

RAPPORT INDIVIDUEL

Table des matières

1.	Mise en situation.....	2
2.	Choix pour l'application.....	2
2.1	justification du choix du langage	2
2.2	Pourquoi choisir l'ide Visual studio ?	3
2.3	ANALYSE	4
3.	mise en place	6
3.1	Visual studio	6
3.2	Définition et installation des bibliothèques nécessaires pour l'application	8
4.	Développement de l'application	9
2.1	XAML	10
2.2	C#.....	10
2.3	Les fonctionnalités de BASE (barre de navigation).....	11
2.4	Première interface l'accueil de l'application.....	14
2.5	deuxième fenêtre le mode instantané	15
2.6	l'interface du mode création	22
2.7	La fenêtre Charger.....	24
2.8	L'interface Periode_scenario.....	28
5.	Test Unitaire	32
5.1	Interface mode instantané	32
5.2	fenêtre de création.....	34
5.3	interface charger	34
5.4	L'interface Periode_scenario.....	36

1. MISE EN SITUATION

Rappel de la situation pour l'application de bureau (étudiant 1):

L'étudiant se charge du développement de l'application bureau « Banc de test éolienne ». Il doit créer une interface de création de scénarios de tests, une interface d'exécution de tests, une interface de commande de la soufflerie. L'application doit communiquer avec les modules Acquisition et Commande de l'étudiant 2 via socket TCP afin de récupérer les données du module Acquisition et commander la soufflerie.

L'installation de bibliothèques .NET pour la communication socket TCP et .NET MySQL pour la communication BDD. L'étudiant doit configurer et mettre en œuvre les bibliothèques .NET socket et MySQL.

2. CHOIX POUR L'APPLICATION

2.1 JUSTIFICATION DU CHOIX DU LANGAGE



Pour l'application du banc de test pour éolienne, le choix du C# est imposé avec l'IDE Visual Studio. Même si cela est imposé le comparatif reste intéressant, pour ce faire j'ai fait deux tableaux qui comparent le C# et le C++ :

Le premier tableau sera un comparatif général sur les avantages et désavantages techniques. Le deuxième tableau traitera des avantages et désavantages, sur un avis personnel.

TABLEAU COMPARATIF TECHNIQUE

C++	C#
C++ est un langage plus complexe	C# est un langage similaire au Java
En C++, vous devez gérer la mémoire manuellement	C# contrôle automatiquement la gestion de la mémoire
C++ inclut des fonctionnalités plus complexes.	C# n'a pas de fonctionnalités complexes. Il a une hiérarchie simple et assez facile à comprendre.
C++ est un langage qui fonctionne sur toutes sortes de plates-formes. Il est également populaire sur les systèmes Unix et Linux.	C#, bien que normalisé, est rarement vu en dehors de Windows.
C++ peut être multiplateforme mais fait considérablement gonfler la taille du code source (sans impact sur les performances)	C# le multiplateforme est inclut avec des bibliothèques .NET
Il est prévu pour développer des applications en console.	La programmation C# peut être utilisée pour créer des applications Windows, mobiles et console.

TABLEAU SUR CRITERE PERSONNEL

	Connaissance du langage	Documentation	Possibilité d'avoir de l'aide (0.5)	Intérêt personnel pour le langage	Ergonomie	Total
C++	0.65	1	0.5	0.5	0.5	3,15
C#	0.25	0.75	0.5	0.75	1	3.25

2.2 POURQUOI CHOISIR L'IDE VISUAL STUDIO ?

Visual Studio est un IDE parfait pour le langage C# car le C# appartient à Microsoft tout comme Visual Studio.

L'extension R# Visual Studio offre des fonctionnalités incroyables de productivité, d'analyse de code et de génération de code pour décupler l'expérience de développeur.

La bibliothèque de classes de base fournie par le Framework .NET est une aide précieuse pour les développeurs C#. Elle est directement intégrée par Visual Studio

2.3 ANALYSE

Pour commencer le développement j'ai besoin de bien comprendre le projet, pour ce faire j'ai fait trois diagrammes de séquence pour chaque partie de mon application :

DIAGRAMME DE SEQUENCE MODE INSTANTANE DE (ETUDIANT 1) :

Ce diagramme permet d'analyser le mode instantané de l'application de bureau

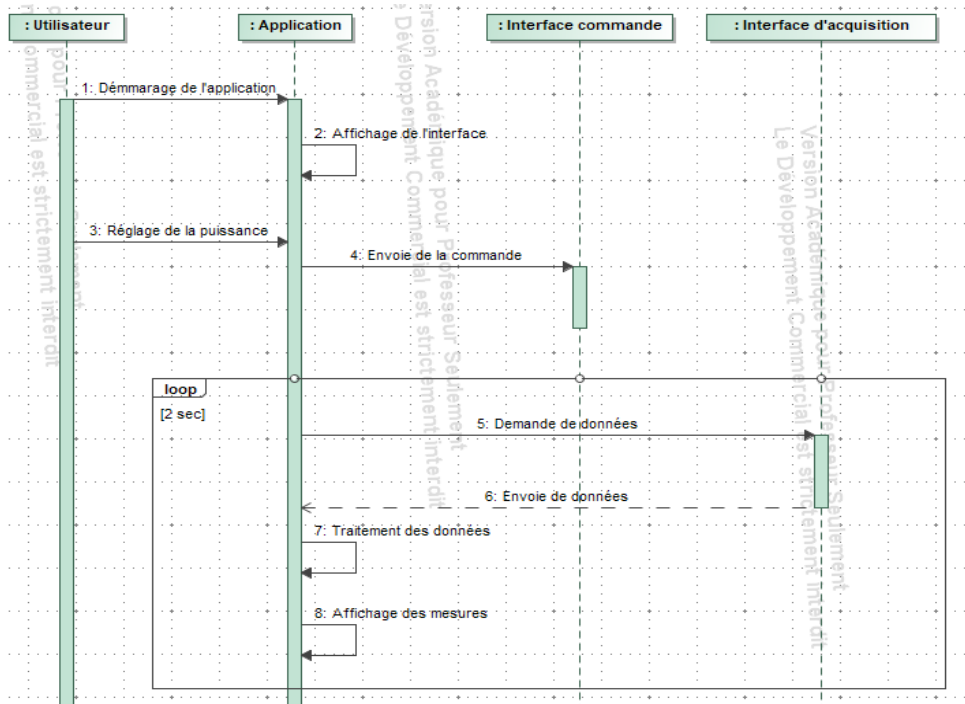


DIAGRAMME DE SEQUENCE L'APPLICATION MODE CREATION D'UN SCENARIO (ETUDIANT 1) :

Au préalable l'utilisateur allume l'application, l'application retourne l'affichage, ce diagramme permet d'avoir une vue d'ensemble sur la création.

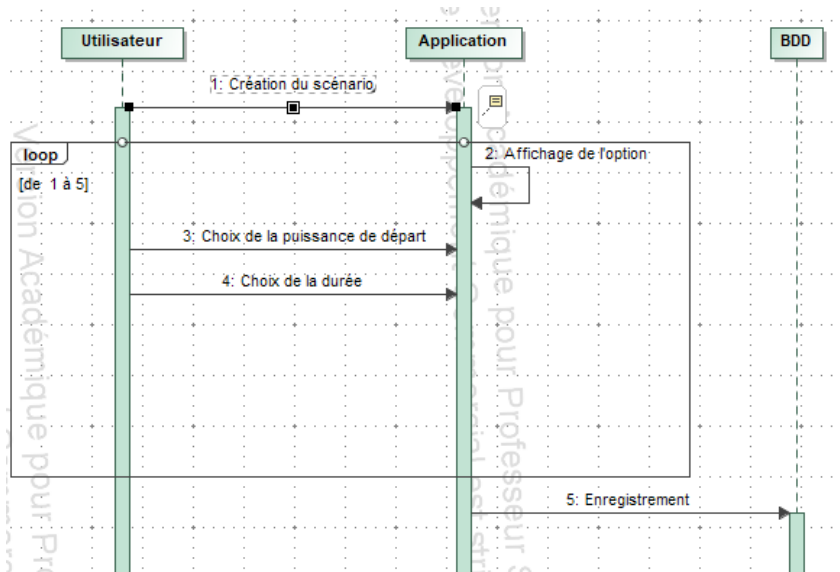
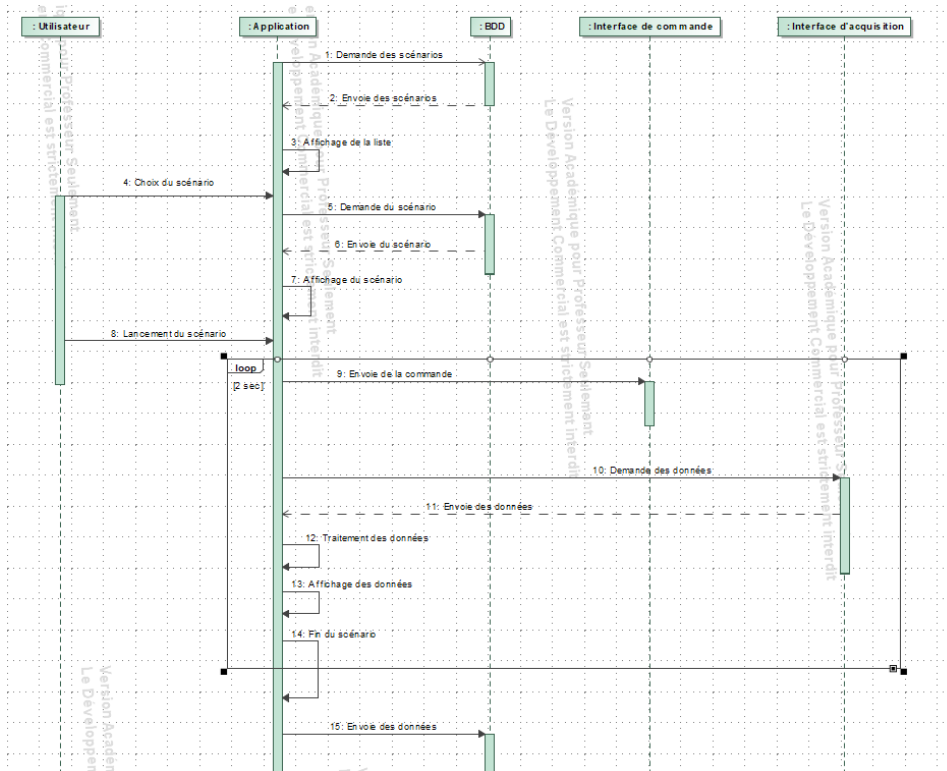


DIAGRAMME DE SEQUENCE MODE « CHARGER UN SCENARIO » (ETUDIANT 1) :

Au préalable l'utilisateur allume l'application, l'application retourne l'affichage. Comment le scénario est chargé.



Les diagrammes segmentent l'application en trois, cela me permet d'avoir une vision claire et précise pour le développement de l'application.

3. MISE EN PLACE

3.1 VISUAL STUDIO

Avant de commencer le développement de l'application de bureau il me faut quelques prérequis, il me faudra tout d'abord l'IDE « Visual Studio », l'installation de Visual Studio reste simple. Il faut se rendre sur le site officiel de Microsoft sur le lien suivant : <https://visualstudio.microsoft.com/fr/>.

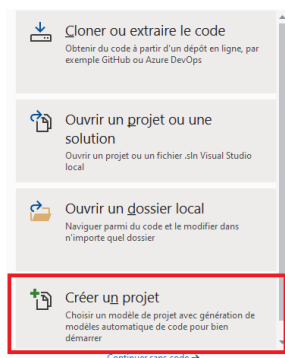
Une fois sur le site on clique sur « *Télécharger Visual Studio* » puis on prend la version « *Community 2022* »



Une fois téléchargé, on lance le fichier d'installation et on suit la procédure d'installation fournie par Microsoft.

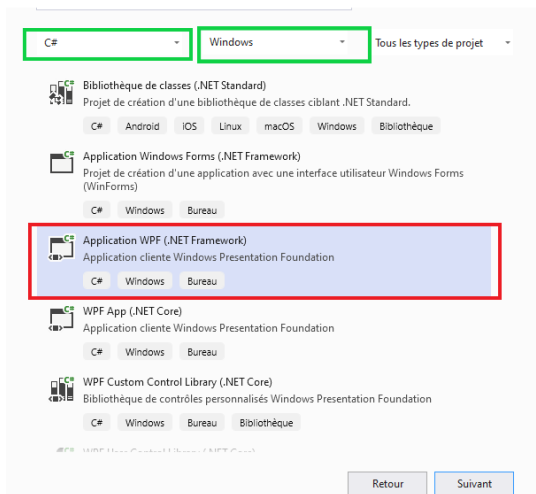
CREATION DU PROJET POUR L'APPLICATION DE BUREAU

Une fois l'étape précédente finie, lancer l'IDE Visual studio.

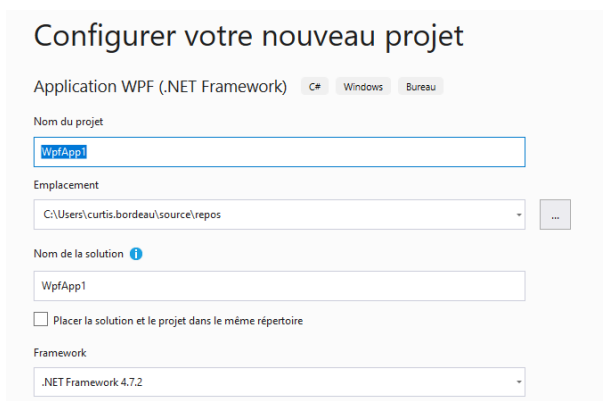


Et faire « Créer un projet »

Plusieurs paramètres sont demandés pour la création d'un projet. Il faut choisir le langage : ce sera le C# pour l'application de bureau « C# », et la plateforme (en vert) : on prend « Windows » (en vert également)



Et pour finir quel type d'application, il faut prendre : « l'application WPF (Windows Presentation Foundation) (.Net Framework) » (en rouge)



Ensuite on choisira le nom du projet et l'emplacement de sauvegarde. Je nomme mon projet : « Application SFL1 ».

3.2 DEFINITION ET INSTALLATION DES BIBLIOTHEQUES NECESSAIRES POUR L'APPLICATION

Nom de la bibliothèque	Description
<i>System</i>	Contient des classes fondamentales et des classes de base qui définissent des types de données de valeur et de référence, des événements et des gestionnaires d'événements, des interfaces, des attributs et des exceptions de traitement couramment utilisés.
System.Windows	Fournit plusieurs classes importantes d'éléments de base WPF (Windows Presentation Foundation), diverses classes prenant en charge la logique d'événement et le système de propriétés WPF.
<i>System.Text</i>	Contient des classes représentant des encodages de caractères ASCII et Unicode, des classes de base abstraites pour la conversion de blocs de caractères vers et à partir des blocs d'octets, et une classe d'assistance qui manipule et met en forme les objets String sans créer d'instances intermédiaires de String.
System.Net	Constitue une interface de programmation simple pour un grand nombre des protocoles réseau employés aujourd'hui.
System.Net.Sockets	Fournit une implémentation gérée de l'interface Windows Sockets (Winsock) pour les développeurs qui doivent contrôler étroitement l'accès au réseau.
System.IO	Fournit des méthodes statiques pour créer, copier supprimer, déplacer et ouvrir un fichier unique, et facilite la création d'objets FileStream.
System.Windows.Controls	Fournit des classes pour créer des éléments, appelés contrôles, qui permettent à un utilisateur d'interagir avec une application.
System.Windows.Input	Fournit des types pour prendre en charge le système d'entrée WPF (Windows Presentation Foundation). Cela inclut des classes d'abstraction de périphérique pour souris, clavier et périphériques de stylet, une classe de gestionnaire d'entrée commune, prise en charge des commandes personnalisées et de leur exécution, et diverses classes d'utilitaires.
MySQL.Data.MySqlClient	Pour la communication SQL

4. DEVELOPPEMENT DE L'APPLICATION

Pour l'application de bureau (WPF) on aura besoin de plusieurs modes comme définis plus haut :

- Le mode instantané

Qui permet de commander la soufflerie en direct. Pour être précis l'application envoie la commande au module de commande, et le module de commande formate les données et envoie la commande à la soufflerie

- Le mode création

Permet de créer un scénario, autrement dit on pourra définir le nom et les périodes du scénario enfin de l'enregistrer dans la base de données (BDD)

- Le mode charger un scénario

Permet de charger un scénario, ce mode va charger les scénarios depuis la BDD

Voilà ce qui est convenu avec le cahier de recettes mais au fil du temps nous avons eu quelques changements pour le mode création et charger scénario.

PETITE PARENTHÈSE POUR LES CHANGEMENTS APPORTÉS DURANT LE PROJET :

- Le mode création

Il permet de créer un scénario, on pourra définir le nom du scénario et l'enregistrer dans la base de données

- Le mode charger scénario

Permet de charger les scénarios depuis la base de données. Cette interface permet de charger les scénarios, supprimer les scénarios, et d'éditer les scénarios.

- Le mode éditer scénario

Ce mode permet d'ajouter ou de supprimer des périodes, mais aussi de modifier le nom du scénario.

Ce changement permet d'avoir un mode « modifier le scénario », sans pour autant rajouter ce mode.

En effet l'utilisateur ne définit que le nom durant la création du scénario, le scénario n'aura donc aucune période de base. Il sera obligé d'aller dans le mode charger scénario pour pouvoir l'éditer enfin d'y ajouter les périodes désirées, et il pourra donc modifier le nom du scénario à volonté tout en ajoutant ou en supprimant des périodes.

2.1 XAML

XAML est un langage basé sur XML qui suit ou s'étend sur les règles de structure XML. Une partie de la terminologie est partagée à partir de ou basée sur la terminologie couramment utilisée pour décrire le langage XML ou le modèle d'objet de document XML.

J'ai commencé par faire une page nommée structure, pour avoir une base graphique de mon application (cela sécurise une sauvegarde). Pour commencer j'ai fait un « Grid » qui prend toute la fenêtre, cela me permettra de modifier le contenu de ma fenêtre dans sa globalité.

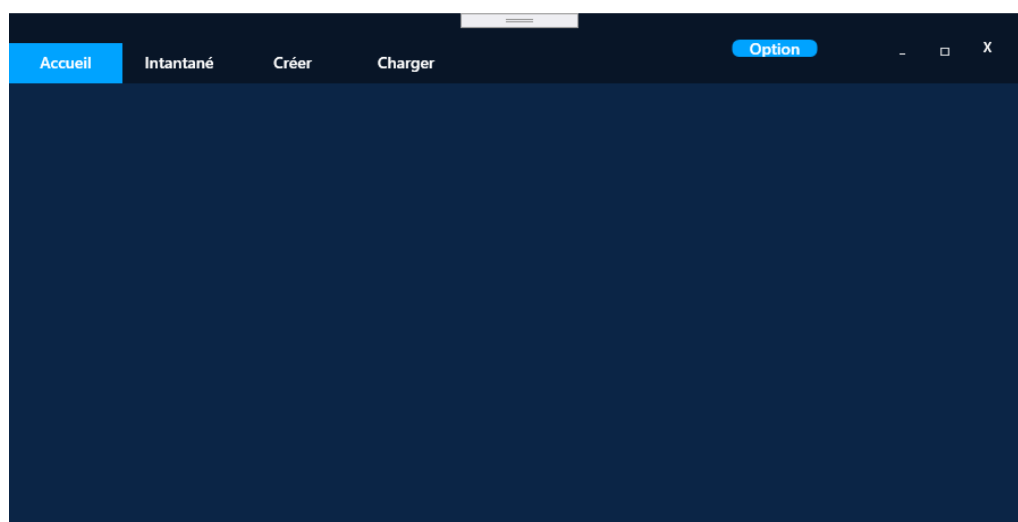
Dans ce premier « Grid » je vais définir plusieurs autres « Grid » qui me permettra de segmenter ma fenêtre en plusieurs morceaux. Et dans ces « Grid » je vais définir un « Grid.ColumnDefinitions » qui me permet de segmenter en plusieurs colonnes mes « Grid » pour avoir une meilleure précision sur le placement de mon contenu (bouton, navigation, contenu...)

2.2 C#

Avant d'entrer dans le vif du sujet j'aimerais expliquer ma façon de fonctionner pour le développement de l'application :

Pour le développement de l'application j'ai fait deux projets que je développe en amont. Le premier projet sera le fonctionnel de l'application et le deuxième sera la partie design de l'application. Cette idée m'est venue durant le développement à cause des soucis que j'ai pu rencontrer (le bouton mal positionné, ou pas la bonne forme, mal arrondi, etc.). J'ai d'abord réalisé un design de base qu'on retrouvera sur chaque fenêtre de l'application : couleurs, forme des boutons, barre de navigation.

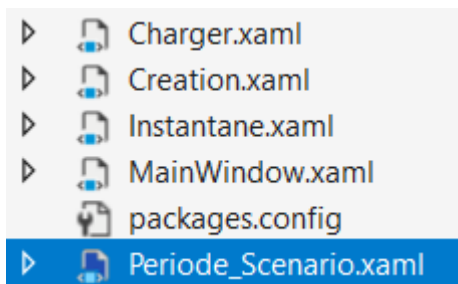
Je ne vais pas m'attarder sur le design de l'application ce n'est pas le but. Cependant je vais montrer le design qui me permettra d'avoir une base sur chaque interface de l'application :



Accueil est en bleu clair quand la souris passe dessus.

2.3 LES FONCTIONNALITES DE BASE (BARRE DE NAVIGATION)

Maintenant que j'ai le design de l'application je vais pouvoir faire la barre de navigation qui sera commune sur chaque interface. Je commence par créer le nombre d'interfaces nécessaires pour l'application.



J'ai créé au total 5 interfaces, (Période_Scenario fait office du mode « éditer le scénario »). Pour pouvoir faire la navigation entre ces interfaces, sinon c'est impossible.

PETITE PARENTHÈSE SUR LA BARRE DE NAVIGATION :

Chaque mot qu'on peut apercevoir ci-dessous (**barre de navigation**) est un bouton. Il faut définir une méthode pour chaque bouton, sinon aucun bouton ne fonctionnera.



POUR CE FAIRE VOICI LES METHODES DE CHAQUE BOUTON :

LA METHODE BTN_ACCUEIL :

```
1 référence
private void Btn_Accueil(object sender, RoutedEventArgs e)
{
    MainWindow omainWindow = new MainWindow();
    omainWindow.Show();
    this.Close();
}
```

- Cette méthode permet la « navigation » vers l'interface de l'accueil, on crée un objet de la fenêtre souhaité et on montre la fenêtre voulue avec la méthode « .Show() » et on ferme la fenêtre actuelle avec un this.Close().

LA METHODE BTN_INSTANTANE :

```
1 référence
private void Btn_Instantane(object sender, RoutedEventArgs e)
{
    Instantane oinstantane = new Instantane();
    oinstantane.Show();
    this.Close();
}
```

- La méthode « Btn_Instantane » permet d'aller sur l'interface du mode instantané

LA METHODE BTN_CREATION :

```
private void Btn_Creation(object sender, RoutedEventArgs e)
{
    Creation ocreation = new Creation();
    ocreation.Show();
    this.Close();
}
```

- Cette méthode permet d'aller vers la fenêtre du mode création

LA METHODE BTN_CHARGER :

```
1 référence
private void Btn_Charger(object sender, RoutedEventArgs e)
{
    Charger ocharger = new Charger();
    ocharger.Show();
    this.Close();
}
```

- Cette méthode permet d'aller vers l'interface du mode chargement

LA METHODE POUR FERMER LA PAGE

```
1 référence
private void Btn_Close(object sender, RoutedEventArgs e)
{
    Close(); // Ferme la fenetre
}
```

- L'action sur le bouton « Btn_Close » fermera l'application.

LA METHODE POUR AGRANDIR OU RETRECIR LA FENETRE

```
private void Btn_Agrandir(object sender, RoutedEventArgs e)
{
    if (WindowState != WindowState.Normal)
    {
        this.WindowState = WindowState.Normal;
    }
    else
    {
        this.WindowState = WindowState.Maximized;
    }
}
```

- Cette méthode permet d'agrandir ou rétrécir la fenêtre. Une méthode un peu plus complexe avec une condition : si la fenêtre est différente du « state normal » (fenêtre flottante) alors on la rend au state flottante, sinon on agrandit la fenêtre.

LA METHODE POUR REDUIRE LA FENETRE

```
1 référence
private void Btn_Reduire (object sender, RoutedEventArgs e)
{
    this.WindowState = WindowState.Minimized; // On agrandi la fenetre
}
```

LA METHODE QUI PERMET DE DEPLACER LA FENETRE

```
1 référence
private void Grid_MouseDown(object sender, MouseButtonEventArgs e)
{
    if (e.LeftButton == MouseButtonState.Pressed) // je fais une con
    {
        DragMove();
    }
}
```

- Cette méthode permet de bouger la fenêtre si le clic gauche de la souris est enfoncé et qu'on bouge la souris en même temps.

Maintenant que la navigation est faite et que toutes les pages ont leur « squelettes », je vais faire interface par interface.

2.4 PREMIERE INTERFACE L'ACCUEIL DE L'APPLICATION

Voici la première fenêtre de mon application. Cette interface apporte une introduction pour l'application afin de guider l'utilisateur de l'application. Un texte explique en quelques lignes l'application.

Ce qui peut être intéressant c'est les deux boutons : Commencer et Créer.



Le bouton Commencer reprend la méthode : « Btn_Instantane » que le bouton « Instantané » ; et le mode Créer reprend la même méthode que le bouton « Créer » dans la barre de navigation.

Pour définir quelle action fera le bouton au moment du clic, cela se passe dans le XAML

```
<Button Content="Commencer"
        Foreground="White" FontSize="14" FontWeight="Medium"
        Grid.Column="1" Grid.ColumnSpan="22"
        VerticalAlignment="Center" HorizontalAlignment="Right"
        Click="Btn_Instantane">
```

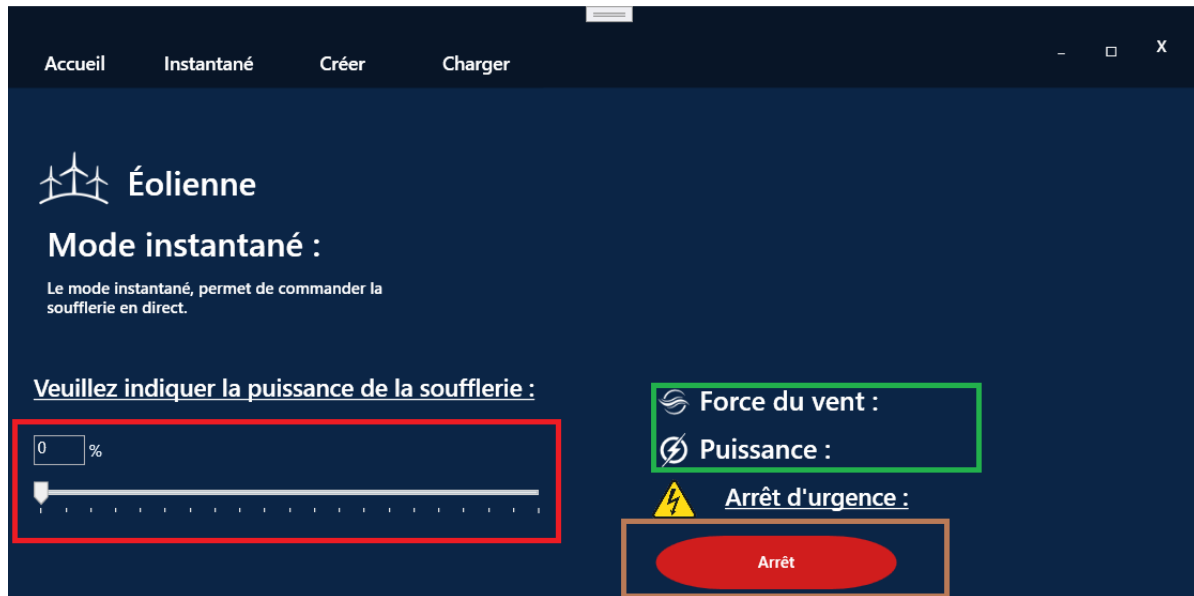
Dans le code XAML j'écris la commande suivante : « Click= "Btn_Instantane " ». Cela permet d'appeler la méthode au moment du clic sur le bouton. La méthode est visible dans la fonctionnalité de base.

A SAVOIR :

Une méthode peut être utiliser **plusieurs fois**. Comme dans cette interface la **méthode** « Btn_Instantane » est utiliser **à la fois** sur le bouton « commencer » **et sur** le bouton « Instantané »

2.5 DEUXIEME FENETRE LE MODE INSTANTANE

Cette deuxième fenêtre permet de commander la soufflerie à distance en mode instantané. On va voir comment la communication est faite et comprendre comment l'application arrive à avoir les données de la force du vent et de la puissance de l'éolienne.



Cette interface a une particularité, la particularité se situe sur le **cadre rouge**, au niveau des interactions entre le **slider** et le **TextBox**. Cette zone permet d'augmenter la puissance de soufflerie instantanément, si nous glissons le **slider** nous remarquons d'abord que le **textbox** indiquera la puissance du **slider**, ceci fonctionne avec un binding.

Définition du « Binding » : Windows Presentation Foundation liaison de données (WPF) fournit un moyen simple et cohérent pour les applications de présenter et d'interagir avec les données. La liaison de données permet de synchroniser les valeurs des propriétés de deux objets différents. Autrement dit un binding (qui est un terme anglais désignant l'action de lier des éléments entre eux)

Nous avons deux éléments reliés par un binding : un **TextBox** et un **slider**.



INFORMATIONS IMPORTANTES :

Avant de commencer l'explication voici les informations nécessaires pour la bonne compréhension :

- Le slider se nomme `slValue`
- Le textbox se nomme `valeur_slider`

Dans le code XAML, j'ai donc introduit un binding dans la balise du TextBox, qui permet de relier le TextBox à mon slider. Voici le code :

```
<TextBox Text="{Binding ElementName=slValue, Path=Value, UpdateSourceTrigger=PropertyChanged}"
    TextAlignment="Left"
    Foreground="White"
    Height="23"
    Background="#0B2546"
    TextWrapping="Wrap"
    Name="valeur_slider"
    Grid.Column="1" Grid.ColumnSpan="2" >
```

« ElementName=slValue » c'est le nom de mon slider (à qui on le relie) puis on met à jour les données avec « updateSourceTrigger=PropertyChanged ». (Les données se mettent à jour quand la valeur change) et « Path=Value » on accepte la valeur double.

CONCLUSION :

Maintenant que les deux boutons ont une interaction directe entre eux, nous devons envoyer la commande au module de commande. La communication se fera à chaque changement de valeur du slider.

La communication sera en TCP socket, un moyen de communication à distance avec un serveur et un ou plusieurs clients.

COMMUNICATION ENTRE LE CAPTEUR DE COMMANDE ET L'APPLICATION

La communication fonctionne avec un TCP Socket. Pour bien comprendre nous avons donc l'application qui est client et le capteur de commande qui est serveur, le capteur de commande permet de traiter la commande envoyée par l'application jusqu'à la soufflerie.

Ma méthode pour la communication se nomme `EnvoiTcpClient()` :

```
1 référence
public void EnvoiTcpClient()
{
    string message = slValue.Value.ToString(); // message contiendra l'information du TextBox et en plus on choisi seulement d'env
    TcpClient oclient = new TcpClient();
    // Création de l'objet client
    try
    {
        oclient.Connect("127.0.0.1", 23); // Connexion NE PAS OUBLIER DE GERER LES ERREURS

        Byte[] data = System.Text.Encoding.ASCII.GetBytes(message); // conversion en ASCII

        NetworkStream stream = oclient.GetStream();

        stream.Write(data, 0, data.Length);

        stream.Close();
        oclient.Close();
    }
    catch
    {
        MessageBox.Show("La connection n'a pas été établie", string.Empty, MessageBoxButton.OK, MessageBoxImage.Exclamation);
    }
}
```

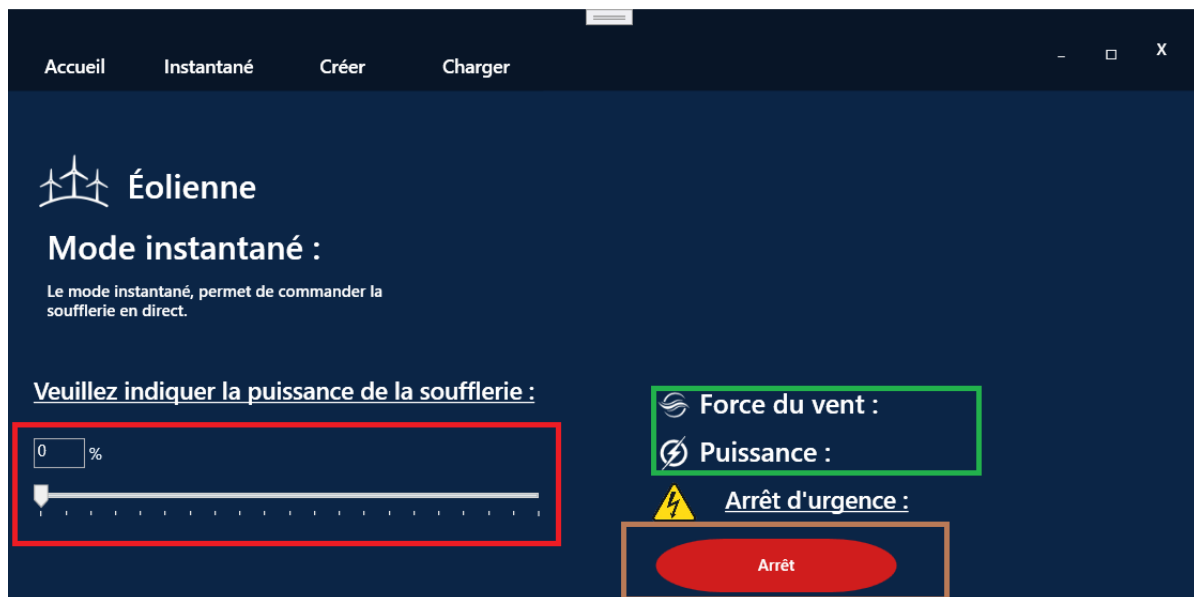
La première ligne de cette méthode permet d'envoyer le message de la valeur du slider. C'est une méthode fournie par Microsoft, je n'ai fait que l'adapter à mon projet. Un objet de `TcpClient` est créé. Sur la suite les erreurs sont gérées avec un « try » puis un « catch » dans le « try » la première ligne permet de se connecter via l'objet de `TcpClient` avec les paramètres nécessaires et on converti le message en ASCII pour l'envoi du message, puis on ferme le client. En cas d'erreur nous avons un catch qui permet d'afficher un message d'erreur ; cela évite que l'application plante.

```
1 reference
private void Slider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e) // On créer un évènement lors du changement
{
    EnvoiTcpClient();
    // l'évènement est l'appel de la fonction EnvoiTcpClient
}
```

Maintenant que la communication est possible via la méthode « `EnvoiTcpClient` », nous devons l'appeler au moment du changement de valeur du slider. Pour ce faire, j'ai créé une méthode « `Slider_ValueChanged` » qui sera appelée lors du changement de valeur, dans cette méthode j'appelle la méthode « `envoiTcpClient` ». L'intérêt de faire ceci est que pendant le changement de valeur du slider ce n'est pas la seule chose à faire, il faut aussi traiter les données.

TEST UNITAIRE :

Pour voir les tests unitaires de cette partie voir : [Ici](#)



Maintenant nous allons voir la partie **verte**. Cette partie contient deux labels non visibles, ils se situent à droite du carré **vert**, cela permettra d'afficher la valeur dans ces labels.

Cette partie est un peu plus complexe car deux méthodes sont nécessaires : une méthode qui permet de recevoir les informations nécessaires et une méthode pour traiter les données en Json.

LA METHODE QUI PERMET DE RECEVOIR LES DONNEES

Avant de pouvoir traiter les données nous devons avoir accès à ces données, une méthode pour recevoir ces informations est donc nécessaire.

```
public void Receive_Acquisition()
{
    {
        IPEndPoint ip = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 23);
        Socket oclient = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

        try
        {
            oclient.Connect(ip);
        }
        catch
        {
            MessageBox.Show("La connection n'a pas était établie", string.Empty, MessageBoxButton.OK, MessageBoxImage.Exclamation);
        }

        byte[] data = new byte[1024];
        int receivedDataLength = oclient.Receive(data);
        string stringData = Encoding.ASCII.GetString(data, 0, receivedDataLength);
        Console.WriteLine(stringData);

        oclient.Shutdown(SocketShutdown.Both);
        oclient.Close();
    }
}
```

Cette méthode est fournie par Microsoft, je l'ai simplement adaptée à mes besoins (adresse ip, nom).

LA METHODE QUI TRAITE LES DONNEES

Les données reçues sont sous le format de Json.

Json : JSon (JavaScript Objet Notation) est un langage léger d'échange de données textuelles. Pour les ordinateurs, ce format se génère et s'analyse facilement. Pour les humains, il est pratique à écrire et à lire grâce à une syntaxe simple et à une structure en arborescence.

Il faut traiter deux types de données : la force du vent et la puissance.

Sur l'interface du mode instantané les deux labels qui ne sont pas visibles dans le cadre vert se nomment :

- *Vent pour la force du vent*
- *Puissance pour la puissance*

Le nom des labels nous sera utile pour mettre les informations dans le label correspondant.

Le JSON a lieu dans la méthode du Slider_ValueChanged

```
private void Slider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e) // On créer un évènement lors du changement
{
    EnvoiTcpClient();
    Receive_Acquisition();
    // l'évènement est l'appel de la fonction EnvoiTcpClient
    try
    // Le try il essaye de faire ce qui est demandé sinon il va dans le catch
    {
        StreamReader oSR = new StreamReader("C:/Users/curtis.bordeau/capteurs.JSON");
        // @ évite de écrire le /
        CapteurAcquisition oCapteurAcquisition = CapteurAcquisition.ToDeserializeCapteurAcquisition(oSR.ReadToEnd());
        Vent.Content = oCapteurAcquisition.force_vent;
        Puissance.Content = oCapteurAcquisition.puissance;
        oSR.Close();
    }
    catch
    // Si n'a pas réussi un message apparaîtra pour signaler l'erreur
    {
        MessageBox.Show("Le fichier Json n'a pas pu être récupéré", string.Empty, MessageBoxButton.OK, MessageBoxImage.Exclamation);
    }
}
```

« **StreamReader** » permet de lire le fichier Json, et entre parenthèses il faut indiquer le chemin du fichier et son nom, car la manipulation est locale pour le moment.

« **CapteurAcquisition** » est une classe que nous verrons juste après, cette ligne permet de dématérialiser le fichier.

Pour finir nous affichons le résultat dans les labels créés au préalable en choisissant les noms correspondant au label.

Exemple : Vent.Content

Puissance.Content

LA CLASSE « CAPTEURACQUISITION »

```
using System.Web.Script.Serialization;
using System.Windows;

namespace WPF_Eolienne
{
    6 références
    public class CapteurAcquisition
    {
        2 références
        public float force_vent { get; set; }

        2 références
        public float puissance { get; set; }

        0 références
        public CapteurAcquisition()
        {
            force_vent = 0;
            puissance = 0;
        }

        1 référence
        public static CapteurAcquisition ToDeserializeCapteurAcquisition(string sCapteurAcquisitionSerialized)
        {
            JavaScriptSerializer ser = new JavaScriptSerializer();
            CapteurAcquisition oCapteurAcquisition = null;

            try
            {
                oCapteurAcquisition = ser.Deserialize<CapteurAcquisition>(sCapteurAcquisitionSerialized);
            }
            catch
            {
                MessageBox.Show("Echec de désérialization du capteur d'acquisition en json.", string.Empty, MessageBoxButton.OK, MessageBoxImage.Exclamation);
            }

            return oCapteurAcquisition;
        }
    }
}
```

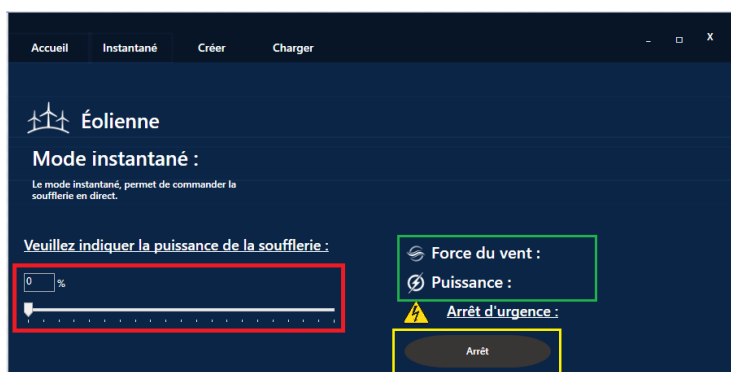
La classe CapteurAcquisition va récupérer les données du fichier, chaque valeur est définie par un nom : force_vent et puissance. Voici un fichier Json pour mieux comprendre :

```
{
  "force_vent": "52.42",
  "puissance": "12.54"
}
```

On remarque que chaque valeur est définie par un nom, qui me permet de différencier les valeurs et de les récupérer.

LE BOUTON ARRÊT D'URGENCE

Pour la dernière méthode j'ai créé un petit bouton qui met le slider à la valeur zéro



Dans le **cadrant jaune**, pour ce faire voici la méthode :

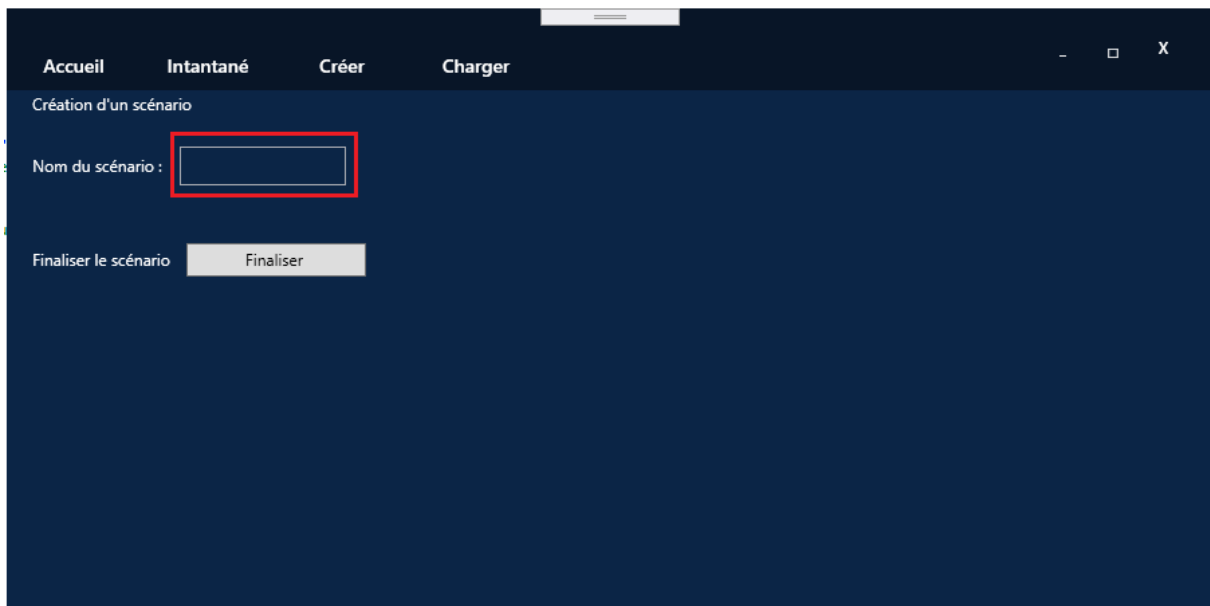
```
private void Button_Arrêt(object sender, RoutedEventArgs e)
{
    valeur_slider.Text = "0";
}
```

Cette méthode remet le slider à la valeur zéro, cette action se déclenche quand on clique sur le bouton « Arrêt ».

TEST UNITAIRE :

Pour voir les tests unitaires de cette partie voir : [ici](#)

2.6 L'INTERFACE DU MODE CREATION



Cette fenêtre permet de créer un scénario, il y a juste un paramètre à définir : le nom du scénario. Pour ce faire j'ai défini un « **TextBox** ».

Ce « **TextBox** » se nomme : **txtScenario**.

La création d'un scénario s'enregistre dans une base de données (BDD), et donc nécessite une connexion à cette BDD, la connexion se fera dans le constructeur par défaut

CONNEXION VERS LA BASE DE DONNEES

```
public partial class Creation : Window
{
    private MySqlConnection conn;

    6 références
    public Creation()
    {
        InitializeComponent();

        string myConnectionString = "server=127.0.0.1;"
                                   + "uid=root;"
                                   + "pwd=";
                                   + "database=eoliennedb;"
                                   + "Charset=latin1;";

        conn = new MySqlConnection(myConnectionString);
    }
}
```

On définit une variable « **MySqlConnection conn** » qui sera utilisée pour reprendre la connexion de la BDD. Puis dans le constructeur par défaut s'ajoute la connexion, il faut définir l'adresse du serveur, le login, le mot de passe et le nom de la BDD.

Après que la connexion fonctionne, une méthode qui rajoute le scénario dans la base de données est nécessaire, pour ce faire :

METHODE QUI INSERE LES DONNEES DANS LA BDD

```
1 référence
private void ajout_Scenario(object sender, RoutedEventArgs e)
{
    string nom = String.Format(txtScenario.Text);
    // DateTime date_creation = DateTime.Parse(txtDate.Text);

    string sql = "INSERT INTO `scenario` ( nom, date_creation ) VALUES ( '" + nom + "', Now());";

    conn.Open();
    MySqlCommand cmd = new MySqlCommand(sql, conn);
    cmd.ExecuteNonQuery();
    conn.Close();

    Creation creation = new Creation();
    creation.Show();
    this.Close();
}
```

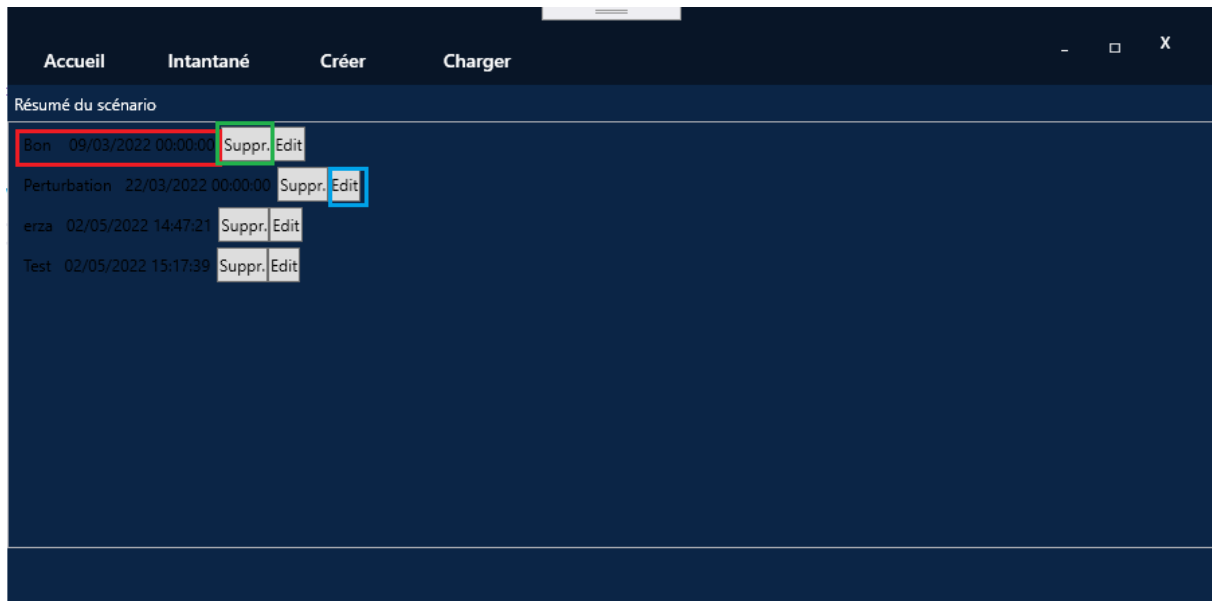
On formate le nom au bon format. Puis on fait la requête SQL nécessaire pour l'enregistrement dans la base de données. Et une fois que la requête SQL est faite la page se relance. Cela permet à l'utilisateur de comprendre que le scénario a bien été enregistré ; sur le futur je compte faire un apparaitre un message pop.

La méthode s'exécute une fois qu'on clique sur le bouton Finaliser.

TEST UNITAIRE :

Pour voir les tests unitaires de cette partie voir : [ici](#)

2.7 LA FENETRE CHARGER



L'interface du mode Charger permet de voir la liste des scénarios **avec le nom et la date de création du scénario**. A cela s'ajoute deux boutons : un bouton pour supprimer le scénario « Suppr » (supprimer) et un bouton « Edit » (éditer) qui permet de modifier le scénario (nom, ajout de période ou supprimer). Nous avons donc deux possibilités sur cette fenêtre :

- Supprimer le scénario
- Editer le scénario

INFORMATION IMPORTANTE :

La liste des scénarios s'affiche dans une ListBox qui se nomme : « listScenario »

L'interface Charger a besoin aussi de se connecter à la base de données ; c'est exactement comme la fenêtre création, fait dans le constructeur par défaut. On ne s'attardera pas dessus.

Voyons d'abord le **cadre rouge**. Dans ce **cadre** nous avons une méthode qui permet d'aller chercher les données depuis la BDD et une classe qui s'occupe de la ListBox.

LA METHODE QUI VA CHERCHER LES DONNEES DANS LA BDD

```
private void MAJListePhases()
{
    string sql = "SELECT * FROM scenario ";

    conn.Open();

    MySqlCommand cmd = new MySqlCommand(sql, conn);
    MySqlDataReader rdr = cmd.ExecuteReader();

    listScenario.Items.Clear();

    if (rdr.HasRows)
    {
        while (rdr.Read())
        {
            int idScenario = Int32.Parse(rdr["id"].ToString());
            string nom = rdr["nom"].ToString();
            string date = rdr["date_creation"].ToString();

            listScenario.Items.Add(new ListBoxItemScenario(idScenario, nom, date, this));
        }
    }
    conn.Close();
}
```

Dans le cadre orange il y a la requête SQL : on prend **toutes les informations de la table « scenario »**. Puis dans le cadre vert on **transforme au type voulu « string »**.

LA CLASSE « LISTBOXITEMSCENARIO »

```
public class ListBoxItemScenario : ListBoxItem
{
    private Charger charger;
    private StackPanel sp;
    private Label txtnom;
    private Label txtdate;
    private Button btnSuppr;
    private Button btnEdit;
    private int idScenario;
    private string nomScenario;

    public ListBoxItemScenario(int idScenario, string name, string date, Charger charger)
    {
        this.charger = charger;
        this.idScenario = idScenario;
        this.nomScenario = name;

        sp = new StackPanel();
        sp.Orientation = Orientation.Horizontal;
        txtnom = new Label();
        txtdate = new Label();
        btnSuppr = new Button();
        btnEdit = new Button();

        btnSuppr.Click += BtnSupprimer_Click;
        btnEdit.Click += BtnEdit_Click;

        txtnom.Content = name;
        txtdate.Content = date;
        btnSuppr.Content = "Suppr.";
        btnEdit.Content = "Edit";

        sp.Children.Add(txtnom);
        sp.Children.Add(txtdate);
        sp.Children.Add(btnSuppr);
        sp.Children.Add(btnEdit);

        this.AddChild(sp);
    }
}
```

Faire cette classe qui hérite de « `ListBoxItem` » permet de personnaliser les items à l'intérieur de la `ListBox` au lieu de n'avoir que du texte. Autrement dit « `ListBoxItem` » représente un élément sélectionnable dans la « `ListBox` »

Nous avons une classe qui se nomme « `ListBoxItemScenario` » qui hérite de « `ListBoxItem` ».

Sur les premières lignes de cette classe des attributs sont déclarés.

On crée un `Stack panel` et l'orientation dans la « `ListBox` ». On crée de même deux labels et deux boutons.

Les labels permettront d'afficher le nom du scénario et la date de création du scénario, les deux boutons seront utiles pour faire l'interaction « supprimer le scénario » et « modifier le scénario »

Le cadre bleu définit la valeur des attributs. On définit le contenu mis dans les boutons, par exemple : pour le bouton « `btnSuppr` » on dit que l'utilisateur verra afficher « `Suppr.` ». On fait de même.

Pour finir on ajoute les items (label, bouton) dans la `ListBox`.

LA METHODE « BTN_SUPPRIMER_CLICK »

```
private void BtnSupprimer_Click(object sender, RoutedEventArgs e)
{
    MessageBoxResult result = MessageBox.Show("Voulez-vous supprimer cette phase ?", "Supprimer", MessageBoxButton.YesNo, MessageBoxImage.Question);
    if (result == MessageBoxResult.Yes)
    {
        charger.supprimerScENARIO(idScENARIO);
    }
}
```

Cette méthode permet d'afficher un « messageBox » (fenêtre pop) pour confirmer la suppression du scénario, et une condition « si l'utilisateur clique sur oui pour confirmer alors il appelle la méthode « `supprimerScENARIO()` ».

LA METHODE « SUPPRIMERSCENARIO() »

```
public void supprimerScENARIO(int id)
{
    string sql = "DELETE FROM `scenario` WHERE `id` = " + id;

    conn.Open();
    MySqlCommand cmd = new MySqlCommand(sql, conn);
    cmd.ExecuteNonQuery();
    conn.Close();

    MAJListePhases();
}
```

Cette méthode permet de faire une requête SQL pour supprimer le scénario sur base de données à l'ID correspondant.

LA METHODE « BTNEDIT_CLICK »

```
private void BtnEdit_Click(object sender, RoutedEventArgs e)
{
    Periode_Scenario operiode_scenario = new Periode_Scenario(idScENARIO, nomScENARIO);
    operiode_scenario.Show();
    charger.Close();
}
```

La méthode permet d'aller vers l'interface du mode « edit ».

INFORMATION

Une interface doit être créée pour que l'utilisateur puisse modifier le scénario. Elle se nomme « Période_Scenario ».

Les tests unitaires : [ici](#)

2.8 L'INTERFACE PERIODE_SCENARIO

Accueil Instantané Créer Charger

Liste des période du scénario

Nom du scénario : dza Modifier

Durée : Puissance : Ajouter

Cette fenêtre permet d'ajouter ou de supprimer des périodes au scénario, et de modifier le nom. Cette interface se connecte à la BDD au même titre que les modes Charger, Création. Nous avons trois méthodes et une classe :

- La méthode pour mettre à jour le nom du scénario
- La méthode qui permet d'ajouter des scénarios
- Une méthode qui sélectionne les périodes du scénario correspondant.
- La classe « ListBoxIteamPeriode »

La méthode pour supprimer le scénario est la même que l'interface « charger ».

INFORMATION IMPORTANTE :

Le label pour le nom du scénario se nomme : txtNomScenario

Le nom du label pour la durée se nomme : txtDuree

Et enfin le nom du label pour la puissance se nomme : txtPuissance

LA METHODE POUR METTRE A JOUR LE NOM DU SCENARIO

```
private void btnModifier_Click(object sender, RoutedEventArgs e)
{
    nomScenario = txtNomScenario.Text;
    string sql = $"UPDATE scenario SET nom = '{nomScenario}' WHERE id = {idScenario}";

    conn.Open();
    MySqlCommand cmd = new MySqlCommand(sql, conn);
    cmd.ExecuteNonQuery();
    conn.Close();
}
```

La première ligne le « **nomScenario** » prend la valeur « **txtNomScenario** » (le label Créer).
Puis on fait une requête sql pour **mettre à jour le nom du scénario dans la BDD**.

LA METHODE QUI AJOUTE DES PERIODES

```
private void BtnAjouter_Click(object sender, RoutedEventArgs e)
{
    int duree = Int32.Parse(txtDuree.Text);
    int puissance = Int32.Parse(txtPuissance.Text);

    string sql = $"INSERT INTO periode (duree, puissance_soufflerie, scenario_id) VALUES ({duree}, {puissance}, {idScenario})";

    conn.Open();
    MySqlCommand cmd = new MySqlCommand(sql, conn);
    cmd.ExecuteNonQuery();
    conn.Close();

    SetListePeriode();
}
```

Cette méthode fait une requête SQL pour introduire les nouvelles périodes dans la BDD et appelle la méthode « SetListePeriode », l'appel permet de rafraichir la liste des périodes.

METHODE SETLISTPERIODE

```
private void SetListePeriode()  
{  
    string sql = $"SELECT * FROM periode where scenario_id = {idScenario}";  
  
    conn.Open();  
  
    MySqlCommand cmd = new MySqlCommand(sql, conn);  
    MySqlDataReader rdr = cmd.ExecuteReader();  
  
    listPeriodes.Items.Clear();  
  
    if (rdr.HasRows)  
    {  
        while (rdr.Read())  
        {  
            int idPeriode = Int32.Parse(rdr["id"].ToString());  
            string duree = rdr["duree"].ToString();  
            string puissance = rdr["puissance_soufflerie"].ToString();  
  
            listPeriodes.Items.Add(new ListBoxItemPeriode(idPeriode, duree, puissance, this));  
        }  
    }  
    conn.Close();  
}
```

On fait une requête SQL pour récupérer les périodes correspondant au scénario avec L'ID.

LA CLASSE LISTBOXITEMPERIODE

```
public class ListBoxItemPeriode : ListBoxItem
{
    private Periode_Scenario PeriodeScenario;
    private StackPanel sp;
    private Label txtDuree;
    private Label txtPuissance;
    private Button btnSuppr;
    private int idPeriode;

    public ListBoxItemPeriode(int idPeriode, string duree, string puissance, Periode_Scenario Peri
    {
        this.PeriodeScenario = PeriodeScenario;
        this.idPeriode = idPeriode;

        sp = new StackPanel();
        sp.Orientation = Orientation.Horizontal;
        txtDuree = new Label();
        txtPuissance = new Label();
        btnSuppr = new Button();

        btnSuppr.Click += BtnSupprimer_Click;

        txtDuree.Content = duree.ToString() + " secondes";
        txtPuissance.Content = puissance.ToString() + "%";
        btnSuppr.Content = "Suppr.";

        sp.Children.Add(txtDuree);
        sp.Children.Add(txtPuissance);
        sp.Children.Add(btnSuppr);

        this.AddChild(sp);
    }

    private void BtnSupprimer_Click(object sender, RoutedEventArgs e)
    {
        MessageBoxResult result = MessageBox.Show("Voulez-vous supprimer cette période ?", "Suppri

        if (result == MessageBoxResult.Yes)
        {
            PeriodeScenario.supprimerPeriode(idPeriode);
        }
    }
}
```

TEST UNITAIRE :

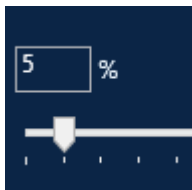
Pour voir les tests unitaires de cette partir voir : [ici](#)

5. TEST UNITAIRE

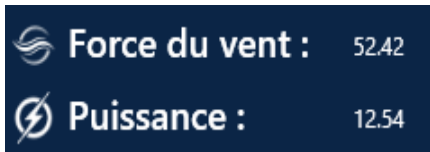
5.1 INTERFACE MODE INSTANTANÉ

Cette partie concerne le mode instantané, nous allons voir si les méthodes fonctionnent correctement :

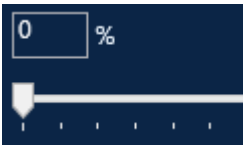
COMMUNICATION ENTRE LE CAPTEUR DE COMMANDE ET L'APPLICATION

Elément testé :	Communication entre l'application et le capteur de commande		
Objectif du test :	Envoyer la valeur indiquée sur le « slider »		
Nom du testeur :	Curtis Bordeaux	Date :	04/05/2022
Procédure du test			
Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation
On glisse le slider d'un trait, qui donne la valeur 5	Le textBox reprend la valeur		Oui
La valeur est envoyée au serveur Socket	La valeur du slider est reçue sur le socket et la communication a eu lieu	<pre>> New Client: 127.0.0.1 5> Client closed conection.</pre>	Oui

TRAITEMENT DES DONNEES EN JSON

Élément testé :	Traitement des données en JSON		
Objectif du test :	Traiter les données sous le format Json en local		
Nom du testeur :	Curtis Bordeaux	Date :	04/05/2022
Procédure du test			
Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation
On glisse le slider	Les données apparaissent dans le label correspondant		Oui

LE BOUTON ARRÊT D'URGENCE

Élément testé :	Le bouton « arrêt »		
Objectif du test :	Le slider revient à zéro		
Nom du testeur :	Curtis Bordeaux	Date :	04/05/2022
Procédure du test			
Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation
On glisse le slider	Le slider revient de 5 à zéro.		Oui

5.2 FENETRE DE CREATION

Cette partie est dédiée aux tests unitaires qui concernent le mode Création.

CONNEXION ET INSERER UN SCENARIO A LA BDD

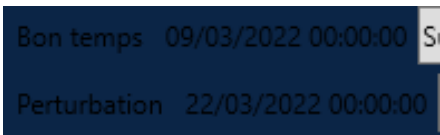
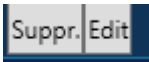
Élément testé :	Connexion et insérer un scénario à la BDD		
Objectif du test :	Enregistrer le scénario dans la BDD		
Nom du testeur :	Curtis Bordeaux	Date :	04/05/2022
Procédure du test			
Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation
Définir le nom « Test » de scénario et valider sur le bouton « finaliser »	Le scénario a été ajouté dans la BDD avec le nom et la date de création	5 Test 2022-05-05 17:08:05	Oui

5.3 INTERFACE CHARGER

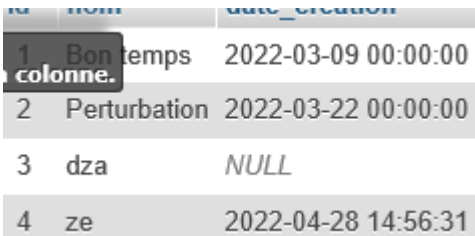
Cette partie est dédiée aux tests unitaires qui concernent le mode Charger.

LA METHODE QUI VA CHERCHER LES DONNEES DANS LA BDD

Élément testé :	La liste des scénarios est chargée avec les items prédéfinis		
Objectif du test :	Les scénarios apparaissent sur la ListBox avec les items		
Nom du testeur :	Curtis Bordeaux	Date :	04/05/2022
Procédure du test			
Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation

Cliquer sur le mode « charger »	Les scénarios apparaissent		Oui
Rien	Les items apparaissent : Supp. Et Edit		Oui

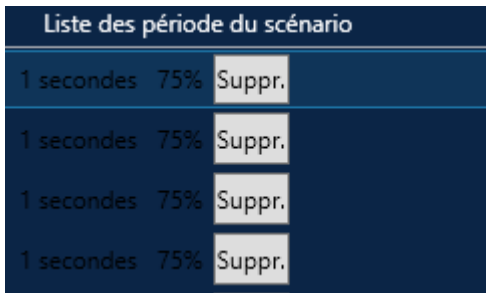
LA METHODE QUI SUPPRIME LE SCENARIO DANS LA BDD

Élément testé :	Le bouton « Suppr »		
Objectif du test :	Le scénario est supprimé sur la BDD		
Nom du testeur :	Curtis Bordeau	Date :	04/05/2022
Procédure du test			
Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation
Cliquer sur le mode « Suppr » et confirmer sur le message pop	Le scénario n'apparaît plus sur la BDD		Oui

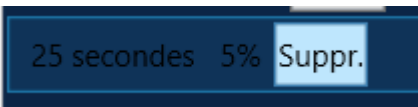
5.4 L'INTERFACE PERIODE_SCENARIO

Cette partie est dédiée aux tests unitaires qui concernent le mode Edit.


L'INTERFACE EDIT

Élément testé :	Apparaître les périodes d'un scénario		
Objectif du test :	Les périodes apparaissent		
Nom du testeur :	Curtis Bordeaux	Date :	04/05/2022
Procédure du test			
Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation
Cliquer sur le bouton « Edit » « Bon Temps »	Les périodes apparaissent		Oui

LA METHODE QUI AJOUTE DES PERIODES

Élément testé :	Ajouter une période		
Objectif du test :	La période est ajoutée		
Nom du testeur :	Curtis Bordeaux	Date :	04/05/2022
Procédure du test			
Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation
Définir une durée : « 25 » et une puissance de « 5 », et cliquer sur ajouter	La période apparait		Oui

LA METHODE QUI SUPPRIME LE SCENARIO DANS LA BDD

Elément testé :	Supprimer une période		
Objectif du test :	La période est supprimée		
Nom du testeur :	Curtis Bordeaux	Date :	04/05/2022
Procédure du test			
Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation
Cliquer sur le bouton supprimer, un message apparait confirmé le résultat	La période est supprimée et n'apparait pas dans la liste		Oui