COMPTE RENDU D'ACTIVITE :

DEVELOPPEMENT DE L'APPLICATION DE BUREAU

MediaTek86

Table des matières

Contexte

InfoTech Services 86 (ITS86) est une ESN spécialisée dans le développement informatique (applications de bureau, web, mobile), l'hébergement de site web, l'infogérance, la gestion de parc informatique et l'ingénierie système et réseau. Elle répond régulièrement à des appels d'offres en tant que société d'infogérance et prestataire de services informatiques.

Parmi les marchés gagnés récemment par ITS86, on trouve celui de la gestion du parc informatique du réseau des médiathèques de la Vienne, *MediaTek86*, ainsi que l'informatisation de plusieurs activités internes des médiathèques ou en lien avec le public.

En tant que technicien développeur junior travaillant dans le pôle Développement d'ITS86, il m'a été confié le développement d'une application de bureau permettant de gérer les personnels de chaque médiathèque.

Mission à effectuer

La mission consiste en le développement d'une application de bureau permettant de gérer les personnels de chaque médiathèque, leur affectation à un service et leurs absences. Il s'agit d'une application monoposte installée sur le poste du service administratif. Plus précisément il s'agit d'une solution applicative permettant :

- une connexion sécurisée à la base de données MySQL comportant les données sur les personnels ;
- l'affichage de la liste des personnels avec la possibilité d'ajouter, modifier, supprimer un personnel ;
- l'affichage de la liste des absences d'un personnel sélectionné avec la possibilité d'ajouter, modifier ou supprimer une absence ;
- le contrôle des saisies de l'utilisateur afin de l'avertir en cas d'oubli de remplissage d'un champ d'information, de l'ajout d'une absence sur la même période qu'une autre déjà renseignée, ou d'incohérences concernant les dates de début et de fin de l'absence en cours de saisie.

En plus du développement de l'application, il a été demandé de fournir une documentation technique au format html afin de faciliter sa maintenance ainsi qu'une vidéo de

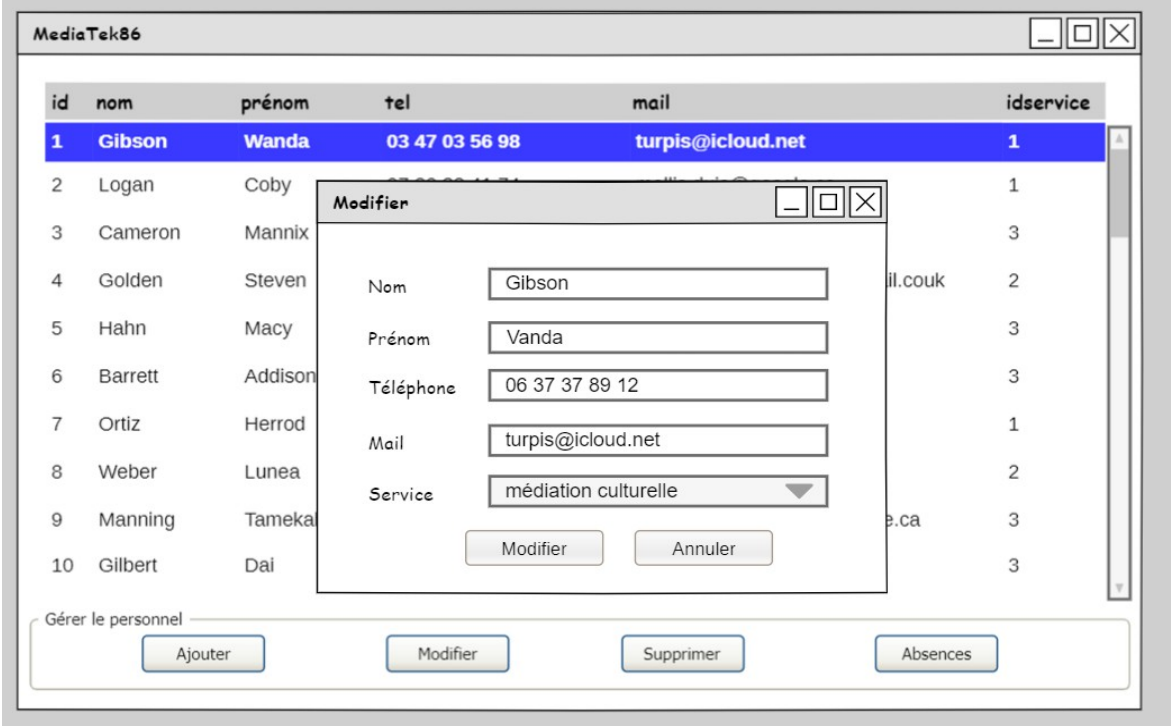
présentation des principales fonctionnalités de cette application à destination des utilisateurs.

Étapes de développement du projet

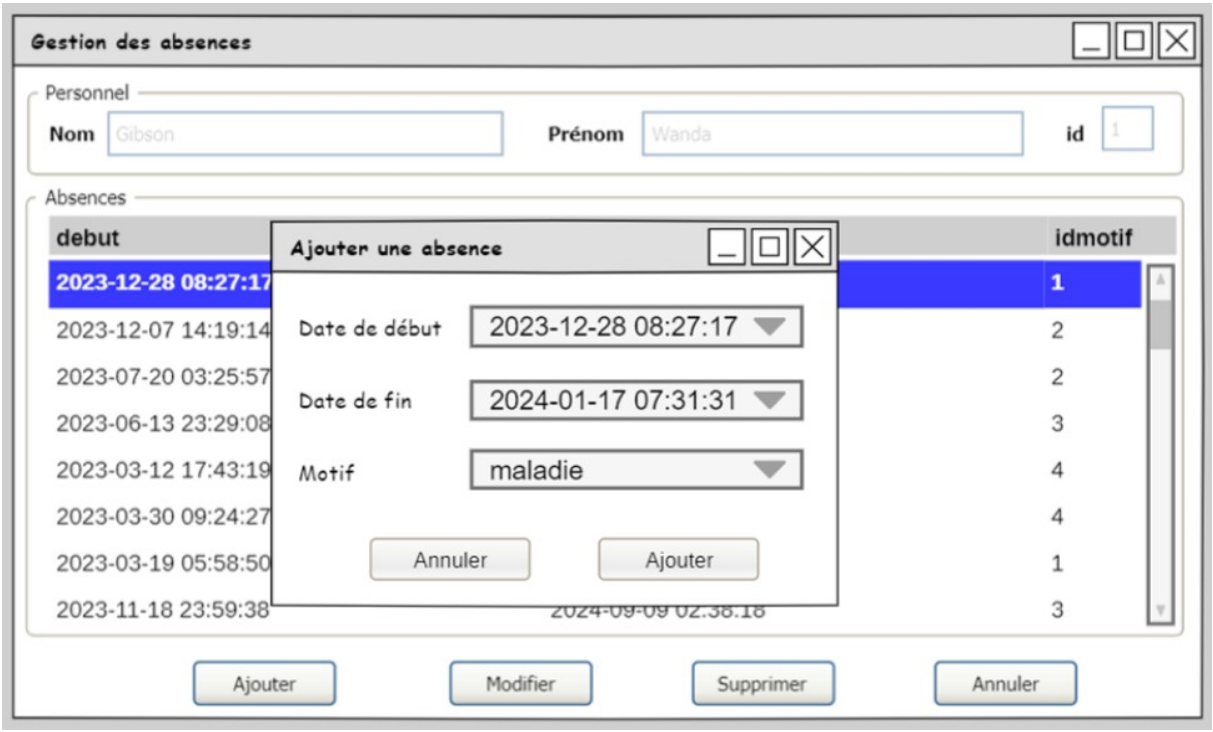
Maquettage de l'interface graphique

La première étape du projet a été la conception et le maquettage de l'interface graphique grâce au logicien Pencil.

Connexion



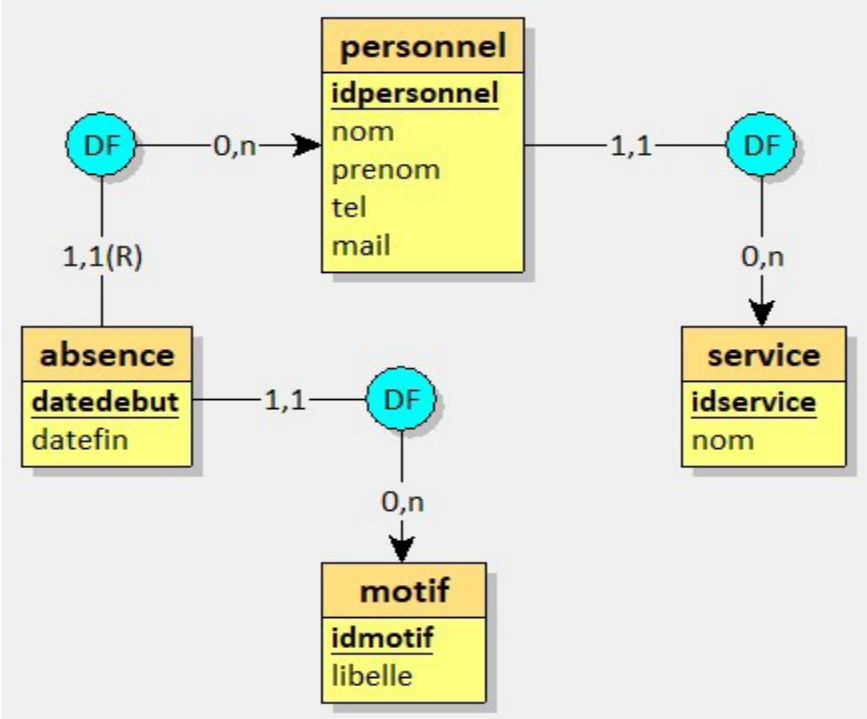
Gestion du personnel



Gestion des absences

Création et remplissage de la base de données

Avant de commencer le développement de l'application, nous avons généré, grâce au logiciel Looping, un script SQL pour créer la structure de la base de données à partir du modèle suivant :



Nous avons ensuite créé les identifiants et mots de passe des utilisateurs pouvant accéder à la base :

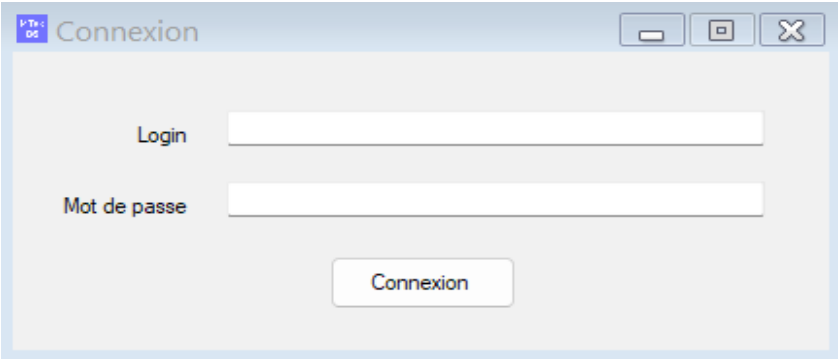
```
--
-- Création des utilisateurs et attribution des privilèges
--
CREATE USER 'respo1'@'localhost'
IDENTIFIED WITH sha256_password BY 'respoMEDIA86@';
GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO 'respo1'@'localhost';
ALTER USER 'respo1'@'localhost' REQUIRE NONE WITH MAX_QUERIES_PER_HOUR 0
MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0;
GRANT ALL PRIVILEGES ON `mediatek86`.* TO 'respo1'@'localhost';

CREATE USER 'mediatek86'@'localhost'
IDENTIFIED WITH sha256_password BY 'yZJP]YCuomae)H5k';
GRANT SELECT ON *.* TO 'mediatek86'@'localhost';
ALTER USER 'mediatek86'@'localhost' REQUIRE NONE WITH
MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0
MAX_USER_CONNECTIONS 0;
GRANT ALL PRIVILEGES ON `mediatek86`.* TO 'mediatek86'@'localhost';
```

Pour remplir la base de données nous avons utilisé le générateur en ligne https://www.generatedata.com/ et grâce au jeu de données produit nous avons écrit un script SQL pour remplir la base de données. Il est à noter que le générateur de données ne produit pas un jeu de données cohérent, notamment concernant les dates d'absence. De nombreux inserts ne respectent pas le principe du non recoupement de deux périodes d'absence pour un même personnel. Par exemple pour le personnel à l'idpersonnel 5, nous constatons qu'il a été absent du 1er mars 2023 au 1er juin 2024 et du 15 décembre 2023 au 30 octobre 2024.

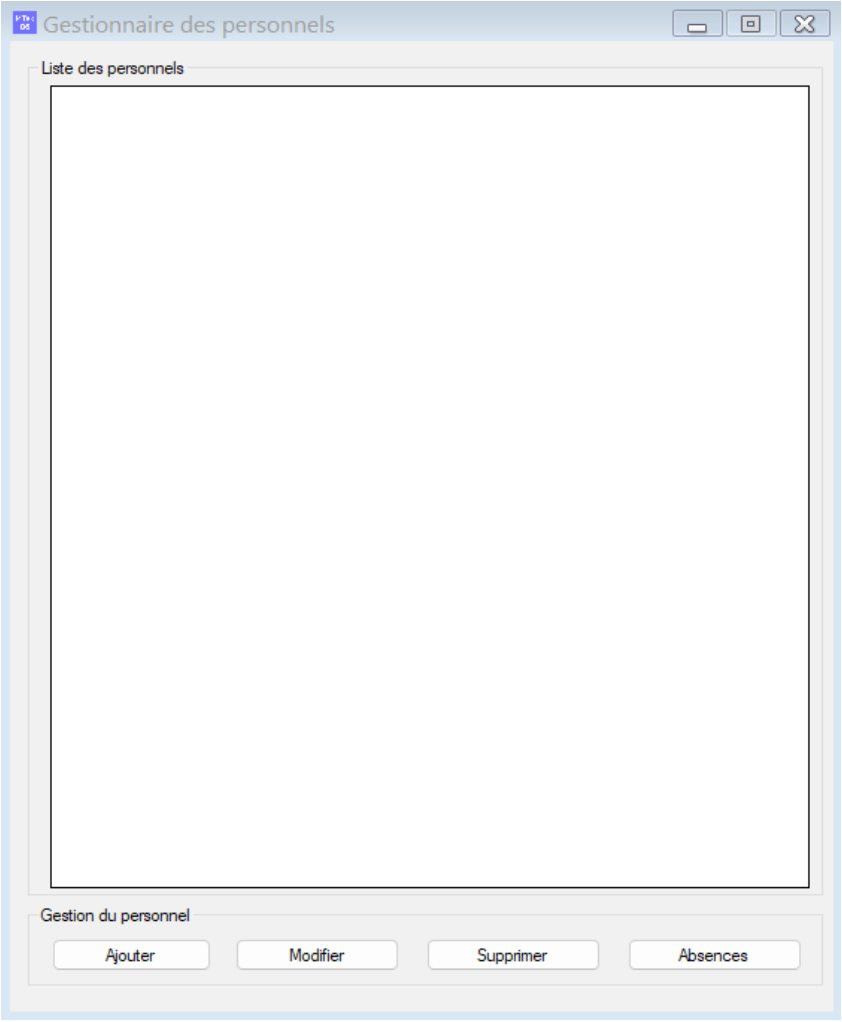
```
INSERT INTO `absence` (`idpersonnel`, `datedebut`, `datefin`, `idmotif`)
VALUES [...]
(5, '2023-03-01 01:56:28', '2024-06-01 18:14:01', 3),
(5, '2023-12-15 20:05:53', '2024-10-30 10:14:43', 3),
(2, '2023-09-04 11:33:27', '2024-12-23 16:08:51', 3),
(5, '2023-05-02 10:27:52', '2024-08-11 22:35:44', 3),
(14, '2023-05-03 16:14:50', '2024-08-20 06:52:00', 3),
(13, '2023-03-03 06:52:42', '2024-04-29 15:07:38', 2),
(14, '2023-10-12 00:44:44', '2024-02-29 15:46:59', 3),
(3, '2023-02-10 21:19:18', '2024-04-10 18:06:45', 4),
(8, '2023-05-17 15:43:45', '2024-06-08 15:18:51', 3),
(14, '2023-01-17 01:48:39', '2024-06-01 09:58:47', 2),
```

Structuration MVC et codage des interfaces

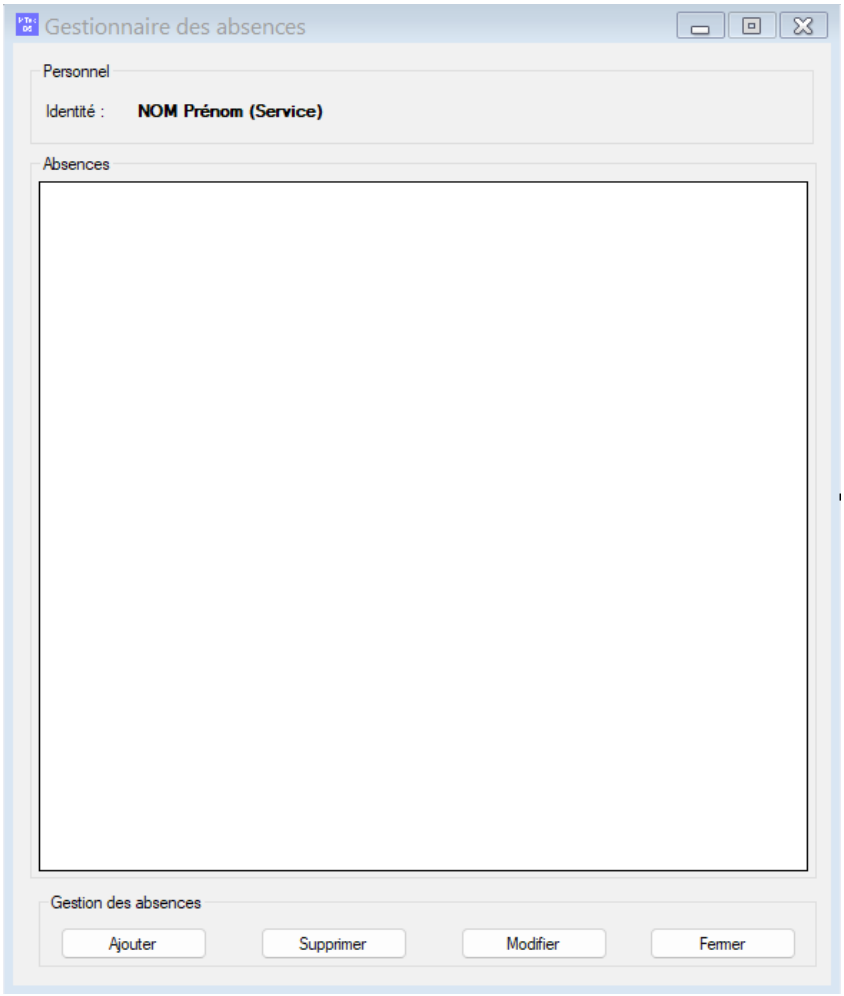
Après avoir vérifié le bon fonctionnement de l'environnement de travail (Visual Studio, WAMP), créé un dépôt distant sur Github et structuré les différents dossiers de l'application selon le modèle MVC, nous avons procédé au codage des interfaces grâce à WinForms en respectant les maquettes déjà conçues sur Pencil.



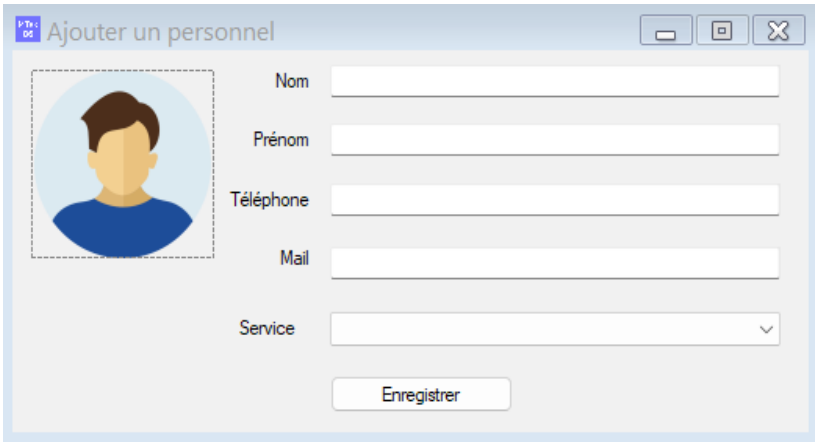
Connexion



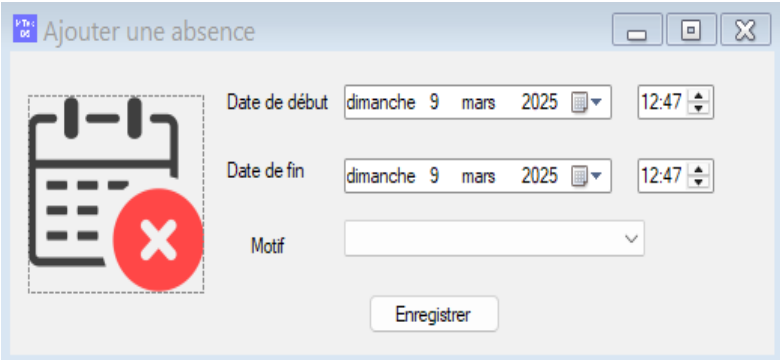
Liste des personnels



Liste des absences



Ajout de personnel



Ajout d'absence

Codage des outils de connexion

Les premières classes codées sont les outils de connexion contenus dans deux paquetages : bddmanager et dal. Le premier paquetage contient une seule classe avec le singleton de connexion et les constructeurs des requêtes SQL (SELECT ou autres). Le deuxième paquetage contient les deux classes qui permettent l'accès et la manipulation des données de la base. La partie essentielle de cette session de codage est l'intégration correcte et efficace des différentes requêtes SQL.

Extrait de code concernant la demande d'ajout d'un personnel :

```
  /// <summary>
  /// Demande d'ajout d'un personnel
  /// </summary>
  /// <param name="personnel"></param>
  public void AddPersonnel(Personnel personnel)
  {
      if (access.Manager != null)
      {
              string req = "insert into personnel(nom, prenom, tel, mail,
idservice) ";
          req += "values (@nom, @prenom, @tel, @mail, @idservice);";
              Dictionary<string, object> parameters = new Dictionary<string,
object>();
          parameters.Add("@nom", personnel.Nom);
          parameters.Add("@prenom", personnel.Prenom);
          parameters.Add("@tel", personnel.Tel);
          parameters.Add("@mail", personnel.Mail);
          parameters.Add("@idservice", personnel.Service.Idservice);
         try
          {
              access.Manager.ReqUpdate(req, parameters);
          }
          catch (Exception e)
          {
```

```
                Console.WriteLine(e.Message);
                Environment.Exit(0);
            }
        }
    }
```

Codage du modèle

L'étape suivante consiste à coder les six classes du modèle : deux qui servent à la connexion à la base des données (Admin, responsable) et quatre qui servent à sauvegarder et manipuler les données issues de la base (personnel, service, absence, motif). Il a été choisi de faire quatre classes des quatre tableaux du modèle de la base de données initiales. Mais peut-être eût-il été plus simple d'en conserver que deux en fusionnant complètement personnel avec service et absence avec motif...

Extrait de code de la classe personnel :

```
    /// <summary>
    /// Demande d'ajout d'un personnel
    /// </summary>
    /// <param name="personnel"></param>
    public void AddPersonnel(Personnel personnel)
    {
        if (access.Manager != null)
        {
                string req = "insert into personnel(nom, prenom, tel, mail,
idservice) ";
            req += "values (@nom, @prenom, @tel, @mail, @idservice);";
                Dictionary<string, object> parameters = new Dictionary<string,
object>();
            parameters.Add("@nom", personnel.Nom);
            parameters.Add("@prenom", personnel.Prenom);
            parameters.Add("@tel", personnel.Tel);
            parameters.Add("@mail", personnel.Mail);
            parameters.Add("@idservice", personnel.Service.Idservice);
           try
            {
                access.Manager.ReqUpdate(req, parameters);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Environment.Exit(0);
            }
        }
    }
```

Codage des contrôleurs

L'étape d'après consiste à coder le paquetage controller qui contient les contrôleurs rattachés aux deux fenêtres principales de l'application : la fenêtre de connexion et celle de la gestion des personnels.

Extrait de la classe du contrôleur de la fenêtre connexion :

```
/// <summary>
/// Contrôleur de FrmAuthentification
/// </summary>
class FrmAuthentificationController
{
    /// <summary>
    /// objet d'accès aux opérations possibles sur Responsable
    /// </summary>
    private readonly ResponsableAccess responsableAccess;

    /// <summary>
    /// Récupère l'acces aux données
    /// </summary>
    public FrmAuthentificationController()
    {
        responsableAccess = new ResponsableAccess();
    }

    /// <summary>
    /// Vérifie l'authentification
    /// </summary>
        /// <param name="admin">objet contenant les informations de connexion</param>
        /// <returns> vrai si les informations de connexion sont correctes</returns>
    public Boolean ControleAuthentification(Admin admin)
    {
        return responsableAccess.ControleAuthentification(admin);
    }
}
```

Codage des fonctionnalités de l'application

Le codage des différentes fonctionnalités à partir des cas d'utilisation a occupé la majeure partie du temps de codage. Il s'agissait cette fois-ci d'intervenir dans les classes du paquetage view afin d'implémenter chacune de ces fonctionnalités.

Dans la mesure où il s'agit d'une application qui utilise plusieurs fenêtres pour répondre aux différents cas d'utilisation, le premier problème qu'il a fallu résoudre est celui de la communication des données nécessaires entre toutes ces fenêtres : la méthode qui a été

choisie est celle des constructeurs.

Exemple de constructeur (fenêtre de modification d'une absence) :

```
  /// <summary>
  /// Initialisation de la fenêtre et du contrôleur
  /// Remplissage des champs avec les informations de l'absence en cours de modification
  /// Sauvegarde de l'absence avant modification pour affichage lors de la confirmation de modification
  /// Remplissage d'une liste avec les absences sans celle en cours de modification
  /// </summary>
  /// <param name="Idpersonnel"></param>
  /// <param name="Datedebut"></param>
  /// <param name="Datefin"></param>
  /// <param name="IdMotif"></param>
  /// <param name="Libelle"></param>
  /// <param name="absenceslistecontrol"></param>
  public FrmModAbsence(int Idpersonnel, DateTime Datedebut, DateTime Datefin, int IdMotif, String Libelle, List<Absence> absenceslistecontrol)
  {
      InitializeComponent();
      controller = new FrmGestionPersonnelController();
      absencesamodifier = absenceslistecontrol;
      foreach (Absence absence in absenceslistecontrol)
      {
          absencessansabsencoursdemodif.Add(absence);
      }
      absencessansabsencoursdemodif.Remove(absencessansabsencoursdemodif.Find(abs => abs.Idpersonnel == Idpersonnel && abs.Datedebut == Datedebut && abs.Datefin == Datefin && abs.IdMotif == IdMotif && abs.Libelle == Libelle));
      idpersonnelencoursdemodif = Idpersonnel;
      dateAbsDeb.Value = Datedebut;
      TimeAbsDeb.Value = Datedebut;
      dateAbsFin.Value = Datefin;
      TimeAbsFin.Value = Datefin;
      CBoxMotifAbs.SelectedIndex = CBoxMotifAbs.FindStringExact(Libelle);
          absEnCoursdeModif.Add(new Absence(Idpersonnel, Datedebut, Datefin, IdMotif, Libelle));
  }
```

Exemple d'appel de ce constructeur (depuis la fenêtre de gestion des absences) :

```
  /// <summary>
  /// Demande de modification d'une absence
  /// </summary>
  /// <param name="sender"></param>
  /// <param name="e"></param>
  private void BtnAbsModif_Click(object sender, EventArgs e)
  {
      Absence absence = (Absence)bdgAbsences.List[bdgAbsences.Position];
      int Idpersonnel = absence.Idpersonnel;
```

```
        DateTime Datedebut = absence.Datedebut;
        DateTime Datefin = absence.Datefin;
        int IdMotif = absence.IdMotif;
        String Libelle = absence.Libelle;
          Form modAbsence = new FrmModAbsence(Idpersonnel, Datedebut, Datefin,
IdMotif, Libelle, absencesPersonnelControllist);
        modAbsence.Owner = this;

        if (modAbsence.ShowDialog() == DialogResult.Yes)
        {
            RemplirListeAbsences(idpersonnelencoursdemodif);
        }
    }
```

Le deuxième problème qui a surgi lors du codage de cette partie de l'application concernait le conflit entre la modification d'une absence et les méthodes de contrôle du recoupement des dates d'absence ci-dessous :

```
    /// <summary>
    /// Contrôle si deux périodes d'absences se chevauchent
    /// </summary>
    /// <param name="datedebut1"></param>
    /// <param name="datefin1"></param>
    /// <param name="datedebut2"></param>
    /// <param name="datefin2"></param>
    /// <returns></returns>
    public static bool ChevauchePeriode(DateTime datedebut1, DateTime datefin1,
DateTime datedebut2, DateTime datefin2)
    {
        return datedebut1 <= datefin2 && datedebut2 <= datefin1;
    }

    /// <summary>
    /// Contrôle si une période d'absence chevauche une autre période d'absence
du personnel
    /// </summary>
    /// <param name="dateAbsdeb"></param>
    /// <param name="dateAbsfin"></param>
    /// <param name="absencesPersonnel"></param>
    /// <returns></returns>
    public static bool GetChevauchementAbs(DateTime dateAbsdeb, DateTime
dateAbsfin, List<Absence> absencesPersonnel)
    {
        foreach (Absence absence in absencesPersonnel)
        {
                if (ChevauchePeriode(dateAbsdeb, dateAbsfin, absence.Datedebut,
absence.Datefin))
            {
                return true;
            }
        }
        return false;
```

```
}
```

Lorsqu'on modifiait une absence, le programme affichait une alerte pour nous informer qu'une absence a déjà été saisie pour cette période. Le problème a été résolu comme montré plus haut en créant un doublon transitoire de la liste d'absences duquel on supprimait l'absence en cours de modification : c'est sur ce doublon qu'on appliquait la méthode de contrôle GetChevauchementAbs().

Concernant les contrôles de saisie utilisateur nous avons également décidé de rajouter une fonctionnalité qui n'était pas exigée dans les cas d'utilisation donnés mais qui nous a semblé utile : avertir l'utilisateur quand il tente d'ajouter à la base un personnel portant le même nom et prénom qu'un autre personnel déjà enregistré.

```
  Service service = new Service(idservice, nomservice);
  Personnel personnel = new Personnel(0, TxtBoxNom1.Text, TxtBoxPrenom1.Text,
TextBoxTel1.Text, TextBoxMail1.Text, service);
  int compteurdoublon = 0;
  foreach (Personnel unpersonnel in lespersonnels)
  {
           if (unpersonnel.Nom.ToUpper() == TxtBoxNom1.Text.ToUpper() &&
unpersonnel.Prenom.ToUpper() == TxtBoxPrenom1.Text.ToUpper())
      {
          compteurdoublon++;
      }
  }
  if (compteurdoublon > 0)
  {
     if (MessageBox.Show("Un personnel ayant les mêmes nom et prénom est déjà
présent dans la base. Voulez-vous vraiment ajouter " + personnel.Nom.ToUpper()
+ " " + personnel.Prenom + " ?", "Confirmation d'ajout",
MessageBoxButtons.YesNo) == DialogResult.Yes)
      {
          controller.AddPersonnel(personnel);
          DialogResult = DialogResult.OK;
      }
  }
  if (compteurdoublon == 0)
  {
      controller.AddPersonnel(personnel);
      DialogResult = DialogResult.OK;
  }
```

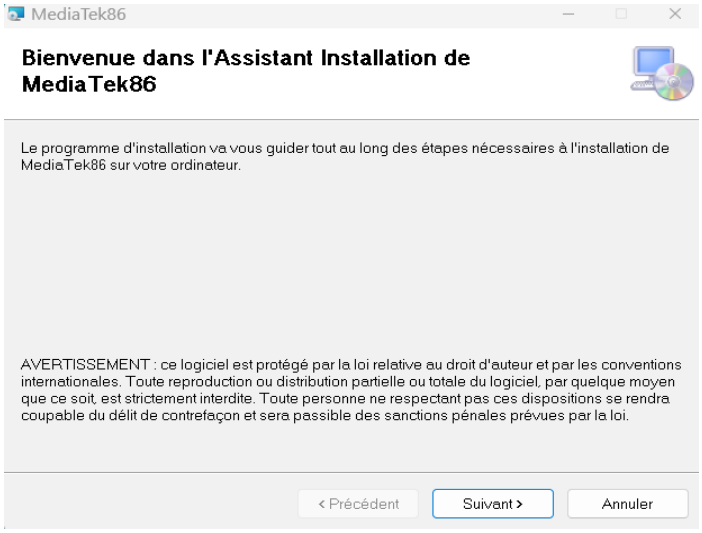
Tests de l'application et création de l'installateur

Une fois l'application codée nous avons procédé à des tests de son fonctionnement tout en vérifiant bien que l'application communiquait correctement avec la base de données et

que toutes les requêtes SQL étaient effectives au niveau de la base.

Nous avons ensuite créé une icône pour l'application grâce à l'éditeur en ligne https://www.xiconeditor.com/ et généré l'application, la documentation xml, l'installateur grâce au « Setup wizard » de Visual studio. Nous avons procédé à l'installation du logiciel et l'avons ensuite testé de nouveau, une fois installé.

Icône de l'application



Installateur

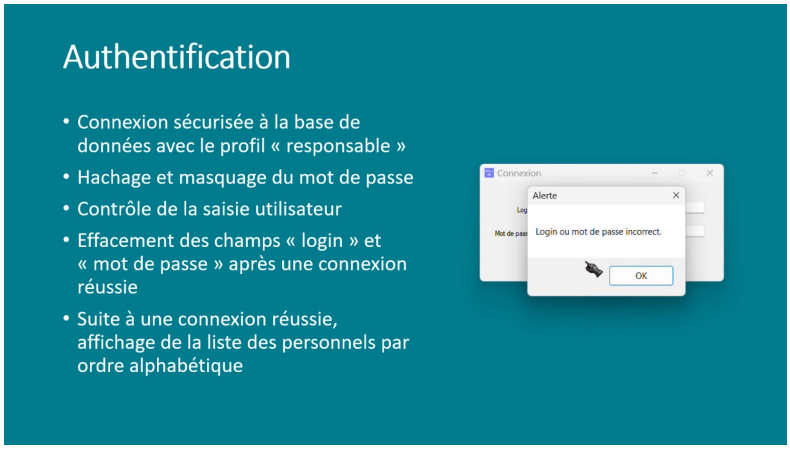
Génération de la documentation technique

L'étape d'après a consisté en la création de la documentation technique au format html. Nous avons essayé d'en créer une avec Sandcastle, sans succès, le logiciel retournant un message d'erreur ou une documentation très partielle de l'application ne contenant qu'une seule classe. Nous n'avons pas trouvé de solution à ce problème mais l'un des messages d'erreur de l'application nous a permis de rectifier une erreur de nommage dans notre code : le namespace de FrmGestionPersonnel était MediaTek86 au lieu de MediaTek86.view.

Nous avons ensuite essayé un autre logiciel, doxygen, et la génération de la documentation sous format html a pu se faire normalement.

Documentation technique générée grâce à doxygen

Création de la documentation utilisateur



Une documentation utilisateur a été créée sous la forme d'une vidéo de 4'54. Pour sa création nous avons utilisé l'outil de capture d'écran de windows afin de filmer les différentes fonctionnalités de l'application.

La création de la vidéo en elle-même a été faite sur Microsoft Powerpoint. La musique, libre de droits, qui accompagne la vidéo a été récupérée sur https://www.fesliyanstudios.com/royalty-free-music/download/easy-going/1250.

Bilan

L'application est fonctionnelle et respecte les cas d'utilisation donnés. La principale difficulté rencontrée a été la façon de transmettre des données d'une forme à une autre : nous avons choisi la méthode des constructeurs parce qu'il s'agit de celle qu'on maîtrisait le mieux mais nous sommes curieux d'en apprendre d'autres comme celle des délégués. Par

ailleurs, le codage de cette application nous a fait réfléchir sur le principe de simplification qui devrait régir la transformation du modèle conceptuel de données en diagramme de classes, un principe qui n'a été que partiellement suivi lors du codage de cette application.