MediaTek86

Présentation de l'application

Sommaire

Contexte

Je travaille en tant que technicien développeur junior pour l'ESN InfoTech Services 86. Nous venons de remporter le marché pour différentes interventions au sein du réseau MediaTek86, notamment dans le domaine du développement d'applications.

On m'a confié la responsabilité de développer une application de bureau qui permettra de gérer le personnel de chaque médiathèque, en assignant les employés à des services spécifiques et en gérant leurs absences.

Cette application sera utilisée sur un seul poste, au sein du service administratif.

En ce qui concerne le système de gestion de base de données relationnelles (SGBDR), j'ai la possibilité de travailler avec MySQL ou MariaDB. Quant au langage de programmation, je vais coder en C#.

Objectif

Créer une application permettant d'ajouter, modifier et supprimer des membres du personnel de la médiathèque.
Il faut aussi pouvoir ajouter, modifier et supprimer les absences pour chaque personnel.
Le tout possible en se connectant a l'aide d'un identifiant et un mots de passe.

Etapes de création

Création du projet

Création de la base de données

Tout d'abord, nous devons créer une base de données et importer les différentes tables qui nous ont été fournis.
Ensuite, nous devons créer un utilisateur qui ai les droits d'accéder a la base de données MySQL

```
CREATE USER 'foo'@'localhost' IDENTIFIED BY 'bar';
GRANT USAGE ON *.* TO 'foo'@'localhost';
GRANT ALL PRIVILEGES ON `mlr1`.* TO 'foo'@'localhost';
```

Puis nous alimentons la base de données avec de fausses données afin d'avoir de la matière pour travailler plus tard.

Création du projet "application"

Nous créons ensuite un projet C# Windows Form vierge afin de créer notre application.
Notre application aura pour structure Model View Controll

Upload sur un dépôt distant

Une fois le projet créé, il faut l'upload sur Github dans notre cas afin de bénéficier d'un versionning de notre application et dans le future pouvoir travailler en collaboration sur le même projet.

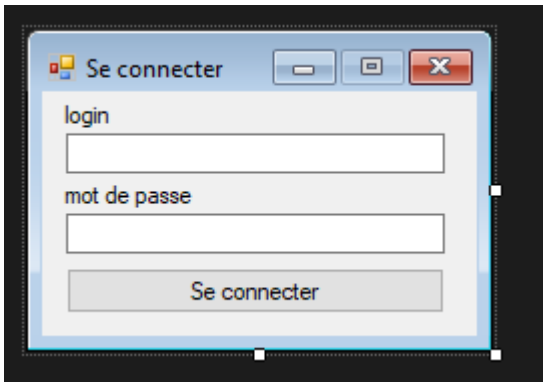
Codage de l'application

Création des interfaces

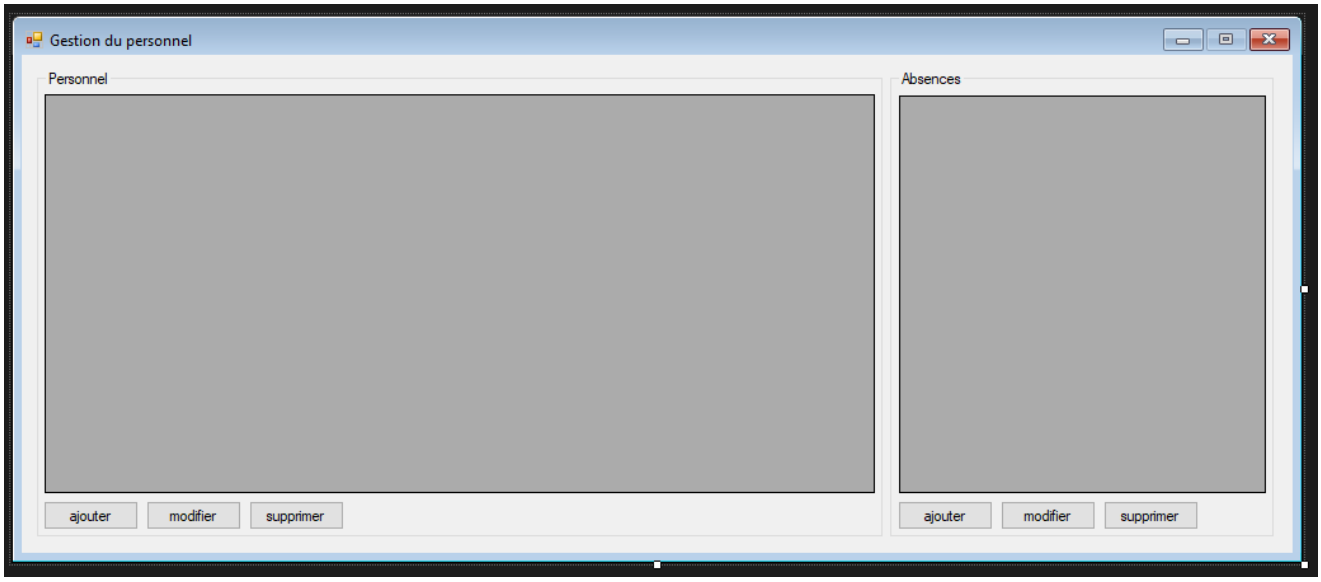
Comme je l'ai dit notre application sera sous le model MVC, ainsi notre application est de cette forme:

```
.
└── MediaTek86/
    ├── model/
    ├── view/
    ├── controller/
    └── Program.cs
```

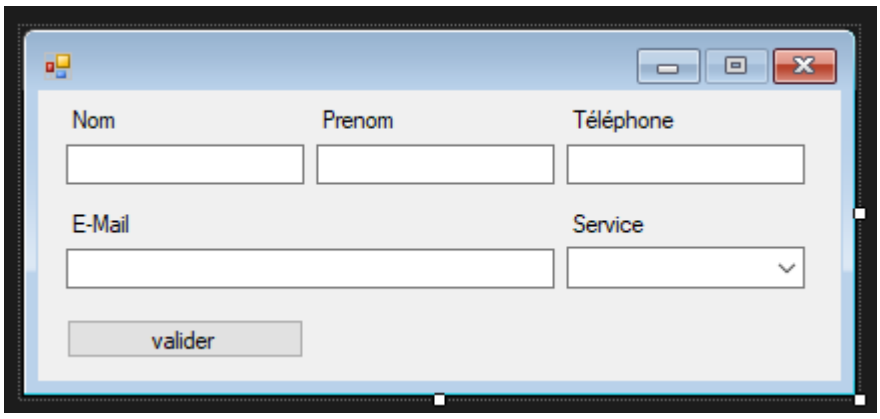
Je commence d'abord par designer la fenêtre de connexion:

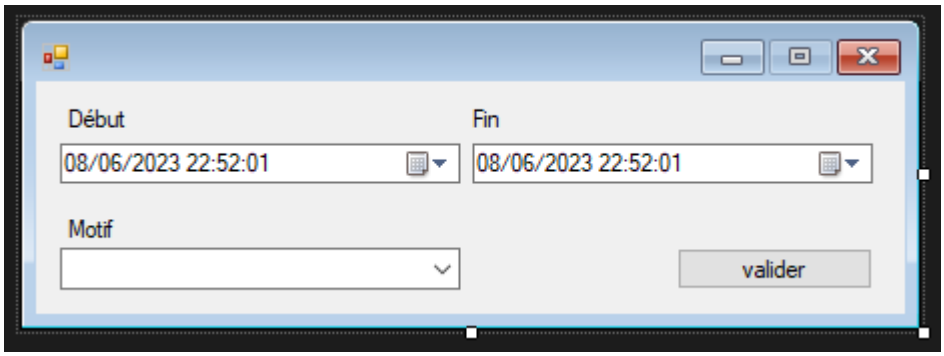


Une fois cette rapide étape passé, je m'attaque au design du corp de l'application



Je crée ensuite les différents formulaires pour ajouter/modifier le personnel et les absences






```
.
└── MediaTek86/
    ├── model/
    ├── view/
    │   ├── Manager/
    │   │   ├── Manager.cs
    │   │   ├── PersonnelDataInput.cs
    │   │   └── AbsenceDataInput.cs
    │   └── Login.cs
    ├── controller/
    └── Program.cs
```

Création connexion a la base de données

Ensuite, j'ajoute un nouveau dossier a mon projet: `bddManager` qui contiendra une class me permettant de communiquer avec la base de données MySQL.

```
.
└── MediaTek86/
    ├── bddManager/
    │   └── BddManager.cs
    ├── model/
    ├── view/
    │   ├── Manager/
    │   │   ├── Manager
    │   │   ├── PersonnelDataInput
    │   │   └── AbsenceDataInput
    │   └── Login.cs
    ├── controller/
    └── Program.cs
```

Le code dans `BddManager.cs` provient de ce repo
J'ai uniquement modifier la méthode `ReqSelect` afin d'avoir en type de retour un dictionnaire

(dans le code d'origine c'est une liste)
Ainsi, mes clés seront tout simplement le nom de mes champs !

```
public List<Dictionary<string, object>> ReqSelect(string stringQuery,
Dictionary<string, object> parameters = null)
{
        MySqlCommand command = new MySqlCommand(stringQuery, connection);

        if (!(parameters is null))
        {
                foreach (KeyValuePair<string, object> parameter in parameters)
                {
                        command.Parameters.Add(new MySqlParameter(parameter.Key,
parameter.Value));
                }
        }

        command.Prepare();

        MySqlDataReader reader = command.ExecuteReader();
        List<Dictionary<string, object>> records = new List<Dictionary<string,
object>>();

        while (reader.Read())
        {
                Dictionary<string, object> row = new Dictionary<string, object>
();

                for (int i = 0; i < reader.FieldCount; i++)
                {
                        string columnName = reader.GetName(i);
                        object value = reader.GetValue(i);
                        row[columnName] = value;
                }

                records.Add(row);
        }

        reader.Close();

        return records;
}
```

Création des models

J'ai ensuite attaquer la création des différents models (`Personnel`, `Absence`, `Motif`, `Service`) qui sont ni plus ni moins des objets reflétant un enregistrement de notre base de

données.

Ajout d'un dossier `dal`

Dans ce dossier, nous allons écrire toute nos requêtes SQL.
Par exemple, nous auront un fichier `PersonnelAccess.cs` qui aura une méthode `GetPersonnels()` qui nous retourne une liste de toute les personnels :

```
public List<Personnel> GetPersonnels()
{
        List<Personnel> personnels = new List<Personnel>();

        if (access == null) return personnels;

        string req = @"SELECT
                personnel.IDPERSONNEL,
                personnel.NOM,
                personnel.PRENOM,
                personnel.TEL,
                personnel.MAIL,
                service.IDSERVICE,
                service.NOM AS NOMSERVICE
        FROM
                personnel
        INNER JOIN service ON service.IDSERVICE = personnel.IDSERVICE";

        try
        {
                List<Dictionary<string, object>> records =
access.Manager.ReqSelect(req);

                foreach (Dictionary<string, object> row in records)
                {
                        personnels.Add(new Personnel(
                                (int)row["IDPERSONNEL"],
                                new Service(
                                        (int)row["IDSERVICE"],
                                        (string)row["NOMSERVICE"]
                                ),
                                (string)row["NOM"],
                                (string)row["PRENOM"],
                                (string)row["TEL"],
                                (string)row["MAIL"]
                        ));
                }
        }
        catch (Exception e)
```

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

        return personnels;
}
```

Et nous faisons ceci pour chaque model, et créons les requêtes dont nous auront besoin dans notre application !

Création des controllers

Comme notre application est relativement simple, les controllers contiennent les mêmes méthodes que nos classes dans le dossier `dal`.
Voici un exemple pour le controller du personnel

```
class PersonnelController
{
        private readonly PersonnelAccess personnelAccess;

        public PersonnelController()
        {
                personnelAccess = new PersonnelAccess();
        }

        public List<Personnel> GetPersonnels()
        {
                return personnelAccess.GetPersonnels();
        }

        public void DeletePersonnels(int personnelId)
        {
                personnelAccess.DeletePersonnel(personnelId);
        }

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

        public void UdpatePersonnel(Personnel personnel)
        {
                personnelAccess.UpdatePersonnel(personnel);
        }
}
```

Utilisation des controllers

Maintenant que la liaison avec les données est possible, nous pouvons ajouter du dynamisme a notre application en important les données de notre BDD MySQL.

```
public Manager()
{
        this.personnelController = new PersonnelController();
        this.absenceController = new AbsenceController();
        this.InitializeComponent();
```

Ici on ajoute les différentes colonnes de notre DataGridView servant a l'affichage du personnel

```
        //// PERSONNEL DATAGRID
        this.dataGridPersonnel.AutoGenerateColumns = false;
        this.refreshPersonnelData();

        // Création des colonnes
        DataGridViewTextBoxColumn nomColumn = new DataGridViewTextBoxColumn();
        nomColumn.Name = "surname";
        nomColumn.DataPropertyName = "nom";
        nomColumn.HeaderText = "Nom";

        DataGridViewTextBoxColumn prenomColumn = new DataGridViewTextBoxColumn();
        prenomColumn.Name = "name";
        prenomColumn.DataPropertyName = "prenom";
        prenomColumn.HeaderText = "Prénom";

        DataGridViewTextBoxColumn telColumn = new DataGridViewTextBoxColumn();
        telColumn.Name = "phone";
        telColumn.DataPropertyName = "tel";
        telColumn.HeaderText = "Téléphone";

        DataGridViewTextBoxColumn mailColumn = new DataGridViewTextBoxColumn();
        mailColumn.Name = "email";
        mailColumn.DataPropertyName = "mail";
        mailColumn.HeaderText = "Email";

        DataGridViewTextBoxColumn serviceColumn = new
DataGridViewTextBoxColumn();
        serviceColumn.Name = "serviceName";
        serviceColumn.DataPropertyName = "service.nom";
        serviceColumn.HeaderText = "Service";

        // Ajout des colonnes au DataGridView `dataGridPersonnel`
        this.dataGridPersonnel.Columns.AddRange(nomColumn, prenomColumn,
```

```
telColumn, mailColumn, serviceColumn);
```

Et ici pour les Absences du personnel

```
        //// ABSENCE DATAGRID
        this.dataGridAbsence.AutoGenerateColumns = false;

        // Création des colonnes
        DataGridViewTextBoxColumn debutColumn = new DataGridViewTextBoxColumn();
        debutColumn.Name = "dateStart";
        debutColumn.DataPropertyName = "dateDebut.dateString";
        debutColumn.HeaderText = "Date début";

        DataGridViewTextBoxColumn finColumn = new DataGridViewTextBoxColumn();
        finColumn.Name = "dateEnd";
        finColumn.DataPropertyName = "dateFin.dateString";
        finColumn.HeaderText = "Date fin";

        DataGridViewTextBoxColumn motifColumn = new DataGridViewTextBoxColumn();
        motifColumn.Name = "reason";
        motifColumn.DataPropertyName = "motif.libelle";
        motifColumn.HeaderText = "Motif";

        // Ajout des colonnes au DataGridView `dataGridPersonnel`
        this.dataGridAbsence.Columns.AddRange(debutColumn, finColumn,
motifColumn);
}
```

Ces méthodes nous permettent d'actualiser, et trier les données de nos DataGrid

```
// Refresh content of the personnel data grid
private void refreshPersonnelData()
{
        List<Personnel> personnels = this.personnelController.GetPersonnels();

        personnels.Sort((x, y) => x.nom.CompareTo(y.nom));

        this.dataGridPersonnel.DataSource = personnels;
}

private void refreshAbsenceData()
{
        Personnel personnel =
(Personnel)this.dataGridPersonnel.CurrentRow.DataBoundItem;
        List<Absence> absences =
this.absenceController.GetAbsences(personnel.id);

        absences.Sort((x, y) => x.dateDebut.CompareTo(y.dateDebut));
```

```
        this.dataGridAbsence.DataSource = absences;
}
```

Comme vous l'avez peut-être remarqué, les données passées dans les DataGrids sont des listes d'objets, qui eux même contiennent d'autres objets.
Un problème s'est donc posé, comment afficher les propriétés d'un objet dans un objet ?
Et bien voici une petite fonction qui nous permet de faire cela:

```
private void dataGridPersonnel_CellFormatting(object sender,
DataGridViewCellFormattingEventArgs e)
        {
                // Get values from child object
                DataGridViewColumn column =
this.dataGridPersonnel.Columns[e.ColumnIndex];
                if (column.DataPropertyName.Contains("."))
                {
                        object data =
this.dataGridPersonnel.Rows[e.RowIndex].DataBoundItem;
                        string[] properties = column.DataPropertyName.Split('.');
                        for (int i = 0; i < properties.Length && data != null;
i++)
                                data =
data.GetType().GetProperty(properties[i]).GetValue(data);

this.dataGridPersonnel.Rows[e.RowIndex].Cells[e.ColumnIndex].Value = data;
                }
        }
```

Elle peut paraitre complexe a première vu, mais cela est relativement simple. Cette méthode est liée a l'évènement du formatting des cellules de notre DataGrid. Ainsi, nous détectons si la propriété `DataPropertyName` de la colonne contient un point " `.` ". Pourquoi un `.` ? Car j'ai décidé que pour accéder a une propriété d'un objet enfant, nous l'identifions avec un point. par exemple dans un objet de type `Personnel`, nous avons la propriété service de type `Service`, et l'objet `Service` lui contient `nom` et `id`. Ainsi, si nous voulons le nom du service, nous devons mettre dans `DataPropertyName` : `service.nom` *(propriété `service` de personnel, et propriété `nom` de service)*.
D'où le point.
Et ensuite nous récupérons juste la valeur de la propriété voulu pour l'afficher !

Utilisation des formulaires

Maintenant, venons-en a l'utilisation des formulaires `PersonnelDataInput` et `AbsenceDataInput`.
Voici le fonctionnement de `PersonnelDataInput` :

```
public partial class PersonnelDataInput : Form
{
        const int defaultId = 0;

        /// <summary>
        /// Personnel object generate with value passed in inputs
        /// </summary>
        public Personnel personnel { get; set; }
        /// <summary>
        /// Base Personnel passed in constructor
        /// </summary>
        public readonly Personnel oldPersonnel;
        private readonly ServiceController serviceController;

        /// <summary>
        /// Constructor of PersonnelDataInput
        /// </summary>
        /// <param name="personnel">Fill inputs with thit datas</param>
        public PersonnelDataInput(Personnel personnel = null)
        {
                this.InitializeComponent();

                this.oldPersonnel = personnel;
                this.serviceController = new ServiceController();
                List<Service> services = serviceController.GetServices();
                this.ddService.DataSource = services;
                this.ddService.DisplayMember = "nom";
                this.ddService.SelectedIndex = -1;
```

Cette condition nous permet de gérer le pré-remplissage des entrées. Ainsi, si un personnel est passé en paramètre du constructeur, ses propriétés seront récupérer pour être affectées aux différentes entrés *(En fonction de cela, le titre du formulaire change aussi)*

```
                if (personnel is Personnel){
                        this.Text = "Modifier un membre du personnel";
                        this.txtSurname.Text = personnel.nom;
                        this.txtName.Text = personnel.prenom;
                        this.txtPhone.Text = personnel.tel;
                        this.txtEmail.Text = personnel.mail;
                        this.ddService.SelectedIndex = services.FindIndex(service
=> service.id == personnel.service.id);
                }
                else{
                        this.Text = "Ajouter un membre au personnel";
                }
        }

        /// <summary>
```

```
        /// Triggered when validation button is pushed
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnAccept_Click(object sender, EventArgs e)
        {
```

On récupère tout les champs pour ensuite vérifier leurs remplissage ou non. Si ca n'est pas le cas, la fonction l'action du clique n'abouti a rien sinon on continu l'exécution.

```
                string surname = txtSurname.Text;
                string name = txtName.Text;
                string tel = txtPhone.Text;
                string mail = txtEmail.Text;
                Service service = (Service)this.ddService.SelectedItem;

                if (String.IsNullOrEmpty(surname) || String.IsNullOrEmpty(name)
|| String.IsNullOrEmpty(tel) || String.IsNullOrEmpty(mail) || service == null)
return;

                int id = PersonnelDataInput.defaultId;
```

Ce bout de code nous permet de gérer la validation de modification *(donc uniquement si un personnel a été passé en constructeur)*

```
                if(this.oldPersonnel != null)
                {
                        DialogResult confirm = MessageBox.Show("Êtes-vous sûr de
vouloir modifier cette personne ?", "Modifier cette utilisateur ?",
MessageBoxButtons.YesNo);

                        if (confirm == DialogResult.No)
                        {
                                this.Close();
                                return;
                        }

                        id = this.oldPersonnel.id;
                }

                this.personnel = new Personnel(
                        id,
                        service,
                        surname,
                        name,
                        tel,
                        mail
                );
```

```
                this.DialogResult = System.Windows.Forms.DialogResult.OK;
                this.Close();
        }
}
```

Puis nous récupérons les données dans notre personnel en reprenant l'instance de notre formulaire, et accédant a la propriété `personnel` !
Exemple pour la création d'un personnel :

```
private void btnAddPersonnel_Click(object sender, EventArgs e)
{
        PersonnelDataInput personnelData = new PersonnelDataInput();

        if (personnelData.ShowDialog() == DialogResult.Cancel) return;

        this.personnelController.CreatePersonnel(personnelData.personnel);
        this.refreshPersonnelData();
}
```

Et on fait exactement la meme chose pour `AbsenceDataInput`
Et voila ! Notre application est maintenant terminé !