page du portfolio.....	18
4	Bilan final.....	18

1 Contexte

Dans le cadre de ma première année en BTS Services Informatiques aux Organisations, option Solutions Logicielles et Applications Métiers, j'ai développé une application de gestion du personnel en C#.

Ce projet a été réalisé dans un cadre fictif dans lequel je travaille comme technicienne développeuse junior au sein de l'entreprise InfoTech Services 86 (ITS 86). Il s'agit d'une Entreprise de Services Numériques (ESN), organisée autour de grands pôles comportant chacun plusieurs activités : le pôle Développement et le pôle Systèmes et Réseau. La partie concernée ici est le pôle Développement qui propose notamment le développement d'applications, la création et la gestion de bases de données, ainsi que des solutions d'hébergement sur des serveurs dédiés.

Dans le cadre de ses activités, et afin de décrocher des contrats, ITS 86 répond régulièrement à des appels d'offres. Dans le cas présent, l'entreprise a répondu à l'appel d'offres du réseau des médiathèques de la Vienne, MediaTek86. Celui-ci met un accent particulier sur le développement du numérique. Par exemple, il propose un portail en ligne, DigiMediaTek86, permettant la consultation et la prolongation des prêts, la réservation de documents, etc. Il met aussi à disposition des équipements numériques en libre-service (tablettes numériques, liseuses...), et organise des formations au numérique, en ligne comme en présentiel.

ITS 86 a remporté ce marché avec MediaTek86. Celui-ci couvre plusieurs domaines et comporte un certain nombre de missions, notamment la gestion du parc informatique des médiathèques ainsi que l'informatisation de plusieurs activités internes ou en lien avec le public. C'est dans le cadre du deuxième volet qu'une mission m'a été confiée : développer une application de bureau permettant, pour chaque médiathèque, de gérer son personnel.

2 Mission globale

L'objectif principal de ce projet est de développer une application de bureau permettant de gérer le personnel au sein de chaque médiathèque du réseau MediaTek86. Elle vise à faciliter le suivi du personnel, leur affectation au sein d'un service ainsi que la gestion de leurs absences. Pour des raisons de sécurité et de simplicité, l'application est installée sur un seul poste informatique par médiathèque, avec un unique compte utilisateur pour chacune.

Pour atteindre cet objectif, plusieurs missions ont été réalisées :

- Conception et structuration d'une base de données SQL, afin de stocker l'ensemble des informations concernant le personnel (nom, prénom, téléphone, mail, service d'appartenance) et leurs absences (dates, motif).
- Développement d'une application de bureau en C# sous Visual Studio 2022 avec
 - ➔ une vérification des droits d'accès pour sécuriser les données personnelles ;
 - ➔ une interface claire et intuitive facilitant la manipulation des données ;
 - ➔ des fonctionnalités permettant de consulter, ajouter, modifier et supprimer des données de manière simple mais sécurisée.
- Test et validations manuelles des fonctionnalités de l'application afin de vérifier leur bon fonctionnement et détecter d'éventuelles erreurs.

Grâce à cette application, les médiathèques du réseau MediaTek86 peuvent désormais gérer leur personnel de manière centralisée, sécurisée et optimisée.

3 Présentation des étapes

3.1 Étape 1 : préparation de l'environnement de développement

Avant de commencer concrètement le projet, il était essentiel de préparer l'environnement de développement. Pour cela, j'ai d'abord vérifié que WampServer et Visual Studio 2022 Entreprise étaient bien installés sur mon poste de travail :

- WampServer est un outil qui regroupe Apache (serveur web), MySQL (gestionnaire de bases de données) et phpMyAdmin (interface de gestion de la base de données). Je m'en sers comme serveur local afin d'héberger la base de données SQL, ce qui facilite le développement et les tests de l'application sans avoir besoin d'une connexion à un serveur distant.
- Visual Studio 2022 est mon IDE (environnement de développement intégré) pour concevoir une application de bureau en C#. Il est très complet, avec une excellente interface intuitive, et offre de nombreux outils très importants pour un développeur : tests unitaires intégrés, gestionnaire de packages NuGet (pour installer des bibliothèques externes), très bon débogueur, etc. De plus, VS 2022 est particulièrement indiqué pour le développement d'applications de bureau sous Windows car il est optimisé pour le framework .NET.

Ensuite, j'ai vérifié que le logiciel de modélisation Looping était également bien installé sur mon ordinateur. Cet outil permet de concevoir différents schémas de bases de données (MCD, MLD), ainsi que la récupération de ceux-ci en script SQL, facilitant ainsi leur importation dans WampServer pour l'initialisation de la base de données de l'application de gestion du personnel.

3.2 Étape 2 : création et alimentation de la base de données

Pour commencer cette étape, j'ai récupéré le schéma conceptuel de données (MCD).

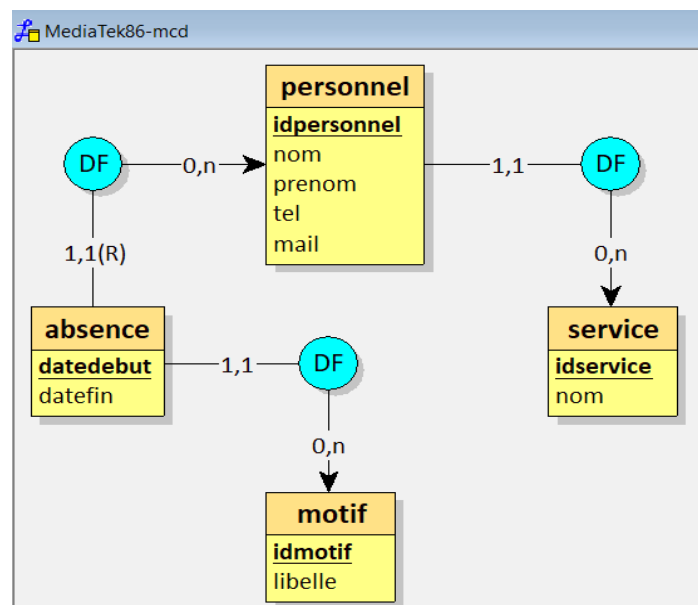


Schéma conceptuel de données (MCD) représentant les différentes entités et leurs relations

À partir de ce schéma, j'ai pu récupérer le script SQL correspondant :

```

SQL
CREATE TABLE service(
  idservice INT AUTO_INCREMENT,
  nom VARCHAR(50) ,
  PRIMARY KEY(idservice)
);

CREATE TABLE motif(
  idmotif INT AUTO_INCREMENT,
  libelle VARCHAR(128) ,
  PRIMARY KEY(idmotif)
);

CREATE TABLE personnel(
  idpersonnel INT AUTO_INCREMENT,
  nom VARCHAR(50) ,
  prenom VARCHAR(50) ,
  tel VARCHAR(15) ,
  mail VARCHAR(128) ,
  idservice INT NOT NULL,
  PRIMARY KEY(idpersonnel),
  FOREIGN KEY(idservice) REFERENCES service(idservice)
);

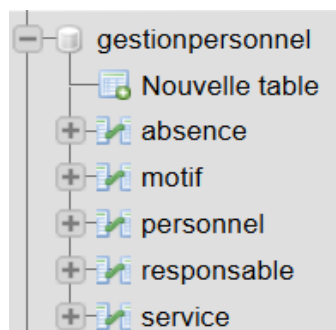
CREATE TABLE absence(
  idpersonnel INT,
  datedebut DATETIME,
  datefin DATETIME,
  idmotif INT NOT NULL,
  PRIMARY KEY(idpersonnel, datedebut),
  FOREIGN KEY(idpersonnel) REFERENCES personnel(idpersonnel),
  FOREIGN KEY(idmotif) REFERENCES motif(idmotif)
);

```

Script SQL généré à partir du MCD pour la création des tables et champs de la base de données

J'ai ensuite créé la base de données `gestionpersonnel` sous MySQL à l'aide de phpMyAdmin, puis configuré un utilisateur disposant des droits d'accès nécessaires à cette base.

J'ai exécuté le script SQL pour générer automatiquement l'ensemble des tables et champs nécessaires à l'application, à l'exception de la table `responsable` que j'ai créée moi-même. Cette dernière a pour objectif de gérer l'authentification des utilisateurs de l'application. Je l'ai rempli avec un login et mot de passe chiffré afin de sécuriser l'application et me permettre également de la tester. Pour hasher le mot de passe, j'ai utilisé la fonction SHA2 : `SHA2('le pwd', 256)`. Il est inutile de créer plusieurs logins, car l'application fonctionne avec un unique compte utilisateur.



Vue des tables de la base de données dans phpMyAdmin

Enfin, pour finaliser la préparation, j'ai alimenté la base de données. Par exemple, pour la table `motif`, j'ai ajouté les quatre motifs possibles d'absence.

Serveur : MySQL:3306 > Base de données : gestionpersonnel > Table : motif

Parcourir Structure SQL Rechercher Insérer Exp

✓ Affichage des lignes 0 - 3 (total de 4, traitement en 0,0023 seconde(s).)

SELECT * FROM `motif`

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

☐ Tout afficher | Nombre de lignes : 25 | Filtrer les lignes: Chercher dans

Options supplémentaires

				idmotif	libelle
<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

Contenu de la table motif avec les différents motifs ajoutés

Pour les tables personnel et absence, j'ai utilisé un générateur automatique d'insert afin de les remplir avec suffisamment de données pouvant servir d'exemples aléatoires pour mes tests.

SELECT * FROM `personnel`

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

☐ Tout afficher | Nombre de lignes : 25 | Filtrer les lignes: Chercher dans cette table | Trier par clé : Aucun(e)

Options supplémentaires

					idpersonnel	nom	prenom	tel	mail	idservice
<input type="checkbox"/>	Éditer	Copier	Supprimer	1	Mclean	Helen	04 47 22 27 94	eros.non@google.edu		1
<input type="checkbox"/>	Éditer	Copier	Supprimer	2	Mcintyre	Emery	03 83 01 49 95	mauris@protonmail.edu		2
<input type="checkbox"/>	Éditer	Copier	Supprimer	3	Lowery	Tara	05 79 73 68 45	augue.id@icloud.couk		2
<input type="checkbox"/>	Éditer	Copier	Supprimer	4	Richards	Arden	08 76 08 88 47	proin.non@aol.net		3
<input type="checkbox"/>	Éditer	Copier	Supprimer	5	Dillon	Elton	04 05 91 87 92	molestie.orci@aol.org		1
<input type="checkbox"/>	Éditer	Copier	Supprimer	6	Farley	Hanna	08 15 44 83 11	viverra@protonmail.ca		1
<input type="checkbox"/>	Éditer	Copier	Supprimer	7	Chaney	Mia	08 35 96 54 46	ipsum@aol.couk		3
<input type="checkbox"/>	Éditer	Copier	Supprimer	8	Crosby	Aretha	07 31 91 67 79	metus.aliquam@hotmail.edu		3
<input type="checkbox"/>	Éditer	Copier	Supprimer	9	Phillips	Alma	08 75 12 56 61	fringilla.purus.mauris@hotmail.com		1
<input type="checkbox"/>	Éditer	Copier	Supprimer	10	Stephenson	Kiayada	07 96 43 15 20	tempor.arcu.vestibulum@yahoo.ca		1

Contenu de la table personnel

À la fin de cet étape, l'environnement de développement était correctement installé et configuré, la base de donnée était créée et alimentée, son accès était sécurisé. Les conditions étaient réunies pour débiter concrètement le projet.

3.3 Étape 3 : représentation des interfaces répondant aux besoins sur Pencil

Avant de commencer le codage de l'application, il est très important de concevoir une maquette représentant les interfaces correspondant aux besoins identifiés. Pour cela, j'ai utilisé le diagramme de cas d'utilisation ainsi que le descriptif de chaque cas d'utilisation. Cette étape

permet d'identifier les interactions entre l'utilisateur et l'application ainsi que de visualiser le design et l'organisation graphique des différentes fonctionnalités de l'application.

Pour ce faire, j'ai utilisé Pencil, un logiciel libre et gratuit de maquettage d'interfaces suffisamment complet, afin de créer plusieurs interfaces correspondant aux principales fonctionnalités de l'application :

- Interface d'authentification sécurisée comprenant les champs pour saisir le login et le mot de passe, ainsi qu'un bouton connexion qui menant à la fenêtre principale.
- Interface de gestion du personnel, affichant la liste du personnel et un formulaire de saisie d'informations, avec des boutons permettant d'ajouter, modifier ou supprimer un employé, se déconnecter, accéder aux absences d'un personnel, etc.
- Interface de gestion des absences, affichant la liste des absences du personnel sélectionné dans l'écran précédent, un formulaire de saisie ainsi que des boutons pour ajouter, modifier, supprimer une absence, ou revenir à l'écran précédent.
- Interfaces de confirmation qui s'affichent avec la suppression ou la modification d'une personne ou d'une absence, l'objectif étant de valider ces actions de en toute sécurité.

Connexion à l'application de gestion du personnel

Saisissez votre login et votre mot de passe pour vous connecter

Login : Mot de passe :

Login ou mot de passe incorrect

Page de connexion permettant l'authentification sécurisée

Gestion des absences

Liste des absences du salarié issue de la base de données avec la date de début, date de fin, le motif

Date de début de l'absence : JJ/MM/YYYY

Date de fin de l'absence : JJ/MM/YYYY

Impossible, une absence est déjà enregistrée dans ce créneau
Attention, la date de fin ne peut pas être antérieure à la date de début

Motif de l'absence : Sélection du motif

Tous les champs doivent être remplis

Enregistrer l'absence

Valider les modifications

Annuler

Ajouter une absence Modifier une absence Supprimer une absence Retour à la gestion du personnel

dessin_interfaces Page de connexion Gestion du personnel de la médiathèque **Gestion des absences** Confirmation Filter Add Page...

Interface de gestion des absences

Confirmation de suppression

Êtes vous sûr de vouloir supprimer cette personne de la liste des employés ?

Oui Non

Interface de confirmation

Cette étape m'a permis d'avoir un premier aperçu visuel de mon application, de réfléchir aux différentes interfaces nécessaires, à la disposition des différentes zones (listes, boutons, zones de texte, etc.) et à la navigation entre les différentes fenêtres. Grâce à cette préparation, j'ai pu commencer le codage de l'application avec une trame claire et un repère visuel. Cela m'a permis d'optimiser la gestion de mon temps et d'organiser mon développement tout en m'assurant de ne rien omettre concernant les fonctionnalités attendues.

3.4 Étape 4 : création du projet sous Visual Studio 2022 et du dépôt distant

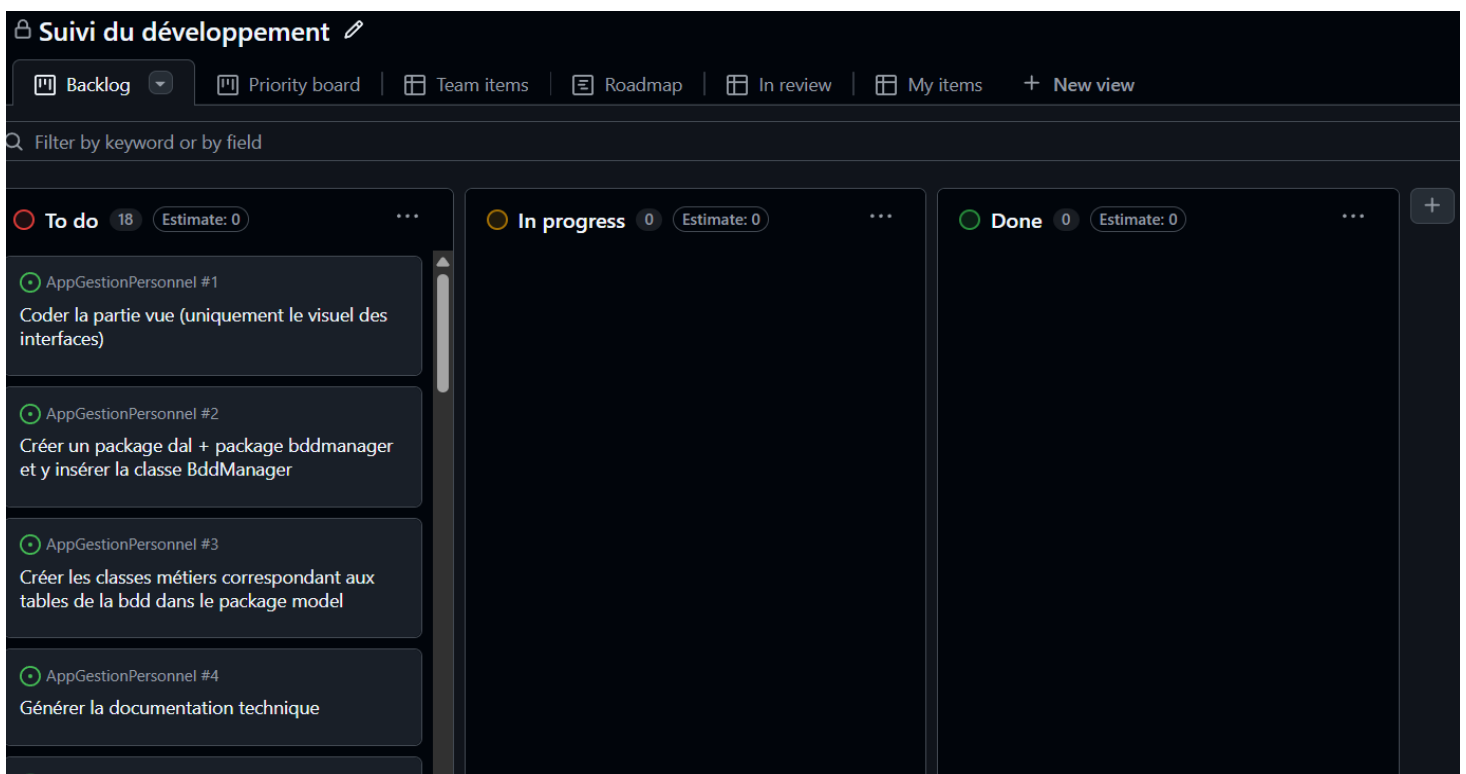
Pour cette étape, j'ai commencé par créer un nouveau projet sous VS 2022 en choisissant comme type de projet, Application Windows Forms (.NET Framework).

J'ai ensuite créé les packages **Model**, **View** et **Controller**, correspondant à l'architecture MVC, afin de structurer mon projet. Celle-ci permet de bien séparer les responsabilités dans le code et de le rendre plus clair. Le Modèle représente les données de l'application et la logique

métier, la Vue correspond aux interfaces utilisateur qui seront utilisées par ce dernier, et enfin, le Contrôleur fait le lien entre le Modèle et la Vue en gérant les actions de l'utilisateur tout en mettant à jour les données et l'interface en fonction de ces actions. Une capture d'écran sera insérée un peu plus loin dans le compte-rendu, présentant l'ensemble des packages de l'application.

Une fois ces packages en place, j'ai créé un dépôt distant sur GitHub et réalisé ma première sauvegarde en effectuant un commit and push depuis VS 2022. Cela permet deux choses importantes : sécuriser mon travail (par exemple, en cas de perte de la version locale) mais aussi le rendre accessible, ce qui est fondamental pour partager son projet ou travailler en équipe.

Enfin, j'ai mis en place un "project" (tableau Kanban) sur GitHub afin de suivre facilement l'avancement du projet : il est très pratique pour organiser visuellement les tâches. Il contient une liste d'issues correspondant chacune à une tâche à réaliser. Cela permet de planifier et gérer le développement de l'application. Au départ, l'ensemble des tâches se trouvait dans la colonne "To Do". Quand je travaillais sur une tâche, je la déplaçais dans la colonne "In Progress", et quand elle était achevée, elle passait dans la colonne "Done".



Aperçu du Kanban lors de sa création


3.5 Étape 5 : codage de la partie Vue de l'application (uniquement le visuel)

Pour cette étape, j'ai utilisé le Windows Forms Designer intégré à VS 2022. Cet outil permet de construire visuellement les interfaces utilisateur sans devoir coder, ce qui facilite leur construction rapide tout en visualisant le rendu final. Il suffit de sélectionner les éléments souhaités (boutons, labels, textBox, groupBox, etc.) et de les glisser-déposer sur l'interface. Cela permet une meilleure organisation et un placement précis des différents composants graphiques.

Je me suis appuyée sur ma maquette réalisée sur Pencil afin de savoir quels éléments placés et à quel endroit. J'ai toutefois ajouté certains éléments afin d'optimiser l'application, notamment des GroupBox qui s'activent ou se désactivent selon les actions de l'utilisateur. Pour mon application, j'ai donc créé trois interfaces :

- Interface d'authentification : composée de plusieurs Labels, de deux TextBox (pour le login et le mot de passe) et d'un bouton de connexion. J'ai également modifié certains

paramètres afin d'optimiser et sécuriser l'application. Par exemple, la TextBox du mot de passe affiche des astérisques à la place des caractères saisis et le Label d'erreur d'authentification est invisible au lancement de l'application.



Connexion à l'application de gestion du personnel

BIENVENUE

Saisissez votre login et votre mot de passe pour vous connecter :

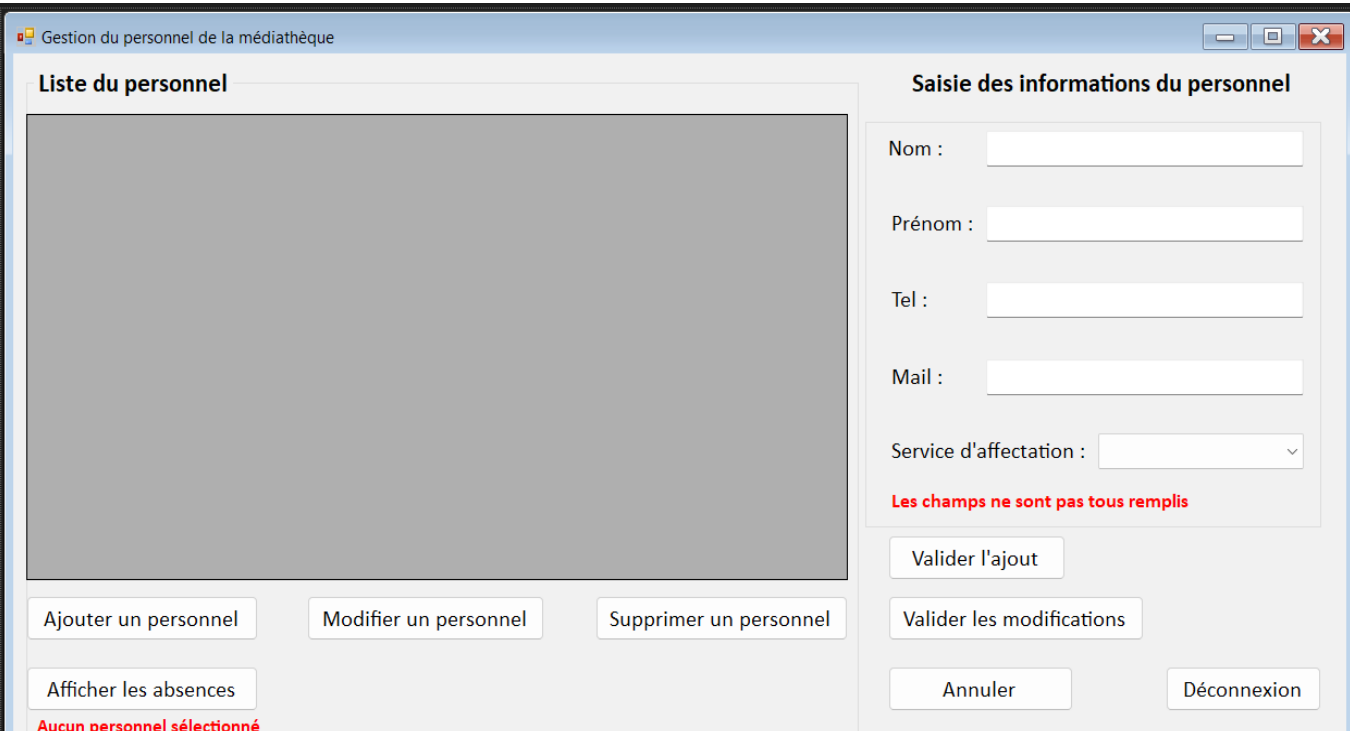
Login : Mot de passe :

Login ou mot de passe incorrect !

Connexion

Interface d'authentification

- Interface de gestion du personnel : composée d'un DataGridView (pour afficher la liste du personnel récupérée depuis la base de données), d'un ComboBox (pour afficher la liste des services), de tous les boutons et TextBox nécessaires, de labels dont ceux pour afficher les messages d'erreur. J'ai également ajouté deux GroupBox afin d'améliorer l'expérience utilisateur et éviter les erreurs de manipulation lors des modification :
 - ➔ Le premier regroupe le DataGridView et les boutons qui ne doivent plus être accessibles lorsque l'utilisateur effectue une action modifiant la base de données (ajout ou modification d'un employé).
 - ➔ Le second concerne l'ensemble des TextBox de saisie des informations. Par exemple, lors de l'ajout d'un nouveau personnel, le premier GroupBox est désactivé et le deuxième activé. À la validation de l'ajout, l'inverse se produit.



Gestion du personnel de la médiathèque

Liste du personnel

Saisie des informations du personnel

Nom :

Prénom :

Tel :

Mail :

Service d'affectation :

Les champs ne sont pas tous remplis

Ajouter un personnel

Modifier un personnel

Supprimer un personnel

Afficher les absences

Aucun personnel sélectionné

Valider l'ajout

Valider les modifications

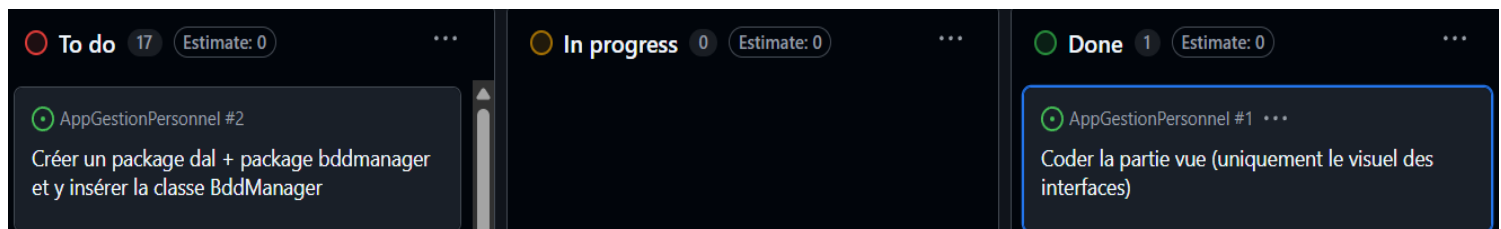
Annuler

Déconnexion

Interface de gestion du personnel

- Interface de gestion des absences : composée d'un DataGridView (affichant la liste des absences du personnel sélectionné sur l'interface précédente, récupérée depuis la base de données), de boutons, de TextBox, de Labels et de DateTimePicker pour la sélection des dates. Comme pour l'interface précédente, j'ai également ajouté deux GroupBox afin de désactiver certains éléments lors de opérations de modification ou d'ajout d'absences, garantissant ainsi un meilleur contrôle sur les actions disponibles.

Interface de gestion des absences



La tâche n°1 est passée en "Done" lors du Commit and Push

Enfin, sur chaque capture d'écran du Kanban GitHub, on pourra suivre l'avancement du projet : avec l'étape venant d'être réalisée et la tâche suivante pour la prochaine partie.

3.6 Étape 6 : constitution des packages dal et bddmanager, codage de la classe Bddmanager

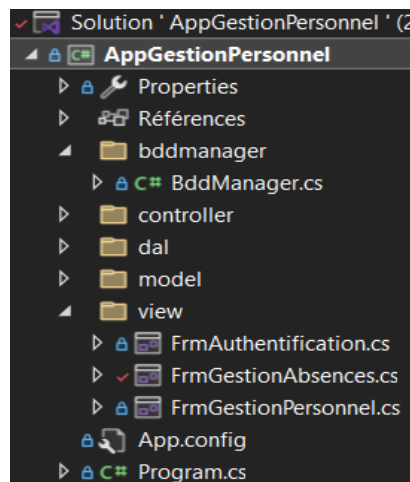
Pour cette étape, j'ai commencé par configurer mon IDE pour pouvoir accéder à la base de données. Cela nécessite l'installation du package `MySQL.Data` en passant par NuGet afin d'obtenir la bibliothèque `MySQL.Data.MySqlClient` qui permettra de gérer la base de données. Ensuite, il est nécessaire d'ajouter la référence `MySQL.Data.dll` au projet.

J'ai ensuite ajouté deux packages au projet :

- `bddmanager` : contient la classe `BddManager`
- `dal` : contient les classes qui répondent aux demandes du contrôleur en exploitant la classe `BddManager`. Ces classes seront remplies au fur et à mesure que les

besoins pour exploiter la base de données apparaîtront dans le développement, comme l'ajout, la modification, la suppression de données.

Cette organisation permet de garantir la lisibilité et la cohérence du projet.



Arborescence finalisée des packages du projet

Ensuite, j'ai créé la classe `BddManager`. Celle-ci gère l'ouverture et la fermeture de la connexion à la base de données. Il ne faut donc pas oublier d'ajouter la bibliothèque précédemment installée. Cette classe est un Singleton, c'est-à-dire qu'une seule instance de la classe peut exister tout au long de l'exécution de l'application. Cela permet de réutiliser la même connexion (ce qui optimise les performances en évitant de multiples ouvertures et fermetures) et de centraliser l'accès à la base de données. J'implémente cette classe toujours de la même manière :

- La méthode `GetInstance` permet de créer une instance de la classe seulement si elle n'existe pas encore, la chaîne de connexion sera alors transmise et la connexion unique est ouverte. Les appels suivants à cette méthode renverront cette même instance déjà créée, évitant ainsi de rouvrir une nouvelle connexion à chaque demande.
- Les méthodes `ReqSelect` et `ReqUpdate` permettent d'exécuter les requêtes. La première permet de récupérer les données depuis la base de données sous forme de listes d'objets, ce qui facilite leur manipulation dans le code. Quant à la deuxième, son but est l'exécution des requêtes `INSERT`, `UPDATE` ou `DELETE`, tout en les sécurisant grâce à l'utilisation de paramètres, évitant ainsi d'insérer directement des données dans une requête et donc de s'exposer à des injections SQL. Ces dernières sont des cyberattaques consistant à modifier une requête SQL mal sécurisée en y glissant du code inattendu.

```
public static BddManager GetInstance(string stringConnect)
{
    if (instance == null)
    {
        instance = new BddManager(stringConnect);
    }
    //retourne soit l'instance créée, soit l'instance déjà existante
    return instance;
}
```

Méthode `GetInstance`

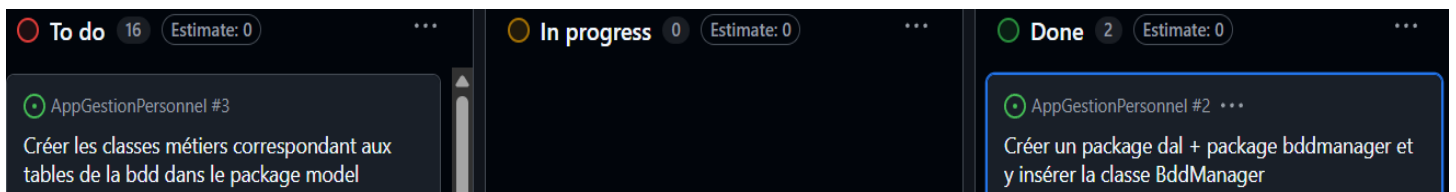
```

/// <summary>
/// Méthode pour exécuter requête select
/// </summary>
/// <param name="stringQuery"> Requête select</param>
/// <param name="parameters"> Dictionnaire des paramètres de la requête</param>
/// <returns> Tableaux contenant les valeurs des colonnes sélectionnées </returns>
5 références | aurelied1991, il y a 8 jours | 1 auteur, 1 modification
public List<Object[]> ReqSelect(string stringQuery, Dictionary<string, object> parameters = null)
{
    //Exécution de la requête sur l'objet command
    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));
        }
    }
}

```

Méthode ReqSelect

Cette architecture a plusieurs avantages : une séparation claire des responsabilités (la classe BddManager pour l'interaction avec la base de données, et le package dal pour les classes qui gèrent les opérations métiers), une sécurisation renforcée des données, ainsi qu'une maintenance facilitée. De plus, il n'y aura plus de raison d'y revenir pendant le reste du développement, car ce sera la classe Access qui gèrera la partie authentification en se chargeant d'interroger la base de données en utilisant les méthodes expliquées ci-dessus.



La tâche n°2 est passée en 'Done' lors du Commit and Push

3.7 Étape 7 : conception des classes métiers dans le package model

Dans cette étape, j'ai conçu, dans le package model, les classes métiers, chacune représentant une table de la base de données, tandis que les propriétés correspondent aux champs de la table. Cela permet de structurer le code avec des objets, ce qui facilite l'accès et la manipulation des données.

Lors de la création de ces classes, j'ai veillé à encapsuler correctement les données. J'ai donc utilisé des getters et des setters. Les premiers permettent un accès en lecture, afin de récupérer des informations sans risquer de les modifier. Par exemples, toutes les propriétés correspondant aux identifiants uniques des tables de la base de données n'ont que des getters, ce qui évite toute modification accidentelle et garantit ainsi l'intégrité des données. Quant aux setters, ils autorisent la modification contrôlée des propriétés lorsque cela est nécessaire.

```

/// <summary>
/// Getter pour avoir accès à l'identifiant du personnel sans pouvoir le modifier
/// </summary>
4 références | aurelied1991, il y a 4 jours | 1 auteur, 2 modifications
public int Idpersonnel { get; }

```

Propriété Idpersonnel en getter

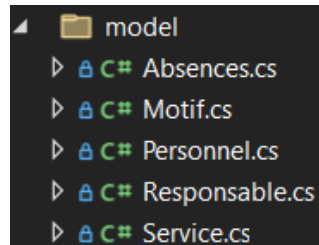
```

/// <summary>
/// Permet de récupérer ou modifier le numéro de téléphone du personnel
/// </summary>
5 références | aurelied1991, il y a 4 jours | 1 auteur, 2 modifications
public string Tel { get; set; }

```

Propriété Tel en getter et setter

J'ai donc créé 5 classes métiers correspondant à la base de données :



Arborescence de model

- **Responsable** : représente le responsable qui peut se connecter à l'application et contient Login et Pwd, tous deux en lecture seule.
- **Personnel** : cette classe représente la table personnel. Les informations gérées sont : Idpersonnel (lecture seule), Nom, Prenom, Tel, Mail (tous en lecture seule et modification), sans oublier Service, le service auquel appartient le personnel, représenté par un objet de type Service. Elle permet de rassembler dans un objet unique l'ensemble des informations d'un employé.
- **Service** : représente le service auquel une personne est rattachée. On y trouve Idservice et Nom, tous deux en lecture seule. J'y redéfinis également la méthode ToString() afin de renvoyer directement le nom du service lors de son affichage dans l'application (DataGridView, ComboBox).

```

/// <summary>
/// Constructeur pour initialiser une nouvelle instance de la classe Service
/// </summary>
/// <param name="idservice">Identifiant unique du service</param>
/// <param name="nom">Nom du service</param>
2 références | aurelied1991, il y a 4 jours | 1 auteur, 2 modifications
public Service(int idservice, string nom)
{
    // Initialisation des propriétés de la classe qui correspondent aux champs de la table "service"
    this.Idservice = idservice;
    this.Nom = nom;
}

```

Constructeur pour initialiser une instance de la classe Service

- **Absences** : représente les absences d'un personnel. Elle contient les informations suivantes : Idpersonnel (lecture seule), Datedebut (lecture et modification), Datefin (lecture et modification), Motif(lecture et modification, représenté par un objet de type Motif). Dans cette classe, j'ai aussi inclus AbsenceId, un identifiant temporaire généré automatiquement à chaque création d'objet afin de faciliter certaines actions.
- **Motif** : représente les motifs des absences. On y trouve aussi la méthode ToString() pour les mêmes raisons que la classe Service.

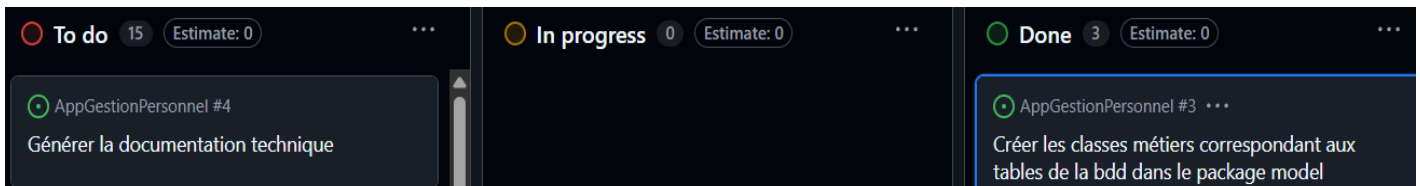

```

/// <summary>
/// Redéfinition de la méthode ToString,
/// </summary>
/// <returns>Le nom du service</returns>
0 références | aurelied1991, il y a 4 jours | 1 auteur, 2 modifications
public override string ToString()
{
    return this.Nom;
}

```

Méthode ToString() de la classe Service

La programmation orientée objet permet donc que le code soit structuré, lisible, maintenable et sécurisé. Chaque table de la base de données est représenté par un objet, favorisant l'organisation et la manipulation des données tout au long du développement.



La tâche n°3 est passée en 'Done' lors du Commit and Push

3.8 Étape 8 : génération de la première documentation technique

La documentation technique consiste à lister les classes, leurs membres et leurs signatures, accompagnés de commentaires du/des développeur(s) pour expliquer le code. Elle est fondamentale car elle rassemble l'ensemble des informations sur les classes et leurs interactions de manière accessible et organisée.

Afin de pouvoir générer la documentation technique, il est nécessaire d'insérer des commentaires normalisés dans le code. Dans VS 2022, il suffit de taper `///` au dessus des classes et méthodes pour générer automatiquement ces commentaires, auxquels on ajoute une description claire et précise.

```

/// <summary>
/// Classe métier correspondant à la table "absences" de la bdd
/// Représente l'absence d'un personnel à un moment donné
/// </summary>
29 références | aurelied1991, il y a 4 jours | 1 auteur, 4 modifications
public class Absences
{
    /// <summary>
    /// Constructeur pour initialiser une instance de la classe Absences avec les informations données
    /// </summary>
    /// <param name="idpersonnel">Identifiant unique du personnel</param>
    /// <param name="datedebut">Date de début de l'absence</param>
    /// <param name="datefin">Date de fin de l'absence</param>
    /// <param name="motif">Motif de l'absence</param>
    3 références | aurelied1991, il y a 4 jours | 1 auteur, 2 modifications
    public Absences(int idpersonnel, DateTime datedebut, DateTime datefin, Motif motif)
    {
        // Initialisation des propriétés de la classe qui correspondent aux champs de la table absences
        this.Idpersonnel = idpersonnel;
        this.Datedebut = datedebut;
        this.Datefin = datefin;
        this.Motif = motif;
    }
}

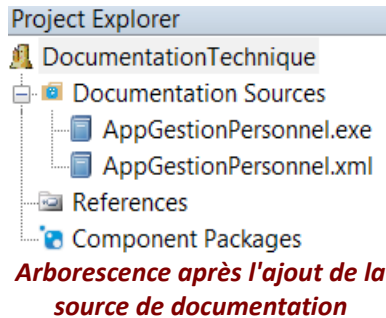
```

Exemples de commentaires normalisés avec la classe Absences et la méthode Absences

Après avoir ajouté tous les commentaires, j'ai modifié les paramètres du projet pour activer la génération d'un fichier XML à chaque build de l'application (dans l'onglet Build, option "Fichier

de documentation XML"). Ce fichier contient l'ensemble des commentaires normalisés.

Cependant, le fichier généré par VS 2022 n'est pas très lisible. Par conséquent, j'ai utilisé SandCastle Help File Builder (SHFB), un outil open-source, afin de transformer ce fichier XML en documentation HTML. Cette dernière est beaucoup plus lisible et on peut également naviguer dedans au sein de l'arborescence du projet. En effet, SHFB permet de convertir les informations au format XML en une interface web navigable avec une organisation par namespaces et classes. J'ai créé un nouveau projet dans SHFB, ajouté le fichier XML comme source de documentation, puis sélectionné le format de sortie (HTML).



Il ne faut pas oublier de configurer l'affichage pour inclure les membres privés, internes et les méthodes héritées, selon les besoins. J'ai ensuite généré la documentation. J'ai obtenu un dossier complet contenant le fichier index.html, consultable dans un navigateur, listant chaque classe, propriété, méthode et description dans une interface Web facile à parcourir. Cette documentation est essentielle et fondamentale pour assurer la maintenance future de l'application, notamment si un développeur doit reprendre le code, ajouter ou modifier des fonctionnalités.

A Sandcastle Documented Class Library

AppGestionPersonnel Namespaces

▼ Namespaces

AppGestionPersonnel

AppGestionPersonnel.bddmanager

AppGestionPersonnel.controller

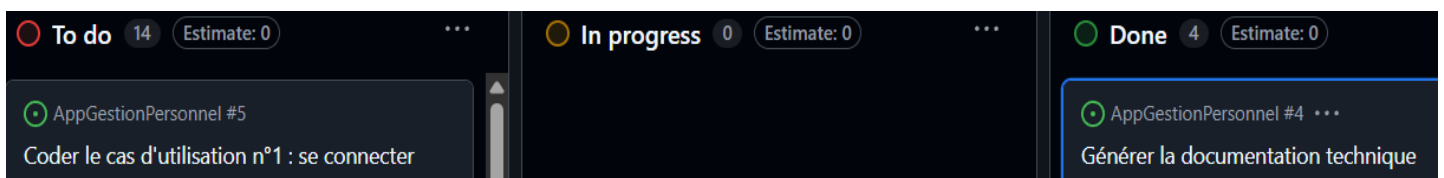
AppGestionPersonnel.dal

AppGestionPersonnel.model

AppGestionPersonnel.view

Aperçu de la documentation au format HTML

Pour finir, j'ai compressé le dossier obtenu et l'ai inclus dans le dossier du projet.



3.9 Étape 9 : développement de l'ensemble des fonctionnalités de l'application

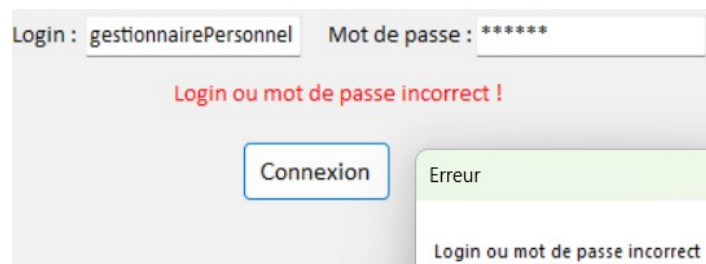
Dans cette étape, j'ai codé toutes les fonctionnalités attendues pour chaque cas d'utilisation. Cette partie détaille la construction de chaque fonctionnalité, en expliquant son objectif et son fonctionnement. Toutefois, pour ne pas alourdir le compte-rendu, je n'inclue pas de captures d'écran des cas d'utilisation.

A) Se connecter

La fonctionnalité "Se connecter" permet à l'utilisateur autorisé d'accéder à l'application en s'identifiant avec un login et un mot de passe. Il n'y a qu'un seul compte utilisateur par médiathèque, l'application étant monoposte et seulement accessible au responsable du personnel. Cette fonction est cruciale pour sécuriser l'accès aux données personnelles des employés de chaque médiathèque et leur modification.

L'interface visuelle a déjà été construite : deux champs à remplir (login et mot de passe) et un bouton pour se connecter. Concernant les nouvelles classes, j'ai ajouté un contrôleur `FrmAuthentificationController` pour la fenêtre d'authentification dans le package `control`, ainsi que les classes `Access` et `ResponsableAccess` dans le package `dal`.

Pour commencer, lors du clic sur le bouton "Connexion", la saisie de l'utilisateur dans les champs est récupérée et transmise au contrôleur pour vérifier si les données correspondent à celles de la base de données. Si jamais la saisie n'est pas correcte, un message d'erreur s'affiche pour informer l'utilisateur que l'authentification a échoué.



Message d'erreur lorsque l'authentification échoue

Le contrôleur utilise la méthode public Boolean `ControleAuthentification(Responsable responsable)` qui récupère l'objet `Responsable` de la vue et appelle la couche d'accès aux données (la classe `ResponsableAccess`) pour vérifier les informations. Cette dernière construit et exécute la requête SQL permettant de vérifier le login et le mot de passe saisi dans la base données via la classe `Access`, qui gère la connexion unique à la base de données en s'appuyant sur la classe `BddManager`. Si tout est correct, la méthode retourne "true". Pendant cette étape, j'ai également construit la chaîne de connexion à la base de données dans la classe `Access`.

```

public Boolean ControleAuthentification(Responsable responsable)
{
    if (access.Manager != null)
    {
        // Création de la requête SQL pour vérifier le login et le mdp
        string requete = "SELECT * FROM responsable 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
        {
            // Exécution de la requête et récupération des résultats
            List<Object[]> records = access.Manager.ReqSelect(requete, parameters);
            if (records != null)
            {
                // Si la liste des résultats n'est pas vide, l'authentification est réussie
                return (records.Count > 0);
            }
        }
        catch (Exception e)
        {
            // Gestion des erreurs
            Console.WriteLine(e.Message);
            MessageBox.Show($"Erreur lors de la connexion à la base de données");
        }
    }
    // Retourne faux si l'authentification échoue
    return false;
}

```

Méthode public Boolean ControleAuthentification(Responsable responsable) **de**
la classe ResponsableAccess

J'ai prêté une grande importance à respecter l'architecture MVC :

- La classe Responsable contient uniquement les données (modèle)
- Le formulaire FrmAuthentification est l'interface utilisateur (vue)
- La classe FrmAuthentificationController gère la logique métier (contrôleur)
- Access et ResponsableAccess s'occupent des interactions avec la base de données.

Enfin, si l'authentification est correcte, le bouton "Connexion" affiche la fenêtre de gestion du personnel. C'est donc lors de cette étape que je me suis occupée de la récupération des listes du personnel et des services dans la base de données, afin qu'elles s'affichent correctement dans le DataGridView et le Combobox.

Les classes Personnel et Service (dans Model) contiennent déjà les données. Pour récupérer ces données dans l'application, j'ai créé les classes PersonnelAccess et ServiceAccess dans le package dal. Elles gèrent les interactions avec la base de données en ce qui concerne les tables Personnel et Service. Par exemple, PersonnelAccess permet de récupérer la liste complète des personnels avec leur service associé grâce à une jointure SQL.

```

public List<Personnel> GetLesPersonnels()
{
    // Création d'une liste locale d'objets de type Personnel pour la remplir avec les données récupérées dans la bdd et la retourner
    List<Personnel> lesPersonnels = new List<Personnel>();

    // Vérification que la connexion à la base de données est bien établie
    if (access != null)
    {
        // Construction de la requête SQL pour récupérer tous les personnels et leurs services
        string requete = "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 ";
        requete += "FROM personnel p JOIN service s ON (p.idservice = s.idservice) ";
        // Résultats triés par nom puis par prénom
        requete += "ORDER BY nom, prenom;";

        //Tentative d'exécution de la requête SQL
        try
        {
            // Exécution de la requête SQL via ReqSelect et récupération des résultats sous la forme de liste d'objets
            List<Object[]> records = access.Manager.ReqSelect(requete);
            // Vérification que la requête retourne un résultat
            if (records != null)
            {
                // Parcours chaque enregistrement retourné par la base de données tant qu'il y a des résultats
                foreach (Object[] record in records)
                {
                    // Création d'un objet service à partir des données récupérées
                    Service service = new Service((int)record[5], (string)record[6]);
                    // Création d'un objet personnel à partir des informations récupérées
                    Personnel personnel = new Personnel((int)record[0], (string)record[1], (string)record[2], (string)record[3], (string)record[4], service);
                    // Ajout du personnel à la liste lesPersonnels
                    lesPersonnels.Add(personnel);
                }
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Environment.Exit(0);
        }
    }

    // Retourne la liste des développeurs récupérés depuis la base de données
    return lesPersonnels;
}

```

Méthode public List <Personnel> GetLesPersonnels() de la classe PersonnelAccess

Ensuite, j'ai créé la classe FrmGestionPersonnelController. Celle-ci récupère les listes via PersonnelAccess et ServiceAccess puis les transmet à la vue pour l'affichage.

The screenshot shows a web application interface for managing personnel. On the left, a table titled 'Liste du personnel' displays a list of personnel with columns for Nom, Prenom, Tel, and Mail. The first row is highlighted in blue. Below the table are three buttons: 'Ajouter un personnel', 'Modifier un personnel', and 'Supprimer un personnel'. At the bottom left is a button 'Afficher les absences'. On the right, a form titled 'Saisie des informations du personnel' contains input fields for Nom, Prénom, Tel, and Mail, a dropdown menu for 'Service d'affectation', and buttons for 'Valider l'ajout', 'Valider les modifications', 'Annuler', and 'Déconnexion'.

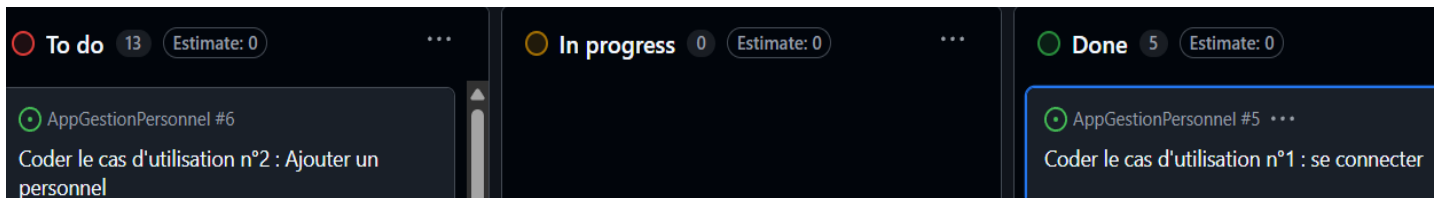
	Nom	Prenom	Tel	Mail
▶	Chaney	Mia	08 35 96 54 46	ipsum@aol.couk
	Crosby	Aretha	07 31 91 67 79	metus.aliquam@hotmail.com
	Dillon	Elton	04 05 91 87 92	molestie.orci@aol.org
	Farley	Hanna	08 15 44 83 11	viverra@protonmail.ca
	Lowery	Tara	05 79 73 68 45	augue.id@icloud.couk
	Mcintyre	Emery	03 83 01 49 95	mauris@protonmail.ec
	Mclean	Helen	04 47 22 27 94	eros.non@google.edu
	Phillips	Alma	08 75 12 56 61	fringilla.purus.mauris@
	Richards	Arden	08 76 08 88 47	proin.non@aol.net
	Stephenson	Kiyada	07 96 43 15 20	tempor.arcu.vestibulum

Le DataGridView rempli avec la liste du personnel

Service
prêt
prêt
administratif
administratif
médiation culturelle
médiation culturelle
administratif
administratif
prêt
administratif

Le DataGridView avec une colonne remplie par le service auquel appartient chaque personnel

Il est important de noter que les blocs try/catch sont très importants dans un code pour capturer les erreurs, éviter que l'application plante brutalement et afficher un message clair. Je les ai utilisés très régulièrement tout au long de mon projet. Par exemple, dans le cas ci-dessus, il permet de capturer les erreurs liées à l'exécution de la requête SQL.



La tâche n°5 est passée en 'Done' lors du Commit and Push

B) Ajouter un personnel

La fonctionnalité "Ajouter un personnel" permet d'enregistrer un nouvel employé dans la base de données en saisissant les informations nécessaires.

Lorsque l'utilisateur clique sur le bouton "Ajouter un nouveau personnel", la zone de saisie des informations devient accessible et le GroupBox qui permet de cliquer sur une action est désactivé.

```
private void BtnAjouterPersonnel_Click(object sender, EventArgs e)
{
    gboSaisieInfos.Enabled = true;
    BtnValiderAjout.Enabled = true;
    BtnAnnuler.Enabled = true;
    gboPersonnel.Enabled = false;
}
```

Activation du formulaire d'ajout d'un personnel

Ensuite, l'utilisateur saisie les informations demandées. Lorsqu'il clique sur le bouton "valider", les données des TextBox sont récupérées et vérifiées. Si un champs reste vide, un label s'affiche pour signaler l'erreur. Si toutes les informations sont présentes, l'objet `Personnel` est créé et transmis au contrôleur.


```
private void BtnValiderAjout_Click(object sender, EventArgs e)
{
    if (!txtNom.Text.Equals("") && !txtPrenom.Text.Equals("") && !txtTel.Text.Equals("") && !txtMail.Text.Equals("") && cboService.SelectedIndex != -1)
    {
        lblProblemeChamps.Visible = false;
        Personnel personnel = new Personnel(0, txtNom.Text, txtPrenom.Text, txtTel.Text, txtMail.Text, (Service)cboService.SelectedItem);
        controller.AjoutPersonnel(personnel);
        RemplirListePersonnel();
        ViderChampsTexte();
    }
    else
    {
        lblProblemeChamps.Visible = true;
    }
}
```

Méthode qui se déclenche lors du clic sur le bouton "Valider"

Le contrôleur récupère l'objet et appelle la méthode `AjoutPersonnel` que j'ai préalablement créée dans la classe `PersonnelAccess`.

```
public void AjoutPersonnel(Personnel personnel)
{
    // Vérification que l'accès à la base de données est bien établi avant de lancer la requête
    if (access.Manager != null)
    {
        // Construction de la requête de type INSERT pour ajouter un nouveau personnel
        string requete = "INSERT INTO personnel(nom, prenom, tel, mail, idservice)";
        requete += "VALUES(@nom, @prenom, @tel, @mail, @idservice)";
        // Création d'un dictionnaire pour stocker les paramètres de la requête
        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);

        // Tentative d'exécution de la requête SQL
        try
        {
            access.Manager.ReqUpdate(requete, parameters);
        }
        // Gestion des erreurs
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Environment.Exit(0);
        }
    }
}
```

Méthode pour ajouter un personnel dans la base de données

Cette dernière a pour rôle d'insérer les informations du nouvel employé dans la base de données en utilisant un dictionnaire de paramètres afin de sécuriser les entrées SQL. Lorsque cet ajout est effectué, la méthode `ViderChampsTexte` de la vue permet de réinitialiser les `TextBox`, désactiver la saisie d'informations et réactiver les boutons de sélection d'une action. La méthode `RemplirListePersonnel` est également utilisée afin de mettre à jour la liste du `DataGriedView` avec les nouvelles informations de la base de données.

Liste du personnel

	Nom	Prenom	Tel	Mail
▶	Chaney	Mia	08 35 96 54 46	ipsum@aol.couk
	Crosby	Aretha	07 31 91 67 79	metus.aliquam@hotmail
	Dillon	Elton	04 05 91 87 92	molestie.orci@aol.org
	Farley	Hanna	08 15 44 83 11	viverra@protonmail.ca
	Lowery	Tara	05 79 73 68 45	augue.id@icloud.couk
	Mcintyre	Emery	03 83 01 49 95	mauris@protonmail.ec
	Mclean	Helen	04 47 22 27 94	eros.non@google.edu
	Phillips	Alma	08 75 12 56 61	fringilla.purus.mauris@
	Richards	Arden	08 76 08 88 47	proin.non@aol.net
	Stephenson	Kiyada	07 96 43 15 20	tempor.arcu.vestibulur

Ajouter un personnel
Modifier un personnel
Supprimer un personnel

Saisie des informations du personnel

Nom : Demange
Prénom : Aurelie
Tel : 06 06 06 06 06
Mail : ademange@gmail.com
Service d'affectation : administratif

Valider l'ajout
Valider les modifications

Liste du personnel				
	Nom	Prenom	Tel	Mail
▶	Chaney	Mia	08 35 96 54 46	ipsum@aol.couk
	Crosby	Aretha	07 31 91 67 79	metus.aliquam@hotmail
	Demange	Aurelie	06 06 06 06 06	ademange@gmail.com
	Dillon	Elton	04 05 91 87 92	molestie.orci@aol.org

DataGridView avant/après l'ajout du nouveau personnel

To do 12 Estimate: 0

AppGestionPersonnel #7

Coder le cas d'utilisation n°3 : supprimer un personnel

In progress 0 Estimate: 0

Done 6 Estimate: 0

AppGestionPersonnel #6 ...


Coder le cas d'utilisation n°2 : Ajouter un personnel

La tâche n°6 est passée en 'Done' lors du Commit and Push

C) Supprimer un personnel

Cette fonctionnalité permet de retirer un personnel de la base de données et, par conséquent, de la liste affichée dans le DataGridView. L'utilisateur sélectionne un employé dans ce dernier et clique sur le bouton "Supprimer un personnel". Dans la méthode correspondant à ce bouton, j'ai inclus une condition afin de vérifier que l'utilisateur a bien sélectionné une ligne. Ensuite, l'application demande une confirmation à l'utilisateur via une MessageBox, ce qui permet d'éviter une suppression accidentelle puisque cette action est définitive.

Confirmation


Voulez-vous vraiment supprimer Aurelie Demange ?

Oui
Non

MessageBox pour confirmer la suppression d'un personnel

```

private void BtnSupprimerPersonnel_Click(object sender, EventArgs e)
{
    if (dgvPersonnel.SelectedRows.Count > 0)
    {
        // Récupère l'objet 'Personnel' directement depuis la ligne sélectionnée
        Personnel personnel = (Personnel)dgvPersonnel.SelectedRows[0].DataBoundItem;
        if (MessageBox.Show($"Voulez-vous vraiment supprimer {personnel.Prenom} {personnel.Nom} ?", "Confirmation"))
        {
            controller.SupprimerPersonnel(personnel);
            RemplirListePersonnel();
            lblAucuneSelection.Visible = false;
        }
    }
    else
    {
        lblAucuneSelection.Visible = true;
    }
}

```

Méthode pour supprimer un personnel de la base de données

Si l'utilisateur valide, l'objet `Personnel` est transmis au contrôleur, et celui-ci appelle la méthode `SupprimerPersonnel` que j'ai préalablement créée dans la classe `PersonnelAccess`. Dans cette méthode se trouve la requête SQL `DELETE` afin de supprimer le personnel sélectionné, toujours avec l'utilisation d'un paramètre pour la sécuriser et récupérer l'`idPersonnel`.

```

public void SupprimerPersonnel(Personnel personnel)
{
    // Vérification que l'accès à la base de données est bien établi avant de lancer la requête
    if (access.Manager != null)
    {
        // Construction de la requête de type DELETE pour supprimer un personnel
        string requete = "DELETE FROM personnel WHERE idpersonnel = @idpersonnel;";
        // Création d'un dictionnaire pour stocker les paramètres de la requête
        Dictionary<string, object> parameters = new Dictionary<string, object>();
        parameters.Add("@idpersonnel", personnel.Idpersonnel);

        // Tentative d'exécution de la requête SQL
        try
        {
            access.Manager.ReqUpdate(requete, parameters);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Environment.Exit(0);
        }
    }
}

```

Méthode SupprimerPersonnel

Une fois la suppression effectuée, la liste du personnel, et donc le `DataGridView`, est mise à jour.

Liste du personnel				
	Nom	Prenom	Tel	Mail
▶	Chaney	Mia	08 35 96 54 46	ipsum@aol.couk
	Crosby	Aretha	07 31 91 67 79	metus.aliquam@hotmail.com
	Demange	Aurelie	06 06 06 06 06	ademange@gmail.com

Liste du personnel				
	Nom	Prenom	Tel	Mail
▶	Chaney	Mia	08 35 96 54 46	ipsum@aol.couk
	Crosby	Aretha	07 31 91 67 79	metus.aliquam@hotmail.com
	Dillon	Elton	04 05 91 87 92	molestie.orci@aol.org
	Farley	Hanna	08 15 44 83 11	viverra@protonmail.ca

DataGridView avant/après la suppression



La tâche n°7 est passée en 'Done' lors du Commit and Push

D) Modifier un personnel

L'objectif de cette fonctionnalité est de mettre à jour les informations d'un personnel existant dans la base de données depuis l'interface utilisateur. L'utilisateur peut mettre une ou plusieurs informations (nom, prenom, tel, mail, service d'affectation).

Tout d'abord, il doit avoir sélectionné un salarié dans le DataGridView et cliquer sur le bouton "Modifier un personnel". Dans le code, j'ai intégré une condition afin de vérifier qu'une ligne est bien sélectionnée et, dans le cas contraire, un label s'affichera pour expliquer le problème à l'utilisateur. Le formulaire devient disponible et pré-rempli avec les informations actuelles du personnel sélectionné. Pour cela, l'objet `personnelSelection` est récupéré depuis la ligne sélectionnée du DataGridView et contient toutes les données du personnel choisi. Celles-ci sont ensuite injectées dans les champs de texte correspondants.

```
personnelSelection = (Personnel)dgvPersonnel.SelectedRows[0].DataBoundItem;

//Remplissage des champs de texte avec les informations du personnel sélectionné
txtNom.Text = personnelSelection.Nom;
txtPrenom.Text = personnelSelection.Prenom;
txtTel.Text = personnelSelection.Tel;
txtMail.Text = personnelSelection.Mail;
```

Extrait de code concernant le pré-remplissage des TextBox

Concernant le ComboBox de la liste des services, celui-ci sélectionne aussi automatiquement le service du personnel choisi grâce à une petite condition dans le code.

```
//Remplissage du combobox avec le service correspondant au personnel sélectionné
if (personnelSelection.Service != null && personnelSelection.Service.Idservice > 0)
{
    cboService.SelectedValue = personnelSelection.Service.Idservice;
}
else
{
    cboService.SelectedIndex = -1;
}
```

Extrait de code concernant la sélection automatique dans le ComboBox

Si un service est associé au personnel, la valeur correspondante sera sélectionnée dans le ComboBox. Si ce n'est pas le cas, l'index sera mis sur -1, c'est-à-dire aucune sélection.

Liste du personnel

	Nom	Prenom	Tel	Mail
	Chaney	Mia	08 35 96 54 46	ipsum@aol.couk
	Crosby	Aretha	07 31 91 67 79	metus.aliquam@hotmail.com
	Dillon	Elton	04 05 91 87 92	molestie.orci@aol.org
	Farley	Hanna	08 15 44 83 11	viverra@protonmail.ca
▶	Lowery	Tara	05 79 73 68 45	augue.id@icloud.couk
	Mcintyre	Emery	03 83 01 49 95	mauris@protonmail.ec
	Mclean	Helen	04 47 22 27 94	eros.non@google.edu
	Phillips	Alma	08 75 12 56 61	fringilla.purus.mauris@
	Richards	Arden	08 76 08 88 47	proin.non@aol.net
	Stephenson	Kiayada	07 96 43 15 20	tempor.arcu.vestibulum

Ajouter un personnel
Modifier un personnel
Supprimer un personnel

Saisie des informations du personnel

Nom :

Prénom :

Tel :

Mail :

Service d'affectation :

Interface graphique lors d'une modification en cours d'un personnel

Une fois que l'utilisateur a effectué ses modifications, il doit cliquer sur le bouton "Valider" pour appliquer les changements (le bouton "Annuler" est également accessible de la même manière que dans les étapes précédentes). Avant d'envoyer les données, j'ai intégré une vérification afin de s'assurer que tous les champs sont bien remplis. Si c'est le cas, un message de confirmation s'affiche, demandant à l'utilisateur de cliquer sur oui ou non. S'il choisit non, il retournera à l'étape de sélection d'une action. Cela permet d'éviter les erreurs liées à une mauvaise manipulation.

Confirmer la modification

Étes-vous sûr de vouloir enregistrer ces modifications ?

MessageBox pour confirmer la modification d'un personnel

Si l'utilisateur confirme, les données affectées à l'objet `personnelSelection` sont envoyées au contrôleur qui appellera la méthode `ModifierPersonnel(Personnel personnel)` et celle-ci transmettra ensuite les données à `PersonnelAccess`, la couche d'accès aux données.

```

public void ModifierPersonnel(Personnel personnel)
{
    personnelAccess.ModifierPersonnel(personnel);
}

```

Méthode ModifierPersonnel de la classe FrmGestionPersonnelController

Dans la classe `PersonnelAccess`, j'ai créé la méthode `ModifierPersonnel` qui gère l'envoi de la requête SQL de type UPDATE à la base de données via les paramètres qui sont récupérés depuis l'objet `Personnel`.

```

public void ModifierPersonnel(Personnel personnel)
{
    // Vérification que l'accès à la base de données est bien établi avant de lancer la requête
    if (access.Manager != null)
    {
        // Construction de la requête de type UPDATE pour modifier un personnel
        string requete = "UPDATE personnel SET nom = @nom, prenom = @prenom, tel = @tel, mail = @mail, idservice = @idservice WHERE idpersonnel = @idpersonnel;";
        // Création d'un dictionnaire pour stocker les paramètres de la requête
        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);
        parameters.Add("@idpersonnel", personnel.Idpersonnel);

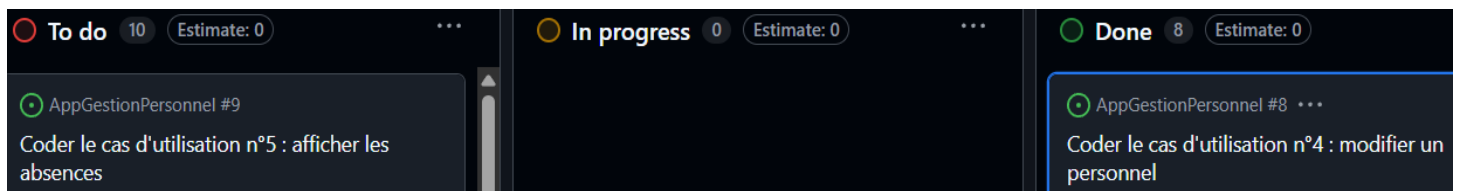
        // Tentative d'exécution de la requête SQL
        try
        {
            access.Manager.ReqUpdate(requete, parameters);
        }
        // Gestion des erreurs
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Environment.Exit(0);
        }
    }
}

```

Méthode ModifierPersonnel de la classe PersonnelAccess

L'exécution de la requête se fait grâce à `access.manage.ReqUpdate()`, qui gère l'envoi et la mise à jour des données. Comme on l'a vu précédemment, la méthode `ReqUpdate` est une méthode de `BddManager` qui gère l'exécution de les requêtes SQL de type INSERT, UPDATE ou DELETE. Elle prend en paramètre la requête et un dictionnaire de paramètres, prépare la commande, injecte les paramètres, puis lance l'exécution sans retour de données. Elle permet de mettre à jour les données de manière sécurisée et centralisée.

Les modifications effectuées sont renvoyées au contrôleur, qui les transmet à la classe `FrmGestionPersonnel` (dans le package View) à la demande de cette dernière afin de mettre à jour le `DataGridView` avec les nouvelles données. Je n'ajoute pas de capture d'écran sur le avant/après modifications car il s'agit du même principe que l'ajout ou la suppression d'un personnel pour la mise à jour du `DataGridView`.



La tâche n°8 est passée en 'Done' lors du Commit and Push

E) Afficher les absences

L'utilisateur peut afficher les absences grâce au bout "Afficher les absences". Cela ouvre une nouvelle fenêtre afin de gérer les absences du personnel qui aura été sélectionné au préalable. Tout d'abord, une vérification est effectuée pour s'assurer qu'une ligne est bien sélectionnée dans le `DataGridView`. Si ce n'est pas le cas, un label est rendu visible pour le signaler à l'utilisateur. En revanche, si une ligne est sélectionnée, l'objet `Personnel` lié à cette ligne est récupéré, permettant ainsi d'obtenir l'identifiant du personnel (`idPersonnel`), essentiel pour retrouver ses absences.

La fenêtre principale est temporairement masquée avec la commande `this.Hide()` et une nouvelle instance de `FrmGestionAbsences` est créée avec l'identifiant du personnel sélectionné passé en paramètre. Cette nouvelle fenêtre affichera uniquement la liste des absences de ce personnel. À la fermeture de cette fenêtre ou en cliquant sur le bouton "Retour à la gestion du

personnel", la fenêtre principale sera de nouveau affichée grâce à `this.Show()`. J'ai choisi de masquer la fenêtre principale afin d'éviter de surcharger l'écran et de conserver une interface lisible pour l'utilisateur.

```
/// <summary>
/// Événement déclenché lors du clic sur le bouton "Afficher les absences"
/// Affiche les absences du personnel sélectionné dans une nouvelle fenêtre
/// </summary>
/// <param name="sender">Objet qui a déclenché l'événement</param>
/// <param name="e">Arguments de l'événement</param>
1 référence | aurelied1991, il y a 5 jours | 1 auteur, 3 modifications
private void BtnAfficherAbsences_Click(object sender, EventArgs e)
{
    Personnel personnelSelectionne = (Personnel)dgvPersonnel.SelectedRows[0].DataBoundItem;
    if (dgvPersonnel.SelectedRows.Count > 0)
    {
        int idPersonnel = personnelSelectionne.Idpersonnel;
        this.Hide();
        FrmGestionAbsences frm = new FrmGestionAbsences(idPersonnel);
        frm.ShowDialog();
        this.Show();
    }
    else
    {
        lblAucuneSelection.Visible = true;
    }
}
```

Méthode permettant d'afficher les absences du personnel sélectionné

Cette nouvelle fenêtre permet donc de gérer les absences du personnel sélectionné. Pour respecter l'architecture MVC, j'ai aussi créé une classe `FrmGestionAbsencesController` dans la package `control` afin de séparer la logique métier de l'interface graphique.

Lors de la création de la fenêtre, en recevant l'identifiant du personnel sélectionné, le contrôleur est initialisé, celui-ci étant la passerelle entre la vue et les données (modèle). Il utilise deux classes pour accéder aux données, qui permettent d'initialiser la fenêtre de gestion des absences.

```

public class FrmGestionAbsencesController
{
    // Objet d'accès aux opérations possibles sur Absences
    private readonly AbsencesAccess absencesAccess;
    // Objet d'accès aux opérations possibles sur Motif
    private readonly MotifAccess motifAccess;

    /// <summary>
    /// Constructeur de la classe et initialise les accès aux données pour opérations sur les absences et les motifs
    /// </summary>
    /// <param name="idPersonnel">id du personnel</param>
    1 référence | aurelied1991, il y a 5 jours | 1 auteur, 2 modifications
    public FrmGestionAbsencesController(int idPersonnel)
    {
        absencesAccess = new AbsencesAccess();
        motifAccess = new MotifAccess();
    }

    /// <summary>
    /// Récupère la liste des absences d'un personnel donné
    /// </summary>
    /// <param name="idPersonnel">Identifiant du personnel pour lequel récupérer les absences</param>
    /// <returns>Liste des absences pour le personnel donné</returns>
    1 référence | aurelied1991, il y a 5 jours | 1 auteur, 2 modifications
    public List<Absences> GetLesAbsences(int idPersonnel)
    {
        return absencesAccess.GetLesAbsences(idPersonnel);
    }

    /// <summary>
    /// Récupère la liste des motifs d'absence disponibles dans la base de données
    /// </summary>
    /// <returns>Liste des motifs disponibles pour les absences</returns>
    1 référence | aurelied1991, il y a 5 jours | 1 auteur, 3 modifications
    public List<Motif> GetLesMotifs()
    {
        return motifAccess.GetLesMotifs();
    }
}

```

Classe FrmGestionAbsencesController avec les méthodes permettant d'accéder aux classes et méthodes pour accéder aux données

La première classe est AbsencesAccess qui construit et exécute la requête SQL permettant de récupérer la liste des absences liées au personnel sélectionné, ainsi que la liste des motifs associés (en joignant la table des motifs dans la requête). La deuxième classe est MotifAccess qui, par l'exécution d'une requête SQL, récupère la liste des motifs existants.

Je ne mettrai pas de captures d'écran de ces deux méthodes ici (pour ne pas encombrer le compte-rendu) : celle pour récupérer la liste des motifs est construite de la même manière que celle pour récupérer la liste des services. Quant à celle pour récupérer la liste des absences, elle fonctionne comme pour la liste des personnels sauf qu'elle utilise un paramètre afin de filtrer les absences pour n'afficher que celles du personnel sélectionné.

Grâce à ces deux classes, le contrôleur récupère ces listes et la vue affiche les absences dans un DataGridView avec leur motif associé. J'ai aussi masqué les colonnes inutiles pour l'utilisateur, comme les id, et formaté les dates pour une meilleure lisibilité. Quant au ComboBox cboMotifAbsence, il est rempli avec la liste des motifs d'absence.


```

public void RemplirListeAbsences()
{
    // Récupération de la liste des absences depuis le contrôleur
    List<Absences> lesAbsences = controller.GetLesAbsences(idPersonnel);
    // Affectation de la liste au DataGridView
    dgvAbsences.DataSource = lesAbsences;
    // Masquage des colonnes inutiles
    dgvAbsences.Columns["IdPersonnel"].Visible = false;
    dgvAbsences.Columns["AbsenceId"].Visible = false;
    // Formatage des colonnes pour afficher les dates au format "dd/MM/yyyy"
    dgvAbsences.Columns["Datedebut"].DefaultCellStyle.Format = "dd/MM/yyyy";
    dgvAbsences.Columns["Datefin"].DefaultCellStyle.Format = "dd/MM/yyyy";
    // Ajustement de la largeur des colonnes pour s'adapter au contenu
    dgvAbsences.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
}

```

Méthode de la classe FrmGestionAbsences pour remplir le DataGridView avec la liste des absences du personnel sélectionné

```

public void RemplirListeMotifs()
{
    // Récupération de la liste des motifs depuis le contrôleur
    List<Motif> lesMotifs = controller.GetLesMotifs();
    // Affectation de la liste des motifs au ComboBox
    cboMotifAbsence.DataSource = lesMotifs;
    cboMotifAbsence.DisplayMember = "Libelle";
    cboMotifAbsence.ValueMember = "IdMotif";
    cboMotifAbsence.SelectedIndex = -1;
}

```

Méthode de la classe FrmGestionAbsences pour remplir le ComboBox avec la liste des motifs d'absence

Gestion des absences du personnel

Liste des absences

	Datedebut	Datefin	Motif
▶	04/10/2024	09/10/2024	vacances
	11/07/2024	02/09/2024	maladie
	01/12/2023	16/12/2023	maladie
	24/11/2023	30/11/2023	congé parental
	29/05/2023	18/08/2023	maladie

Saisie des informations sur l'absence

Date de début de l'absence : 11/05/2025

Date de fin de l'absence : 25/05/2025

Motif de l'absence :

Enregistrer

Enregistrer les modifications

Annuler

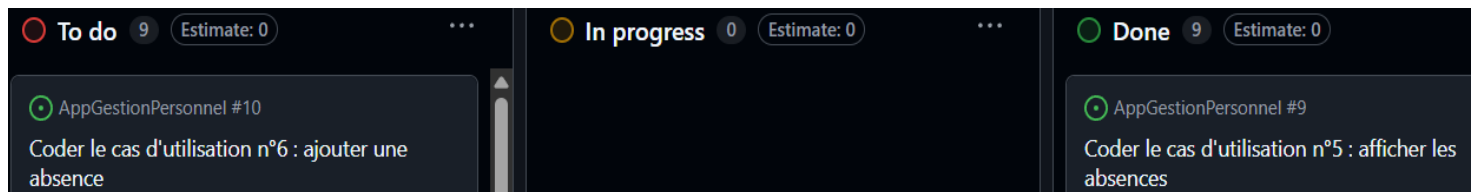
Ajouter une absence

Modifier une absence

Supprimer une absence

Retour à la gestion du personnel

Interface de gestion des absences du personnel sélectionné avec la liste affichée dans le DataGridView



La tâche n°9 est passée en 'Done' lors du Commit and Push

F) Ajouter une absence

L'objectif de cette fonctionnalité est de permettre l'ajout d'une nouvelle absence pour le personnel sélectionné, tout en faisant attention à l'intégrité des données, notamment en évitant les conflits de dates avec les absences déjà existantes ou bien les dates incohérentes.

Pour commencer, lorsque l'utilisateur clique sur le bouton "Ajouter une absence", le formulaire de saisie est activé tandis que la liste des absences est désactivée. Cela permet d'éviter que l'utilisateur modifie autre chose par erreur sans s'en rendre compte, par exemple en cliquant sur le bouton "Supprimer" .

```
private void btnAjoutAbsence_Click(object sender, EventArgs e)
{
    gboSaisieInfosAbsence.Enabled = true;
    btnEnregistrerAbsence.Enabled = true;
    btnAnnulerAction.Enabled = true;
    gboListeAbsences.Enabled = false;
}
```

Modification du formulaire de saisie lors du clic sur le bouton "Ajouter une absence"

Ensuite, l'utilisateur doit saisir l'ensemble des informations nécessaires puis cliquer sur le bouton "Enregistrer l'absence". Plusieurs vérifications sont alors effectuées avant que l'absence ne soit réellement ajoutée à la base de données. Tous les champs doivent être remplis, sinon un label d'erreur s'affiche. Pour s'assurer de la cohérence des données, la date de fin ne peut pas être antérieure à la date de début de l'absence, sinon un label est également affiché pour signaler à l'utilisateur le problème. Pour finir, pour garantir l'intégrité des données et leur logique, les dates ne doivent pas entrer en conflit avec une absence déjà existante pour ce personnel, sinon un label est aussi affiché.

Liste des absences

	Datedebut	Datefin	Motif
▶	04/10/2024	09/10/2024	vacances
	11/07/2024	02/09/2024	maladie
	01/12/2023	16/12/2023	maladie
	24/11/2023	30/11/2023	congé parental
	29/05/2023	18/08/2023	maladie

Saisie des informations sur l'absence

Date de début de l'absence : 11/05/2025

Date de fin de l'absence : 25/05/2025

Motif de l'absence : motif familial

Enregistrer

Enregistrer les modifications

Annuler

Retour à la gestion du personnel

Ajouter une absence


Modifier une absence


Supprimer une absence

Interface lors de la saisie d'une nouvelle absence


	Datedebut	Datefin	Motif
▶	11/05/2025	25/05/2025	motif familial
	04/10/2024	09/10/2024	vacances
	11/07/2024	02/09/2024	maladie
	01/12/2023	16/12/2023	maladie
	24/11/2023	30/11/2023	congé parental
	29/05/2023	18/08/2023	maladie


Mise à jour de la liste avec la nouvelle absence


Date de début de l'absence : 25/05/2025 

Date de fin de l'absence : 13/05/2025 


La date de fin ne peut pas être antérieure à la date de début

Motif de l'absence : congé parental 

Date de début de l'absence : 12/05/2025 

Date de fin de l'absence : 14/05/2025 

Erreur, une absence est déjà enregistrée pour cette période

Motif de l'absence : congé parental 

Affichages de messages d'erreurs lorsque l'utilisateur ajoute des dates incohérentes

```

private void btnEnregistrerAbsence_Click(object sender, EventArgs e)
{
    // Masque les messages d'erreur à chaque nouvelle tentative d'ajout
    lblProblemeDate.Visible = false;
    lblCreneauNonLibre.Visible = false;

    //Vérification des champs obligatoires
    if (cboMotifAbsence.SelectedIndex != -1 && dtpDebutAbsence.Value != null && dtpFinAbsence.Value != null)
    {
        // Vérification que la date de début est antérieure à la date de fin
        if (dtpDebutAbsence.Value < dtpFinAbsence.Value)
        {
            List<Absences> lesAbsences = (List<Absences>)dgvAbsences.DataSource;
            bool creneauLibre = true;
            foreach (Absences absences in lesAbsences)
            {
                // Vérification de l'absence de conflit de dates avec les autres absences
                if (dtpDebutAbsence.Value < absences.Datefin && dtpFinAbsence.Value > absences.Datedebut)
                {
                    creneauLibre = false;
                    break;
                }
            }
            if (creneauLibre)
            {
                //Création et ajout de l'absence dans la base de données
                Absences absence = new Absences(idPersonnel, dtpDebutAbsence.Value, dtpFinAbsence.Value, (Motif)cboMotifAbsence.SelectedItem);
                controller.AjoutAbsence(absence);
                RemplirListeAbsences();
                ReinitialiserFormulaire();
            }
            else
            {
                lblCreneauNonLibre.Visible = true;
            }
        }
        else
        {
            lblProblemeDate.Visible = true;
        }
    }
    else
    {
        MessageBox.Show("Veuillez remplir tous les champs obligatoires.");
    }
}

```

Méthode qui vérifie que la saisie des informations pour l'ajout d'une nouvelle absence est correcte et intègre celle-ci à la liste

Si tout est correct, un objet de type Absences est créé et envoyé à la méthode AjoutAbsence(Absences absence) du contrôleur.

```

public void AjoutAbsence(Absences absence)
{
    absencesAccess.AjoutAbsence(absence);
}

```

Méthode AjoutAbsence du contrôleur

Cette dernière appelle la méthode correspondante dans la classe AbsencesAccess, AjoutAbsence(Absences absence). Cette dernière prépare et exécute la requête SQL avec des paramètres sécurisés grâce à un dictionnaire afin d'éviter les risques d'injection SQL.


```

public void AjoutAbsence(Absences absences)
{
    // Vérification que la connexion à la base de données est bien établie avant de lancer la requête
    if (access.Manager != null)
    {
        // Construction de la requête SQL pour insérer une nouvelle absence dans la base de données
        string requete = "INSERT INTO absence(idpersonnel, datedebut, datefin, idmotif) ";
        requete += "VALUES(@idpersonnel, @datedebut, @datefin, @idmotif)";
        // Préparation des paramètres nécessaires pour l'insertion dans la bdd
        Dictionary<string, object> parameters = new Dictionary<string, object>();
        parameters.Add("@idpersonnel", absences.Idpersonnel);
        parameters.Add("@datedebut", absences.Datedebut);
        parameters.Add("@datefin", absences.Datefin);
        parameters.Add("@idmotif", absences.Motif.Idmotif);

        // Tentative d'exécution de la requête SQL pour insérer la nouvelle absence
        try
        {
            access.Manager.ReqUpdate(requete, parameters);
        }
        // Gestion des erreurs lors de l'exécution de la requête SQL
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Environment.Exit(0);
        }
    }
}

```

Méthode AjoutAbsence de la classe AbsencesAccess

Une fois la nouvelle absence ajoutée, le DataGridView est mis à jour avec la nouvelle absence. Le formulaire est réinitialisé grâce à la méthode ReinitialiserFormulaire afin de vider les champs, désactiver la saisie puis réactiver l'accès à la liste des absences et aux boutons d'action. Tout au long de l'étape (mais aussi pour modifier une absence), l'utilisateur peut cliquer sur le bouton "Annuler" afin d'enclencher le même processus de réinitialisation et revenir au point de départ.



La tâche n°10 est passée en 'Done' lors du Commit and Push

G) Supprimer une absence

L'objectif de cette fonctionnalité est de permettre à l'utilisateur de supprimer une absence déjà enregistrée pour le personnel sélectionné.

Tout d'abord, l'utilisateur doit sélectionner une absence à supprimer. Sinon, lors du clic sur le bouton "Supprimer une absence", un label s'affichera pour signaler l'erreur. Si une ligne est bien sélectionnée, une boîte de dialogue s'ouvrira afin de demander à l'utilisateur de confirmer son souhait de supprimer l'absence. Cela permet d'assurer l'intégrité des données et d'éviter toute suppression accidentelle. Si l'utilisateur confirme, l'absence sera supprimée, sinon, l'action est annulée.

```
private void btnSupprimerAbsence_Click(object sender, EventArgs e)
{
    if (dgvAbsences.SelectedRows.Count > 0)
    {
        lblAucuneSelection.Visible = false;
        // Récupération de l'absence sélectionnée
        Absences absence = (Absences)dgvAbsences.SelectedRows[0].DataBoundItem;
        // Demande de confirmation à l'utilisateur avant de supprimer l'absence
        if (MessageBox.Show("Êtes-vous sûr de vouloir supprimer cette absence ?", "Confirmer la suppression", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
        {
            // Suppression de l'absence via le contrôleur
            controller.SupprimerAbsence(absence);
            // Mise à jour de la liste des absences affichée
            RemplirListeAbsences();
        }
    }
    else
    {
        lblAucuneSelection.Visible = true;
    }
}
}
```

Événement du clic sur le bouton de suppression d'une absence

Si la suppression est confirmée, l'absence est transmise au contrôleur via la méthode `SupprimerAbsence()`, qui appelle ensuite la méthode du même nom dans la classe `AbsencesAccess`. Celle-ci prépare et exécute la requête SQL de suppression de l'absence dans la base de données. Pour la requête, j'ai choisi d'utiliser uniquement `idPersonnel` et `dateDebut` comme critère de suppression car ces deux champs suffisent pour identifier de manière unique une absence. Cela permet une suppression précise en évitant les erreurs de suppression multiple. Comme d'habitude, les paramètres sont protégés contre les risques d'injection SQL grâce à l'utilisation d'un dictionnaire.

```
public void SupprimerAbsence(Absences absence)
{
    absencesAccess.SupprimerAbsence(absence);
}
```

Méthode `SupprimerAbsence` du contrôleur

```
// Construction de la requête SQL pour supprimer une absence de la base de données
string requete = "DELETE FROM absence WHERE idpersonnel = @idpersonnel AND datedebut = @datedebut;";
// Préparation des paramètres nécessaires pour la suppression dans la bdd
Dictionary<string, object> parameters = new Dictionary<string, object>();
parameters.Add("@idpersonnel", absences.Idpersonnel);
parameters.Add("@datedebut", absences.Datedebut.Date);
```

Requête SQL pour supprimer une absence dans la base de données


Une fois l'absence supprimée, la liste dans le `DataGridView` est mise à jour avec la méthode `RemplirListeAbsences()` afin de refléter l'état actuel de la base de données.

Liste des absences

	Datedebut	Datefin	Motif
▶ ▶	04/10/2024	09/10/2024	vacances
	11/07/2024	02/09/2024	maladie
	01/12/2023	16/12/2023	maladie
	24/11/2023	30/11/2023	congé parental
	29/05/2023	18/08/2023	maladie

Mise à jour de la liste avec la suppression de l'absence

Confirmer la suppression


 Êtes-vous sûr de vouloir supprimer cette absence ?

Oui Non

Suppression d'une absence avec une fenêtre pour confirmer l'action

To do 7 Estimate: 0

AppGestionPersonnel #12

Coder le cas d'utilisation n°8 : modifier une absence

In progress 0 Estimate: 0

Done 11 Estimate: 0

AppGestionPersonnel #11 ...

Coder le cas d'utilisation n°7 : supprimer une absence

La tâche n°11 est passée en 'Done' lors du Commit and Push

H) Modifier une absence

L'objectif de cette fonctionnalité est de permettre à l'utilisateur de modifier les informations d'une absence déjà existante. Ceci est très pratique, par exemple, si un personnel prolonge un arrêt maladie, ou si l'utilisateur a entré de mauvaises informations à un moment donné. Si cette fonctionnalité n'existait pas, il faudrait ajouter une nouvelle absence.

Lorsque l'utilisateur clique sur le bouton "Modifier une absence", l'application vérifie qu'une absence est bien sélectionnée, et si ce n'est pas le cas, un label rouge s'affiche. Sinon, le formulaire de saisie des absences est activé tandis que la liste des absences et les boutons d'actions sont désactivés. Le programme récupère ensuite les informations de l'absence sélectionnée depuis le DataGridView puis les injecte dans les champs correspondants (DateTimePicker et ComboBox). L'identifiant temporaire de l'absence, `absenceIdTemporaire`, est conservé pour pouvoir l'identifier facilement lors de la validation des modifications.

```
// Identifiant temporaire de l'absence sélectionnée pour les modifications
private Guid absenceIdTemporaire;
```

Propriété privée `absenceIdTemporaire`

```
private void btnModifAbsence_Click(object sender, EventArgs e)
{
    if(dgvAbsences.SelectedRows.Count > 0)
    {
        lblAucuneSelection.Visible = false;
        gboSaisieInfosAbsence.Enabled = true;
        btnEnregistrerModifications.Enabled = true;
        btnAnnulerAction.Enabled = true;
        gboListeAbsences.Enabled = false;

        // Récupération les informations de l'absence sélectionnée
        Absences absence = (Absences)dgvAbsences.SelectedRows[0].DataBoundItem;
        absenceIdTemporaire = absence.AbsenceId;
        dtpDebutAbsence.Value = absence.Datedebut;
        dtpFinAbsence.Value = absence.Datefin;
        if(absence.Motif != null)
        {
            cboMotifAbsence.SelectedValue = absence.Motif.Idmotif;
        }
        else
        {
            cboMotifAbsence.SelectedIndex = -1;
        }
    }
    else
    {
        lblAucuneSelection.Visible = true;
    }
}
```

Événement lors du clic sur le bouton "Modifier une absence"

Ensuite, l'utilisateur doit cliquer sur le bouton "Valider les modifications". Cela entraîne plusieurs vérifications, sinon un label avec un message d'erreur s'affiche, ce qui permet de garantir l'intégrité des données. Tous les champs doivent être remplis, la date de fin ne doit pas être antérieure à la date de début et les nouvelles dates ne doivent pas entrer en conflit avec d'autres absences déjà enregistrées pour ce personnel.

J'ai mis en place un identifiant temporaire (`absenceIdTemporaire`) lorsque je me suis rendue compte que l'`idPersonnel` couplé avec la date de début posait problème lors d'une modification dans la base de données. Lors des vérifications, si on compare les dates avec toutes les absences

sans faire de distinction, on se retrouve à également comparer l'absence en cours de modification avec elle-même, ce qui entraîne la détection d'un chevauchement. L'identifiant temporaire permet de stocker l'identifiant unique de l'absence en cours de modification, ce qui permet, lors de la vérification, d'ignorer cette absence et de ne comparer qu'avec les autres absences. Cela permet de modifier l'absence sans déclencher d'erreur tout en évitant de laisser passer de véritables chevauchements avec d'autres absences.

```
private void btnEnregistrerModifications_Click(object sender, EventArgs e)
{
    // Masque les messages d'erreur à chaque nouvelle tentative
    lblProblemeDate.Visible = false;
    lblCreneauNonLibre.Visible = false;

    // Vérifie que tous les champs sont remplis
    if (cboMotifAbsence.SelectedIndex != -1 && dtpDebutAbsence.Value != null && dtpFinAbsence.Value != null)
    {
        // Vérifie que la date de début est antérieure à la date de fin
        if (dtpDebutAbsence.Value < dtpFinAbsence.Value)
        {
            List<Absences> lesAbsences = (List<Absences>)dgvAbsences.DataSource;
            bool creneauLibre = true;
            Absences absenceAModifier = null;

            foreach (Absences absences in lesAbsences)
            {
                // On vérifie seulement les absences qui ne sont pas en cours de modification
                if (absences.AbsenceId == absenceIdTemporaire)
                {
                    // On récupère l'ancienne absence
                    absenceAModifier = absences;
                }
                else if (dtpDebutAbsence.Value < absences.Datefin && dtpFinAbsence.Value > absences.Datedebut)
                {
                    creneauLibre = false;
                    break;
                }
            }
        }
    }

    // Si le créneau est libre et que l'absence à modifier a été trouvée
    if (creneauLibre && absenceAModifier != null)
    {
        // Demande de confirmation à l'utilisateur avant de modifier l'absence
        if (MessageBox.Show("Êtes-vous sûr de vouloir enregistrer ces modifications ?", "Confirmer la modification", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
        {
            // Création de l'objet Absences mis à jour
            Absences absence = new Absences(idPersonnel, dtpDebutAbsence.Value, dtpFinAbsence.Value, (Motif)cboMotifAbsence.SelectedItem);
            // Enregistrement des modifications dans la base de données via le contrôleur
            controller.ModifierAbsence(absence, absenceAModifier.Datedebut);
            // Mise à jour de la liste des absences affichée et réinitialisation du formulaire
            RemplirListeAbsences();
            ReinitialiserFormulaire();
        }
        else
        {
            ReinitialiserFormulaire();
        }
    }
    else
    {
        lblCreneauNonLibre.Visible = true;
    }
}
else
{
    lblProblemeDate.Visible = true;
}
```

Événement lors du clic sur le bouton "Enregistrer les modifications"

Si toutes les vérifications sont validées, une boîte de dialogue s'ouvre pour demander à l'utilisateur de confirmer, ce qui évite les accidents. Si l'utilisateur ne valide pas, l'application retourne au point de départ de la fenêtre de gestion des absences, sinon un nouvel objet de type Absences est créé avec les informations mises à jour.

Ensuite, le contrôleur appelle la méthode `ModifierAbsence` pour envoyer les modifications vers la base de données via la classe `AbsencesAccess`. Cette dernière, dans sa

méthode `ModifierAbsence`, exécute la requête `UPDATE`. La mise à jour est réalisée en identifiant l'absence par l'`idPersonnel` et les anciennes informations (celles d'origine, avant modification) de la date de début. En effet, les dates peuvent être modifiées mais sont nécessaires pour identifier l'absence car si on recherchait avec la nouvelle date, la requête ne trouverait rien à modifier. L'ancienne date sert donc de repère pour retrouver l'enregistrement d'origine afin de lui appliquer les modifications. Comme d'habitude, les paramètres sont protégés contre les risques d'injection SQL grâce à l'utilisation d'un dictionnaire de paramètres.

```
public void ModifierAbsence(Absences absence, DateTime ancienneDatedebut)
{
    absencesAccess.ModifierAbsence(absence, ancienneDatedebut);
}
```

Méthode `ModifierAbsence` du contrôleur

```
public void ModifierAbsence(Absences absences, DateTime ancienneDatedebut)
{
    // Vérification que l'accès à la base de données est bien établi avant de lancer la requête
    if (access.Manager != null)
    {
        // Construction de la requête SQL pour mettre les informations d'une absence à jour dans la bdd
        string requete = "UPDATE absence SET datedebut = @datedebut, datefin = @datefin, idmotif = @idmotif ";
        requete += "WHERE idpersonnel = @idpersonnel AND datedebut = @ancienneDatedebut";
        // Préparation des paramètres nécessaires pour la mise à jour dans la bdd
        Dictionary<string, object> parameters = new Dictionary<string, object>();
        parameters.Add("@idpersonnel", absences.Idpersonnel);
        parameters.Add("@ancienneDatedebut", ancienneDatedebut); // ICI on utilise l'ancienne date
        parameters.Add("@datedebut", absences.Datedebut); // ICI la nouvelle date
        parameters.Add("@datefin", absences.Datefin);
        parameters.Add("@idmotif", absences.Motif.Idmotif);

        // Tentative d'exécution de la requête SQL pour mettre à jour l'absence
        try
        {
            access.Manager.ReqUpdate(requete, parameters);
        }

        // Gestion des erreurs lors de l'exécution de la requête SQL
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Environment.Exit(0);
        }
    }
}
```

Méthode `ModifierAbsence` de la classe `AbsencesAccess`

Pour finir, la liste des absences dans le `DataGridView` est actualisée avec les nouvelles données. Le formulaire est réinitialisé grâce à la méthode `ReinitialiserFormulaire` afin de vider les champs, désactiver la saisie et réactiver l'accès à la liste des absences et aux boutons d'action.

Saisie des informations sur l'absence	
Date de début de l'absence :	<input type="text" value="12/05/2025"/>
Date de fin de l'absence :	<input type="text" value="19/05/2025"/>

Informations sur l'absence avant modification

Liste des absences

	Datedebut	Datefin	Motif
▶	11/05/2025	25/05/2025	motif familial
	04/10/2024	09/10/2024	vacances
	11/07/2024	02/09/2024	maladie
	01/12/2023	16/12/2023	maladie
	24/11/2023	30/11/2023	congé parental
	29/05/2023	18/08/2023	maladie

Saisie des informations sur l'absence

Date de début de l'absence : 11/05/2025

Date de fin de l'absence : 25/05/2025

Motif de l'absence : motif familial

	Datedebut	Datefin	Motif
▶	12/05/2025	19/05/2025	motif familial
	04/10/2024	09/10/2024	vacances
	11/07/2024	02/09/2024	maladie
	01/12/2023	16/12/2023	maladie
	24/11/2023	30/11/2023	congé parental
	29/05/2023	18/08/2023	maladie

Modification de la date de fin de l'absence puis mise à jour de la liste

To do 6 Estimate: 0

AppGestionPersonnel #13
More actions

Mettre à jour la documentation technique +
corriger les commentaires normalisés

In progress 0 Estimate: 0

Done 12 Estimate: 0

AppGestionPersonnel #12

Coder le cas d'utilisation n°8 : modifier une absence

La tâche n°12 est passée en 'Done' lors du Commit and Push

I) Retour à la gestion du personnel et déconnexion

J'ai trouvé important d'avoir un bouton de navigation sur chaque fenêtre pour revenir à la précédente. Sur la fenêtre de la gestion des absences, j'ai créé un bouton intitulé "Retour à la gestion du personnel". Lorsque l'utilisateur clique dessus, l'instruction suivante est exécutée :

```
private void btnRetourPersonnel_Click(object sender, EventArgs e)
{
    // Ferme la fenêtre de gestion des absences
    this.Close();
}
```

Événement lors du clic sur le bouton "Retour à la gestion du personnel"

Le `this.Close` permet de fermer la fenêtre actuelle, ce qui a pour effet de revenir automatiquement à la fenêtre parente, ici la fenêtre de gestion du personnel. Cela est possible car, lors de l'ouverture de la fenêtre de gestion des absences, j'ai utilisé la méthode `ShowDialog()`. Cette dernière suspend l'exécution de la fenêtre parente jusqu'à la fermeture de la fenêtre enfant. Dès que `this.Close` est exécuté, l'exécution reprend là où elle s'était arrêtée, ce qui entraîne le réaffichage de la fenêtre gestion du personnel.


```
private void BtnAfficherAbsences_Click(object sender, EventArgs e)
{
    if (dgvPersonnel.SelectedRows.Count > 0)
    {
        Personnel personnelSelectionne = (Personnel)dgvPersonnel.SelectedRows[0].DataBoundItem;
        int idPersonnel = personnelSelectionne.Idpersonnel;
        this.Hide();
        FrmGestionAbsences frm = new FrmGestionAbsences(idPersonnel);
        frm.ShowDialog();
        this.Show();
    }
}
```

Événement lors du clic sur le bouton "Afficher les absences" qui permet l'ouverture de la fenêtre de gestion des absences

Sur cette dernière, j'ai ajouté le bouton "Déconnexion" afin de revenir à la fenêtre d'authentification.

```
private void BtnDeconnexion_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Événement lors du clic sur le bouton "Déconnexion"

Ce `this.Close()` ferme complètement la fenêtre de gestion du personnel.

Le retour à la fenêtre de connexion se passe de la même manière que pour le retour à la fenêtre gestion du personnel. J'ai juste ajouté une réinitialisation des champs de texte lors de ce retour à la fenêtre afin de garantir la confidentialité des informations d'authentification et renforcer la sécurité de l'application.

```
// Si l'authentification est réussie, on cache la fenêtre d'authentification et on ouvre la fenêtre de gestion du personnel
this.Hide();
FrmGestionPersonnel frm = new FrmGestionPersonnel();
frm.ShowDialog();
this.Show();
// Réinitialisation des champs de saisie et cacher le message d'erreur
txtLogin.Clear();
txtPwd.Clear();
lblErreurLogin.Visible = false;
```

Réinitialisation des champs de saisie du formulaire d'authentification

3.10 Étape 10 : ajout/correction des commentaires normalisés et mise à jour de la documentation technique

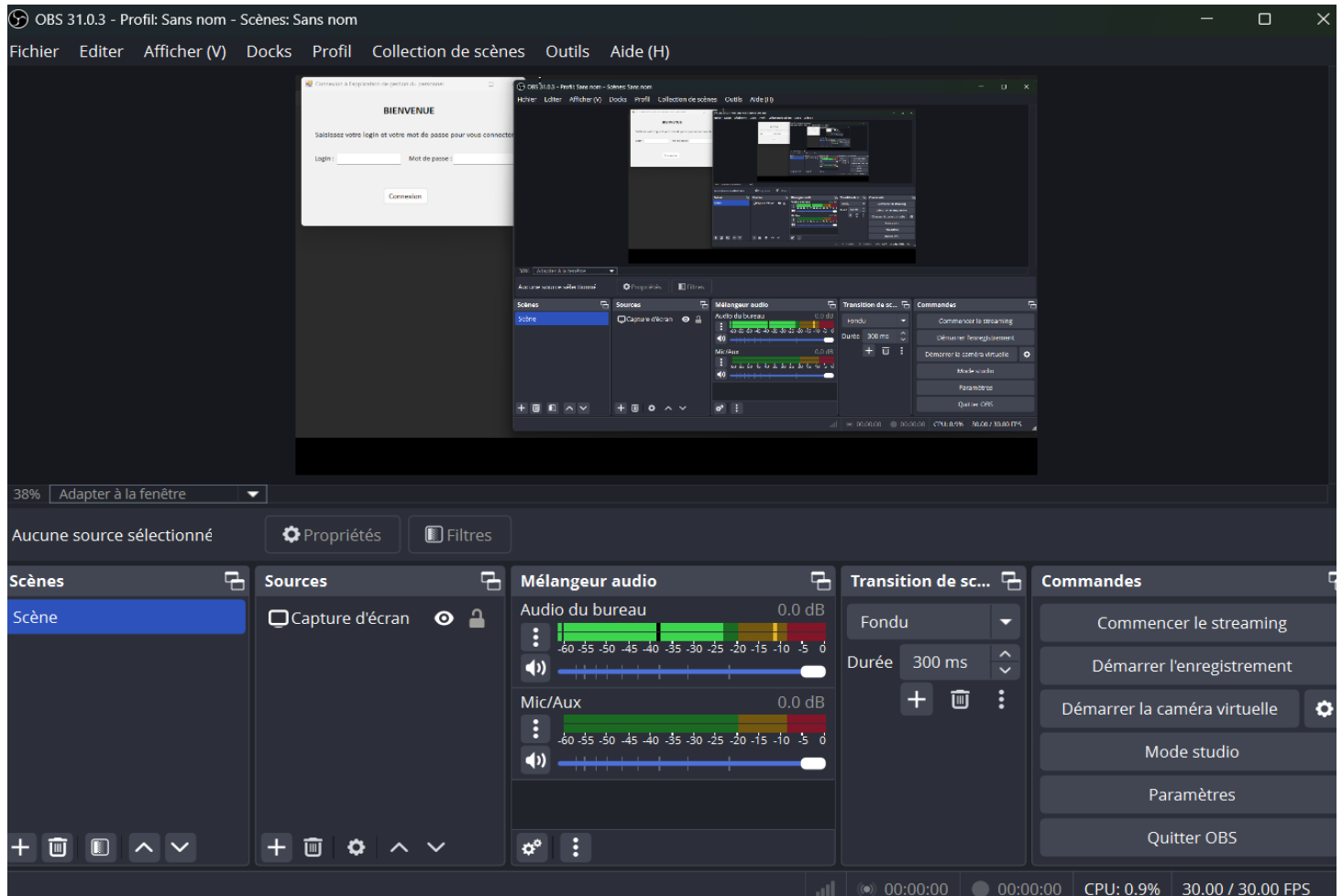
Pour assurer la maintenabilité et la compréhension du code, j'ai ajouté ou corrigé les commentaires normalisés sur l'ensemble des méthodes et des classes. Je n'entre pas ici dans les détails car tout est déjà expliqué dans l'étape de la première génération de la documentation.

Puis, j'ai régénéré la documentation technique afin d'inclure les modifications effectuées depuis la dernière génération. Ainsi, la documentation est maintenant à jour et reflète l'application dans sa version finale.

3.11 Étape 11 : production de la documentation utilisateur sous format vidéo

Pour réaliser la documentation utilisateur, j'ai utilisé le logiciel OBS Studio, que je maîtrise grâce à mon expérience passée. J'ai donc capturé mon écran tout en commentant oralement les différentes fonctionnalités de l'application.

L'objectif était de créer une vidéo claire et pédagogique d'environ 5 minutes, présentant une démonstration complète de l'application et de ses fonctionnalités. J'y explique les étapes de connexion, de déconnexion, ainsi que les différentes actions possibles dans la gestion du personnel, des absences, etc.



Capture d'OBS avant de lancer l'enregistrement de la vidéo de démonstration de l'application

Ce support visuel a pour but de permettre à l'utilisateur de comprendre rapidement et le plus simplement possible comment utiliser l'application. En effet, suivre un guide détaillé en vidéo, combinant explications orales et démonstrations à l'écran, est souvent plus intuitif qu'un guide écrit. C'est un support efficace et accessible pour la prise en main de l'application.

3.12 Étape 12 : création d'un installateur

Un installateur est un programme permettant de regrouper l'ensemble des fichiers nécessaires au bon fonctionnement d'une application et de les déployer sur un ordinateur. Il permet l'installation des fichiers, la création de raccourcis (bureau, barre des tâches...). L'objectif est que l'installation de l'application soit simplifiée autant que possible pour l'utilisateur final.

Pour obtenir cet installateur, j'ai commencé par ajouter l'extension nécessaire sur VS 2022, nommée Microsoft Visual Studio Installer Projects 2022, via le menu Extensions → Gérer les extensions. Une fois l'installation finalisée, j'ai créé un nouveau projet au niveau de la solution en choisissant le modèle "Setup Wizard". J'ai ensuite suivi les différentes étapes en choisissant "Create a setup for a Windows application", la sortie principale (donc l'exécutable de l'application) et en validant les options par défaut pour les fichiers à inclure. Pour que ce soit plus représentatif, j'ai également créé une icône au format .ico que j'ai intégrée dans le projet setup. J'ai aussi créé, avec cette même icône, des raccourcis dans le menu Démarrer et sur le bureau.

Puis, j'ai configuré les propriétés du projet setup en mettant mon nom comme auteur et le nom de l'application. J'ai généré la solution en mode release, ce qui a créé un fichier .msi (installateur) et un setup.exe (exécutable) dans le dossier release. Le mode release est optimisé pour l'exécution finale : il supprime les informations de débogage ce qui rend l'exécutable plus léger, le code généré est plus rapide, et il optimise certaines choses, par exemple en supprimant les variables non utilisées.

Pour finir, j'ai vérifié que mon installateur fonctionnait en copiant le fichier à un autre emplacement et en lançant l'installation. Le processus s'est déroulé sans problème, en s'installant dans le dossier Program Files et avec des raccourcis. L'application s'est ensuite lancée et a fonctionné sur l'ensemble de ses fonctionnalités.

4 Bilan final

Ce premier projet m'a permis de mettre en pratique et développer les compétences acquises durant ma formation, notamment dans le développement d'application de bureau en C# et la gestion des données SQL. Chaque étape a amené son lot de défis, mais m'a permis d'apprendre à résoudre des bugs et trouver des solutions même lorsque cela demandait des heures de travail. Et il faut bien l'avouer : quelle satisfaction de voir son petit projet enfin fonctionner après avoir débusqué et corrigé un bug récalcitrant. J'ai pu réaliser une application de A à Z : la conception des interfaces visuelles, l'implémentation des différentes fonctionnalités attendues pour la gestion du personnel et des absences, l'ajout de commentaires normalisés, la génération de la documentation technique, la réalisation de la vidéo de démonstration et la création de l'installateur.

Pour clore ce projet, j'ai rédigé le présent compte-rendu détaillant les différentes étapes suivies ainsi que les choix techniques effectués. J'ai aussi généré un script complet de la base de données, garantissant une installation fiable et rapide sur d'autres environnements. Pour finir, j'ai créé une page dédiée à ce projet sur mon portfolio afin de le présenter de manière claire et professionnelle.

En conclusion, ce projet a été une véritable expérience formatrice, me permettant de consolider mes compétences tout en découvrant les différentes phases d'un projet de développement d'une application, de la conception jusqu'à la mise en production. Cela me donne encore plus envie de réaliser de nouveaux projets afin de continuer à progresser et à avancer dans cette voie professionnelle.