C# - Patron de conception

L'architecture de la solution s'articule autour de 6 projets :

- **MyApp** : est notre projet WPF, gérant l'affichage de la vue et les évènements, ne contenant aucune instance du modèle ce qui permet une meilleure maintenabilité en passant par la façade (BackEnd). Le projet est lui-même découpé en 4 dossiers : *audio*, *UserControls* et *Windows*.
  Les dossiers *audio* et *images* nous permettent de stocker les données de base de l'application, à savoir quelques fichiers mp3 et des images pour faire une **Stub**.
  UserControls contient les Contrôles Utilisateurs de notre projet et *Windows* les Fenêtres WPF, cela nous permet de vider la MainWindow et découper le projet en plusieurs parties et limiter les responsabilités de chaque vue.

- **Biblio** est une bibliothèque de classe et aussi notre modèle qui gère et permet d'instancier les données s'affichant dans la vue.
  L'idée est d'avoir deux classes principales (User et Music), implémentant une interface chacune définissant un pattern de propriétés et de méthodes pour la classe, cette classe étant non instanciable directement (Internal), elles obligent à utiliser leur Créateurs (UserMaker et MusicMaker) afin de respecter des conditions d'instanciation (via des exceptions) et encapsuler ces classes. La classe Player peut ensuite manager les Musics, les Users grâce aux méthodes de chaque classe (Lecture/Pause, Connexion/Déconnexion, etc ...).
  Les UserControls/Windows ont accès à ces données via la façade ce qui permet une encapsulation entière du modèle.
  Une Music contient donc un certain nombre d'éléments qualificatifs d'une musique unique et également des Comments (des avis utilisateurs) propres à celle-ci par exemple. Une Playlist va contenir une ou plusieurs Music pour créer une liste de lecture. Un User va être composé de plusieurs informations le concernant, mais aussi d'une Playlist lui étant propre. Enfin une UserDB contient tous les Users inscrits.
  Un Player hérite de MediaElement afin de lui donner un comportement propre à son usage dans l'application (uniquement pour lire des musiques) ce qui le rend plus simple d'utilisation.
  Au niveau de la persistance, la classe Serialize, dont Playlist et UserDB héritent, permet d'avoir un objet commun de sérialisation, ce qui simplifie les implémentations de stratégie de sérialisation. Data nous permet de n'avoir qu'une seule classe à modifier si

nous décidons de changer de méthode de sérialisation.

- **BackEnd** est une bibliothèque de classe faisant office de façade du projet, elle permet d'instancier les classes du modèle et permettre à la vue d'y accéder mais sans que la vue n'ait à les stocker directement. Elle permet grâce aux éléments statiques une meilleure accessibilité des données dans le Visual Tree entier.

- **MainTest** est une Application Console permettant d'effectuer des tests unitaires et fonctionnelles de notre solution en Application Console.

- **Stub** est une bibliothèque de classe qui permet d'avoir un panel de données de base pour l'application lorsque l'utilisateur la lance pour la première fois.
  Elle nous sert également de données de référence pour nos tests.

- **Resources** est une bibliothèque de classe qui permet de stocker les images dans une dll accessible simplement via des Packs URLs dans le XAML comme dans la Stub, ce qui simplifie le chargement d'images et permet le déploiement de l'application sans contrainte de chemin d'accès.