Preuve

Documentation C#

Je sais concevoir un diagramme de classe qui représente mon application
Le diagramme de classe a été réalisé grâce à StarUML. Il se situe dans Documentation_C_sharp.pdf.

Je sais réaliser un diagramme de paquetage qui illustre bien l'isolation entre les parties de mon application
Le diagramme de paquetage se situe dans Documentation_C_sharp.pdf et a été réalisé avec StarUML.

Je sais décrire mes deux diagrammes en mettant en valeur les éléments essentiels
Pour le diagramme de classes, nous avons mis des couleurs afin de mettre en évidence certaines parties du diagramme. Par exemple, les classes représentant des détails du personnage sont représentées avec un fond bleu. Nous avons également décrit différentes parties du diagramme séparément.

Code_C# :

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

```
/// <summary>
/// Permet de supprimer un groupe de personnages.
/// </summary>
/// <param name="nom">Le nom du groupe à supprimer.</param>
3 références
public void SupprimerGroupe(string nom)
{
    Groupes.Remove(nom);
    OnPropertyChanged(nameof(NomsGroupes));
}
```

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

```
[DataContract]
23 références
public abstract class Nommable : IEquatable<Nommable>, IComparable<Nommable>, IComp
{
    // Attributs


    33 références
    public string Nom { get => nom; }
    [DataMember (Order = 0, Name = "nom")]
    private string nom;
```

La classe (abstraite) `Nommable`.

Je sais gérer des collections simples (tableaux, listes…) :

```
public ICollection<string> NomsGroupes { get { return new List<string>(Groupes.Keys); } }

6 références
public IList<Personnage> Personnages
{
    get
    {
```

Collections présentes dans le `Manager`.

Je sais gérer des collections avancées (dictionnaires) :

```
public IDictionary<string, ObservableCollection<Personnage>> Groupes { get; }
```

Le dictionnaire des groupes présents dans le `Manager`.

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

Nous avons encapsulé les collections. Ainsi, depuis l'extérieur de la classe, il est impossible de modifier directement la collection, car une `ReadOnlyCollection` est renvoyée.

```
public ReadOnlyObservableCollection<string> Citations { get => new ReadOnlyObservableCollection<string>(citations); }
[DataMember(EmitDefaultValue = false, Name = "citations")]
private ObservableCollection<string> citations;
```

La collection `citations` et sa version publique en lecture seule, dans la classe `Personnage`.

De manière générale, on utilise la structure { get; set; } spécifique à C# lorsque nous n'avons pas besoin d'effectuer des vérifications spécifiques lorsque nous assignons une nouvelle valeur à la propriété, et que nous pouvons directement renvoyer l'objet ou la variable sans transformation ou conversion :

```
[DataMember(EmitDefaultValue = false, Name = "annee")]
public int? AnneeDeCreation { get; set; }
```

Propriété `AnneeDeCreation` de `JeuVideo`.

Je sais tester mon application :

```
Manager manager = new Manager(new Stub());
manager.ChargeDonnees();
//Test de l'affichage des relations d'un personnage (ayant des relations) (+ ajout et suppression Relation)
manager.RechercherUneSerie("mario", out Serie serie);
manager.RechercherUnPersonnage("Mario", "mario", out Personnage perso1);
AfficherLesRelations(perso1);
```

Extrait de `Test_Personnage`

Les Test fonctionnels avec les projets : `Test_Personnage` et `Tests_Series_et_Groupes.`

```
public void Test_AjouterSerie_SerieExistante()
{
    Manager mgr = new Manager(new StubP.Stub());
    mgr.ChargeDonnees();

    Serie serie = new Serie("zelda");

    Assert.IsTrue(mgr.LesSeries.Series.Contains(serie));
    Assert.IsFalse(mgr.AjouterSerie("zelda", out _));
}
```

Extrait de `UnitTest.`

Les Tests_Unitaires avec le projet `UnitTest.`

Je sais utiliser LINQ :

```
return new ObservableCollection<Personnage>(LesSeries.Series.SelectMany(serie => serie.Personnages).OrderBy(n => n.Nom));
```

Utilisation de SelectMany pour former la liste de tous les personnages.

Je sais gérer les évènements :

```
5 références
public class NotificationRelationEvent : EventArgs
{
    2 références
    public Relation LaRelation { get; private set; }
    2 références
    public Personnage LePersonnage { get; private set; }

    /// <summary>
    /// Constructeur
    /// </summary>
    /// <param name="personnage">Le personnage à notifier</param>
    /// <param name="relation">La relation dans laquelle il est mentionné</param>
    1 référence
    public NotificationRelationEvent(Personnage personnage, Relation relation)
    {
```

Création de notre propre événement `NotificationRelationEvent.`

```
public EventHandler<NotificationRelationEvent> NotificationRelation;
1 référence
protected virtual void OnNotificationRelation(NotificationRelationEvent args)
    => NotificationRelation?.Invoke(this, args);

public event PropertyChangedEventHandler PropertyChanged;
2 références
protected virtual void OnPropertyChanged(string PropertyName)
    => PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(PropertyName));
```

Déclaration de l'événement `NotificationRelationEvent` dans la classe `Personnage`.

Déclaration de l'événement `PropertyChanged` (la classe `Personnage` implémente `IpropetyChanged`).

Invocation de l'événement `NotificationRelation` dans la Méthode `OnNotificationRelation`.
Invocation de l'événement `PropertyChanged` dans la Méthode `OnPropertyChanged`.

```
public bool AjouterRelation(string type, Personnage perso)
{
    Relation relation = new Relation(type, perso);
    if (!Relations.Contains(relation))
    {
        Relations.Add(relation);
        OnNotificationRelation(new NotificationRelationEvent(perso, relation));
        return true;
    }
    return false;
}
```

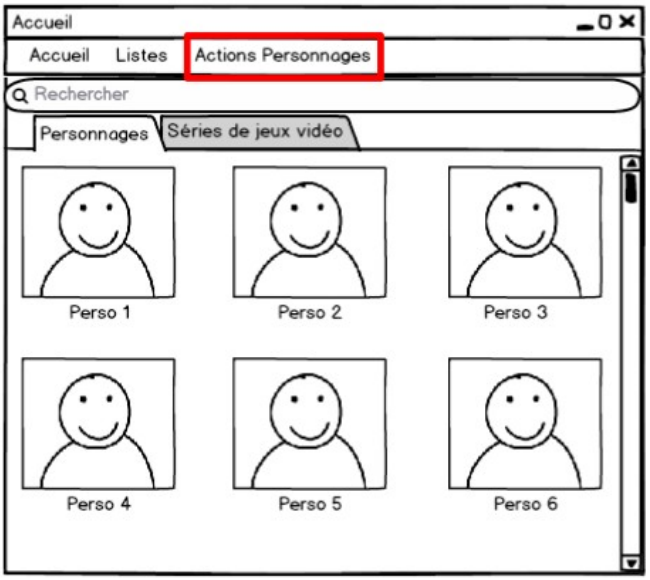
Appelle la de la méthode `OnNotificationRelation` dans la méthode `AjouterRelation`.

Documentation IHM

Je sais décrire le contexte de mon application pour qu'il soit compréhensible par tout le monde
Après chaque version du contexte, nous avons l'avons fait lire à des membres de nos familles respectives et à des amis qui n'ont pas forcément de connaissances en informatique. Cela nous a permis d'améliorer le contexte.

Je sais dessiner des sketchs pour concevoir les fenêtres de mon application
Nous avons utilisé le logiciel Balsamiq pour réaliser les sketchs de notre application. Ils sont présents dès la page 2 de Documentation_IHM.pdf.

Je sais enchaîner mes sketchs au sein d'un storyboard
Pour rendre nos storyboards plus clairs, nous avons encadré certains éléments. Par exemple, pour indiquer que l'utilisateur doit cliquer sur « Actions personnages », nous avons encadré cet élément du menu en rouge.



Je sais concevoir un diagramme de cas d'utilisation qui représente les fonctionnalités de mon application
Le diagramme de cas d'utilisation a été réalisé grâce à StarUML et se trouve dans Documentation_IHM.pdf.

Je sais concevoir une application ergonomique
Les détails concernant l'ergonomie de l'application sont situés à la dernière page du document Documentation_IHM.pdf.

Code Interface Homme-Machine :

Je sais choisir mes layouts à bon escient :

Le `DockPanel` nous permet de placer facilement le Master à gauche, le Detail au centre, et le menu en haut.

```
<DockPanel>
    <DockPanel.Background>
        <ImageBrush ImageSource="/Bibliothèques_Images;Component/Images_Fonds/arriere_plan_grille_
    </DockPanel.Background>
    <Grid Background="#FFF9E2C6" DockPanel.Dock="Top" VerticalAlignment="Bottom">
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <local1:UC_Menu Grid.Column="0"/>
        <Menu Background="#FFF9E2C6" Grid.Column="1" HorizontalAlignment="Right" Margin="5, 0">
            <MenuItem Header="Actions Personnage Sélectionné" Height="37" >
                <MenuItem Header="Supprimer" Click="SupprimerPersonnageClick"/>
                <MenuItem Header="Modifier" Click="ModifierPersonnageClick"/>
                <MenuItem Header="Exporter" Click="Exporter"/>
                <MenuItem Header="Ajouter à" Click="AjouterAUnGroupe">
                </MenuItem>
            </MenuItem>
```

Utilisation d'un `DockPanel` dans le `UC_Principal_MainWindow`.

Le `StackPanel` nous permet de positionner facilement l'image du personnage au-dessus de son nom dans la mosaïque.

```
<ListBox.ItemTemplate>
    <DataTemplate>
        <StackPanel>
            <Image Source="{Binding Image, Converter={StaticResource string2ImageConverter}}"
                   Width="100" Height="100" VerticalAlignment="Center"/>
            <TextBlock Text="{Binding Nom}" Style="{StaticResource textInformatif}"/>
        </StackPanel>
    </DataTemplate>
</ListBox.ItemTemplate>
```

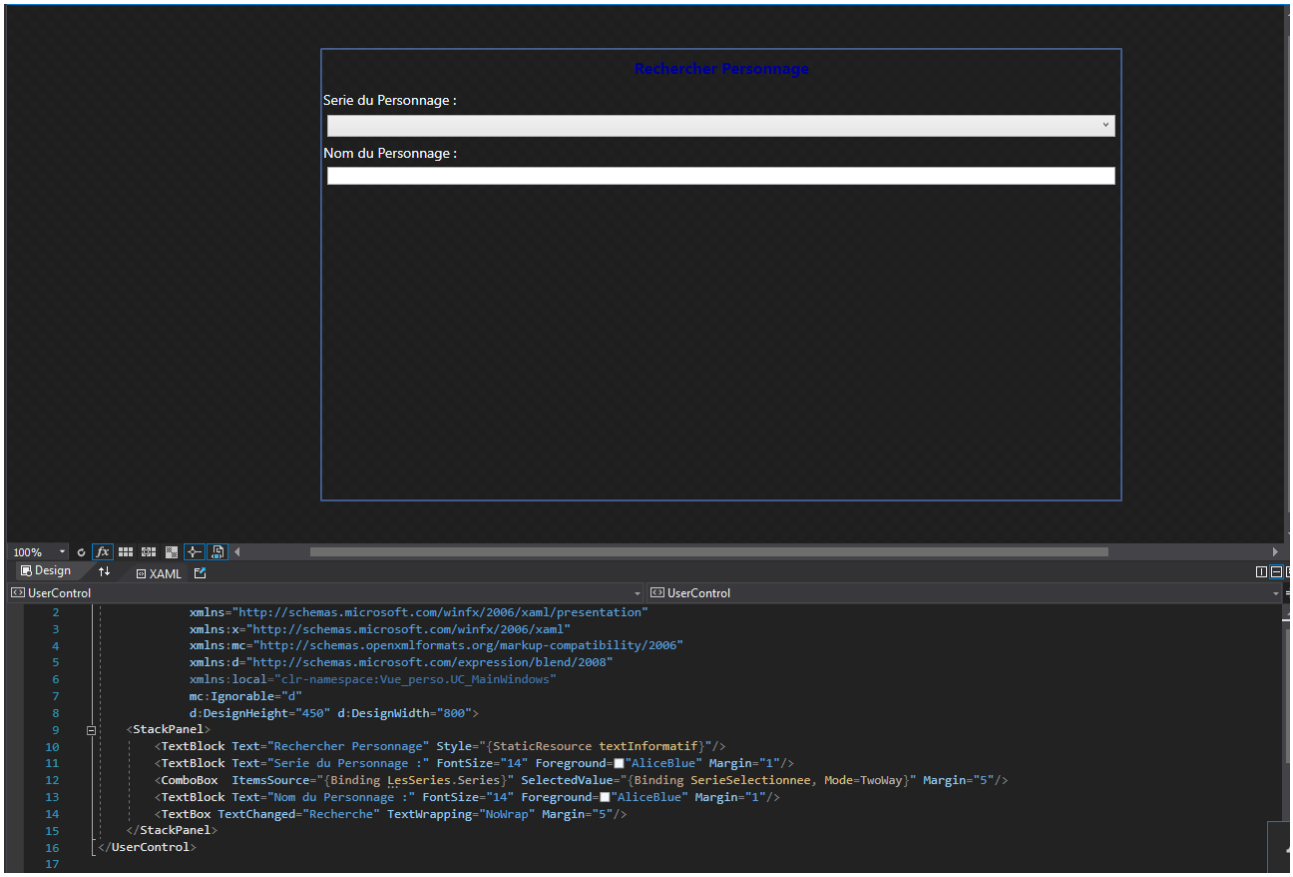
Utilisation d'un `StackPanel` dans le `DataTemplate` de la `ListBox` de `MosaiquePersonnages_UC`

Je sais choisir mes composants à bon escient :

```
<ListBox Background="{StaticResource BackgorundSalmon}" Grid.Row="1" Grid.Column="1" ItemsSource="{Binding JeuxV
    <ListBox.ItemTemplate>
        <DataTemplate>
            <WrapPanel>
                <TextBlock Style="{StaticResource textInformatif}" Text="{Binding Nom}"/>
                <TextBlock Style="{StaticResource textInformatif}" Text="{Binding AnneeDeCreation}"/>
            </WrapPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

Utilisation d'une ListBox pour stocker la liste des personnages dans UC_Principal_MainWindow

Je sais créer mon propre composant :



Création des différents UserControl par exemple le `UC_Recherche_Perso_Classique`.

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

```
        xmlns:conv="clr-namespace:Vue_perso.Converters">
    <Application.Resources>
        <conv:String2ImageConverter x:Key="string2ImageConverter"/>
        <conv:String2ColorConverter x:Key="string2ColorConverter"/>
        <conv:OppositeThemeTypeConverter x:Key="oppositeThemeTypeConverter" />
        <conv:ThemeTypeConverter x:Key="themeTypeConverter" />
        <Style TargetType="Button" x:Key="CommonButtonDisabler">
            <Style.Triggers>
                <DataTrigger Binding="{Binding ElementName=MandatoryTextBox, Path=Text.Length}" Value="0">
                    <Setter Property="IsEnabled" Value="False" />
                </DataTrigger>
            </Style.Triggers>
        </Style>
        <Style TargetType="TextBlock" x:Key="textInformatif">
            <Setter Property="FontFamily" Value="Comic-San" ></Setter>
            <Setter Property="HorizontalAlignment" Value="Center"></Setter>
            <Setter Property="FontSize" Value="16"></Setter>
            <Setter Property="VerticalAlignment" Value="Center"></Setter>
            <Setter Property="Foreground" Value="DarkBlue"></Setter>
            <Setter Property="FontWeight" Value="Bold"></Setter>
            <Setter Property="Margin" Value="10"></Setter>
        </Style>
```

On peut voir ici les styles placés en ressources de l’application.

Je sais utiliser les DataTemplate (locaux et globaux) :

```
<TextBlock Style="{StaticResource textInformatif}" Text="Jeux video :" Grid.Row="1"/>
<ListBox Background="{StaticResource BackgorundSalmon}" Grid.Row="1" Grid.Column="1" ItemsSource="{Bi
    <ListBox.ItemTemplate>
        <DataTemplate>
            <WrapPanel>
                <TextBlock Style="{StaticResource textInformatif}" Text="{Binding Nom}"/>
                <TextBlock Style="{StaticResource textInformatif}" Text="{Binding AnneeDeCreation}"/>
            </WrapPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

DataTemplate utilisé dans la ListBox présentant les jeux d'un personnage dans `UC_Principal_MainWindow`.

Je sais intercepter les événements de la vue

Dans ModifierPerso.xaml, nous interceptons l'événement Click de différents boutons avec Click="RenseignerChamp" :

```
<Label   Grid.Column="0" Grid.Row="2" Content="Citations :"/>
<Button Grid.Column="1" Grid.Row="2" Content="Modifier..." Name="CitationsButton" Click="RenseignerChamp"/>

<Label   Grid.Column="0" Grid.Row="3" Content="Jeux vidéos :"/>
<Button  Grid.Column="1" Grid.Row="3" Content="Modifier..." Name="JVButton" Click="RenseignerChamp"/>

<Label   Grid.Column="0" Grid.Row="4" Content="Thèmes musicaux :"/>
<Button  Grid.Column="1" Grid.Row="4" Content="Modifier..." Name="ThemeButton" Click="RenseignerChamp"/>

<Label   Grid.Column="0" Grid.Row="5" Content="Relations :"/>
<Button  Grid.Column="1" Grid.Row="5" Content="Modifier..." Name="RelationsButton" Click="RenseignerChamp"/>
```

On définit la méthode associée dans ModifierPerso.xaml.cs :

```
private void RenseignerChamp(object sender, RoutedEventArgs e)
{
    Button senderButton = sender as Button;
    Window newWindow;
    /*
     * On regarde quel bouton a été cliqué.
     * Ensuite, on instancie un Window, contenant une référence vers une fenêtre demandée.
     */
    if (senderButton.Equals(CitationsButton))
    {
        newWindow = new CitationsDialog(Perso);
        newWindow.ShowDialog();
    }
    else if (senderButton.Equals(JVButton))
    {
        newWindow = new JeuxVideoDialog(Perso);
        newWindow.ShowDialog();
    }
    else if (senderButton.Equals(ThemeButton))
    {
        newWindow = new ThemeMusical();
        newWindow.ShowDialog();
    }
    else if (senderButton.Equals(RelationsButton))
    {
        newWindow = new RelationsDialog();
        newWindow.ShowDialog();
    }
}
```

Je sais notifier la vue depuis des événements métiers

Exemple de propriété de la classe Personnage :

```
[DataMember(EmitDefaultValue = false, Name = "image")]
private string image;
public string Image
{
    get
    {
        if (image == null) return "image_par_defaut.png"; // On renvoie l'image par défaut
        else return image;
    }
    set
    {
        image = value;
        OnPropertyChanged(nameof(Image));
    }
}
```

```
public event PropertyChangedEventHandler PropertyChanged;
protected virtual void OnPropertyChanged(string PropertyName)
    => PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(PropertyName));
```

À noter que la classe Personnage implémente InotifyPropertyChanged afin de pouvoir notifier la vue lors de l'invocation de PropertyChanged.

Je sais gérer le DataBinding sur mon Master

Dans `UC_Principal_MainWindow.xaml`, on utilise un Binding sur une DependencyProperty `ListePersonnage`, afin d'obtenir la liste des personnages effective.

```
<ListBox Background="{StaticResource BackgorundSalmon}" Name="ListBo
         ItemsSource="{Binding ListePersonnages, ElementName=root}"
```

En effet, dans le cas de l'affichage des personnages d'une série spécifique, la liste des personnages n'est pas la même.

Je sais gérer le DataBinding sur mon Detail

Dans `UC_Principal_MainWindow.xaml`, on utilise modifie le DataContext de la grille du Detail :

```
<Grid Background="{StaticResource BackgorundSalmon}" DataContext="{Binding PersonnageSelect, ElementName=root}">
```

PersonnageSelect est une DependencyProperty. Elle est mise à jour à chaque changement de l'élément sélectionné, via un DataBinding sur SelectedItem de la ListBox du Master.

```
<ListBox Background="{StaticResource BackgorundSalmon}" Name="ListBoxPersonnages"
         Grid.Row="2" ItemsSource="{Binding ListePersonnages, ElementName=root}"
         SelectedItem="{Binding PersonnageSelect, ElementName=root, Mode=TwoWay}"
         SelectionChanged="ListBoxPersonnages_SelectionChanged">
```

Ensuite, chaque élément contenant une information du personnage sélectionné est bindé sur les propriétés du personnage en question :

```
<Image Width="150" Height="150" Source="{Binding Image, Converter={StaticResource string2ImageConverter}}" HorizontalAlignment="left" Grid.Row="0" Grid.Colu
<Grid Grid.Column="1">
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition/>
    </Grid.RowDefinitions>
    <TextBlock Style="{StaticResource textInformatif1}" Grid.Row="0" Grid.Column="1" Text="{Binding Nom}"/>
    <TextBlock Style="{StaticResource textInformatif1}" FontSize="30"  Text="{Binding CitationAleatoire}" Grid.Column="1" Grid.Row="1" TextWrapping="Wrap" T
</Grid>

<TextBlock Style="{StaticResource textInformatif}" Text="Jeux video :" Grid.Row="1"/>
<ListBox Background="{StaticResource BackgorundSalmon}" Grid.Row="1" Grid.Column="1" ItemsSource="{Binding JeuxVideo}">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <WrapPanel>
                <TextBlock Style="{StaticResource textInformatif}" Text="{Binding Nom}"/>
                <TextBlock Style="{StaticResource textInformatif}" Text="{Binding AnneeDeCreation}"/>
            </WrapPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
<TextBlock Style="{StaticResource textInformatif}" Text="Thème musical" Grid.Row="2"/>
<ListBox Background="{StaticResource BackgorundSalmon}" Grid.Row="2" Grid.Column="1" ItemsSource="{Binding Theme.Titres}" MouseDoubleClick="OuvertureLienTit
```

Je sais gérer le DataBinding et les DependencyProperties sur mes UserControl

L'une des DependencyProperties de MosaiquePersonnage_UC est MosaicWidth :

```
public double MosaicWidth
{
    get { return (double)GetValue(MosaicWidthProperty); }
    set { SetValue(MosaicWidthProperty, value); }
}

// Using a DependencyProperty as the backing store for MosaicWidth.  This enables animation, styling, binding, etc...
public static readonly DependencyProperty MosaicWidthProperty =
    DependencyProperty.Register("MosaicWidth", typeof(double), typeof(MosaiquePersonnages_UC), new PropertyMetadata(0.0));
```

Cette DependencyProperty nous permet de définir la largeur du WrapPanel, utilisé en ItemsPanel de la ListBox affichant les personnages :

```
<ListBox ItemsSource="{Binding Personnages}" Background="{x:Null}" MouseUp
    <ListBox.ItemsPanel>
        <ItemsPanelTemplate>
            <WrapPanel Width="{Binding MosaicWidth, ElementName=root}"/>
        </ItemsPanelTemplate>
    </ListBox.ItemsPanel>
```

Dans Accueil.xaml, elle est passée via un Binding :

```
<ContentControl x:Name="contentControlAccueil">
    <UC_accueil:MosaiquePersonnages_UC MosaicWidth="{Binding ElementName=contentControlAccueil, Path=ActualWidth}"/>
</ContentControl>
```

Ainsi, la largeur du WrapPanel est égale à la largeur effective du ContentControl parent du UserControl.

Je sais développer un Master/Detail
Notre application est un Master/Detail.

Documentation projet tuteuré

Je sais mettre en avant dans mon diagramme de classe la persistance de mon application
Les classe en lien avec la persistance sont représentées avec un fond marron. De manière similaire aux autres parties du diagramme de classes dans la documentation C#, nous avons isolé une partie du diagramme afin de le décrire plus clairement.

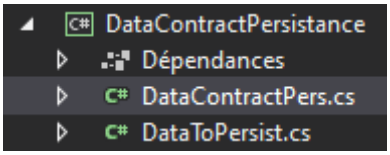
Je sais mettre en avant dans mon diagramme de classes ma partie personnelle
Nous avons encadré dans le diagramme de classes les méthodes et propriétés spécifiques à nos ajouts personnels, puis nous avons décrit le fonctionnement technique de ces ajouts.

Je sais mettre en avant dans mon diagramme de paquetages la persistance de mon application
Dans notre diagramme de paquetage, dans Documentation_Projet_Tuteure.pdf, nous avons entouré le projet spécifiques à la persistance de notre application.

Je sais réaliser une vidéo de 1 à 3 minutes qui montre la démo de mon application
La vidéo se situe dans le dossier Documentation.

Code Projet Tuteuré

Je sais coder la persistance au sein de mon application :
Le projet DataContractPersistance gère la persistance dans notre application.



DataContractPers gère la sauvegarde et le chargement. DataToPersist ne possède que la SérieTheque et le dictionnaire représentant les groupes.

La classe DataContractPers implémente notre interface IPersistance (faisant partie du modèle), imposant l'implémentation des fonctions Sauvegarder et Charger :

```
public interface IPersistance
{
    (SeriesTheque lesSeries, IDictionary<string, ObservableCollection<Personnage>> groupes) Charger();

    void Sauvegarder(SeriesTheque lesSeries, IDictionary<string, ObservableCollection<Personnage>> groupes);
}
```

Pour chaque classe à sérialiser, on ajoute les décorateurs DataContract pour les classes et DataMember pour les attributs.

```
[DataContract(Name = "jeuVideo")]
/// <summary>
/// Classe JeuVideo
/// </summary>
public class JeuVideo : Nommable
{
    [DataMember(EmitDefaultValue = false, Name = "annee")]
    public int? AnneeDeCreation { get; set; }
```

Je sais coder une fonctionnalité qui m'est personnelle
Les groupes de personnages sont un exemple de fonctionnalité qui ne nous était pas demandée, mais que nous avons tout de même réalisée.

Je sais documenter mon code
Les classes et les méthodes possèdent une documentation avant leurs définitions. Exemple avec la classe JeuVideo :

```
[DataContract(Name = "jeuVideo")]
/// <summary>
/// Classe JeuVideo
/// </summary>
public class JeuVideo : Nommable
{
    [DataMember(EmitDefaultValue = false, Name = "annee")]
    public int? AnneeDeCreation { get; set; }

    /// <summary>
    /// Constructeur de la classe JeuVideo
    /// </summary>
    /// <param name="nom">nom du jeu </param>
    /// <param name="annee">Année de publication du jeu</param>
    public JeuVideo(string nom, int? annee) : base(nom)
    {
        AnneeDeCreation = annee;
    }
```

Je sais utiliser SVN

Nous avons utilisé SVN pour gérer les versions de notre code. Notre application est disponible sur la forge :

Dernières révisions

#	Date	Auteur	Commentaire
155	13/06/2021 15:30	Théotime MAILLARBAUX	AJout des raccourcis Alt et du focus des boîtes de dialogue
154	13/06/2021 13:49	Théotime MAILLARBAUX	Correction de l'affichage en mosaïque
153	13/06/2021 11:31	Antoine VITON	(Correction commit précédent)
152	13/06/2021 11:29	Antoine VITON	Corrections du bug au niveau de l'ajout d'un personnage à un groupe alors qu'on est en train de consulter un groupe. Correction bug "retrouver image lors de l'importation". Correction bug sur RechercherUnPersonnage dans une série. Repassage du stub à la version pour les tests....
151	12/06/2021 23:52	Théotime MAILLARBAUX	Données de l'application stockées dans AppData\Roaming
150	12/06/2021 19:38	Antoine VITON	Correction d'un petit bug (image qui ne s'affiche pas) au niveau de l'installeur. Ajout d'un Readme.txt avec le setup.
149	12/06/2021 18:58	Antoine VITON	Mise en place de l'installeur. Création du projet. Génération de l'installeur.
148	12/06/2021 17:11	Antoine VITON	Ajout d'élément au stub. Modification d'un chemin d'accès Première publication du projet.
147	12/06/2021 15:43	Antoine VITON	Dernières Modifications avant déploiement. Résolution bug importation avec l'utilisation de la méthode clearEstMentionneDans. Légères Modifications du design des menus. Nettoyage de codes laissés en commentaire. Changement de l'icone de l'application. Modifications légères de la méthode d'affichage d'un Personnage après click sur une relation.
146	12/06/2021 13:04	Théotime MAILLARBAUX	Ouverture du lien d'un titre musical et de la fiche d'un personnage lors d'un double-clic sur respectivement un titre possédant un lien et une relation ayant un personnage enregistré.

Exemple de certains de nos commits

<u>Je sais développer une application qui compile</u>
Notre application compile.

<u>Je sais développer une application fonctionnelle</u>
Notre application ne présente aucun bug à notre connaissance.

<u>Je sais mettre à disposition un outil pour déployer mon application</u>
Nous avons utilisé l'extension Microsoft Visual Studio Installer Project pour déployer notre application.