

Preuves

IHM : Interface Homme-Machine (XAML, WPF)

DOCUMENTATION

Je sais décrire le contexte de mon application pour qu'il soit compréhensible par tout le monde.

Cf DocumentationIHM.pdf

Je sais dessiner des sketches pour concevoir les fenêtres de mon application.

Cf DocumentationIHM.pdf

Je sais enchaîner mes sketches au sein d'un story-board.

Cf DocumentationIHM.pdf

Je sais concevoir un diagramme de cas d'utilisation qui représente les fonctionnalités de mon application.

Cf DocumentationIHM.pdf

Conception et Programmation Orientées Objets (C#, .NET)

DOCUMENTATION

Je sais concevoir un diagramme de classes qui représente mon application.

Cf DocumentationCPOO.pdf

Je sais réaliser un diagramme de paquetages qui illustre bien l'isolation entre les parties de mon application.

Cf DocumentationCPOO.pdf

Je sais décrire mes deux diagrammes en mettant en valeur et en justifiant les éléments essentiels.

Cf DocumentationCPOO.pdf

IHM : Interface Homme-Machine (XAML, WPF)

DOCUMENTATION

Je sais concevoir une application ergonomique.

Cf DocumentationIHM.pdf

Je sais concevoir une application avec une prise en compte de l'accessibilité.

Cf DocumentationIHM.pdf

Projet Tuteuré S2

DOCUMENTATION

Je sais mettre en avant dans mon diagramme de classes la persistance de mon application.

Cf DocumentationProjetTuteuréS2.pdf

Je sais mettre en avant dans mon diagramme de classes ma partie personnelle.

Cf DocumentationProjetTuteuréS2.pdf

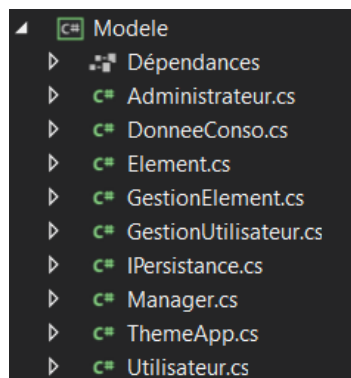
Je sais mettre en avant dans mon diagramme de paquets la persistance de mon application.

Cf DocumentationProjetTuteuréS2.pdf

Conception et Programmation Orientées Objets (C#, .NET)

CODE

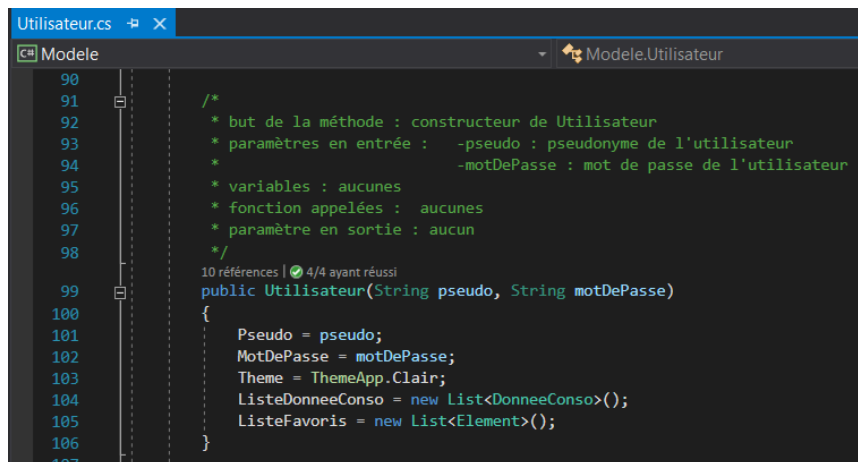
Je maîtrise les bases de la programmation C# (classes, structures, instances...).



Le Modele contient plusieurs classes, enum, interfaces

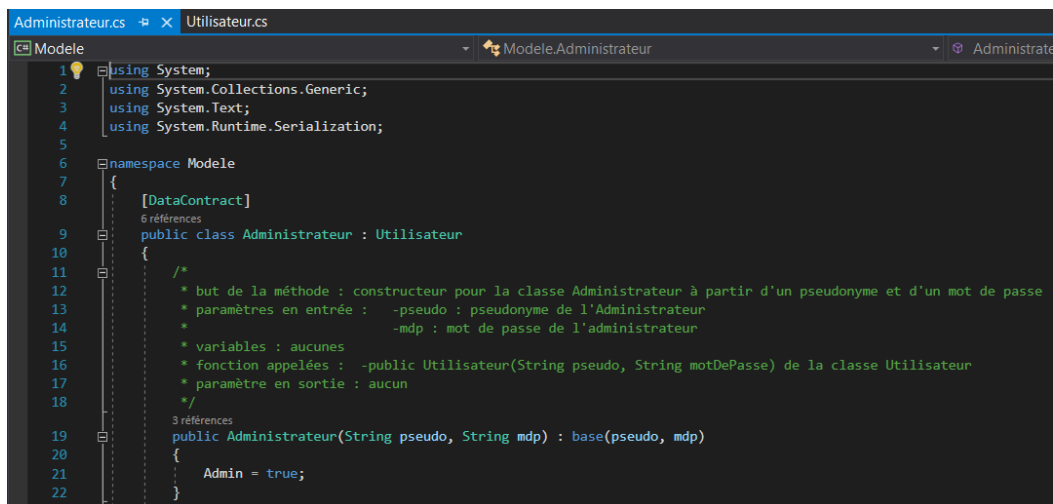
L'ensemble des classes possède au moins un constructeur d'instance.

Exemple :



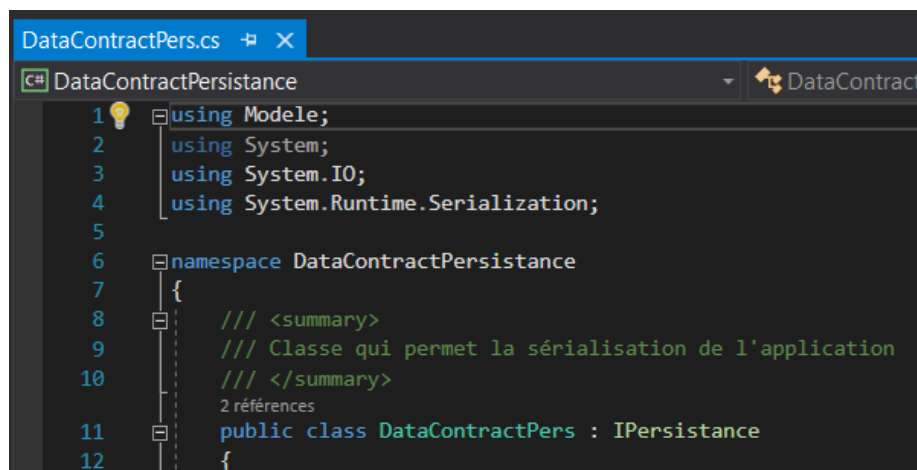
```
90
91
92  /*
93   * but de la méthode : constructeur de Utilisateur
94   * paramètres en entrée : -pseudo : pseudonyme de l'utilisateur
95   *                         -motDePasse : mot de passe de l'utilisateur
96   * variables : aucunes
97   * fonction appelées : aucunes
98   * paramètre en sortie : aucun
99   */
100 public Utilisateur(String pseudo, String motDePasse)
101 {
102     Pseudo = pseudo;
103     MotDePasse = motDePasse;
104     Theme = ThemeApp.Claire;
105     ListeDonneeConso = new List<DonneeConso>();
106     ListeFavoris = new List<Element>();
107 }
```

Je sais utiliser l'abstraction à bon escient (héritage, interfaces, polymorphisme).



```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.Runtime.Serialization;
5
6 namespace Modele
7 {
8     [DataContract]
9     public class Administrateur : Utilisateur
10     {
11         /*
12          * but de la méthode : constructeur pour la classe Administrateur à partir d'un pseudonyme et d'un mot de passe
13          * paramètres en entrée : -pseudo : pseudonyme de l'Administrateur
14          *                         -mdp : mot de passe de l'administrateur
15          * variables : aucunes
16          * fonction appelées : -public Utilisateur(String pseudo, String motDePasse) de la classe Utilisateur
17          * paramètre en sortie : aucun
18          */
19         public Administrateur(String pseudo, String mdp) : base(pseudo, mdp)
20         {
21             Admin = true;
22         }
23     }
24 }
```

Un exemple de classe qui utilise l'héritage est la classe Administrateur qui hérite de Utilisateur.



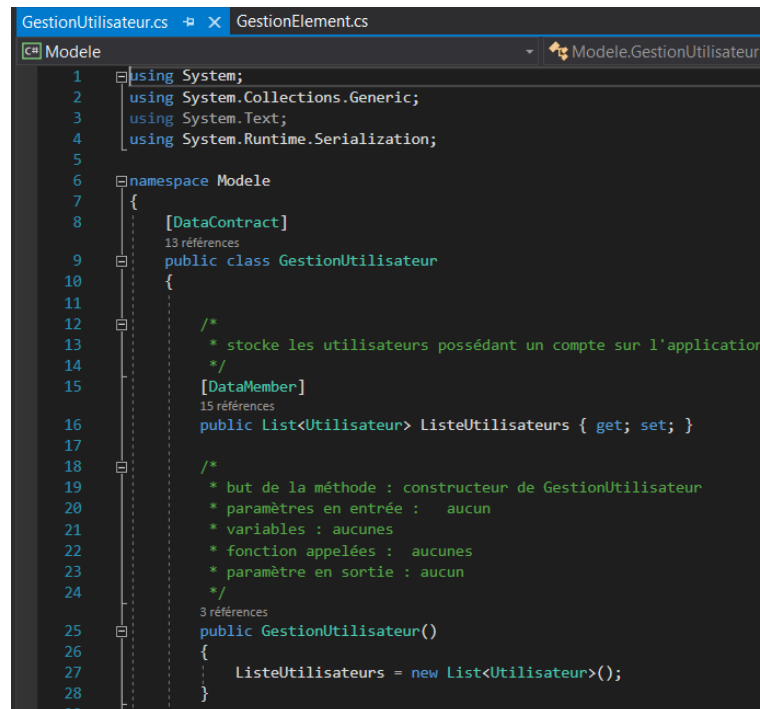
```
1 using Modele;
2 using System;
3 using System.IO;
4 using System.Runtime.Serialization;
5
6 namespace DataContractPersistance
7 {
8     /// <summary>
9     /// Classe qui permet la sérialisation de l'application
10    /// </summary>
11    public class DataContractPers : IPersistance
12    {
13    }
```

La classe DataContractPers implémente par exemple l'interface IPersistance.

Je sais gérer des collections simples (tableaux, listes...).

2 classes utilisent des collections simples : GestionUtilisateur et GestionElement.

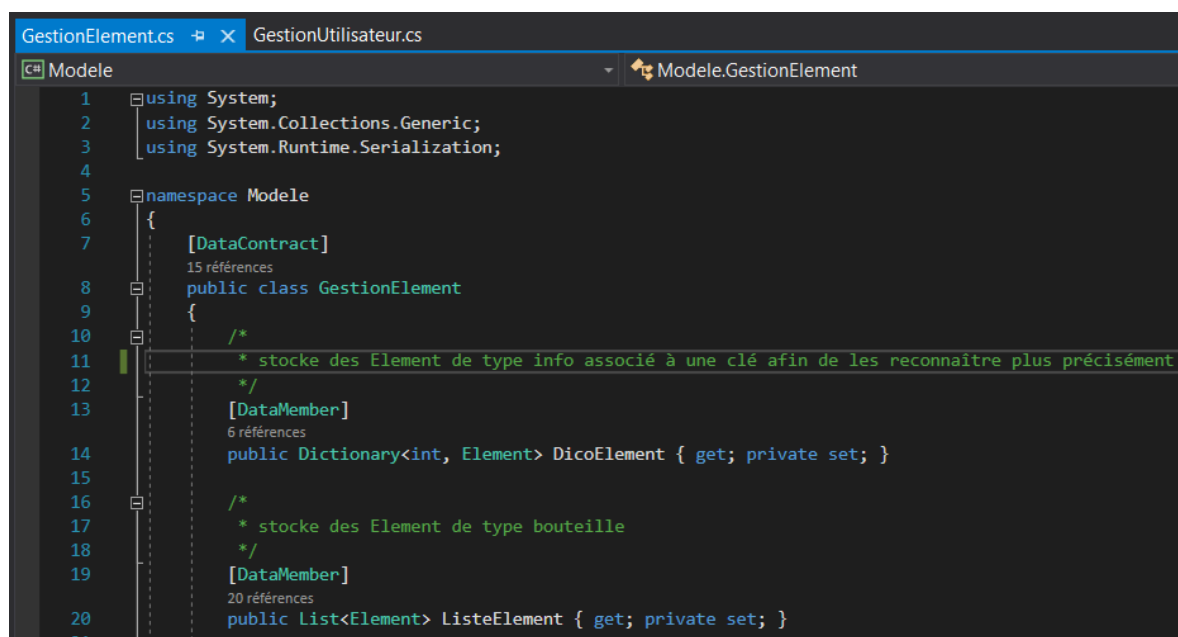
Exemple pour GestionUtilisateur :



```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.Runtime.Serialization;
5
6 namespace Modele
7 {
8     [DataContract]
9     public class GestionUtilisateur
10     {
11         /*
12          * stocke les utilisateurs possédant un compte sur l'application
13          */
14         [DataMember]
15         public List<Utilisateur> ListeUtilisateurs { get; set; }
16
17         /*
18          * but de la méthode : constructeur de GestionUtilisateur
19          * paramètres en entrée : aucun
20          * variables : aucunes
21          * fonction appelées : aucunes
22          * paramètre en sortie : aucun
23          */
24         public GestionUtilisateur()
25         {
26             ListeUtilisateurs = new List<Utilisateur>();
27         }
28     }
29 }
```

Je sais gérer des collections avancées (dictionnaires).

La classe GestionElement utilise un Dictionary<int, Element> et une List simple:



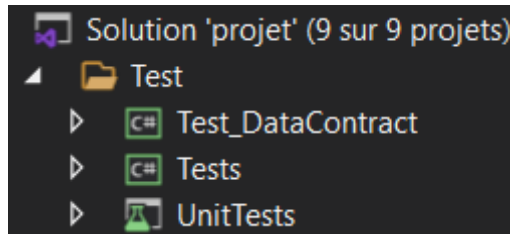
```
1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.Serialization;
4
5 namespace Modele
6 {
7     [DataContract]
8     public class GestionElement
9     {
10         /*
11          * stocke des Element de type info associé à une clé afin de les reconnaître plus précisément
12          */
13         [DataMember]
14         public Dictionary<int, Element> DicoElement { get; private set; }
15
16         /*
17          * stocke des Element de type bouteille
18          */
19         [DataMember]
20         public List<Element> ListeElement { get; private set; }
21     }
22 }
```

Je sais contrôler l'encapsulation au sein de mon application.

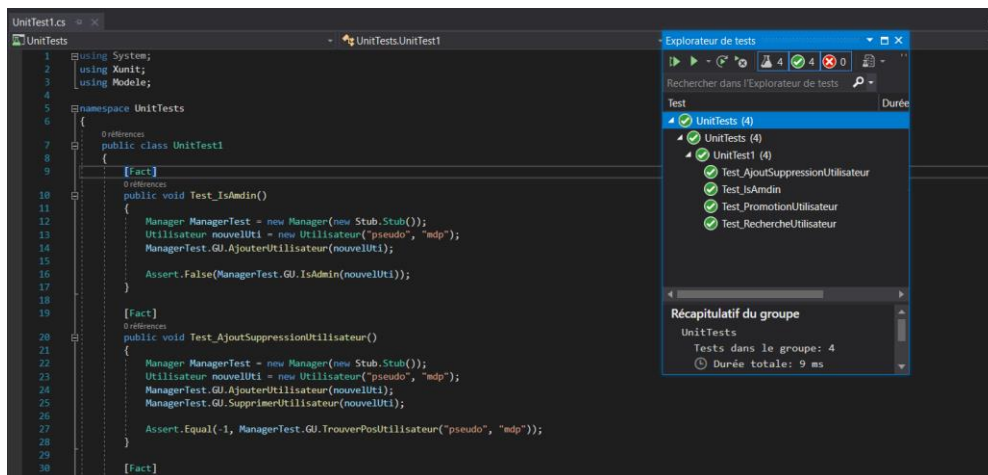
La majorité des éléments au sein de l'application sont public.

Je sais tester mon application.

Différents tests ont été réalisés :

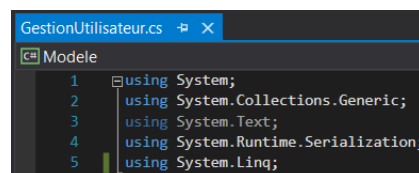


Exemple des test unitaires :

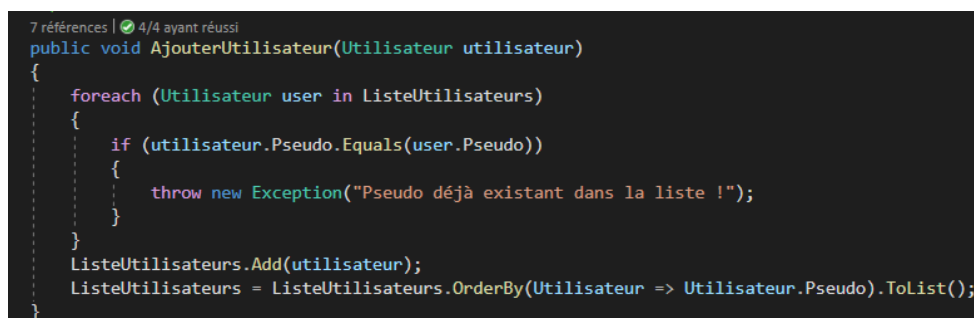


Je sais utiliser LINQ.

La classe GestionUtilisateur utilise System.Linq :



Avec LINQ, l'ajout dans la liste des utilisateurs est trié par ordre alphabétique grâce à OrderBy :



Je sais gérer les évènements.

La classe Element utilise un event :

```
public class Element : INotifyPropertyChanged
{
    /*
     * permet de notifier la vue d'un changement de la variable lienImage
     */
    public event PropertyChangedEventHandler PropertyChanged;
```

Lorsque la propriété LienImage est changée, elle déclenche un événement pour permettre à la Vue d'être « à jour » :

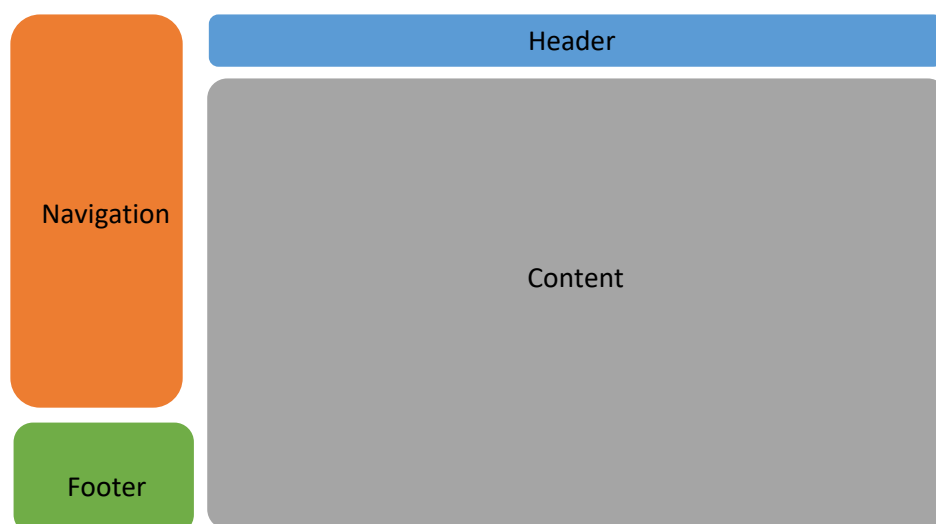
```
public String LienImage
{
    /*
     * but de la méthode : retourne lienImage
     * paramètres en entrée : aucun
     * variables : aucunes
     * fonction appelées : aucune
     * paramètre en sortie : -lienImage
     */
    get
    {
        return lienImage;
    }
    /*
     * but de la méthode : attribuer une valeur à lienImage
     * paramètres en entrée : -value : valeur de type string
     * variables : aucunes
     * fonction appelées : -PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("LienImage")) de la classe PropertyChangedEventArgs
     * paramètre en sortie : -lienImage
     */
    set
    {
        lienImage = value;
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("LienImage"));
    }
}
```

IHM : Interface Homme-Machine (XAML, WPF)

CODE

Je sais choisir mes layouts à bon escient.

Le layout globale de l'application ressemble à ceci :

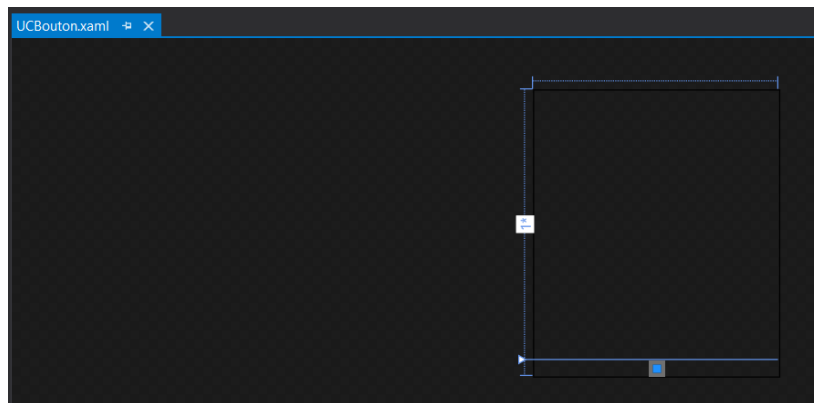


Je sais choisir mes composants à bon escient.

Les composants de la Vue sont essentiellement des Grids, StackPanels, WrapPanels, TextBox et TextBlocks, Listbox, Buttons ou encor CheckBox.

Je sais créer mon propre composant.

Nous avons créé un style de bouton à l'aide d'un UserControl :



L'image et le contenu du bouton sont définis avec le code behind.

Je sais personnaliser mon application en utilisant des ressources et des styles.

```
App.xaml
Application
1 <Application x:Class="WpfApp1.App"
2   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:local="clr-namespace:WpfApp1" xmlns:sys="clr-namespace:System;assembly=netstandard"
5   StartupUri="Loginpage.xaml">
6   <Application.Resources>
7     <SolidColorBrush x:Key="CouleurBoutonsEtTitres" Color="#0056b3"/>
8     <SolidColorBrush x:Key="CouleurBackGround" Color="White"/>
9     <SolidColorBrush x:Key="CouleurContenus" Color="DimGray"/>
10
11     <FontFamily x:Key="FontTitresEtBoutons">Showcard Gothic</FontFamily>
12     <sys:Double x:Key="fontSize">15</sys:Double>
13     <sys:Double x:Key="fontSizeTitres">18</sys:Double>
14
15     <Style x:Name="stylebouton" TargetType="Button">
16       <Setter Property="Background" Value="{DynamicResource CouleurBoutonsEtTitres}"/>
17       <Setter Property="FontFamily" Value="{DynamicResource FontTitresEtBoutons}"/>
18       <Setter Property="FontSize" Value="{DynamicResource fontSize}"/>
19       <Setter Property="Foreground" Value="White"/>
20     </Style>
21     <Style x:Name="stylecheckbox" TargetType="CheckBox">
22       <Setter Property="Background" Value="White"/>
23       <Setter Property="FontFamily" Value="Arial Black"/>
24       <Setter Property="FontSize" Value="{DynamicResource fontSize}"/>
25       <Setter Property="Foreground" Value="Gray"/>
26       <Setter Property="Width" Value="100"/>
27       <Setter Property="Height" Value="20"/>
28     </Style>
29     <Style x:Key="Titres" TargetType="TextBlock">
30       <Setter Property="Background" Value="{DynamicResource CouleurBackGround}"/>
31       <Setter Property="Foreground" Value="{DynamicResource CouleurBoutonsEtTitres}"/>
32       <Setter Property="FontSize" Value="{DynamicResource fontSizeTitres}"/>
```

Je sais utiliser les DataTemplate (locaux et globaux).

Création et utilisation de DataTemplate dans UCListeBouteille.xaml :

```
18 <ListBox x:Name="ListeElem" Grid.Row="1" Background="{x:Null}" ItemsSource="{Binding GE.ListeElement}" SelectionChanged="ListeElem_SelectionChanged">
19 <ListBox.ItemTemplate>
20 <DataTemplate>
21 <WrapPanel>
22 <Image Source="{Binding LienImage}" Width="50" Height="50"/>
23 <TextBlock Text="{Binding Nom}" Style="{StaticResource ResourceKey=Contenus}" />
24 </WrapPanel>
25 </DataTemplate>
26 </ListBox.ItemTemplate>
27 </ListBox>
```

Je sais intercepter les événements de la vue.

Exemple : Méthode Click d'un bouton :

```
UCLogin.xaml.cs  WpfApp1.UCLogin
1 référence
private void Login_Click(object sender, RoutedEventArgs e)
{
    if (ManagerUCLogin.GU.TrouverPosUtilisateur(Username.Text, mdp.Password) != -1)
    {
        ManagerUCLogin.SetUtilisateurCourant(new Utilisateur(Username.Text, mdp.Password));
        if (ManagerUCLogin.UtilisateurCourant.Theme == ThemeApp.Sombre)
        {
            (App.Current as App).ThemeCourant = ThemeApp.Sombre;
            App.PasserEnSombre();
        }
        else
        {
            (App.Current as App).ThemeCourant = ThemeApp.Clair;
            App.PasserEnClair();
        }
        MainWindow fenetre = new MainWindow();
        fenetre.Show();
        Window win = Window.GetWindow(this);
        win.Close();
    }
    else
    {
        incorrect.Visibility = Visibility.Visible;
    }
}
```

Je sais notifier la vue depuis des événements métier.

La classe Element implémente INotifyPropertyChanged :

```
public class Element : INotifyPropertyChanged
{
    /*
     * permet de notifier la vue d'un changement de la variable lienImage
     */
    public event PropertyChangedEventHandler PropertyChanged;
```

Lorsque la propriété LienImage est changée, elle déclenche un événement de notification pour permettre à la Vue d'être « à jour » :


```

public String LienImage
{
    /*
     * but de la méthode : retourne lienImage
     * paramètres en entrée : aucun
     * variables : aucunes
     * fonction appelées : aucune
     * paramètre en sortie : -lienImage
     */
    get
    {
        return lienImage;
    }
    /*
     * but de la méthode : attribuer une valeur à lienImage
     * paramètres en entrée : -value : valeur de type string
     * variables : aucunes
     * fonction appelées : -PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("LienImage")) de la classe PropertyChangedEventArgs
     * paramètre en sortie : -lienImage
     */
    set
    {
        lienImage = value;
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("LienImage"));
    }
}

```

Je sais gérer le DataBinding sur mon master.

```

<ListBox Grid.Row="0" Background="{x:Null}" Name="ListeBouteilles" ItemsSource="{Binding GE.ListeElement}" SelectedItem="Bouteille" SelectionChanged="ListeBouteilles_SelectionChan
<ListBox.ItemTemplate>
  <DataTemplate>
    <WrapPanel>
      <Image Source="{Binding LienImage}" Width="50" Height="50"/>
      <TextBlock Text="{Binding Nom}" Style="{StaticResource ResourceKey=Titres}"/>
    </WrapPanel>
  </DataTemplate>
</ListBox.ItemTemplate>
</ListBox>

```

Cf. UCInfoBouteille.xaml

Je sais gérer le DataBinding sur mon détail.

```

<Grid Grid.Column="0" DataContext="{Binding ElementName=ListeBouteilles, Path=SelectedItem}">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="0.16*" />
    <ColumnDefinition Width="0.33*" />
    <ColumnDefinition Width="0.33*" />
    <ColumnDefinition Width="0.16*" />
  </Grid.ColumnDefinitions>
  <Button x:Name="previous" Content="previous" HorizontalAlignment="Center" Height="30" VerticalAlignment="Bottom" Width="90" Margin="5" Click="previous_Click" />
  <Button x:Name="supprimer" Content="supprimer" Grid.Column="1" Height="30" VerticalAlignment="Bottom" Width="90" Margin="5" HorizontalAlignment="Center" Visibility="Hidden" Click="
  <Button x:Name="modifier" Content="modifier" Grid.Column="2" Height="30" VerticalAlignment="Bottom" Width="90" Margin="5" HorizontalAlignment="Center" Visibility="Hidden" Click="C
  <Border Grid.Column="1" BorderThickness="2" BorderBrush="Black" MaxHeight="200" MaxWidth="200">
    <Image x:Name="ImageBouteille" HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Stretch="Fill" Source="{Binding LienImage}" />
  </Border>
  <StackPanel HorizontalAlignment="Center" Grid.Column="2" VerticalAlignment="Center">
    <TextBlock TextAlignment="Center" TextWrapping="Wrap" VerticalAlignment="Center" Text="{Binding Nom}" Style="{StaticResource ResourceKey=Titres}" />
    <TextBlock TextAlignment="Center" TextWrapping="Wrap" VerticalAlignment="Center" Text="{Binding Contenu}" Style="{StaticResource ResourceKey=Contenus}" />
  </StackPanel>
  <Button x:Name="next" Grid.Column="3" Content="next" HorizontalAlignment="Center" Height="30" VerticalAlignment="Bottom" Width="90" Margin="5" Click="next_Click" />
</Grid>

```

Cf. UCInfoBouteille.xaml

Je sais gérer le DataBinding et les Dependency Property sur mes UserControl.

La quasi-totalité des éléments de la vue sont des User Control. Le Databinding est alors géré dessus.

Je sais développer un Master/Detail.

Cf. UCInfoBouteille.xaml ou encore UCListeUtilisateur.xaml

Projet Tuteuré S2

DOCUMENTATION

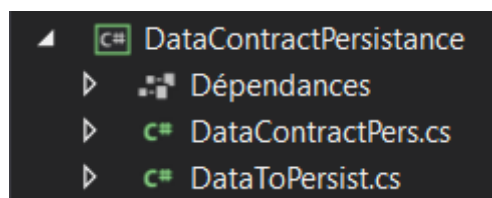
Je sais réaliser une vidéo de 1 à 3 minutes qui montre la démo de mon application.

Cf. Vidéo du répertoire « Documentation » du projet

CODE

Je sais coder la persistance au sein de mon application.

Classes permettant la persistance :



Les données sont enregistrées dans un dossier « Persistance » au niveau de l'exécutable dans le fichier donnees.xml

Persistance	12/06/2021 14:34	Dossier de fichiers	
DataContractPersistance.dll	12/06/2021 12:54	Extension de l'app...	7 Ko
DataContractPersistance.pdb	12/06/2021 12:54	Program Debug D...	11 Ko
Images.dll	12/06/2021 12:54	Extension de l'app...	1 430 Ko
Images.pdb	12/06/2021 12:54	Program Debug D...	12 Ko
Inf'Eau - Raccourci	12/06/2021 14:33	Raccourci	2 Ko
Inf'Eau.exe	12/06/2021 14:33	Application	156 Ko
Modele.dll	12/06/2021 12:54	Extension de l'app...	15 Ko
Modele.pdb	12/06/2021 12:54	Program Debug D...	13 Ko
Stub.dll	12/06/2021 12:54	Extension de l'app...	15 Ko
Stub.pdb	12/06/2021 12:54	Program Debug D...	10 Ko
WpfApp1.deps.json	12/06/2021 12:54	JSON File	2 Ko
WpfApp1.dll	12/06/2021 12:54	Extension de l'app...	71 Ko
WpfApp1.pdb	12/06/2021 12:54	Program Debug D...	25 Ko
WpfApp1.runtimeconfig.json	12/06/2021 12:54	JSON File	1 Ko

donnees.xml	12/06/2021 14:34	XML Document	10 Ko
-------------	------------------	--------------	-------

Exemple parmi beaucoup : Fonctionnalité d'ajout favoris :

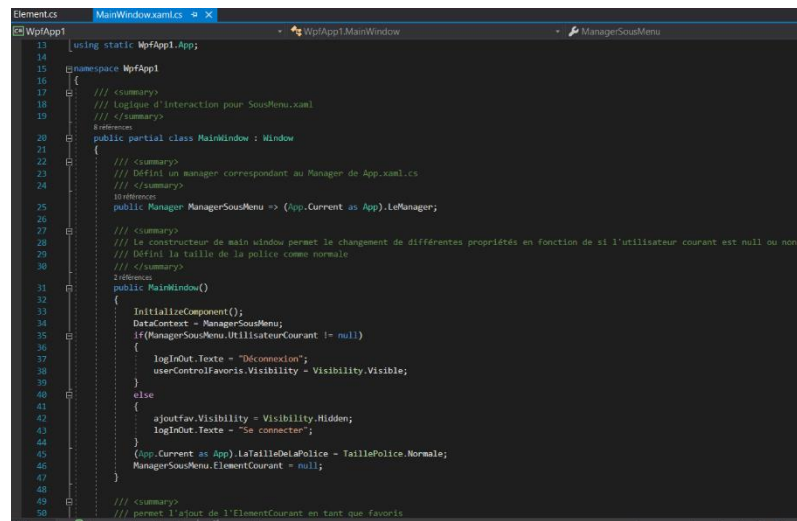
```

/// <summary>
/// permet l'ajout de l'ElementCourant en tant que favoris
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void ajoutfav_Click(object sender, RoutedEventArgs e)
{
    if(ManagerSousMenu.ElementCourant != null && ManagerSousMenu.UtilisateurCourant != null)
    {
        ManagerSousMenu.AjouterFavori();
    }
}

```

Je sais documenter mon code.

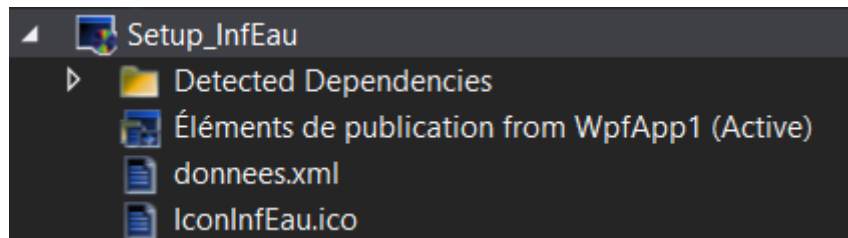
Toutes les parties de codes C# sont commentées, que ce soit le Modele ou WpfApp1. 2 exemples :





Je sais développer une application fonctionnelle.

L'entièreté de l'application fonctionne. L'exécutable présent dans le dossier code/Publication en est la preuve.

Je sais mettre à disposition un outil pour déployer mon application.



 InfEau_Setup.msi	14/06/2021 14:30	Package Windows ...	2 330 Ko
 setup.exe	14/06/2021 14:30	Application	1 080 Ko