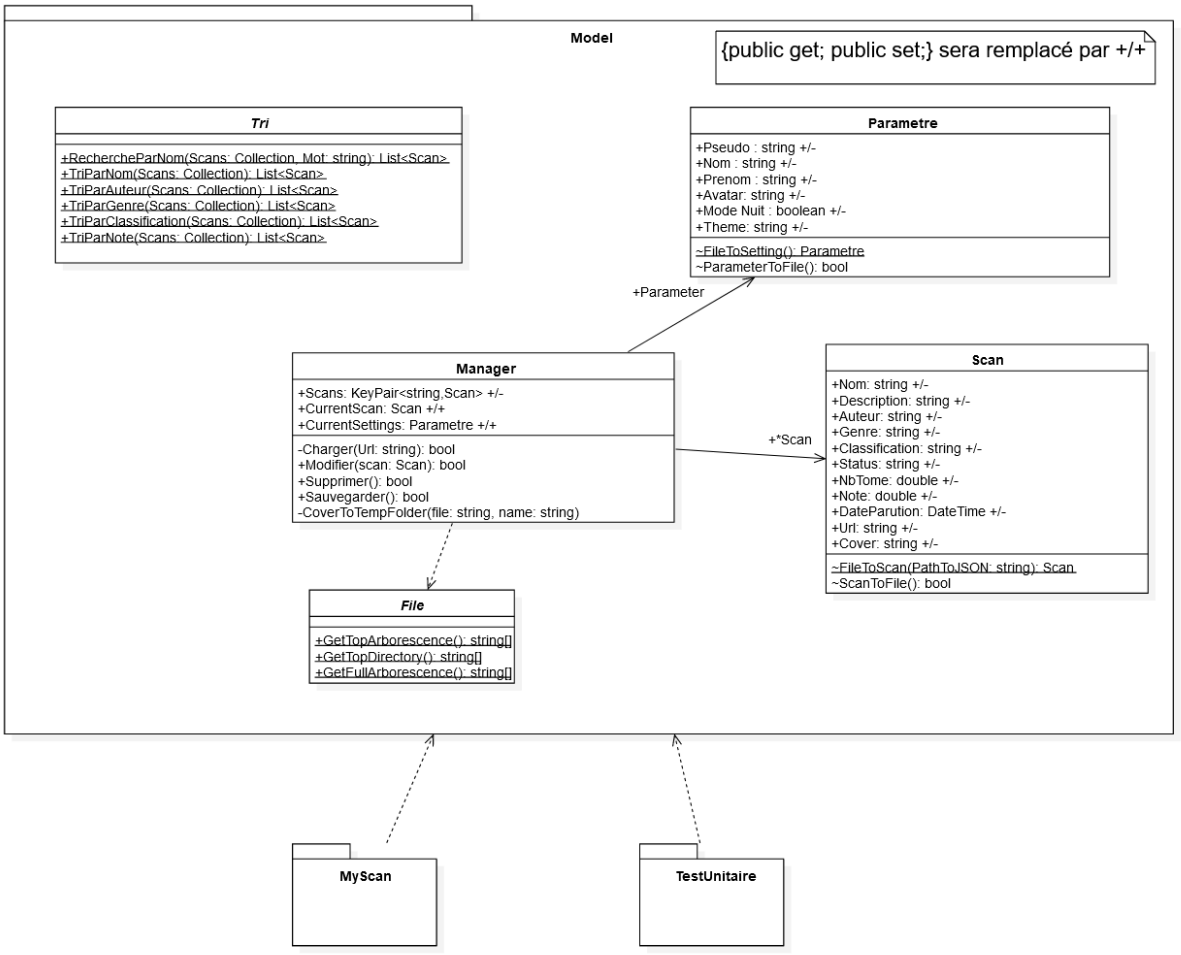
Architecture – MyScan



Patron de Conception

➔ Facade

Le Patron de Conception « Facade », permet de fournir une interface simple (Manager) du sous-système (Model). Il permet ainsi de rendre plus lisible et plus facile l'utilisation de la bibliothèque de classe. Il permet aussi d'encapsuler les différente classes en maitrisant les accès pour éviter une utilisation non prévue. Cela réduit aussi les dépendances ce qui offre une maintenance et une évolution plus simple.

Ici c'est manager la facade des autres classes de Model. Il permet ainsi de maîtriser les utilisations des classes.

Packages NuGet

MyScan

➔ MahApps.Metro et MahApps.IconPacks

Model

➔ Newtonsoft.Json

Description Diagramme de Classes

Model

➔ Classe Scan

Cette classe a pour but de stocker les différente information d'un Scan, donc tout ce que l'on pourrait attendre d'une œuvre littéraire (Genre, Description, Status…) mais aussi des informations telle que le chemin du dossier et de l'image de couverture (Cover). Le private set empêche quiconque d'attribuer des valeurs directement.

Méthodes

La classe possède deux constructeurs :

```
public Scan(string nom, double nbTome, string url, string cover)
```

Le premier permet d'initialiser un scan sans informations (Genre, Description…), par conséquent les autres propriété sont null.

```
public Scan(string nom, string? description, string? auteur, string? genre, string? classification, string? status, double nbTome, double? note, DateTime? dateParution, string? url, string? cover) : this(nom, nbTome, url, cover)
```

Le deuxième constructeur permet d'initialiser un Scan complet, il fait appel au premier constructeur.

```
Internal static Scan? FileToScan (string PathToScan)
internal bool ScanToFile (Scan scan,string PathToSave)
```

Elle permette la conversion d'un Scan en fichier et inversement.
L'un renvoie le Scan, l'autre renvoie si le fichier a été créé.

➔ Classe Paramètre

Initialement, il était prévu d'y intégrer un système d'utilisateur, elle sert maintenant uniquement a enregistré les paramètres et afficher quelques informations à propos de l'utilisateur (qui est unique).

Méthode

De même que Scan, paramètres possède deux constructeur, L'un ne nécessite pas de paramètre et met des valeurs par défaut et l'autre initialise toute les valeurs ;

```
Internal static Parametre? FileToParameter()
internal bool ParameterToFile (Parameter parameter, string PathToSave)
```

Permettent de faire exactement la même chose que la conversion des Scan mais cette fois-ci pour les paramètres.

➔ Classe File

Classe de méthode, elle ne sert qu'à grouper des méthodes donc elle est abstraite, de plus pour que les méthodes soient appelées sans instantation les méthodes sont static.

➔ Classe Tri

De même que la classe File, c'est une classe qui regroupe des méthodes.
Cette classe est uniquement appelée depuis la vue, pour trier une collection de scan selon certains critères (LINQ) et la renvoie.

➔ Classe Manager

C'est la classe qui fait le lien entre le Model et la vue.

Propriété

```
public Dictionary<string,Scan> ScanDictionnary {get ;private set}
```

Cette propriété contient tous les scans, elle est accessible à tout le monde mais seul manager peut la modifier.

```
public Scan CurrentScan { get; set; }
```

Cette propriété contient le Scan qui est sélectionné dans la vue.
Elle facilite l'accès du scan depuis les autres classes. Tout le monde peut l'avoir et la modifier.

```
public Parameter CurrentParameter { get; set; }
```

De même que CurrentScan, elle permet de faciliter l'accès des paramètres depuis les autres classes.

Méthodes

```
private bool Charger(string PathToScans)
```
Cette méthode est de loin la plus importante car elle a pour tâche d'analyser le dossier (entré en paramètre) et générer des Scan à partir des sous dossiers. Ces scans sont ajoutés au dictionnaire.
Pendant l'analyse du dossier le nom du scan est le nom par défaut du dossier, le nombre de dossier correspond au nombre de tomes sauf si le scan possède un fichier contenant les informations (JSON).

De plus il charge aussi les paramètres, par défaut s'il n'y a pas encore de fichier, sinon les paramètre sont restauré.
Il renvoie true si le fichier existe, sinon false.

```
private string? CoverToTempFolder(string file, string name)
```
Cette méthode est une sous méthode de la méthode Charger.
Elle est en charge de copier la Cover (si elle existe) dans un dossier de l'application.
Cela a été nécessaire car lors de la suppression d'un scan, la cover est utilisé par la vue.
Si la cover n'existe pas ou n'a pas pu être copier, alors il renvoie null.
Sinon il renvoie le nouveau chemin de la cover.

```
public bool Modifier(Scan scan)
```
Première méthode public. Elle permet de modifier un Scan dans le dictionnaire.
Elle supprime le Scan dans du dictionnaire, puis ajoute le nouveau et assigne le nouveau scan au CurrentScan.
Elle renvoie true si le scan a bien été supprimé sinon false.

```
public bool Supprimer()
```
Permet de supprimer le scan actuel (CurrentScan) dans le dictionnaire et le dossier.
Il renvoie false si la suppression du dossier ou du scan dans le dictionnaire échouent, sinon renvoie true.

Fonctionnement de l'application au démarrage (non détaillé)

App.xaml.cs

C'est lui qui est appelé lors du lancement de l'application, il instancie le Manager en propriété, permettant ainsi un accès global.

MainWindow

MainWindow est la fenêtre, elle est en charge de la navigation entre les pages.

C'est elle qui contient le menu hamburger menu et qui gère la navigation grâce à celui-ci.

ModelVue contient les méthodes pour générer (MenuItem) et qui génére (ShellViewModel) les élément de l'hamburger menu.

Et Navigation gère la navigation entre les pages.

Lors du lancement, elle appelle une page par défaut qui est Accueil.xaml.

Lorsqu'une page est appelée, elle est affichée (xaml) et son contenu et interaction sont géré par le code behind (cs).