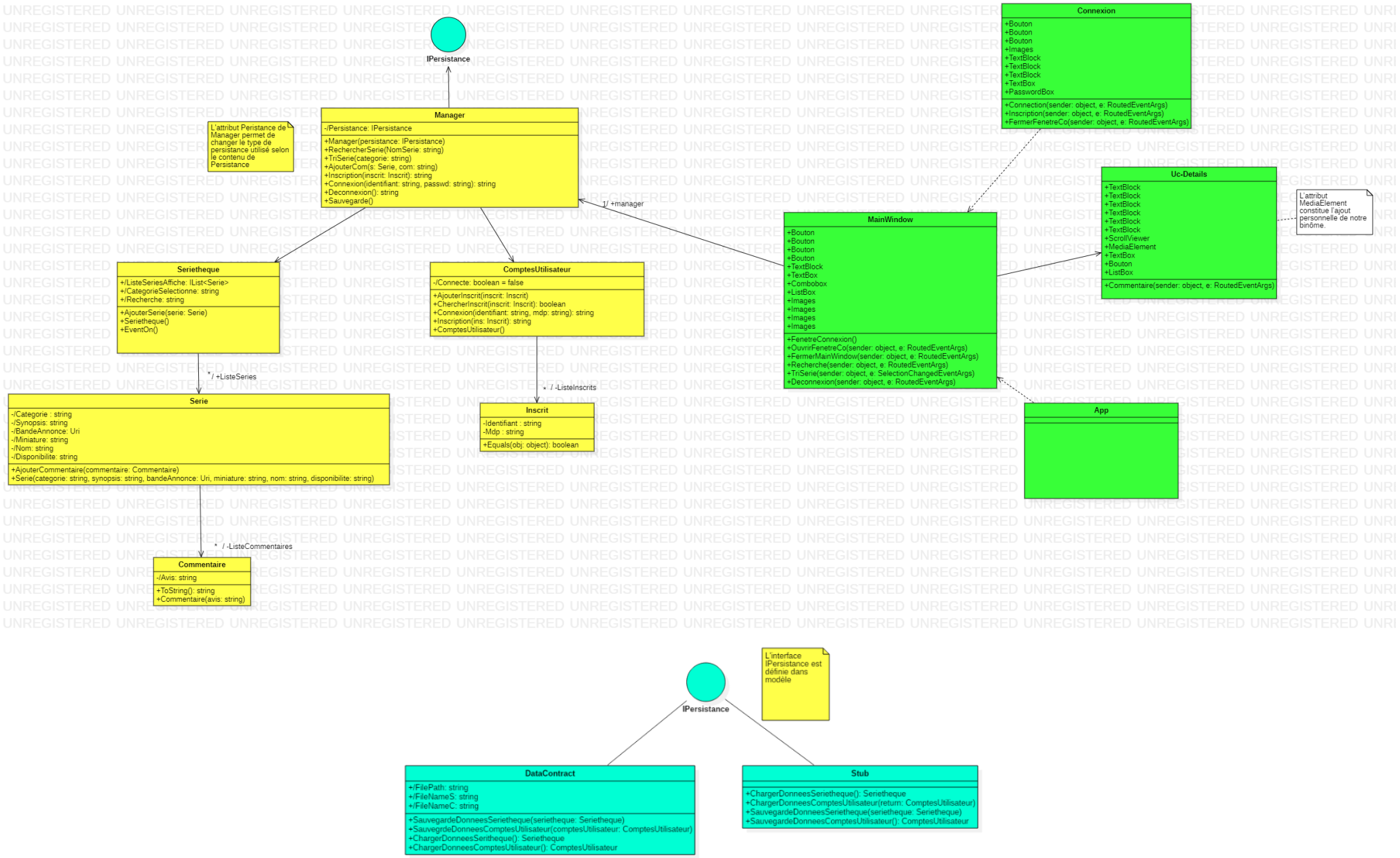
Bonaz Clément

Cellier Etienne

Documentation Conception et Programmation Orienté Objet

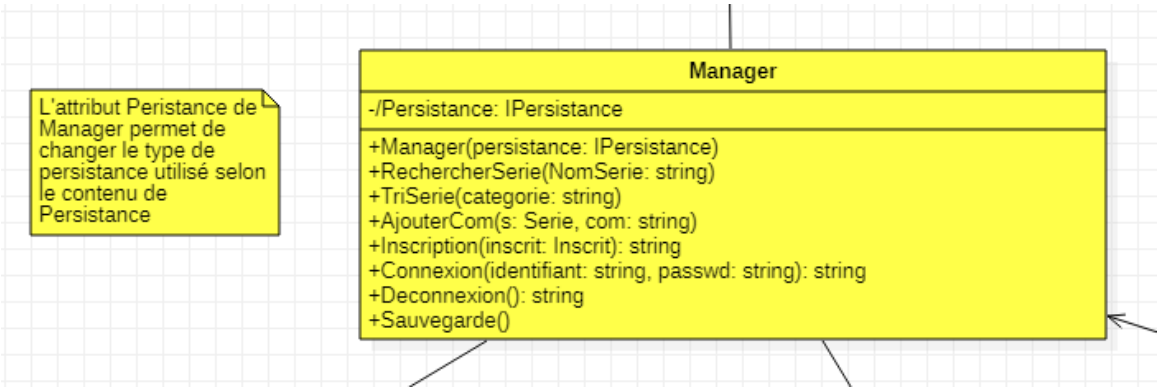
Diagramme de classes :


IPersistance
Manager
-/Persistance: IPersistance
+Manager(persistance: IPersistance)
+RechercherSerie(NomSerie: string)
+TriSerie(categorie: string)
+AjouterCom(s: Serie, com: string)
+Inscription(inscrit: Inscrit): string
+Connexion(identifiant: string, passwd: string): string
+Deconnexion(): string
+Sauvegarde()
L'attribut Peristance de Manager permet de changer le type de persistance utilisé selon le contenu de Persistance
1/ +manager
Connexion
+Bouton
+Bouton
+Bouton
+Images
+TextBlock
+TextBlock
+TextBlock
+TextBox
+PasswordBox
+Connection(sender: object, e: RoutedEventArgs)
+Inscription(sender: object, e: RoutedEventArgs)
+FermerFenetreCo(sender: object, e: RoutedEventArgs)
Uc-Details
+TextBlock
+TextBlock
+TextBlock
+TextBlock
+TextBlock
+TextBlock
+ScrollViewer
+MediaElement
+TextBox
+Bouton
+ListBox
+Commentaire(sender: object, e: RoutedEventArgs)
L'attribut MediaElement constitue l'ajout personnelle de notre binôme.
MainWindow
+Bouton
+Bouton
+Bouton
+Bouton
+TextBlock
+TextBox
+Combobox
+ListBox
+Images
+Images
+Images
+Images
+FenetreConnexion()
+OuvrirFenetreCo(sender: object, e: RoutedEventArgs)
+FermerMainWindow(sender: object, e: RoutedEventArgs)
+Recherche(sender: object, e: RoutedEventArgs)
+TriSerie(sender: object, e: SelectionChangedEventArgs)
+Deconnexion(sender: object, e: RoutedEventArgs)
Serietheque
+/ListeSeriesAffiche: IList<Serie>
+/CategorieSelectionne: string
+/Recherche: string
+AjouterSerie(serie: Serie)
+Serietheque()
+EventOn()
ComptesUtilisateur
-/Connecte: boolean = false
+AjouterInscrit(inscrit: Inscrit)
+ChercherInscrit(inscrit: Inscrit): boolean
+Connexion(identifiant: string, mdp: string): string
+Inscription(ins: Inscrit): string
+ComptesUtilisateur()
* / +ListeSeries
* / -ListeInscrits
Serie
-/Categorie : string
-/Synopsis: string
-/BandeAnnonce: Uri
-/Miniature: string
-/Nom: string
-/Disponibilite: string
+AjouterCommentaire(commentaire: Commentaire)
+Serie(categorie: string, synopsis: string, bandeAnnonce: Uri, miniature: string, nom: string, disponibilite: string)
Inscrit
-Identifiant : string
-Mdp : string
+Equals(obj: object): boolean
App
* / -ListeCommentaires
Commentaire
-/Avis: string
+ToString(): string
+Commentaire(avis: string)
IPersistance
L'interface IPersistance est définie dans modèle
DataContract
+/FilePath: string
+/FileNameS: string
+/FileNameC: string
+SauvegardeDonneesSerietheque(serietheque: Serietheque)
+SauvegrdeDonneesComptesUtilisateur(comptesUtilisateur: ComptesUtilisateur)
+ChargerDonneesSeritheque(): Serietheque
+ChargerDonneesComptesUtilisateur(): ComptesUtilisateur
Stub
+ChargerDonneesSerietheque(): Serietheque
+ChargerDonneesComptesUtilisateur(return: ComptesUtilisateur)
+SauvegardeDonneesSerietheque(serietheque: Serietheque)
+SauvegardeDonneesComptesUtilisateur(): ComptesUtilisateur

Architecture :

Nous avons choisi de mettre en place une structure de façade. Ce type de patron de conception permet de séparer les sous-systèmes et de rendre chaque sous-système « hermétique » et accessible uniquement grâce à point d'accès. Nous avons dans notre diagramme le sous-système Model qui contient tout le code qui permet de faire fonctionner notre application. Le point d'accès de Model et la classe Manager, grâce à la classe manager la Vue peut utiliser les méthodes contenues dans Model.

Manager :

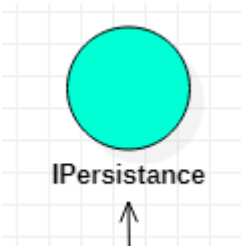


La classe Manager est une classe qui sert d'interface entre le Model et la Vue. Le Manager a pour objectif de donner à la vue les éléments dont elle a besoin pour son fonctionnement sans que la vue ait accès directement au code des éléments. La séparation du model et de la vue permet de ne pas provoquer de problème dans toute l'application si jamais la vue ou le model est modifié.

Le manager contient une Sérietheque qui possède différentes méthodes et une liste de séries et ComptesUtilisateur qui, comme Sérietheque contient plusieurs méthodes et une liste d'inscrit. Les méthodes de Manager sont des liens entre la vue et le model, par exemple la méthode TriSerie est appelé par la vue quand l'utilisateur souhaite réaliser un tri par catégorie, la méthode va alors appeler la méthode de tri propre à Sérietheque. Cet exemple montre comment le « bloc » Model interagi avec le « bloc » Vue sans que la vue ait accès au code contenu dans Model puisque le manger fait office de correspondance entre le Model et la Vue.

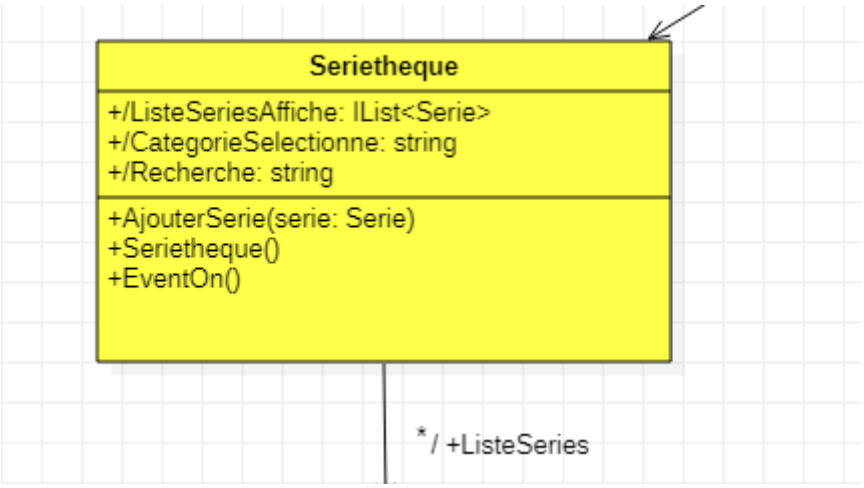
Enfin, la persistance des données est mise en place grâce à l'attribut Persistance de type IPersistance qui permet d'utiliser dans la méthode Sauvegarde, un des types de persistance disponible dans Data pour sauvegarder les données et pour charger les données lors de l'initialisation.

IPersistance :



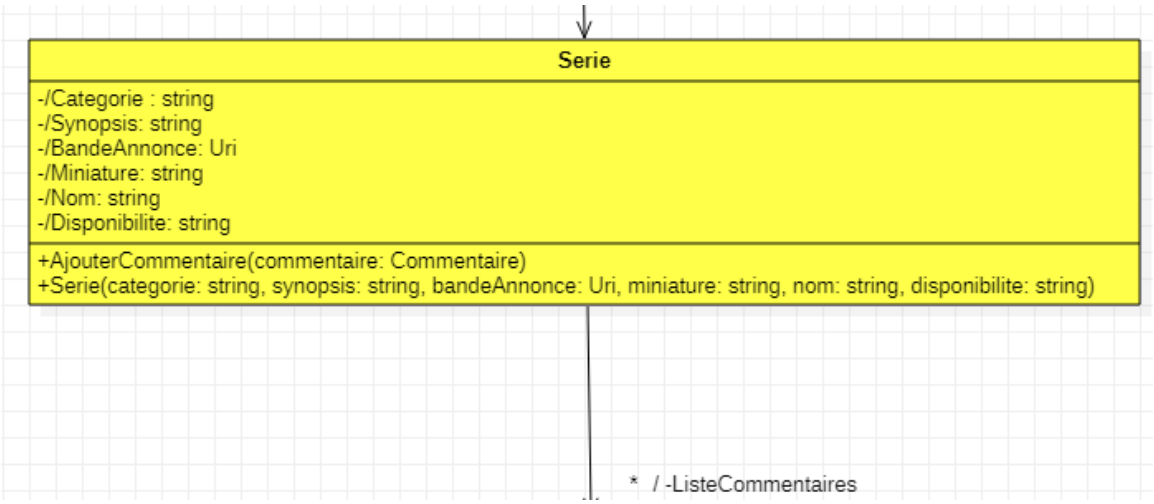
L'interface IPersistance implémente dans le Model la persistance des données, grâce aux méthodes de sauvegarde et de chargement que contient l'interface. Le manager peut selon le contenu de l'attribut Persistance réaliser une sauvegarde et un chargement avec les méthodes d'un certain type de persistance défini dans Data. Par exemple, le type DataContract réalise la sauvegarde sur un fichier XML alors qu'une base de données sauvegarde sous une autre forme. De plus, nous avons choisi de mettre en place 2 point de persistance des données, ces points sont Serietheque et ComptesUtilisateur, nous avons choisi ces deux points, car ils contiennent la liste de séries et la liste des inscrits. Les deux listes citées précédemment sont des éléments qui peuvent être modifiées à chaque utilisation et dont les modifications doivent être ajoutées à l'application. Par exemple, la liste de séries contient des séries qui ont chacune une liste de commentaires, l'utilisateur peut écrire un commentaire qui sera ajouté à cette liste.

Serietheque :



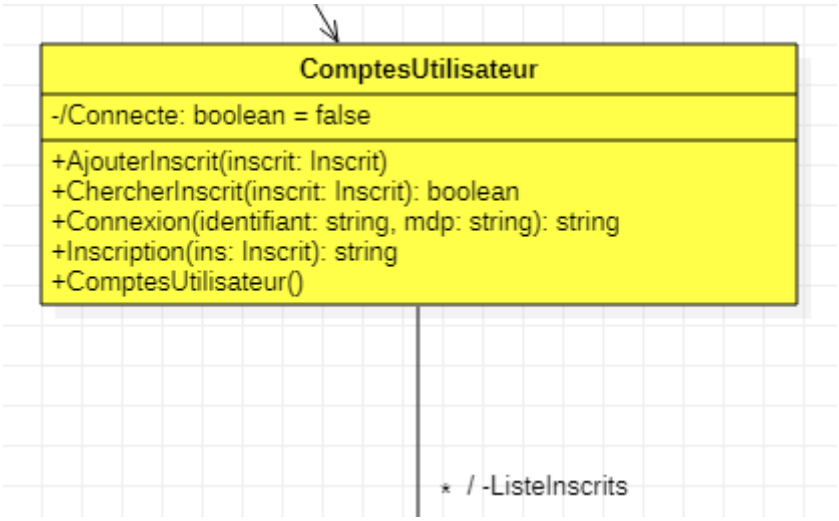
La classe "Serietheque" est une classe qui permet la création de plusieurs listes de "Serie" tel que "ListeSeries" qui est une ObservableCollection de "Serie". ListeSeries nous sert de réserve de données pour la liste "ListeSeriesAffiche" qui est une propriété calculée qui varie selon la recherche écrite ou de la catégorie sélectionnée. Par exemple lorsque la catégorie sélectionnée dans la comboBox et changé la ListeSeriesAffiche s'actualise pour ne contenir que les séries qui sont de la catégorie sélectionnée. La "Serietheque" a d'autres méthodes tel que AjouterSeries qui permet d'ajouter une "Serie" à "ListeSeries" et EventOn car la "Serietheque" hérite d'INotifyPropertyChanged afin que la liste s'actualise selon la recherche ou la catégorie.

Serie :



La classe “Serie” est une classe qui est composé de plusieurs attributs tel que la catégorie, le synopsis, le nom, la bande-annonce, la miniature et la disponibilité. L’attribut bande annonce et de type Uri, le type Uri est utilisé pour déterminer la source utilisée dans la balise MediaElement. Chaque “Serie” a aussi une liste de commentaire qui lui est propre. “ListeCommentaires” est une ObservableCollection afin que les commentaires soient sérialisés par le “DataContract”. La classe “Serie” a aussi quelques méthodes tel que AjoutCommentaire qui permet d’ajouter un commentaire à une série.

ComptesUtilisateur :

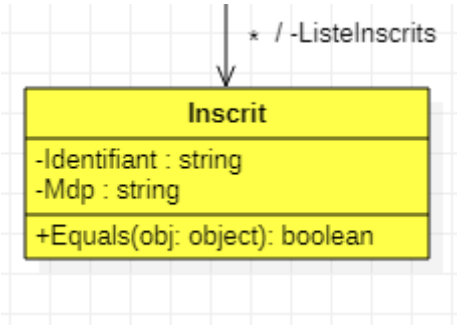


La classe “ComptesUtilisateur” est une classe qui permet de gérer les connexions, les déconnexions et les inscriptions. Elle a une List qui contient les ”Inscrit” et un seul attribut “Connect” qui permet de vérifier via son type qui est booléen, si un Inscrit est connecté ou non. Le fait d’être connecté permet la rédaction de commentaire dans le détail des séries. La classe a de nombreuses méthodes :

- Inscription : cette dernière permet de créer un nouvel “Inscrit”, néanmoins cette inscription respecte plusieurs conditions tel que le nouvel “Inscrit” ne peut pas avoir un identifiant et un mot-de-passe qui est vide. Mais aussi la vérification que l’identifiant saisie n’est pas le même qu’un “Inscrit” déjà existant.
- ChercherInscrit : qui permet de rechercher dans la liste si un inscrit passé en argument est présent dans la liste ou non.
- AjouterInscrit : qui permet d’ajouter à la liste un nouvel inscrit.

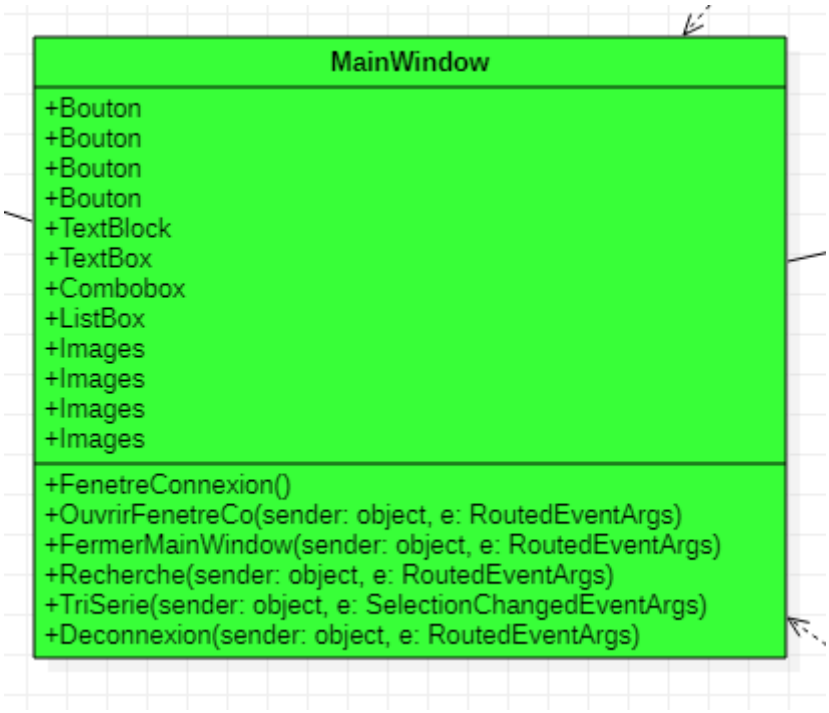
- Connexion : qui vérifie avec ce qui est passé en arguments, c'est-à-dire un identifiant et un mot de passe, si la connexion est au possible grâce à "ChercherInscrit".
- Deconnexion : qui vérifie si l'on est connecté ou non, si c'est le cas, l'utilisateur est déconnecté sinon un message est envoyé pour dire qu'il l'est déjà.

Inscrit :



La classe "Inscrit" est composé de deux attributs : Identifiant et MotDePasse. La classe "Inscrit" implémente DataContract afin de sérialiser les MotDePasse et Identifiant des utilisateurs. La méthode equals a été redéfini dans cette classe permettant de comparer un Inscrit passé en argument avec un autre.

MainWindow :



La classe "MainWindow" comporte des éléments. Elle contient quatre boutons qui engendre des événements à partir du moment où l'on leur clique dessus.

Le premier (celui à côté de la recherche) permet de récupérer le texte saisi dans la TextBox afin d'effectuer grâce à la méthode Recherche une recherche dans la liste de série pour retourner la série chercher.

Le deuxième permet d'ouvrir la fenêtre de connexion grâce aux deux méthodes OuvrirFenetreCo et FenetreConnexion.

Le troisième permet la déconnexion, le clic sur le bouton appelle la méthode Deconnexion qui rend l'attribut Connect = false.

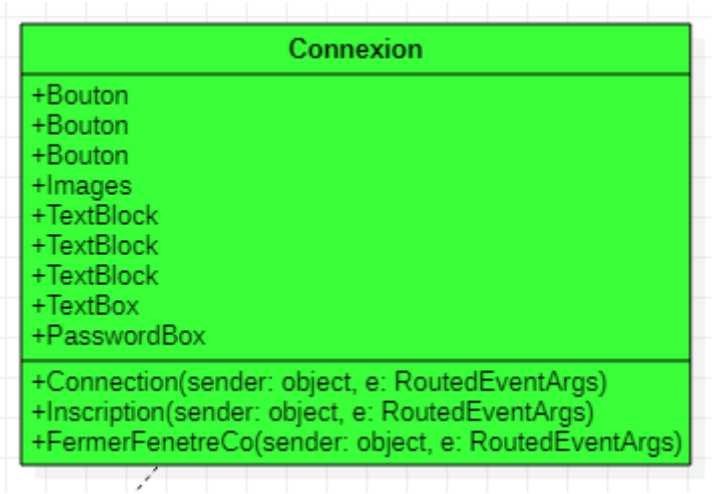
Le dernier permet de fermer la fenêtre grâce à la méthode FermerMainWindow.

La ComboBox recense toutes les catégories possibles, la sélection d'une de ses catégories fait appel à la méthode TriSerie qui fait appelle à la méthode TireSeries du Manager afin d'avoir une nouvelle liste de séries uniquement de même catégorie.

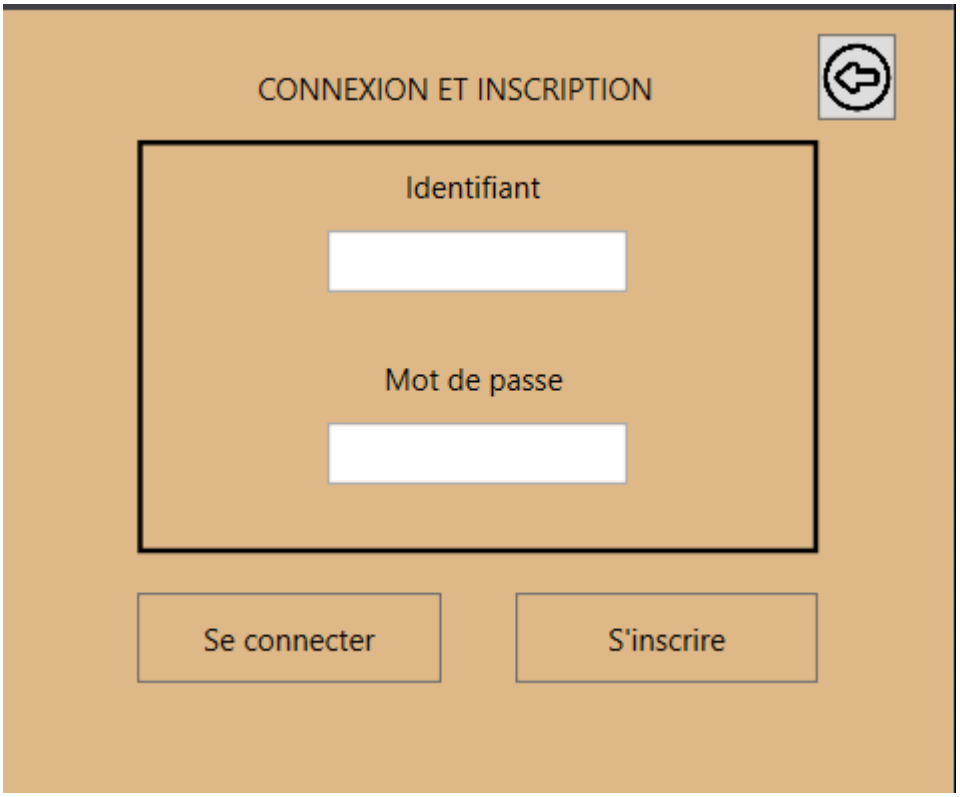
La ListBox est bindé sur ListeSeriesAffiche afin d'afficher les séries présentes dans la liste et on fait apparaître uniquement le Nom et la miniature de chaque série grâce au Binding.

Enfin, nous avons le ContentControl bindé sur l'élément sélectionné de la liste afin d'afficher ses détails dans le UserControl.

Connexion :



La classe "Connexion" se présente sous cette forme.

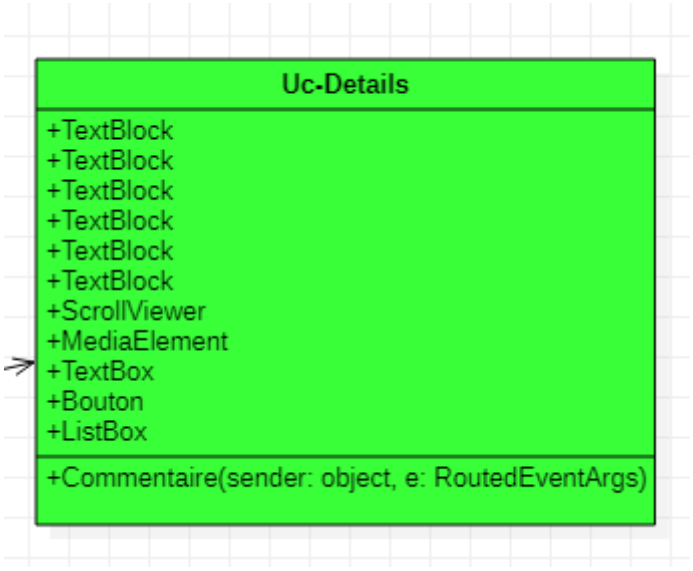


Pour la connexion, la méthode Connexion récupère l'identifiant ainsi que le mot de passe. Puis, elle fait appel à la méthode "Connexion" présente dans le Manager.

L'inscription est similaire à Connexion, elle récupère ce qui est saisie dans la TextBox et la PasswordBox puis fais appel à la méthode Inscription du Manager.

Le bouton permet de fermer la fenêtre grâce à la méthode FermerFenetreCo.

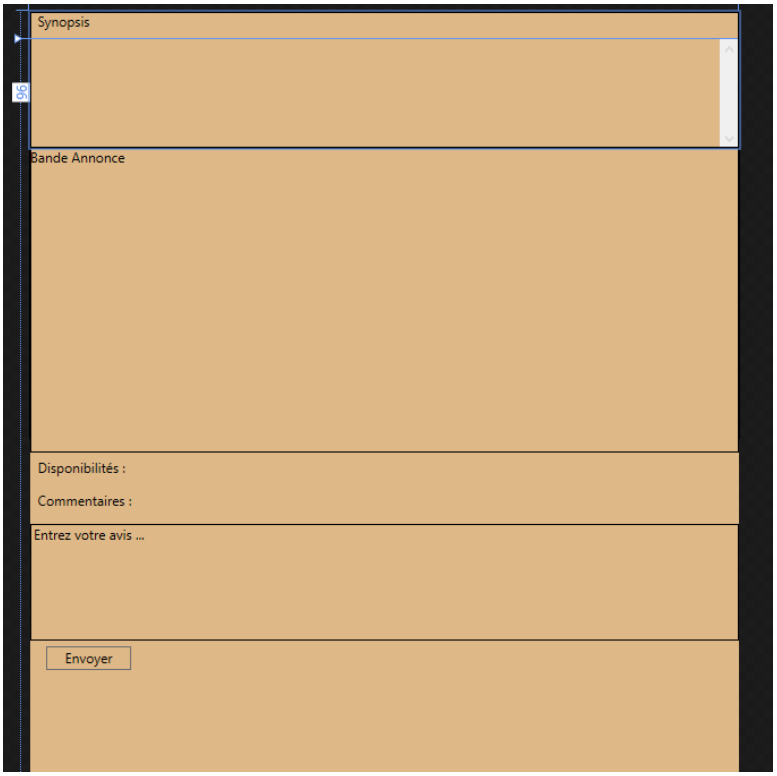
UC-Details :



La fenêtre/la classe UC-Details contient 6 TextBlock, les TextBlock sont des zones de texte où l'on affiche le synopsis, les disponibilités et un commentaire. Le contenu de ces 3 TextBlock est relié grâce au Binding au propriété contenue dans la Série sélectionné dans la MainWindow. Le TextBlock qui affiche un commentaire est lui bindé sur l'avis que contient un commentaire. Ce commentaire est isue de la listBox qui est bindé a la liste de commentaire de la série sélectionnée dans la MainWindow. Le reste des TextBlock sert à afficher du texte.

La classe MediaElement est utilisé afin de diffuser la bandeAnnonce de la série sélectionné la source de la vidéo à lire et afficher et renseigner dans l'attribut BandeAnnonce de Série sur laquelle bindé la source de MediaElement.L'attribut BandeAnnonce contient une chaîne de carractère qui représente le chemin d'accès à la vidéo de bande annonce.

La balise ScrollViwer permet de faire défilerle son contenu afin d'y accéder entièrement.EnfinEnfin le bouton déclanche lorsqu'on clique dessus l'événement Commentaire. Cette événement/méthode récupère grâce à App la série actuellement sélectionné dans la listbox de MainWindow pour ensuite appeler la méthode AjouterCom deManager qui vas ajouter le commentaire saisi dans la TextBox a la liste des commentaires de la série sélectionnée.

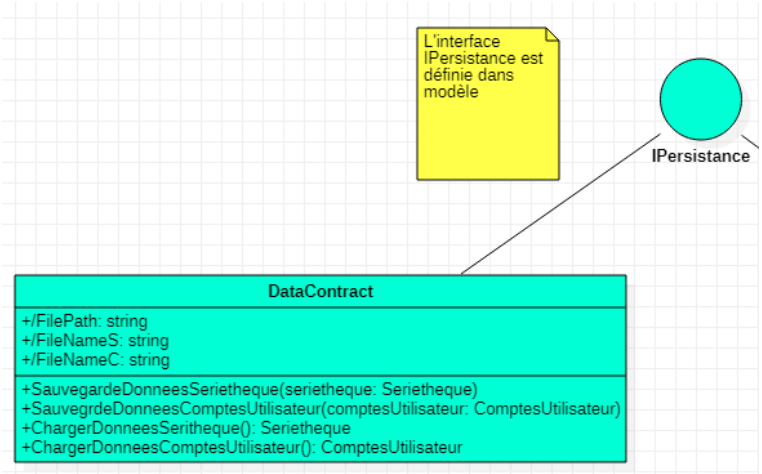


App :

La fenêtre App et une fenêtre et une classe qui est commune pour les fenêtres MainWindow, Connexion et UC-details. Grâce à cette proprité nous utilisont des attributs de la MainWindow depuis le code behind de UC-Details.

```
public void Commentaire(object sender, RoutedEventArgs e) // évenement déclaché par l'appuie sur le bo
{
    if (((MainWindow)Application.Current.MainWindow).Manager.ComptesUtilisateur.Connecte == true)
    {
        string CommentaireSaisie = EcrireCommentaire.Text; // on récupère le commentaire écrit dans la
        Serie temp = ((MainWindow)Application.Current.MainWindow).ListBoxSerie.SelectedItem as Serie;
```

DataContract :

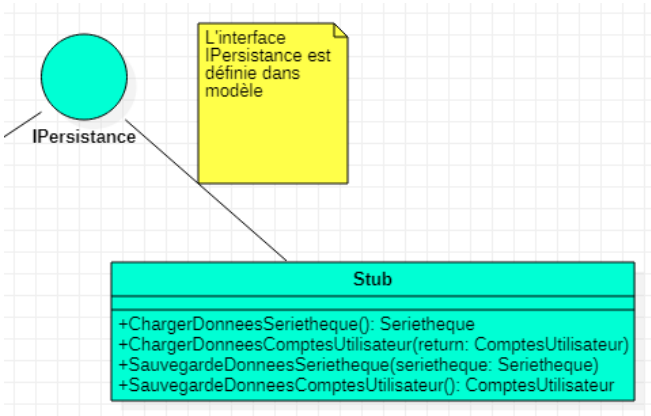


DataContract est une classe qui implémente l'interface IPersistance, les méthodes de sauvegarde et de chargement sont définies dans le but de sauvegarder les données dans des fichiers xml. Les définitions des méthodes de sauvegarde et de chargement ne sont qu'un procédé possible parmi les différentes possibilités. Il est totalement possible de faire la persistance sans utiliser de fichier xml. En effet si on le souhaite on mettre en place une classe BD qui définit la sauvegarde et le chargement par l'utilisation d'une base de données. Pour utiliser cette nouvelle méthode il nous suffira juste de changer le contenue de la variable Persistance de Manager. Les attributs FilePath, FileNameS et FileNameC permettent de définir le dossier où se situe les fichiers xml. FilePath représente le répertoire qui contient les fichier, FIleNameS et C représente le nom du fichier xml.

Les méthodes SauvegardeDonneesSerietheque et ComptesUtilisateur sont issues de l'interface, les 2 méthodes prennent en paramètre une Serietheque et ComptesUtilisateur afin de sauvegarder les attritut défini Datamembre dans le fichier xml grâce à un stream.

Les méthodes ChargerDonneesSerietheque et ComptesUtilisateur sont aussi issue de l'interface IPersistance, les 2 méthodes retourne une serietheque et un comptesUtilisateur qui contiennent les données des fichier xml récupéré grâce à un stream.

Stub :



Stub est une classe qui implémente l'interface IPersistance, les méthodes de sauvegarde et de chargement ne sont pas définies dans le but de réellement charger et sauvegarder les données. Ici les données sont chargées « en dure » c'est-à-dire que les données ne sont pas issues d'un fichier mais directement déclaré dans le corps de la méthode.

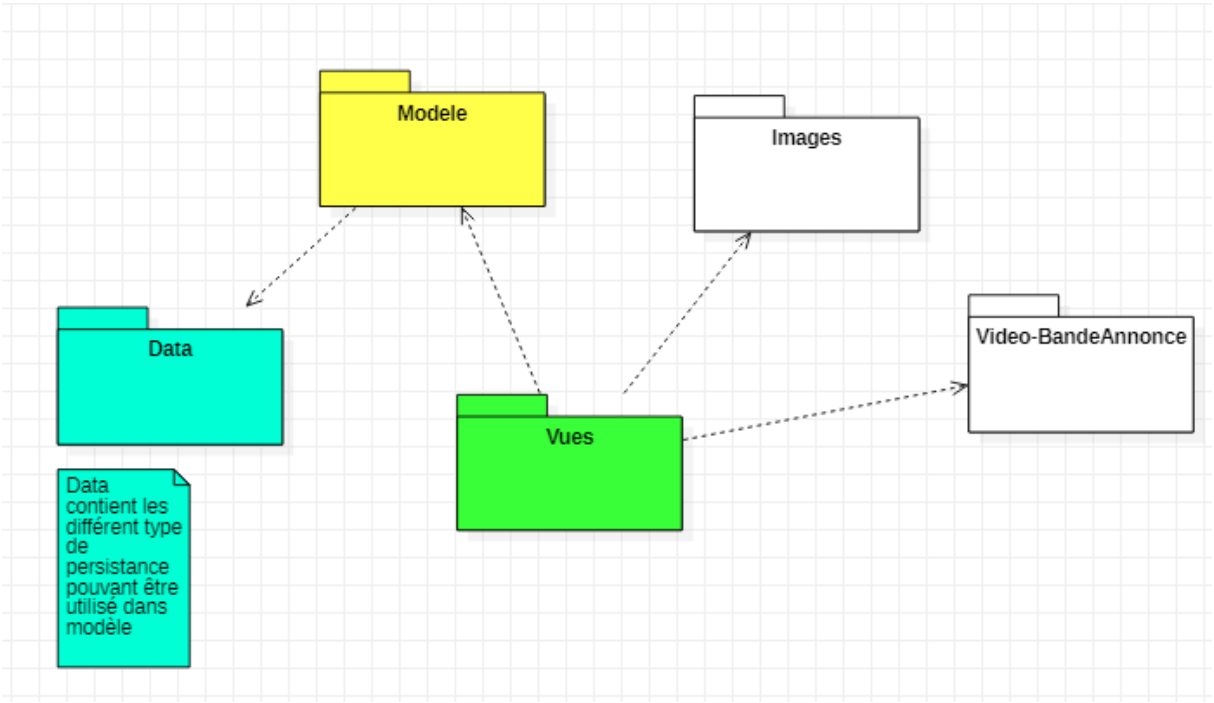
```
public  Serietheque ChargerDonneesSerietheque()
{
    Serietheque serietheque = new Serietheque();

   Serie s = new Serie("SF et Action", "A Hawkins, en 1983 dans l'Indiana.Lorsque Will Byers disparaît de son
    s.AjouterCommentaire(new Commentaire("Cette série est incroyable!!!"));
    serietheque.ListeSeries.Add(s);
    serietheque.ListeSeries.Add(new Serie("SF et Action", "Takeshi Kovacs est un ancien soldat et seul surviva
    serietheque.ListeSeries.Add(new Serie("SF et Action", "En 1989, le même jour, quarante-trois bébés sont in
    serietheque.ListeSeries.Add(new Serie("SF et Action", "Après un siècle de silence, les klingons refont sur
    serietheque.ListeSeries.Add(new Serie("SF et Action", "Après une apocalypse nucléaire causée par l'Homme l

    serietheque.ListeSeries.Add(new Serie("Sitcom", "Les péripéties de 6 jeunes newyorkais liés par une profon
    serietheque.ListeSeries.Add(new Serie("Sitcom", "Petit génie malgré lui, Malcolm vit dans une famille hors
    serietheque.ListeSeries.Add(new Serie("Sitcom", "Leonard et Sheldon pourraient vous dire tout ce que vous
    serietheque.ListeSeries.Add(new Serie("Sitcom", "Ted se remémore ses jeunes années, lorsqu'il était encore
    serietheque.ListeSeries.Add(new Serie("Sitcom", "Fils aîné d'une famille de classe moyenne et martyrisé pa

    serietheque.ListeSeries.Add(new Serie("Manga", "Dans un monde ravagé par des titans mangeurs d'homme depui
    serietheque.ListeSeries.Add(new Serie("Manga", "Cinq ans après le mariage de Son Gokû, Raditz, un mystérie
    serietheque.ListeSeries.Add(new Serie("Manga", "Dans le village de Konoha vit Naruto, un jeune garçon déte
    serietheque.ListeSeries.Add(new Serie("Manga", "Dans un monde magique au beau milieu du pays de Fiore, la
    serietheque.ListeSeries.Add(new Serie("Manga", "En 2022, l'humanité a réussi à créer une réalité virtuelle
```

Puisque le Stub n'est pas une « vrai » manière de conserver les données les méthodes de sauvegarde ne sont pas fonctionnelles elles contiennent juste un message qui indic sur la console de Debug que la sauvegarde a était effectué.

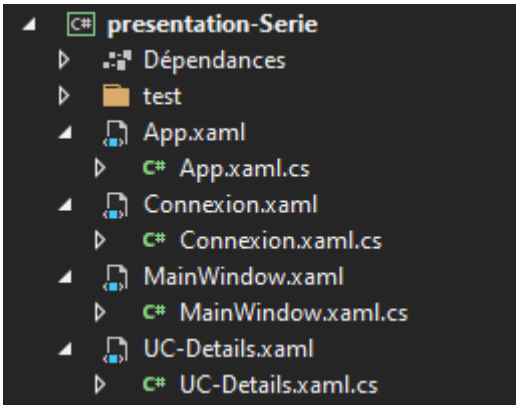
```
public void SauvegardeDonneesComptesUtilisateur(ComptesUtilisateur comptesUtilisateur)
{
    Debug.WriteLine("Données enregistré");
}

3 références
public void SauvegardeDonneesSerietheque(Serietheque serietheque)
{
    Debug.WriteLine("Données enregistré");
}
```

Le Stub est principalement utilisé dans les programmes de test puisque le Stub permet de modifier les données sans aucun impact sur l'application.

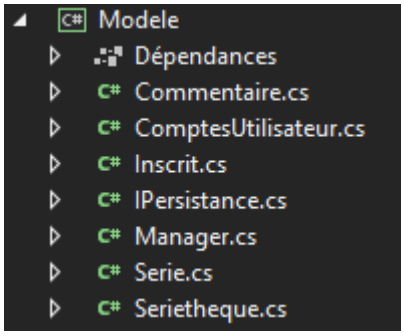
Diagramme de paquetages :



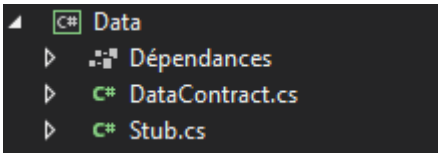
Vues : Le package de Vues est séparé du Modele et des autres packages afin qu'elle n'ait accès au code dont elle a besoin. Il est composé des différentes vues, c'est-à-dire la fenêtre de connexion, la fenêtre principale ainsi que le l'UC Détails. La Vue interagit avec le Modele au travers du Manager afin de d'afficher de récolter les données pour le MasterDetail. La Vue est aussi liée à la banque d'Images ainsi que Video-BandeAnnonces afin que chaque série ai sa propre miniature et bande-annonce.



Model :

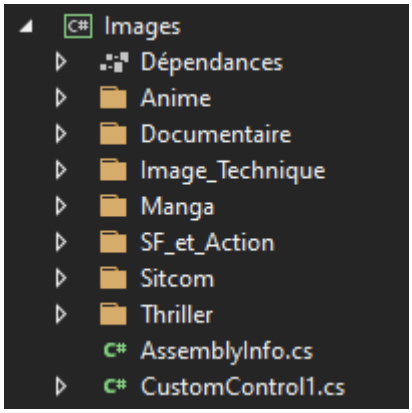


Le package Modele contient tout le code de notre application, c'est-à-dire les classes qui permettent de faire fonctionner notre projet. Nous avons décidé de séparer le code de l'application de la Vue et de la Persistance pour éviter les conflits et bugs mais surtout pour quand on effectue des modifications sur les classes et leurs méthodes, le changement n'impacte pas directement la Vue. De plus, la séparation de la Vue et du Model nous permet d'empêcher l'utilisateur d'accéder au code de l'app et de le modifier puisqu'il ne peut interagir qu'avec la clase Manager au travers de la Vue. Enfin, nous avons décidé d'implémenter la persistance dans Model mais de la définir dans le package Data afin que le model ne soit pas lié de manière directe à la persistance mais plutôt de manière indirecte.



Data : Le Data a été écrit en dehors du Modele afin que si les classes de Data sont modifiés, cela n'aura pas spécialement de l'influence sur le Modele. Eventuellement, si l'on veut améliorer les méthodes de sauvegarde et de chargement ou mettre en place un nouveau type de persistance comme une Base de données pour notre application il suffira uniquement de les écrire et de mettre à jour dans le code du Modele le type de persistance à utiliser avec ces dernières. Le Data est uniquement lié au Modele et ne touche en aucun cas aux autres package.

Images : Le package Images recense les miniatures utilisées pour les séries, ces dernières sont organisées par catégories. Nous avons décidé de les séparer des autres packages afin ce qui permet un accès plus pratique à la Vue et au Modèle à la banque d'Images. Le Modele ne contient uniquement que des références utilisées dans la Vue pour accéder aux Images.



Video-BandeAnnonce : Le package Video-BandeAnnonce contient lui aussi uniquement des données qui sont des vidéos .mp4 destinées à la Vue et référencé par le Modele, ces dernières sont triées par catégorie.

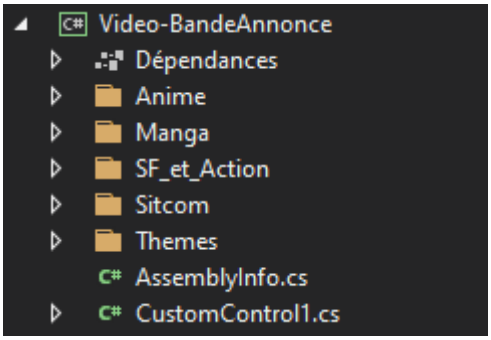


Diagramme de séquence :