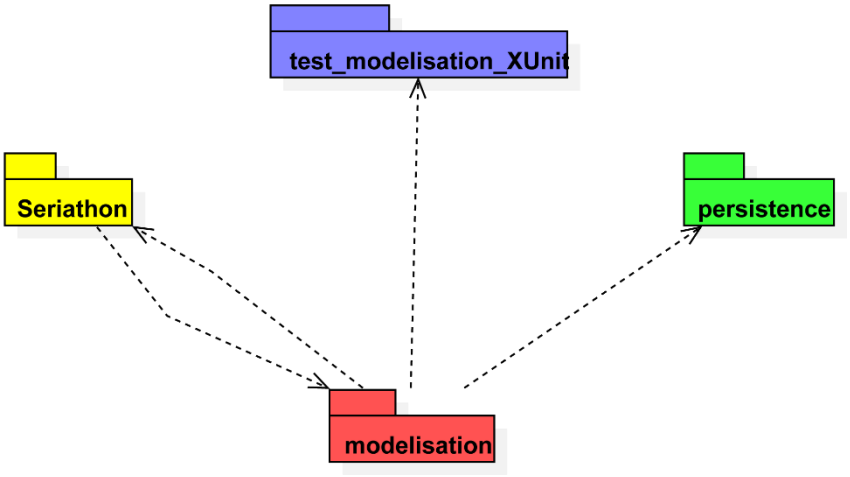
FOUCRAS Baptiste THEME Julien	**Documentation Finale**	*1AG1H*

Dans la documentation suivante, nous allons d'abord vous présenter l'architecture de notre application, puis les considérations ergonomiques et l'accessibilité réellement mise en place, que nous avions promis lors de la première partie de la documentation.

Commençons par l'architecture des projets :



Il existe 4 packages, modélisation correspond au C# pur, Seriathon correspond à la partie XAML, la persistance (Stub, DataContract…), ainsi que les tests unitaires de la partie modélisation (nommé test_modelisation_XUnit).
Bien entendu, le package Seriathon et modélisation dépende l'un de l'autre, étant donné le binding pouvant tout autant donner des informations au front end (XAML), tout comme l'utilisateur peut saisir des informations depuis le front-end vers le back-end (c#).

Le découpage global c'est fait selon l'utilité. Un package pour le front-end, un pour le back-end, et un pour la persistance.

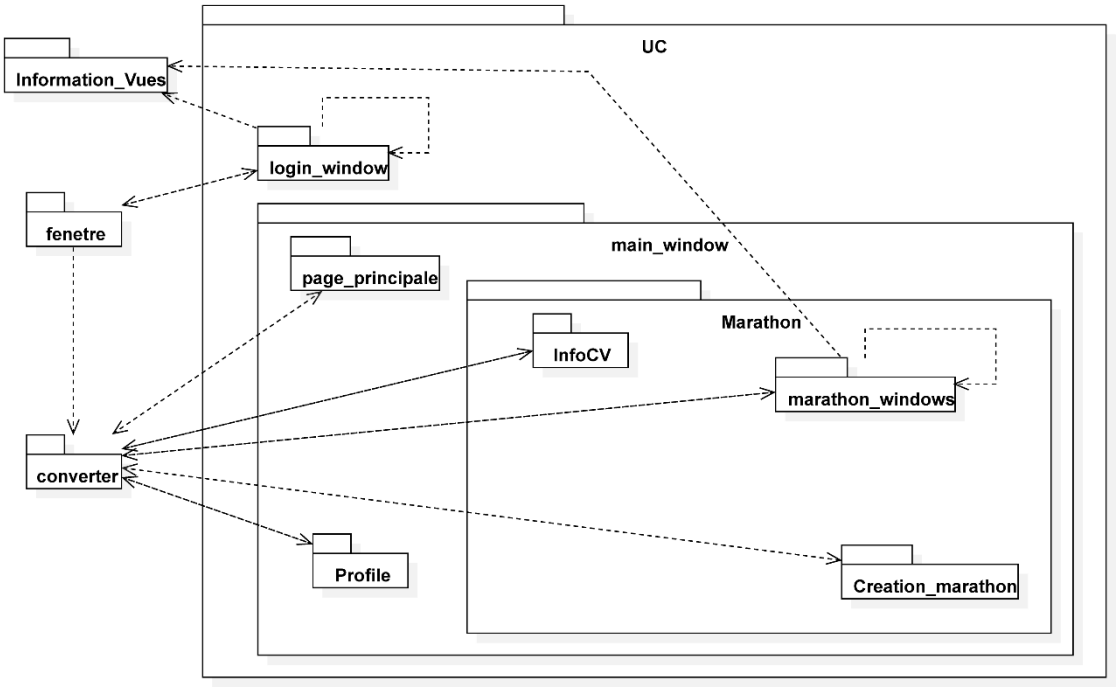
Le C# utilise le package persistance, pour obtenir les informations de bases au lancement de l'application (venant du stub ou d'un DataContract), et pour enregistrer les ajouts, suppressions et modifications sur ses informations.

Le package « test_modelisation_XUnit » importe la plupart des classes du package modélisation, afin de pouvoir tester ces classes.

Les packages et classes présentes dans ces trois packages sont explicité dans les schémas suivants.

Nous allons dans l'ordre vous détailler le projet de XAML (Seriathon), puis la modélisation, et enfin la persistance.

Le package de tests unitaires xUnit ne sera pas détaillé, étant donné qu'il contient seulement des classes de tests pour la plupart des classes de la partie modélisation.



Information_vue :

Information_vue contient tous les pop-ups qui seront utilisé dans le projet.

Fenetre :

Fenetre contient les pages principales de l'application, la première est la page qui apparaîtra lors de l'ouverture de l'application, elle est appelée Sériathon, elle est utilisée avant la connexion ou l'inscription d'un utilisateur. La seconde fenetre est utilisé une fois que l'utilisateur se sera identifié ou se sera inscrit, elle permet la navigation entre le marathon, le profil de l'utilisateur et la page d'accueil.

Converter :

Converter contient les navigateurs ainsi que les converters utilisé pour la navigation mais aussi le converter d'image.

UC :

UC contient tous les user contrôles utilisés dans le projet. Il contient lui-même plusieurs dossiers tel que login_window, main_windows.

→Login_window contient les user contrôles de connexion et d'inscription.

→ Main_window contient profile, page_principale et Marahon.

- ➔ Profile inclue toutes les user contrôles relatifs au profil.
- ➔ Page_principale contient la page principale utilisé dans Sériathon.xaml et dans Accueil.xaml mais aussi la bar de navigation.

- ➔ Marathon qui englobe tous les user contrôles relatifs au marathon : InfoCV, marathon_windows et Creation_marathon.
    - InfoCV contient les pages d'informations sur les films, séries (la série en elle-même, les saisons et les épisodes) et animes (l'anime en lui-même, les saisons ainsi que les épisodes).
    - Marathon_windows contient la page principale du marathon, avec tous les films, séries et animes que l'utilisateur doit regarder, répartie en jour.
    - Creation_marathon contient toutes les user contrôles qui permettent de créer un marathon.

Interactions :

Fenetre :

Seriathon.xaml utilise login_windows pour afficher connexion.xaml et inscription.xaml.

Accueil.xaml utilise converter pour afficher la page principale ainsi que la bar de navigation (page_principale.

Login_window :

connexion.xaml et inscription.xaml utilise fenetre pour passer de Sériathon.xaml à Accueil.xaml, qui est la page qui apparait après la connexion. Connexion.xaml peut utiliser Informations_Vues pour afficher la fenêtre pop-up qui dit que le mot de passe ou l'identifiant n'est pas valable. Login_window s'utilise car connexion peut appeler inscription et inversement.

Page_principale :

Page_principale utilise converter pour afficher la page d'accueil (page_principale), la page des profils (Profile) mais aussi la page des marathon (marathon_windows) si un marathon a déjà été créer ou la page de création d'un marathon (Creation_marathon) si aucun marathon n'a été créé.

InfoCV :

InfoCV utilise converter pour afficher afficher d'autre infos sur les contenues vidéoludiques, par exemple depuis une série on peut aller à une saison puis accéder à un épisode en particulié.

Marathon_windows :

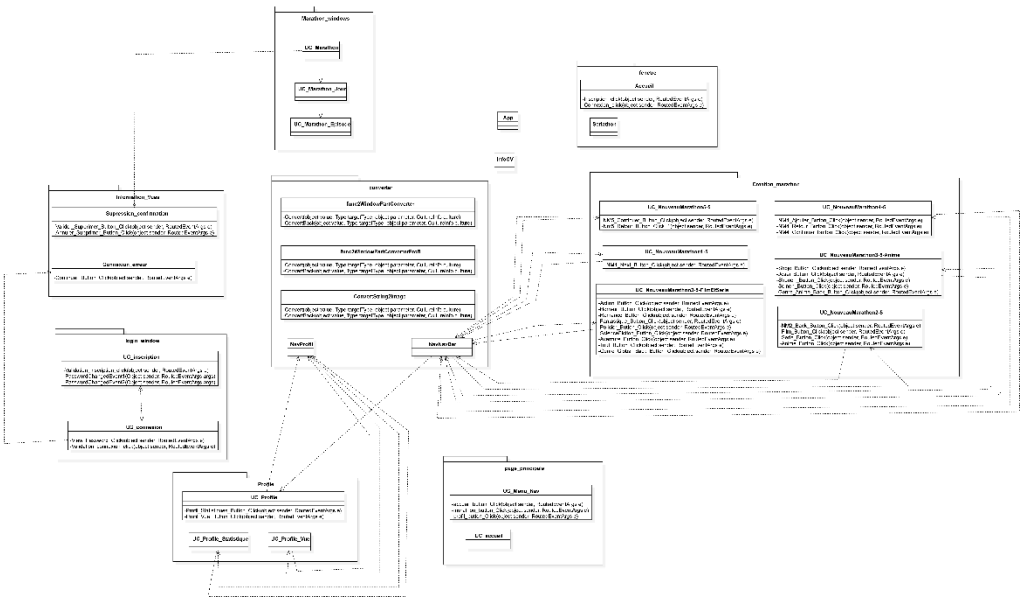
Marathon_windows utilise lui-même, car UC_Marathon utilise UC_Marathon_Jour qui utilise UC_Marathon_Episode. Il utilise Information_Vues pour afficher la pop-up de confirmation de suppression du marathon. Il utilise également converter pour afficher les informations des conteneus vidéoludiques (InfoCV).

Création_marathon :

Creation_marathon utilise converter pour pouvoir passer d'étape en étape de création du marathon donc dans le même dossier mais aussi d'afficher à la fin de création du marathon la page du marathon (marathon_windows).

Profile :

Profile utilise converter pour afficher la page des contenues vidéoludiques vue mais aussi la page des statistiques.



Marathon_window :

- UC_Marathon : contient le marathon de l'utilisateur, avec chaque jours, puis dans chaque chaque jours, il contient tout les épisodes/films à regarder.
- UC_Marathon_Jour : permet de regrouper plusieurs épisodes/films à regarder ce jour ci.
- UC_Marathon_Episode : permet de créer un épisode/film a regarder.

Creation_marathon :

- UC_NouveauMarathon1-5 : on entre le nom du marathon, la période que durera le marathon ainsi que le nombre d'heure par jours.
- UC_NouveauMarathon2-5 : permet de sélectionner quel type de contenue vidéoludique l'utilisateur a envie de regarder.
- UC_NouveauMarathon3-5-Anime : permet de sélectionner quel genre d'anime l'utilisateur a envie de regarder.
- UC_NouveauMarathon3-5-FilmEtSerie : permet de sélectionner quel genre de film/série l'utilisateur a envie de regarder.
- UC_NouveauMarathon4-5 : permet de continuer ou de rajouter un type et/ou genre.
- UC_NouveauMarathon5-5 : il récapitule tout infos du marathon, puis permet de finaliser.

InfoCV :

- InfoCV : il permet d'avoir les informations sur tel film/série/anime/saison ou épisode.

Profile :

- UC_Profile : permet de changer entre l'onglet Vue et statistique.
- UC_Profile_Vue : permet de voir tout les contenues vidéoludiques déjà regarder.
- UC_Profile_Statistique : permet de regarder toutes les statistiques (nombre de films, série, anime vue, les genres et réalisateur favori.

Page_principale :

- UC_accueil : est le fond d'écran de l'accueil.
- UC_Menu_Nav : permet de naviguer entre UC_accueil, UC_marathon et UC_Profile.

Login_window :

- Inscription : permet de s'inscrire.
- Connexion : permet de se connecter.

Information_Vues :

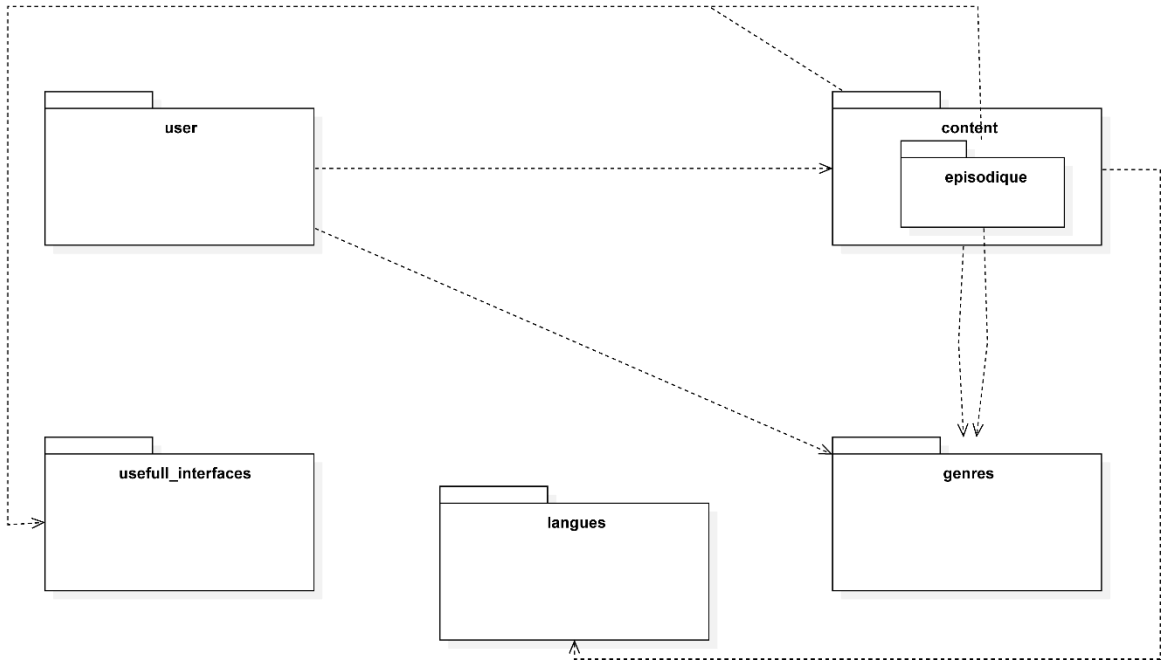
- Supression_confirmation : permet de valider ou non la suppression d'un marathon.
- Connexion_erreur : permet d'indiquer à l'utilisateur qu'une erreur s'est produite lors de la connexion (mot de passe ou adresse e-mail incorrecte).

Fenetre :

- Accueil : affiche UC_accueil, puis permet d'afficher UC_Inscription mais aussi UC_Connexion.
- Sériathon : est la fenêtre utilisé après la connexion de l'utilisateur est contient UC_Menu_Nav qui permet de naviguer entre UC_accueil, UC_marathon et UC_Profile.

Converter :

- func2WindowPartConverter : permet de convertir un func en user contrôle dans le content contrôle de UC_Profile.
- func2WindowPartConverterProfil : permet de convertir un func en user contrôle dans le content contrôle de Sériathon.
- ConvertString2Image : permet de convertir un string (nom d'une image) en un chemin vers celle-ci.
- NavProfil : permet la navigation dans le content contrôle de UC_Profile.
- NavNavBar : permet la navigation dans le content contrôle de Sériathon.



Le package user contient des classes liées directement à l'utilisateur, tel la classe Utilisateur, et la classe Marathon, lié intimement à un Utilisateur.

Le package content rassemble tout le contenu de remplissage de la plateforme, donc tout les contenus Vidéoludiques (films, séries animes…). Le sous-package episodique représente tous les contenus vidéoludiques sous forme d'épisodes (séries et animes), ainsi que leur composants (épisodes et saisons).

Le package usefull_interfaces, langues, et genres sont tous trois des packages de services, ils servent aux classes des packages user et content. Ils à la racine du projet de modélisation, dans un souci logique. En effet ce sont des ressources que nous pourrions utiliser dans d'autre contextes que content ou user, expliquant ces namespaces.

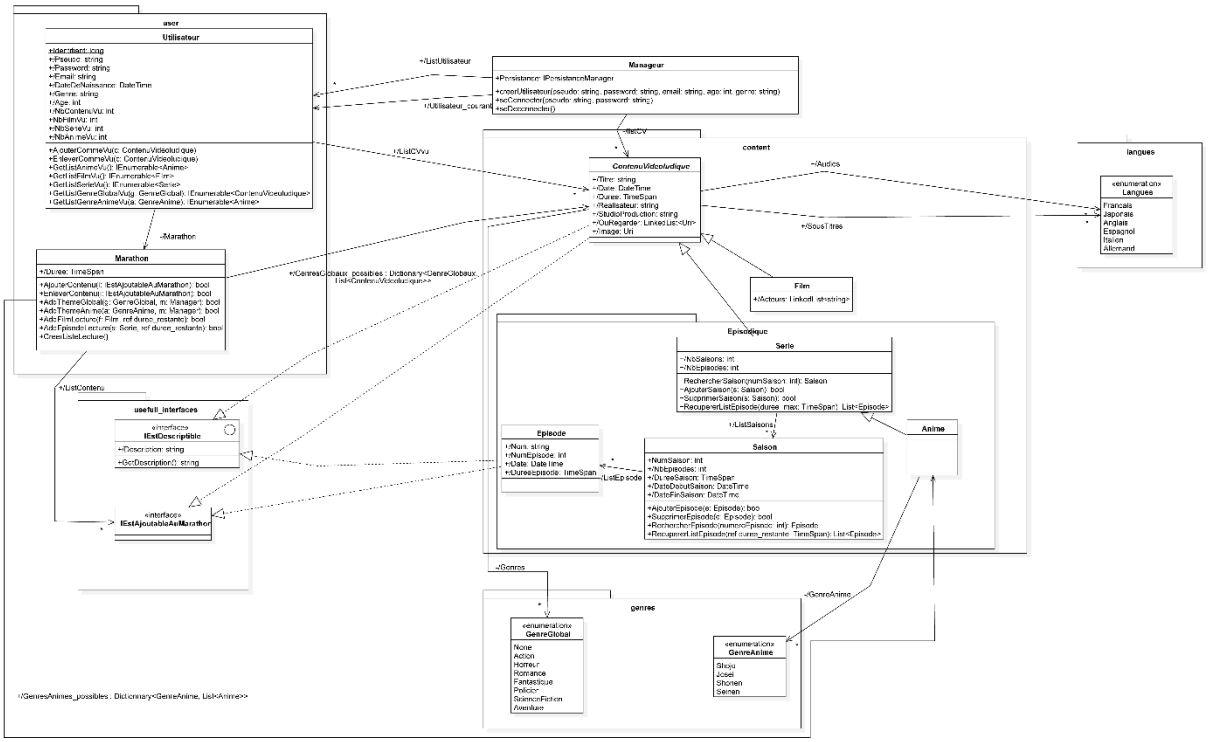
usefull_interfaces regroupe toutes les interfaces pouvant être utilisées par toutes les classes, genres regroupe deux enums pour représenter les genres des contenus vidéoludiques (action, shonen…) et langues regroupe les langues pour les contenus vidéoludiques.

Dans usefull_interfaces, une interface IEstDescriptible est présente, et est importé par ContenuVideoludique et Episode qui l'implémentent, car sont descriptibles. Ces deux classes implémentent aussi IEstAjoutableAuMarathon, pour pouvoir les ajouter à la liste de lecture d'un marathon. usefull-interfaces permet ainsi de regrouper toutes interfaces susceptibles d'être utilisée dans tout les packages.

Le package genres est importé par content, pour décrire les genres Globaux de ContenuVideoludique, mais est aussi importé par episodique, pour décrire les genres d'animes de la classe Anime.

Le package langues contient un enum pour des langages, qui est importé par content, pour spécifier les langues audios et des sous titres des contenus audiovisuels.

Le package user utilise le package content, pour la piste de lecture d'un marathon ou bien les contenus vidéoludiques déjà vu par l'utilisateur



Le Manager possède un attribut de type IPersistanceManager, qui lui permet de charger les données de manière statique (Stub) ou bien depuis des fichiers (DataContract). Il possède aussi tous les Utilisateurs, les ContenusVideoludique en liste. Il possède aussi bien entendu l'utilisateur courant, pour savoir ce que l'on doit afficher.

La classe Utilisateur se trouve dans le package user, ainsi que le marathon, car un marathon est intimement lié à un utilisateur. Tout les ContenusVideoludiques déjà vu par l'utilisateur sont stockés dans ListCVvu, et Linq permet de connaitre le nombre de Contenu, de Film, de Serie (Anime exclu), et d'Anime déjà vu. Linq permet toujours de récupérer la liste filtrée selon la classe, mais aussi selon le fait que le contenu possède ou non un genre global ou d'anime. L'élément renvoyé sera un IEnumerable, renvoyé directement par Linq. Cela permet de construire ce qui est souhaité à partir de ceci (List, LinkedList…).

Marathon est possédé en un unique exemplaire par Utilisateur. Celui-ci possède une durée maximale de sa liste de lecture ListContenu. On peut ajouter et supprimer des IEstAjoutableAuMarathon (cf partie IEstAjoutableAuMarathon). On peut ajouter des thèmes aux dictionnaires, possédant en clé un genre, et en value la liste de tout les ContenuVideoludique. Cela permet de simplifier grandement l'algorithmie de création de la playlist de lecture, en permettant notamment d'ajouter à la playlist plus facilement un contenu vidéoludique dont le thème est déterminé à l'avance. De plus, cela permet d'enlever les contenus vidéoludiques déjà ajouté, afin d'éviter d'ajouter deux fois le même contenu.
La plupart des fonctions sont explicités plus en précision dans le diagramme de séquence, présentant la création de la liste de lecture du Marathon.

AddEpisodeLecture permet de répondre au souci de ne pas ajouter toute la playlist, mais seulement au maximum 3 épisodes pour chaque

Il existe 3 enums, deux pour les genres des contenus vidéoludiques, et une pour les langues.

Comme dis précédemment, ils sont dans des packages à la racine du projet de modélisation, pour être logique en réutilisation.

GenreGlobal possède un type none, car les Animes (possédant un genre global), peuvent ne pas en avoir en réalité, et c'est plus rapide que d'écrire « GenreAnime? Genre = null; ».

GenreAnime ne possède pas de none, car un anime possède forcément un genre d'animes.

IEstDescriptible est une interface, avec pour contrat le fait de posséder un string pour décrire la classe, ainsi qu'une méthode pour get la description.

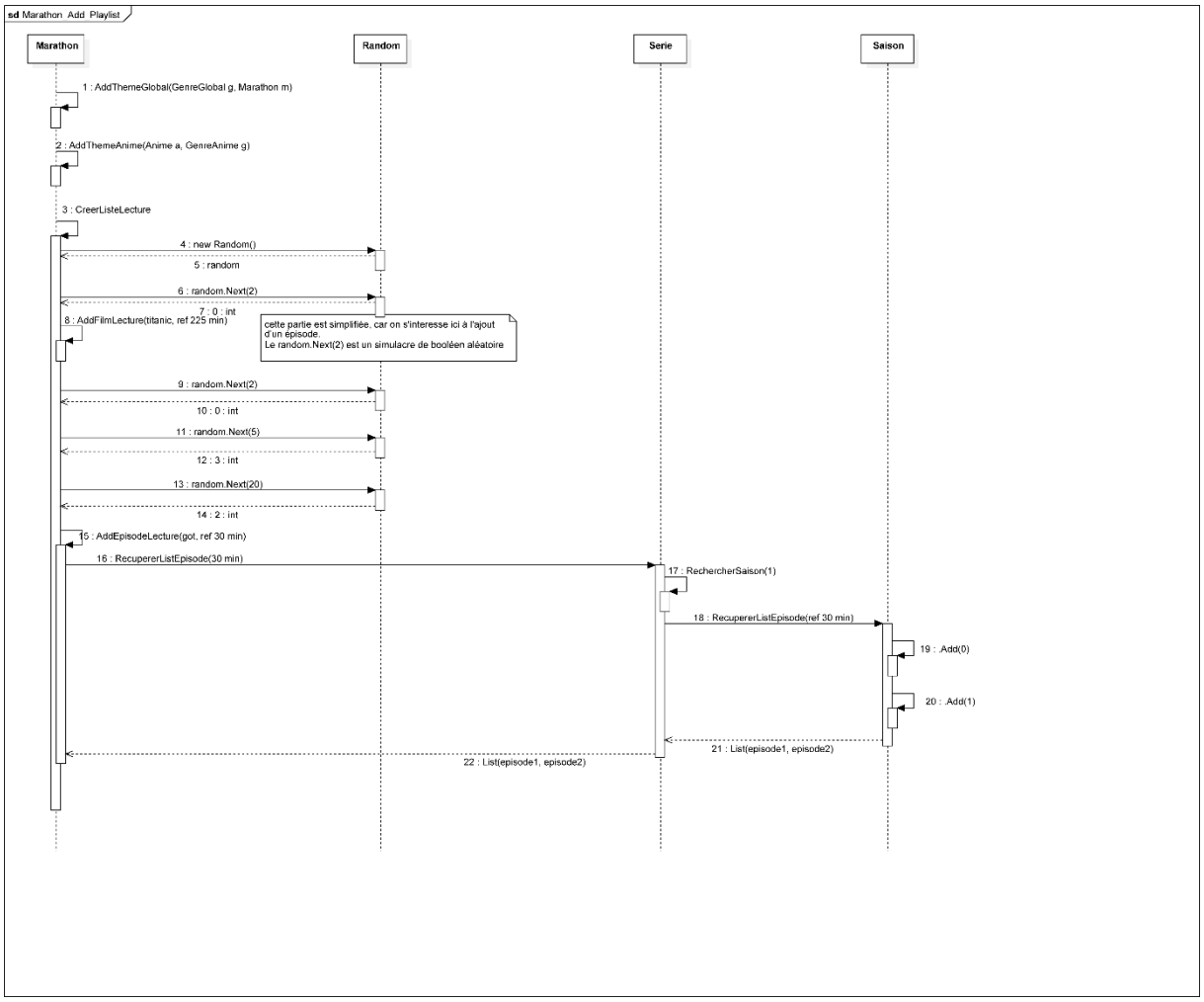
IEstAjoutableAuMarathon permet de répondre au souci d'ajouter des ContenusVideoludique, et des Episodes dans la liste de lecture ListContenu des Marathons. L'interface est préférée au fait de construire a partir des 3 épisodes max choisis au hasard une nouvelle série, car plus léger en terme d'instanciation (pas d'intenciation d'une série) ; et aussi car plus simple.

Tous les films, séries et animes dérivent, directement ou non, de la classe abstraite ContenuVideoludique. Cela permet déjà de compacter le code, car le film et la série partages de nombreuses caractéristiques communes (titre, réalisateur…). De plus, cela permet de stocker dans une unique liste tout les contenus vidéoludiques, plutôt qu'une liste pour les films et une pour les séries. Cela permet donc de manipuler films et séries de manières communes. Anime dérive de série, car un anime est globalement une série, avec un genre d'anime en plus.

Film ajoute juste une liste d'acteurs, et ensuite Serie possèdes une liste de saisons, possédant eux même une liste d'épisodes. La saison ne possède pas réellement de date de début ou de fin, ni de durée, Linq est utilisée afin de déterminer la date minimale et maximale, ainsi que la durée totale des épisodes de la saison.

Anime ne possède rien de nouveau, c'est juste une série, avec une liste de GenreAnime en plus.

Nous détaillons ici la complétion de la liste de lecture d'un marathon nouvellement créé :



Le diagramme de séquence présente donc l'algorithmie globale de la création des thèmes (genres possibles) par des dictionnaires, jusqu'à la création de la playlist random de contenu et épisodes.

Afin de faciliter la création d'une playlist de lecture, deux dictionnaires ont été créé, liant un genre, avec tous les contenus vidéoludiques

Etape 1 :

On commence par ajouter un thème possible au dictionnaire de thèmes globaux possibles. Cela a pour effet, si jamais la clé n'est pas déjà présente, de l'ajouter, avec une liste de tous les ContenuVidéoludiques possédant ce genreGlobal dans leur liste de genres globaux.

Etape 2 :

Dans ce cas, le fonctionnement est le même, mais l'on veut cette fois ajouter un thème d'animé, ainsi que la liste des animes possédant ce genre, dans le dictionnaire de thèmes d'animes possibles.

Etape 3 :

CréerListeLecture correspond au fait d'ajouter dans la liste des contenus composant le marathon un maximum de contenu, en respectant les thèmes possibles, et aussi le temps maximum.

Etape 4&5 :

On construit un nouveau Random, de référence random. Il servira de choisir de façon aléatoire les contenus à ajouter a la playlist.

Etape 6&7 :

On génère un int aléatoire entre 0 et 1, un semblant de booléen, afin de choisir si l'on ajoute un contenu provenant des thèmes globaux possibles, ou bien plutôt des épisodes provenant des animes possibles ayant un des thèmes d'animes possibles. Ici on renvoit 0, on prend donc un contenu vidéoludique de thème global possible.

Etape 8 :

On a simplifié ici, mais entre l'étape 7 et 8, se trouve un retirage d'entier aléatoire, pour choisir une des listes en value avec un thème global en key, et un second pour tirer un contenu aléatoire dans la liste choisie avec le random précédent. On tire ici aléatoirement un ContenuVidéoludique de type Film, et on l'ajoute à la liste de lecture du marathon.

Etape 9&10 :

On retire un int entre 0 et 1 (cf etape 6&7). C'est un simulacre de booleen. Ici on tire 0, on choisit donc un contenu parmi les values du dictionnaire des genres globaux disponibles.

Etape 11&12 :

On cherche un nombre aléatoire entre 0 et le nombre de paires clé-value dans le dictionnaire de genres globaux possibles. Ici on a 3 paires, donc on choisit entre 0 et 4 (indexage par Linq). Ainsi, on tire ici le 3eme élément « en index » du dictionnaire. On récupère la value à l'index correspondant, étant une liste de contenuVideoludique

Etape 13&14 :

On tire un int aléatoire entre 0 et le nombres de nodes contenu dans la list récupéré à l'etape précédente, pour choisir un contenu à un index alétoire tiré juste avant.

Etape 15 :

On a tiré une série. On ne veut pas ajouter toute la série, mais seulement au maximum 3 épisodes, ou une durée se rapprochant le plus de celle passée en paramètres, la majorant de la durée d'un épisode au max

Etape 16 :

On appelle la fonction de serie, pour demander à récupérer une suite d'épisode, d'environ le temps passé en paramètre, ou bien de max 3 épisodes.

Etape 17 :

On commence par récupérer la référence de la première saison de GOT

Etape 18 :

On appelle la fonction de saison, pour demander à récupérer une suite d'épisode, d'environ le temps passé en paramètre, ou bien de max 3 épisodes, afin que Serie puisse le renvoyer à Marathon.

Etape 19 :

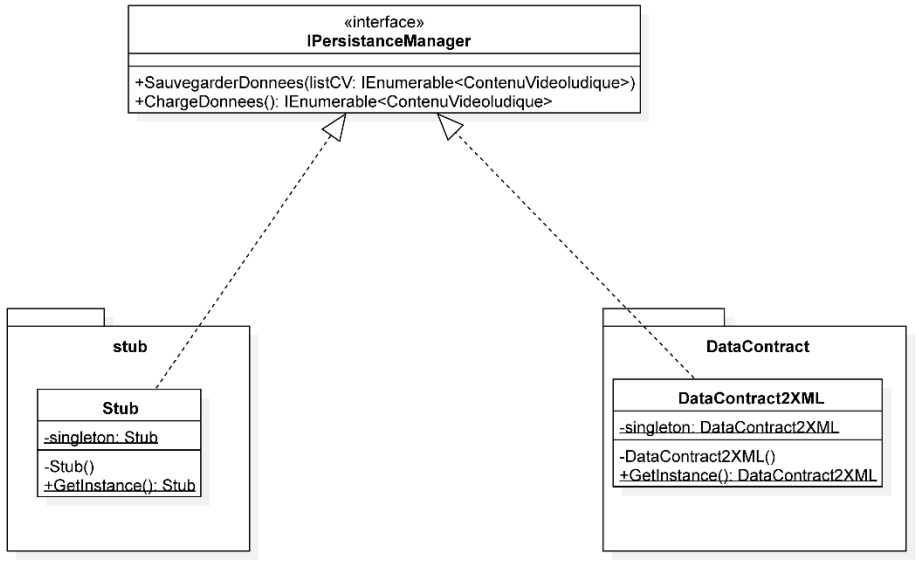
On ajoute un premier épisode, et on décrémente le temps passé en paramètre du temps de l'épisode

Etape 20 :

On ajoute un second épisode, le temps défini comme limite à l'étape 18 est soit réduit à 0, soit majorer de maximum la durée de l'épisode. On cesse donc d'ajouter.

Etape 21&22 :

On renvoi la liste composée des épisodes ajouté vers la méthode RecupererListEpisode de Saison, que l'on renvoi directement au marathon, pour qu'il puisse tout ajouter.



Toutes les classes de persistances possèdent les mêmes méthodes de chargements et de sauvegarde. Seul diffère l'algorithme derrière. Il est donc intéressant de mettre en place le patron de conception stratégie, avec l'interface IPersistanceManager. De plus cela permet plus facilement de prendre plus facilement une source de persistance plus facilement.

L'interface IPersistance, outre le fait qu'elle est importée par Manager, elle précise aussi les méthodes à avoir, SauvegarderDonnées prendra en paramètres toutes les listes de données (la liste des ContenusVideoludiques, et des Utilisateurs, afin de les enregistrer en XML.

ChargeDonnées renverra des IEnumerables de ContenusVideoludique et Utilisateurs, pour charger les données.

De plus, Stub et DataContract2XML exploite la stratégie du singleton, car il est inutile d'avoir plusieurs instances de ces classes, une est amplement suffisante. Ainsi, chacune de ces classes possèdes un attribut privé singleton, possédant la référence de l'instance de la classe (ou bien null si aucune instance n'existe). Le constructeur est dès lors privé, et c'est dans Getnstance que l'on appelle le constructeur si jamais le singleton est null, sinon il renvoie l'instance déjà existante.

La sauvegarde de données sur le stub ne sauvegardera rien, étant donné que le stub est un chargement de données en dur, statique.
DataContract2XML sauvegardera dans des fichiers XML, et chargera les données depuis les fichiers crées.

Ergonomie et Accessibilité :

Comme nous avons pu le dire précédemment, nous considérons l'ergonomie comme une part importante d'une application.

Nous avions ainsi promis que nous inclurons le minimum de boutons possible, ce qui est le.

Nous avions aussi parlé de l'affordance, nous avons donc respecté ce que nous avions dit, c'est-à-dire des couleurs explicite sur l'utilisation des boutons. Les boutons validés, et continuer sont en vert ; tandis que les boutons supprimer et retour sont en rouge.

Nous avons mis de base un thème noir car il permet une meilleure lisibilité et une fatigue des yeux minime. Les couleurs des textes sont claires (blanc), ce qui permet une meilleure visibilité. Dans les boutons à côté du texte des icones ont été mis, une croix pour supprimer, un check pour valider et une flèche pour retour.

La colorimétrie de toute nos pages ont été vérifiées pour qu'elles soient adapté au maximum au utilisateurs daltoniens, le thème sombre avec ses nuances de gris aidant à cette fin.

La police Comic sans MS , comme dit précédemment, a été utilisé comme police principale de notre application. Elle permet une meilleure lisibilité avec une écriture aérée et une distinction avec les barres des p, q, d, et b, notamment utile pour les dyslexiques.