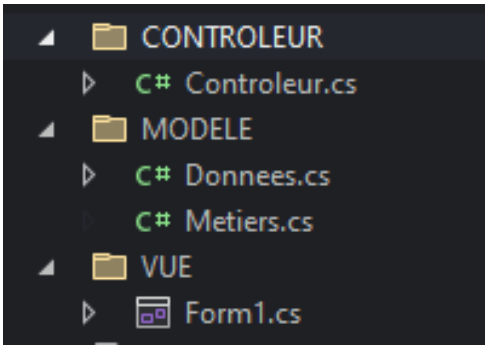
DOCUMENTATION Vichy Papeterie

Le projet a été conçu avec une architecture MVC, soit Modèle-Vue-Contrôleur :



Dans le dossier “**CONTROLEUR**” se trouve le fichier “**Controleur.cs**” qui sert à manipuler les classes métiers et d’envoyer les informations, sous formes de chaînes, à la vue.

```
public static string CreerFacture(string Client, string Article, bool r, int rchiffre, int qte)
{
    Client c = new Client(int.Parse(Client.Split('-')[0]), Client.Split('-')[1]);
    Article a = new Article(int.Parse(Article.Split('-')[0]), Article.Split('-')[1],
        decimal.Parse(Article.Split('-')[2].Replace("€", "")));
    Facture fact = new Facture(c);
    Ajouter(fact, a, qte, r, rchiffre);
    //SQL.EnregistrerFacture(fact, true);
    return fact.ToString();
}
```

Par exemple, pour créer une facture on envoie toutes les informations requises que contient la vue, utilisant des “CheckBox, TextBox ou ListBox” pour construire une facture et la renvoyer en chaîne.

Quand on a terminé de créer une facture, on peut l’enregistrer :

```
public static void EnregistrerFacture(string facture)
{
    SQL.EnregistrerFacture(ToFacture(facture), true);
}
```

À ce moment-là on utilise la classe technique de la couche modèle, elle se situe dans le dossier “**MODELE**” et dans le fichier “**Donnees.cs**”.

Il y a deux classes techniques dans ce fichier, la classe SQL et la classe WCF, l'un utilise une base de données, l'autre un WebService.
Pour enregistrer la facture on utilise la classe SQL :

```
public class SQL
{
    private OdbcConnection connexion;
    private OdbcCommand commande;
    private OdbcDataReader reader;

    public string requête;

    // Constructeur

    public SQL(string r = "")...

    public static List<List<string>> Select(string colonne, string table)...

    public static Client Client(string raison_social)...

    public static Client Client(int codeClient)...

    public static Article Article(string designation)...

    public static Article Article(int code)...

    public static void EnregistrerFacture(Facture f, bool tout)...

    public static Facture ExporterFacture(int numero)...
}
```

Cette classe ne contient que des méthodes/fonctions static, cela permet de créer l'objet SQL dans les méthodes/fonctions directement ce qui est plus pratique car, on crée l'objet, on l'utilise pour récupérer les données et on le détruit.
L'utilisation n'en est que plus facile :

```
BindingList<Client> aRetourner = new BindingList<Client>();
foreach (List<string> ls in SQL.Select("*", "Client"))
{
    aRetourner.Add(new Client(int.Parse(ls[0]), ls[1]));
}
return aRetourner;
```

Pour récupérer les clients d'un table ou même les articles

```
BindingList<Article> aRetourner = new BindingList<Article>();
foreach (List<string> ls in SQL.Select("*", "Article"))
{
    aRetourner.Add(new Article(int.Parse(ls[0]), ls[1], decimal.Parse(ls[2])));
}
return aRetourner;
```

Enfin quand le contrôleur a terminer de tous traiter il renvoie à la vue. A noter que le contrôleur est une classe statique. Voici comment la vue récupère les données :

Ici la vue Bind les clients et articles reçu du contrôleur. Le paramètre booléen sert à régler la sources des données, WebServices en XML ou Base de données.

```
private void RB_WS_CheckedChanged(object sender, EventArgs e)
{
    this.Combo_Client.DataSource = Controleur.getLesClientsDSC(this.RB_BD.Checked);
    this.Combo_Article.DataSource = Controleur.getLesArticlesDSA(this.RB_BD.Checked);
}
```

Ici la vue envoie des données au contrôleur :

```
private void Enregistrement_Click(object sender, EventArgs e)
{
    Controleur.EnregistrerFacture(this.RichTextBox_Details.Text);
}
```

En effet en envoyant la facture en chaîne le contrôleur pourra en recréer une de toute pièce :

```
public static Facture ToFacture(string f)
{
    Facture aRetourner = new Facture(ToClient(f), Convert.ToInt32(GetNumeroFacture(f)), DateTime.Now);
    aRetourner.SetLesLignes(ToLLFacture(f));
    return aRetourner;
}
```

En envoyant toute la facture en chaîne, des méthodes viennent lire les données et renvoient l’objet demandé.

Ceci sert aussi quand on veut rajouter une ligne à la facture :

```
private void Button_Ajouter_Click(object sender, EventArgs e)
{
    if (this.FactureEnCours)
    {
        this.RichTextBox_Details.Text = Controleur.RajouterLigne2(this.RichTextBox_Details.Text,
            this.Combo_Article.SelectedItem.ToString(), this.CheckB_Oui.Checked,
            this.ChercheRemise(), this.RechercheQte());

    }
```

On transforme la facture qui est en chaîne, sous la forme d'un objet facture, on ajoute les lignes à cet objet, puis on renvoie ce nouvel objet en chaîne.

```
public static string RajouterLigne2(string Facture, string Article, bool r, int rchiffre, int qte)
{
    Facture f = ToFacture(Facture);
    f.Ajout(new Article(int.Parse(Article.Split('-')[0]), Article.Split('-')[1],
        decimal.Parse(Article.Split('-')[2].Replace("€", ""))), qte, r, rchiffre);
    return f.ToString();
}
```

Vue :