COMPTE RENDU D'ACTIVITÉ

Atelier de Professionnalisation 2 • 1ère année

BTS SIO option SLAM
POPCZYK Louise
le 31 mai 2022

Sommaire

Présentation du contexte

Lors de ce projet, j'ai travaillé dans un contexte réaliste d'entreprise, en tant que technicienne développeuse pour l'ESN InfoTech Services 86. Accompagnée par l'équipe de ITS86, je devais réaliser la conception et le déploiement d'une application de bureau, MediaTek, permettant de gérer le personnel de chaque médiathèque de la Vienne (86).

Mediatek86 est un réseau qui regroupe les responsables des médiathèques du département de Vienne. Avec le déploiement de ses activités, MediaTek86 fait appel à l'ESN InfoTech86 afin de gérer le parc informatique des médiathèques, son infastructure système et réseau, ainsi que de développer son portail web et ses applications internes.

InfoTech Services 86, est une Entreprise de Services Numériques (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.

MediaTek est une application de bureau (Windows forms C#) qui permet de gérer le personnel, leur affectation à un service et leurs absences, des médiathèques du département de la Vienne. Reliée à une base de données MySQL créée localement grâce à WampServer, les modifications apportées via l'application seront donc prises en compte dans celle-ci.

Caractéristiques de l'application

Langage choisi : C#

Environnement de déploiement (IDE) : Visual Studio

Système de gestion de base de données : MySQL

Serveur : WAMPServeur

Versionning : Github

ÉTAPE 1

1.1 Génération du script SQL

1.2 Création de la base de données

1.3 Liaison Visual Studio à la base de données

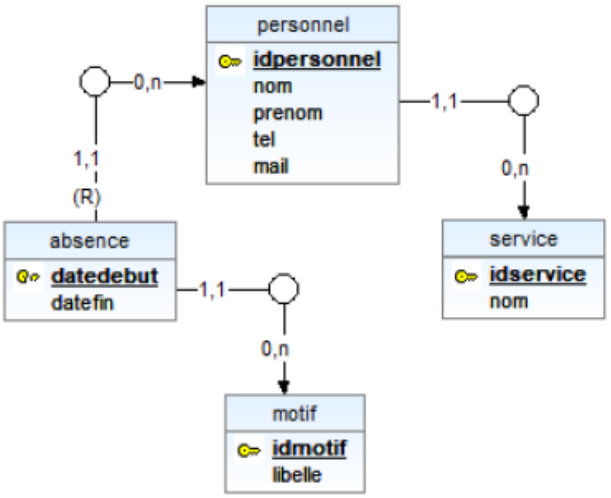
La création d'une base de données est généralement précédée par l'élaboration d'un modèle conceptuel de données (MCD), dans lequel sont définies les données utilisées et leurs relations.

Le MCD fournit une description graphique pour représenter des modèles de données sous la forme de diagrammes pouvant contenir des entités ou des associations. Il peut être utilisé pour décrire les besoins en information ou par exemple le genre d'information nécessaire à l'élaboration du cahier des charges.

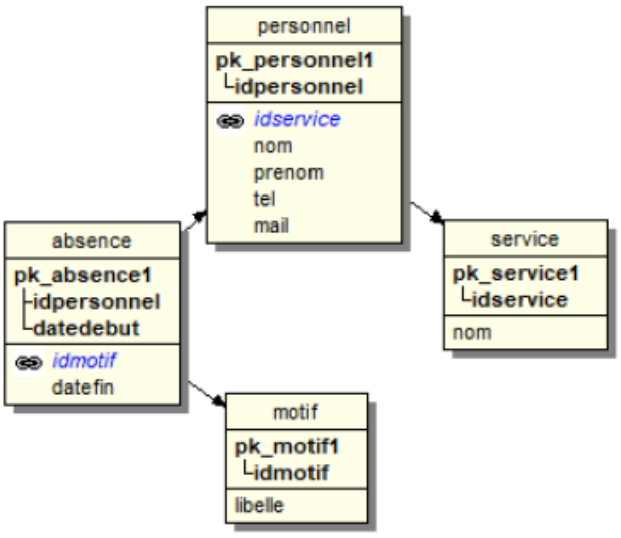
Le processus de modélisation de données se poursuit par la génération d'un script SQL, qui est facilité par divers outils comme Windesign. Ce logiciel permet de générer automatiquement les différents modèles de données, à partir d'un modèle de conceptuel de données.

1.1 Génération du script SQL

1.1.1 Modèle conceptuel de données (disponible dans les consignes du devoir)



1.1.2 Modèle logique de données (généré grâce à WinDesign)



1.1.3 Script SQL (généré grâce à WinDesign)

```
# ------------------------------------------------------------------------
#       TABLE : absence
# ------------------------------------------------------------------------

CREATE TABLE IF NOT EXISTS absence
 (
   idpersonnel INTEGER NOT NULL  ,
   datedebut DATETIME NOT NULL  ,
   idmotif INTEGER NOT NULL  ,
   datefin DATETIME NULL
   , PRIMARY KEY (idpersonnel,datedebut)
 )
ENGINE=InnoDB;
# ------------------------------------------------------------------------
#       TABLE : motif
# ------------------------------------------------------------------------

CREATE TABLE IF NOT EXISTS motif
 (
   idmotif INTEGER NOT NULL AUTO_INCREMENT ,
   libelle VARCHAR(128) NULL
   , PRIMARY KEY (idmotif)
 )
ENGINE=InnoDB;
# ------------------------------------------------------------------------
#       TABLE : service
# ------------------------------------------------------------------------

CREATE TABLE IF NOT EXISTS service
 (
   idservice INTEGER NOT NULL AUTO_INCREMENT ,
   nom VARCHAR(50) NULL
   , PRIMARY KEY (idservice)
 )
ENGINE=InnoDB;
# ------------------------------------------------------------------------
#       TABLE : personnel
# ------------------------------------------------------------------------

CREATE TABLE IF NOT EXISTS personnel
 (
   idpersonnel INTEGER NOT NULL AUTO_INCREMENT ,
   idservice INTEGER NOT NULL  ,
   nom VARCHAR(50) NULL  ,
   prenom VARCHAR(50) NULL  ,
   tel VARCHAR(15) NULL  ,
   mail VARCHAR(128) NULL
   , PRIMARY KEY (idpersonnel)
 )
ENGINE=InnoDB;

# ------------------------------------------------------------------------
#       CREATION DES REFERENCES DE TABLE
# ------------------------------------------------------------------------


ALTER TABLE absence
  ADD FOREIGN KEY FK_absence_motif (idmotif)
      REFERENCES motif (idmotif) ;


ALTER TABLE absence
  ADD FOREIGN KEY FK_absence_personnel (idpersonnel)
      REFERENCES personnel (idpersonnel) ;


ALTER TABLE personnel
  ADD FOREIGN KEY FK_personnel_service (idservice)
      REFERENCES service (idservice) ;
```

1.2 Création de la base de données

1.2.1 Création de la base de données

Grâce au script SQL, il nous est possible de créer la base de données qui correspond au modèle de conceptualisation de données du projet MediaTek. Nous utiliserons la plateforme de développement Web WAMPServer qui nous permettra d'utiliser le système de gestion de base de données phpMyAdmin.

Pour ce faire, il faut :

▹ Lancer wampmanager.exe, puis vérifier la connexion de WAMPServer (si l'icône est verte )

▹ Ouvrir phpMyAdmin, puis se connecter

▹ Créer la base de données (nom : "mediatek", encodage : "utf8_general_ci")

▹ Exécuter le script SQL (onglet SQL > coller le script > exécuter)

1.2.2 Création d'un utilisateur ayant les droits d'accès à la base

Pour sécuriser l'accès à la base de données, il est nécessaire de créer un utilisateur avec un login et mot de passe.
L'utilisateur devra posséder tous les privilèges dont dispose phpMyAdmin afin de pouvoir apporter les modifications nécessaires à la base données.

▹ mediatek > Privilèges > Ajouter un compte utilisateur
(nom d'utilisateur : mediatekadmin mot de passe: ******** privilèges: tout cocher)

Lorsqu'on réalise une action grâce à l'interface graphique de phpMyAdmin, celui-ci nous renvoie le script SQL correspondant.

Ici, le script SQL correspondant à la création du nouvel utilisateur.

```
CREATE USER 'mediatekadmin'@'%' IDENTIFIED WITH mysql_native_password AS '***';GRANT ALL PRIVILEGES ON *.* TO
'mediatekadmin'@'%' REQUIRE NONE WITH GRANT OPTION MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0
MAX_USER_CONNECTIONS 0;GRANT ALL PRIVILEGES ON `mediatek`.* TO 'mediatekadmin'@'%';
```

1.2.3 Ajouter une table “responsable” (avec colonne login et pwd)

La table responsable contiendra deux champs, login et pwd, de type varchar et de longueur 64, qui contiendront eux le nom d’utilisateur et le mot de passe qui nous permettront par la suite de se connecter à l’application.

Le mot de passe contenu dans la colonne “pwd” devra être haché, en cas d’accès malveillant à la base de données. Ainsi, le mot de passe ne pourra être retrouvé.
Nous utiliserons la fonction SHA2.

Pour ce faire :

▹ dans la zone de texte de requête SQL entrer :

```
1 UPDATE responsable
2 SET pwd = SHA2('P@$$word', 256)
```

▹ Résultat :

login	pwd
mediatekadmin	e4bc29dc8068a4f8be92dff03fa14336d460bf5bb185a28f77...

1.2.4 Alimenter la base de données

Les tables “personnel” et “absence” devront être remplies d’exemples aléatoires générés par generatedata.com (https://generatedata.com/).

Après avoir sélectionné le type des exemples et le langage que l’on souhaite générer, nous devons entrer le nom des colonnes de notre table et spécifier le format souhaité.

Le site web nous retourne par la suite le script SQL correspondant, il nous suffira de le coller dans la zone de texte de requête SQL de phpMyAdmin, puis de l'exécuter.

```sql
INSERT INTO `myTable` (`idpersonnel`,`idservice`,`nom`,`prenom`,`tel`,`mail`)
VALUES
  (1,3,"Patience","Spears","0776364555","quisque@icloud.org"),
  (2,2,"Erich","Galloway","0555572932","egestas.sed@protonmail.net"),
  (3,1,"Gannon","Hahn","0501646758","integer.urna@yahoo.net"),
  (4,2,"Mercedes","Bass","0976264433","dui@icloud.couk"),
  (5,3,"Tatyana","Hester","0208737637","dolor.dapibus@hotmail.net");
```

(Ici, il ne m’était pas possible de modifier le nom de ma table, j’ai donc dû le modifier dans phpMyAdmin, avant de l'exécuter.)

1.3 Liaison Visual Studio à la base de données

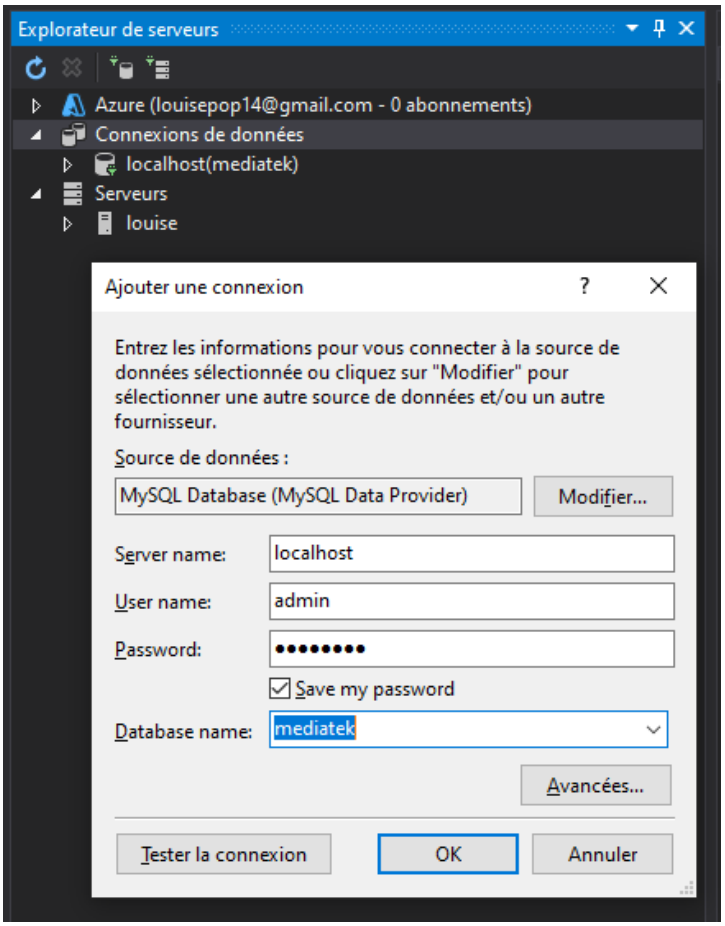
1.3 Connexion de la base de données à l'IDE Visual Studio

L'utilisation de notre base de données dans l'IDE nécessite au préalable que l'on connecte celle-ci à Visual Studio.

Pour ce faire:

▹ Affichage > Explorateur de solution > clic droit sur Connexion de donnée

▹ Remplir la page "Ajouter une connexion" :



ÉTAPE 2

2.1 Prototype de l'application
2.2 Créer un dépôt
2.3 Structurer l'application en MVC
2.4 Coder le visuel

Grâce au dossier complémentaire constitué par les équipes de développement, nous pourrons créer le prototype GUI (Graphical User Interface) de l'application, grâce à Pencil, un outil de maquettage. Par la suite, les interfaces graphiques seront créées grâce à Windows Forms, sous VIsual Studio 2019.

Le dossier complémentaire contient des informations importantes pour le développement de l'application, comme le diagramme d'utilisation et ses cas d'utilisation :

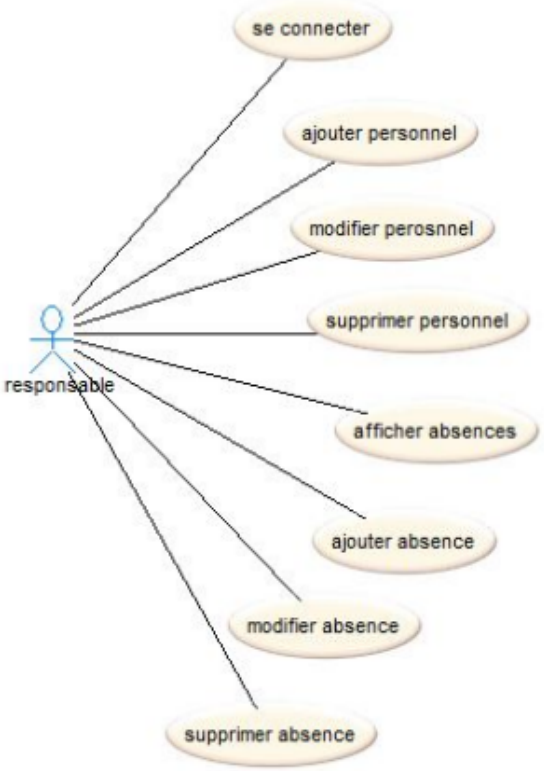
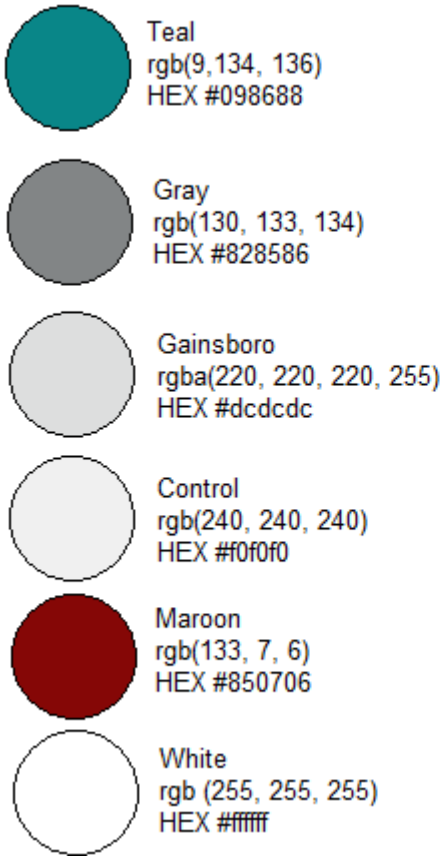


Diagramme d'utilisation

2.1 Prototype de l'application

2.1.1 Charte Graphique

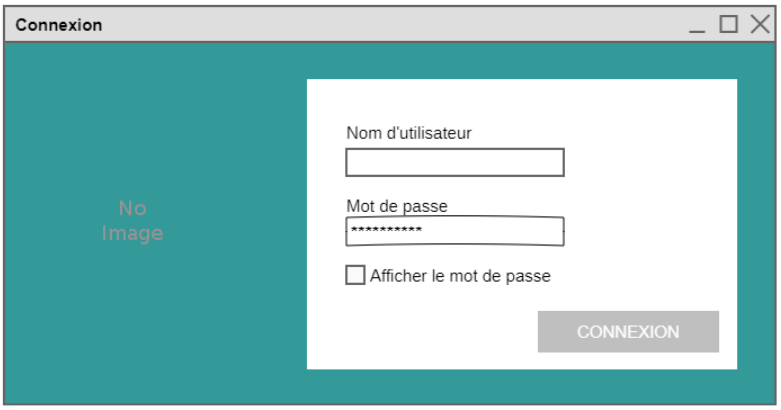


Police d'écriture:
Microsoft Sans Serif

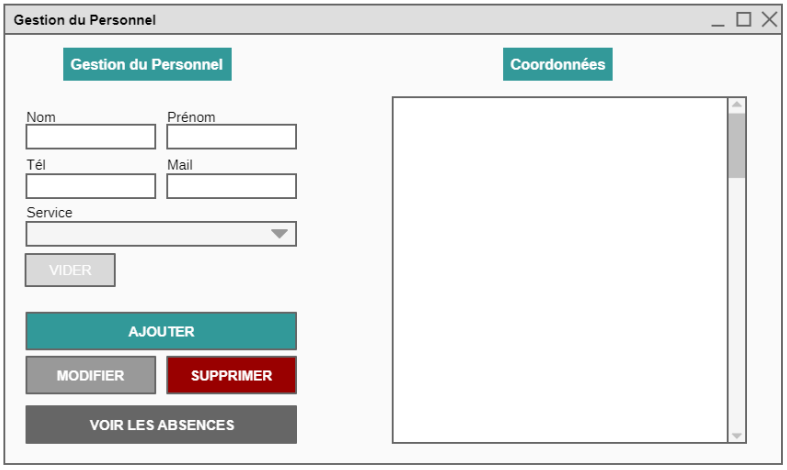
2.1.2 Création des interfaces sous Pencil

J'ai décidé de créer mon application grâce à deux fenêtres, ce qui me semblait être le plus simple et le plus rapide.
Une fenêtre d'authentification, où l'utilisateur pourra indiquer son login et son mot de passe pour se connecter. Puis une fenêtre de gestion, qui affichera ou non des objets graphiques en fonction de ce que souhaite faire l'utilisateur (gestion du personnel ou gestion des absences).

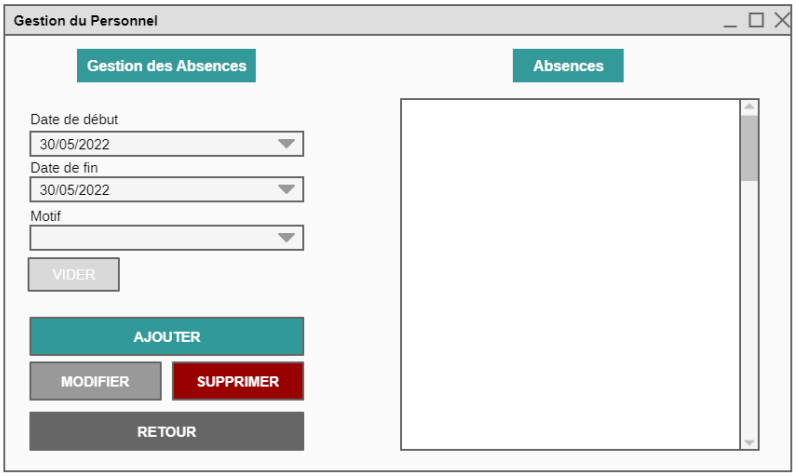
Fenêtre d'authentification :



Fenêtre de gestion pour la Gestion du Personnel :



Fenêtre de gestion pour la Gestion des Absences :



L'action de modifier ou supprimer un personnel/une absence demandera une confirmation via une popup.

2.2 Créer un dépôt

Un dépôt distant est un dépôt qui sert à centraliser un dépôt. C'est un type de dépôt qui est important (voire indispensable) lorsque l"on travaille à plusieurs sur le même projet puisqu'il permet de centraliser le travail de chaque développeur.

Dans notre contexte, l'utilisation d'un dépôt distant est intéressant pour annoter chaque sauvegarde réalisée. Il nous permet de mieux nous retrouver et de partager facilement le code source de notre application.

Pour ce faire :

▹ Se connecter à Github (https://github.com)

▹ Create a new repository

▹ Code > Clone > Cloner l'adresse du dépôt

Puis dans Visual Studio 2019 :

▹ Cloner un projet > Coller l'adresse > Choisir l'emplacement local

Une fois le dépôt créé, il est possible de sauvegarder localement notre travail grâce à des "commits", puis de les envoyer sur notre github grâce à des "push".
Tous les changements envoyés sont consultables sur le repository du Github.

Mon profil Github : https://github.com/uu-0

2.3 Structurer l'application en MVC

Le pattern MVC est une façon d'organiser un code source autour de trois piliers :

- Le Model : qui représente la structure des données, leurs définitions ainsi que les fonctions qui leur sont propres et qu'elles peuvent avoir.

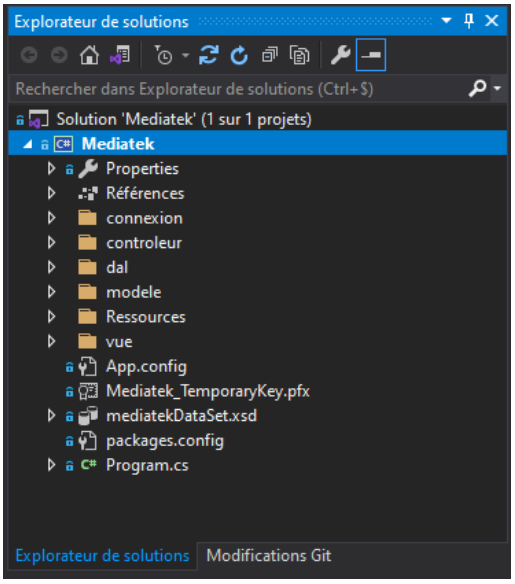
- La View (ou les vues) : qui représente l'interface ou les interfaces graphiques

- Le Controller : qui permet de lier le model à la view. Les requêtes qu'un client va effectuer via l'interface graphique vont être dirigées vers le controller, qui sera en charge de manipuler les données dont il a besoin grâce au model.

Le pattern MVC permet de retrouver les différents modules et fonctions du code beaucoup plus facilement, et permet une modification du code source plus rapide. Il rend également le code plus visible et structuré pour les différents développeurs.

Ainsi, un développeur back-end pourra faire des modifications sans intervenir sur la partie front-end et inversement.

Structuration en MVC de MediaTek: *(fenêtre explorateur de solution de Visual Studio)*



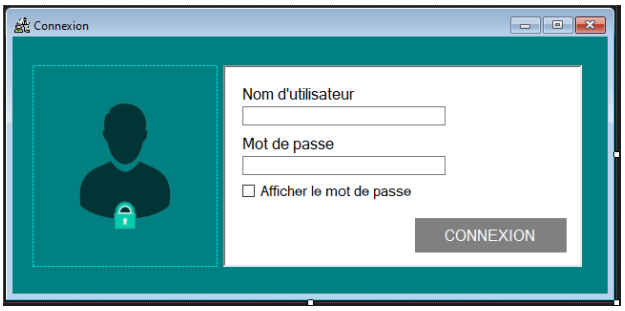
2.4 Coder le visuel

Les interfaces graphiques seront de type Forms, et seront codées dans le package vue de l'application. J'ai donc créé deux Forms, comme expliqué dans "2.1.2 Création des interfaces sous Pencil". L'un pour l'authentification de l'utilisateur, l'autre pour la gestion des absences ou du personnel. J'ai choisi cette option car j'avais peur de manquer de temps, le projet me paraissait à première vue très compliqué. Mais, après une bonne relecture de mes cours, et création d'un planning, j'ai été rassurée et donc remotivée.

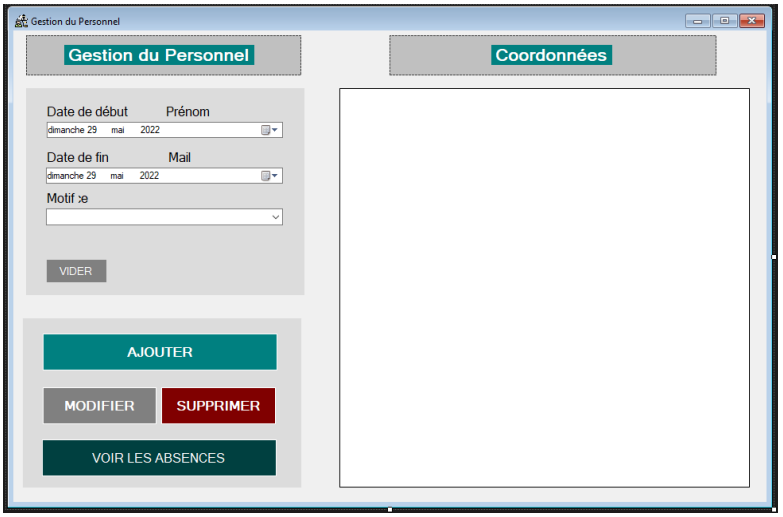
Les composants graphiques seront affichés grâce à une variable de type booléen, selon le choix de l'utilisateur.

Lorsque l'on ouvre la solution, l'utilisateur est invité à entrer son nom d'utilisateur et son login. Si les informations sont correctes, la page de Gestion de Personnel s'affiche. L'utilisateur pourra afficher la page de Gestion des Absences grâce au bouton "VOIR LES ABSENCES".

Fenêtre d'authentification finale :



Fenêtre de Gestion du Personnel/Gestion des Absences finale :



Quand ce bouton est activé, la variable de type booléen affiche ou non les composants graphiques. Ainsi, les composants permettant la gestion des absences sont affichés, et ceux permettant la gestion du personnel, masqués.

```
/// <summary>
/// affiche ou cache les composants graphiques
/// </summary>
/// <param name="show"></param>
3 références | louise, Il y a 41 minutes | 1 auteur, 7 modifications
public void ShowComponents(bool show)
{
    // Personnel
    txtNom.Enabled = !show;
    txtPrenom.Enabled = !show;
    lbService.Visible = !show;
    cboBoxService.Visible = !show;
    lstPersonnel.Visible = !show;

    // Absence
    lstAbs.Visible = show;
    lbDatedeb.Visible = show;
    lbDatefin.Visible = show;
    dateTimeDebut.Visible = show;
    dateTimeFin.Visible = show;
    lbMotif.Visible = show;
    cboBoxMotif.Visible = show;

    if (!show)
    {
        // Personnel
        btnVider.Location = new Point(26, 214);
        lbGestion.Location = new Point(48, 12);
        lbInfo.Location = new Point(128, 12);
        lbNom.ForeColor = System.Drawing.Color.Black;
        lbPrenom.ForeColor = System.Drawing.Color.Black;
        lbMail.ForeColor = System.Drawing.Color.Black;
        lbService.ForeColor = System.Drawing.Color.Black;
        btnAbs.Text = "VOIR LES ABSENCES";
        lbInfo.Text = "Coordonnées";
        lbGestion.Text = "Gestion du Personnel";
    }
    else
    {
        // Absence
        btnVider.Location = new Point(26, 214);
        lbGestion.Location = new Point(48, 12);
        lbInfo.Location = new Point(148, 12);
        //change la couleur des deux labels pour qu'ils aient la même que le fond
        lbNom.ForeColor = System.Drawing.Color.Gainsboro;
        lbPrenom.ForeColor = System.Drawing.Color.Gainsboro;
        lbMail.ForeColor = System.Drawing.Color.Gainsboro;
        lbService.ForeColor = System.Drawing.Color.Gainsboro;
        //change les textes des labels pour qu'ils correspondent aux actions demandées
        btnAbs.Text = "← RETOUR";
        lbInfo.Text = "Absences";
        lbGestion.Text = "Gestion de l'absence";
    }
}
```

ÉTAPE 3

3.1 Coder le package connexion

3.2 Coder le package dal

3.3 Coder les modèles

3.4 Générer la documentation technique

3.1 Coder le package connexion

La classe connexion est composée d'un singleton qui permet la connexion à la base de données. Ici, nous utilisons un singleton car la connexion à la base de données est chronophage : elle ne doit être instanciée qu'une seule fois.
Puisque nous avions déjà réalisé un exercice semblable en cours, j'ai simplement récupéré la classe connexion que j'avais créée à ce moment, puisqu'elle est réutilisable. Cela m'a évité de faire des erreurs et de perdre du temps.

```
namespace MediaTek.connexion
{
    /// <summary>
    /// Connexion à la base de données + exécution des requêtes
    /// </summary>
    24 références | louise, il y a 3 jours | 1 auteur, 1 modification
    public class ConnexionBDD
    {
        /// <summary>
        /// Objet de connexion à la BDD à partir d'une chaîne de connexion
        /// </summary>
        private MySqlConnection connection;

        /// <summary>
        /// Constructeur privé pour créer la connexion à la BDD et l'ouvrir
        /// </summary>
        /// <param name="stringConnect">chaîne de connexion</param>
        1 référence | louise, il y a 3 jours | 1 auteur, 1 modification
        private ConnexionBDD(string stringConnect)
        {
            try
            {
                connection = new MySqlConnection(stringConnect);
                connection.Open();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Application.Exit();
            }
        }
```

L'utilisation de "MySqlConnection" nécessite l'utilisation de "MySql.Data.MySqlClient", une librairie qui permet de travailler avec MySQL. Il faut donc rajouter sa référence au dessus du code, en tapant "using MySql.Data.MySqlClient".

```
using MySql.Data.MySqlClient;
using System;
using System.Collections.Generic;
using System.Windows.Forms;
```

3.2 Coder le package dal

Ce package contiendra la classe "AccesDonnes" puis les méthodes qu'aura besoin l'application pour répondre aux demandes du contrôleur.
Nous utilisons une chaîne de connexion afin de se connecter à la base de données.

Ici, nous retrouvons aussi la méthode qui permettra de vérifier si les informations entrées, lors de l'authentification de l'utilisateur, correspondent au login et pwd entrés plus tôt, dans la table responsable de la base de données.

```
10 références | louise, il y a 2 jours | 1 auteur, 3 modifications
class AccesDonnees
{
    /// <summary>
    /// Chaine de connexion à la BDD
    /// </summary>
    private static string connectionString = "Server=localhost;Database=mediatek;User Id=admin;Password=P@$$word;";
    /// <summary>
    /// Authentification (login/pwd)
    /// </summary>
    /// <param name="login"></param>
    /// <param name="pwd"></param>
    /// <returns></returns>
    1 référence | louise, il y a 2 jours | 1 auteur, 1 modification
    public static Boolean Authentification(string login, string pwd)
    {
        string req = "SELECT * FROM responsable ";
        req += "WHERE login = @login AND pwd = SHA2(@pwd, 256);";
        Dictionary<string, object> parameters = new Dictionary<string, object>();
        parameters.Add("@login", login);
        parameters.Add("@pwd", pwd);
        ConnexionBDD curseur = ConnexionBDD.GetInstance(connectionString);
        curseur.ReqSelect(req, parameters);

        if (curseur.Read())
        {
            curseur.Close();
            return true;
        }
        else
        {
            curseur.Close();
            return false;
        }
    }
```

3.3 Coder les modèles

Ce package contiendra les classes métiers correspondant aux tables de la base de données. (“personnel”, “absence”, “motif” et “service”).
Des “getter” sont créés afin d’accéder aux données de la table.
Comme présenté ici, avec la classe personnel :

```
27 références | louise, il y a 2 jours | 1 auteur, 1 modification
public class Personnel
{
    private int idpersonnel;
    private string nom;
    private string prenom;
    private string tel;
    private string mail;
    private int idservice;
    private string service;

    /// <summary>
    /// Getter : idpersonnel
    /// </summary>
    13 références | louise, il y a 2 jours | 1 auteur, 1 modification
    public int Idpersonnel { get => idpersonnel; }

    /// <summary>
    /// Getter : nom
    /// </summary>
    4 références | louise, il y a 2 jours | 1 auteur, 1 modification
    public string Nom { get => nom; }

    /// <summary>
    /// Getter : prenom
    /// </summary>
    4 références | louise, il y a 2 jours | 1 auteur, 1 modification
    public string Prenom { get => prenom; }

    /// <summary>
    /// Getter : tel
    /// </summary>
    2 références | louise, il y a 2 jours | 1 auteur, 1 modification
    public string Tel { get => tel; }

    /// <summary>
    /// Getter : mail
    /// </summary>
    2 références | louise, il y a 2 jours | 1 auteur, 1 modification
    public string Mail { get => mail; }

    /// <summary>
    /// Getter : idservice
    /// </summary>
    2 références | louise, il y a 2 jours | 1 auteur, 1 modification
    public int Idservice { get => idservice; }

    /// <summary>
    /// Getter : service
    /// </summary>
    0 références | louise, il y a 2 jours | 1 auteur, 1 modification
    public string Service { get => service; }
```

Il est également nécessaire d'utiliser des constructeurs pour ajouter un personnel, lorsqu'il est appelé, ses paramètres sont remplis et envoyés, puis les propriétés sont valorisées :

```
3 références | louise, il y a 2 jours | 1 auteur, 1 modification
public Personnel(int idpersonnel, string nom, string prenom, string tel, string mail, int idservice, string service)
{
    this.idpersonnel = idpersonnel;
    this.nom = nom;
    this.prenom = prenom;
    this.tel = tel;
    this.mail = mail;
    this.idservice = idservice;
    this.service = service;
}
```

Les classes Motif, Service et Absence auront la même forme que cette classe Personnel.

3.4 Générer la documentation technique.

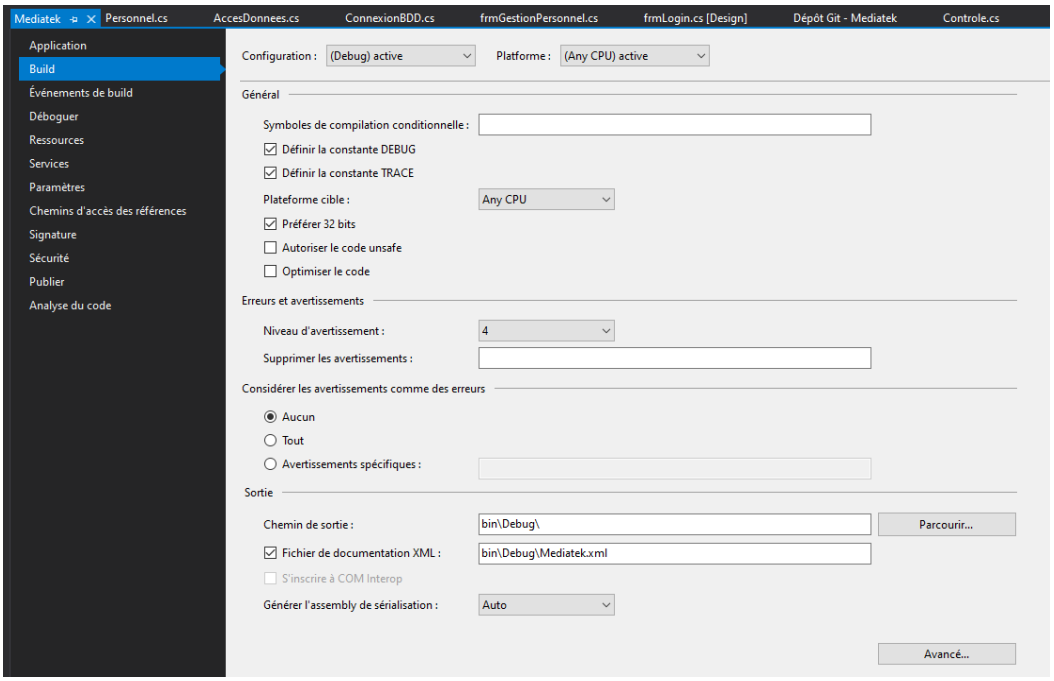
Le fichier XML de documentation technique est généré automatiquement grâce à Visual Studio.

Pour ce faire :

▹ Clic droit sur la solution > Propriétés >

▹ Build > Sortie > cocher la case "Fichier de documentation XML"

Ce fichier se situera dans le fichier bin\Debug de l'application.



ÉTAPE 4

Coder les fonctionnalités de l'application

J'ai tout d'abord écrit les requêtes SQL en fonction des cas d'utilisation du dossier complémentaire à l'aide de phpMyAdmin, dans ce sens, j'étais sûre de ne pas faire d'erreur.

Puis, j'ai écrit les méthodes restantes de la classe "AccesDonnees", contenant les requêtes SQL écrites plus tôt, l'instance de connexion, ainsi qu'un return pour retourner les données si la requête était de type SELECT.

Ici, pour récupérer la table personnel de la base de données :

```
1 référence | louise, il y a 2 jours | 1 auteur, 1 modification
public static List<Personnel> GetLesPersonnels()
{
    List<Personnel> lesPersonnels = new List<Personnel>();
    string req = "SELECT p.idpersonnel as idpersonnel, p.nom AS nom, p.prenom AS prenom, p.tel AS tel, p.mail AS mail, s.idservice AS idservice, s.nom AS 'service'";
    req += "FROM personnel p JOIN service s USING (idservice)";
    req += "ORDER BY nom, prenom;";

    ConnexionBDD curseur = ConnexionBDD.GetInstance(connectionString);
    curseur.ReqSelect(req, null);
    while (curseur.Read())
    {


        Personnel personnel = new Personnel((int)curseur.Field("idpersonnel"),
                                            (string)curseur.Field("nom"),
                                            (string)curseur.Field("prenom"),
                                            (string)curseur.Field("tel"),
                                            (string)curseur.Field("mail"),
                                            (int)curseur.Field("idservice"),
                                            (string)curseur.Field("service"));
        lesPersonnels.Add(personnel);

    }
    curseur.Close();
    return lesPersonnels;
}
```

Il a fallu par la suite créer les méthodes Get, Add, Del et Udpate de la classe Controle du package controleur afin de gérer les différentes demandes, car la classe "AccesDonnes" ne peut être appelée directement dans le package vue.
J'ai ensuite écrit les procédures événementielles du package vue, qui permettront à l'utilisateur de faire ses différentes requêtes.

Mais aussi les différentes méthodes qui sollicitent le controleur afin d'effectuer les diverses manipulations.

Ici , la méthode "RemplirListePersonnel", qui permet de remplir l'objet graphique DataGridView nommé "lstPersonnel".
Et la méthode "RemplirComboBoxService", qui permet de remplir l'objet graphique comboBox nommé "cboBoxService".

```
namespace MediaTek.connexion
{
    /// <summary>
    /// Connexion à la base de données + exécution des requêtes
    /// </summary>
    24 références | louise, Il y a 14 heures | 1 auteur, 2 modifications
    public class ConnexionBDD
    {
        /// <summary>
        /// Objet de connexion à la base de données
        /// </summary>
        private MySqlConnection connection;

        /// <summary>
        /// Constructeur privé pour créer la connexion + l'ouvrir
        /// </summary>
        /// <param name="stringConnect">chaîne de connexion</param>
        1 référence | louise, il y a 3 jours | 1 auteur, 1 modification
        private ConnexionBDD(string stringConnect)
        {
            try
            {
                connection = new MySqlConnection(stringConnect);
                connection.Open();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Application.Exit();
            }
        }
```

lstPersonnel se nomme ainsi car j'avais tout d'abord utilisé une ListBox en pensant que cet objet était idéal. J'ai d'abord eu du mal à afficher les informations de la base de données, et stagné pendant un moment, pensant que le problème était un problème de connexion à la BDD. J'ai donc fait des recherches et ai découvert l'objet graphique DataGridView.

Une fois l'affichage réussi, je ne comprenais pas comment afficher les informations d'un personnel dans les txt correspondants. J'ai

trouvé un site expliquant les procédures événementielles CellClick et CellCenter. Pour ne pas m'embrouiller, j'ai préféré ne pas renommer lstPersonnel.
Ici la procédure événementielle "btnAjt_Click" qui permet d'ajouter un personnel (ou une absence) en fonction des objets graphiques affichés. La procédure vérifie également si les champs Nom, Prénom, Tél et Mail sont remplis et si un élément est sélectionné, dans la comboBox Service :

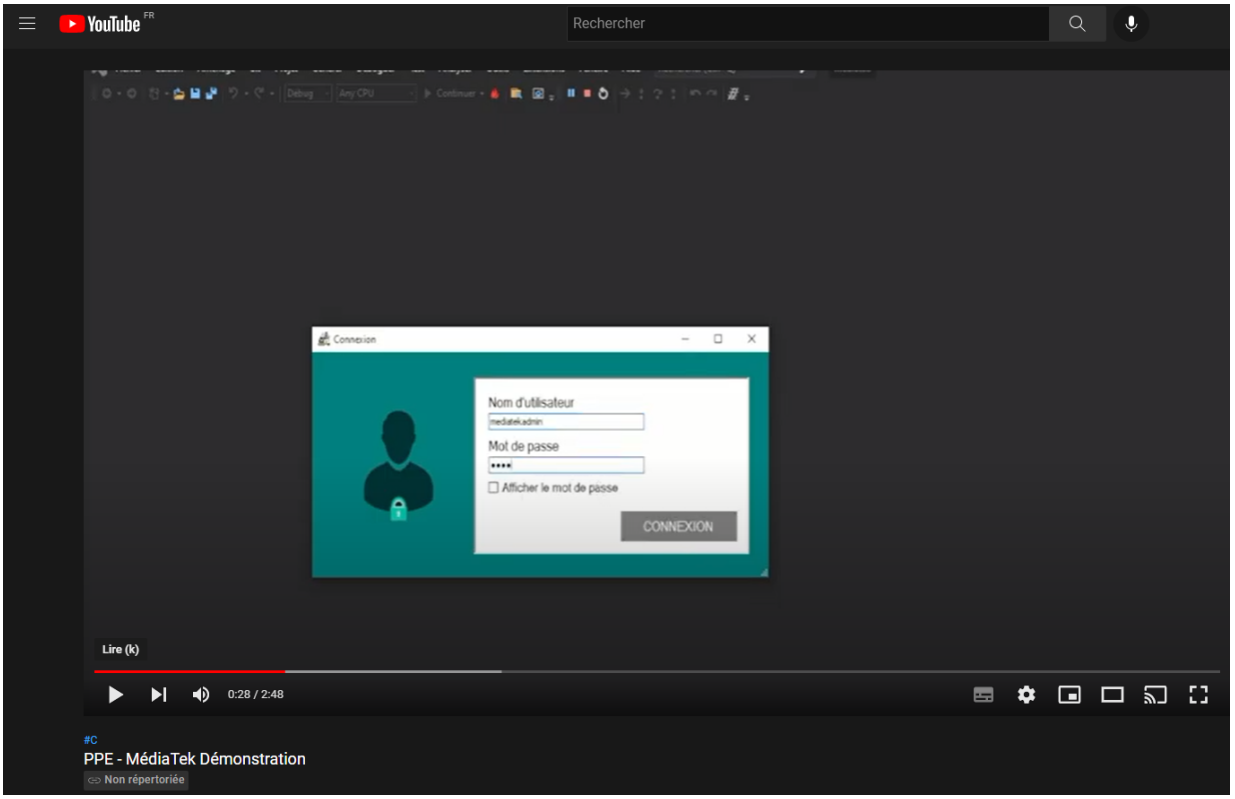
```
private void btnAjt_Click(object sender, EventArgs e)
{
    // Ajout d'un personnel
    if (!show)
    {
        //si les champs ne sont pas vides et q'un champs est séléctionné dans cboBoxService
        if (!txtNom.Text.Equals("") && !txtPrenom.Text.Equals("") && !txtTel.Text.Equals("") && !txtMail.Text.Equals("") && cboBoxService.SelectedIndex != -1)
        {
            int idpersonnel = 0;
            Service service = (Service)bdgService.List[bdgService.Position];
            Personnel personnel = new Personnel(idpersonnel, txtNom.Text, txtPrenom.Text, txtTel.Text, txtMail.Text, service.Idservice, service.Nom);
            controle.AddPersonnel(personnel);

            int nbRecord = lstPersonnel.Rows.Count;
            RemplirListePersonnel();
            if (nbRecord < lstPersonnel.Rows.Count)
            {
                lstPersonnel.ClearSelection();
                Vider();
                MessageBox.Show("Le personnel a été ajouté avec succès.", "Information");
            }
            else
            {
                // Le personnel existe dans la BDD
                MessageBox.Show("Veuillez entrer un nouveau personnel.", "Information");
            }
        }
        else
        {
            // Les champs ne sont pas tous remplis
            MessageBox.Show("Veuillez remplir tous les champs.", "Information");
        }
    }
```

De plus, pour éviter toute erreur, il était nécessaire d'ajouter un test qui vérifie que le personnel ajouté n'existe pas déjà dans la base de données.

ÉTAPE 5

Création d'une documentation utilisateur vidéo



J'ai réalisé la capture vidéo grâce au logiciel OBS Studio, et ai enregistré l'audio simplement avec l'application microphone de mon téléphone.
J'ai ensuite monté la vidéo avec le logiciel iMovie.

Afin d'ajouter facilement la vidéo à mon portfolio, je l'ai mise en ligne en non-répertoriée sur YouTube (la vidéo est en ligne, mais non publique, elle n'est accessible que via un lien).

lien de la vidéo : https://youtu.be/bEbOUj1URsI

ÉTAPE 6

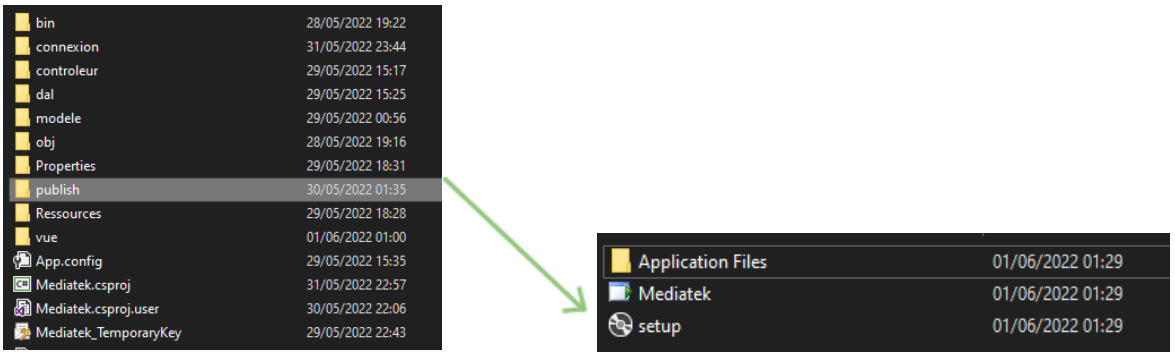
6.1 Générer le déploiement de l'application

6.2 Créer le portfolio en ligne

6.3 Bilan

6.1 Générer le déploiement de l'application

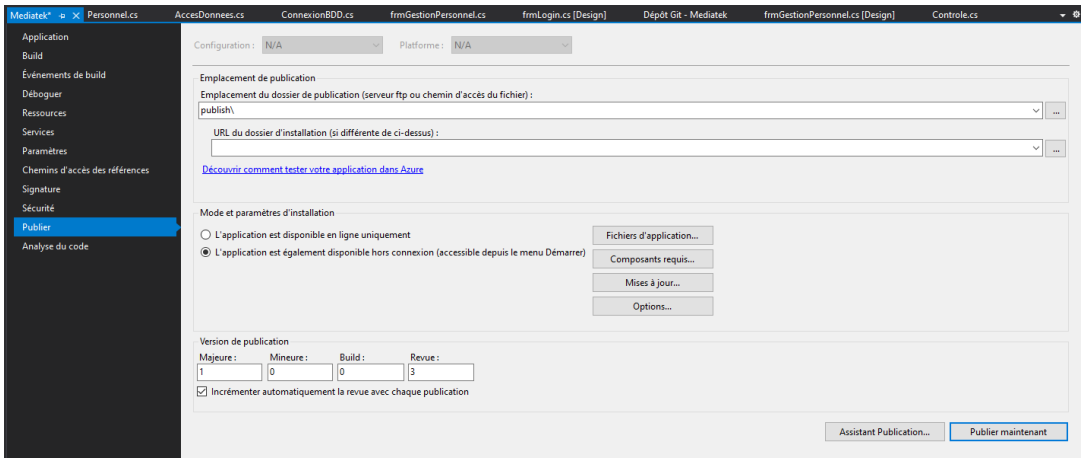
Le déploiement de l'application permet de générer un fichier "publish" contenant un setup, qui permet d'installer MediaTek, sa solution, ainsi qu'un dossier avec des fichiers nécessaires au fonctionnement de Mediatek.
Il suffira simplement de compresser ce fichier (grâce à Winrar par exemple) pour l'envoyer et l'installer sur un autre poste.



Pour ce faire :

▹ Menu Projet > Propriété Mediatek > Publier > Publier Maintenant

6.2



J'ai choisi de coder moi-même mon site, sans utiliser de CMS comme Wix ou WordPress. J'ai toujours affectionné le front-end, et code depuis que je suis en seconde, grâce à l'option informatique choisie au lycée. J'ai beaucoup aimé réaliser mon site web.

Comme demandé dans la consigne, mon portfolio disposera d'un lien renvoyant vers le dépôt Github de l'application Mediatek. Il sera aussi possible de télécharger ce compte rendu. Je l'hébergerai grâce à Github.

Dépôt distant du portfolio sur Github : https://github.com/uu-0/portfolio--PPE

Portfolio publié: https://uu-0.github.io/portfolio--PPE/

Mon profil Github : https://github.com/uu-0

BILAN

Bien qu'à première vue, j'ai cru passer un mauvais moment en réalisant ce projet, je me suis quand même bien amusée. Certaines étapes m'ont bloquée pendant de longs moments, et je me suis sentie démotivée, mais j'ai tout compte fait réussi à surmonter les obstacles qui s'étaient présentés. J'ai beaucoup apprécié réaliser le design de l'application, mais également celui de mon portfolio, que j'ai recommencé à plusieurs reprises, étant trop indécise. Je suis fière d'avoir persévéré et de m'être autant investie dans ce devoir. Cela m'a permis de mettre en situation les compétences apprises en cours, et d'accroître mes acquis grâce à diverses recherches afin de mener à bien ce projet. Les points vus lors de ma formation me sont dorénavant beaucoup plus parlants. Réaliser cette application m'a fait comprendre que rien n'est insurmontable, avec une approche méthodique et organisée.