2015-16

HEIG-VD, section TIC

Auteurs : Baehler Simon,
Moret Jérôme,
Purro Jan,
Berney Léonard &
Roubaty Anthony

Date du rendu final du projet : 4 janvier 2016

Professeur : René Rentsch

[DOCUMENTATION FLAT5]

Rapport du projet de semestre nommé Flat5

Table des matières

1 Introduction

Le but de ce projet est de concevoir et de développer un logiciel fonctionnel de toute pièce, et par groupe devant être composé de 4 ou 5 personnes constitué aléatoirement. Cela permet de mieux comprendre le travail en équipe et l'importance d'avoir une bonne coopération de chaque collaborateur au sein d'un projet.

Pour la réalisation d'un tel projet, certains points ont dû être mis en place tels que :

- Une planification raisonnable du projet.
- Découpe efficace des différentes tâches.
- Transmettre et communiquer correctement les informations au sein du groupe.

1.1 Choix de projet

A la différence avec PRO, dans le cours PDG nous n'avons pas de restriction de langage. La seule restriction imposée c'est le fait de ne pas pouvoir faire de jeux, autrement nous avons carte blanche.

A partir de cela, et après un moment de réflexion une idée nous est venue. Nous voulons créer un programme permettant à deux, puit à plusieurs personnes de pouvoir regarder un film ou écouter de la musique « comme s'il était à côté ». C'est-à-dire, en ayant un système de synchronisation de média.

En poussant un peu le concept, nous sommes arrivés à une idée générale regroupant, justement, ce concept de synchronisation avec une bibliothèque de médias. Notre application proposera alors une gestion facilitée de divers médias (musiques, films, séries), tout en ajoutant une fonctionnalité de synchronisation. Cette dernière va justement permettre d'envoyer les commandes qui sont effectué sur un lecteur aux autres. Par exemple, il y a deux personnes qui regarde un film ensemble, et que l'une d'entre elle appuie sur « pause », alors cela mettra pause aussi chez l'autre personne.

En termes de technologie, les différents lecteurs (sons et vidéos) seront entièrement créés par nous. Nous allons coder notre application en Java avec une interface graphique en JavaFX.

Remarque : Les choix des outils utilisés sont expliqués plus en détails dans les chapitres de ce document.

1.2 Objectif de Flat 5

Proposé une interface facile à prendre en main pour gérer efficacement une collection de média audiovisuel. Tant en proposant la possibilité de partager son expérience en temps réel avec d'autres personnes.

1.3 Utilisation de l'application

Flat 5 s'adresse à toutes les personnes voulant gérer et partager lors médias.

Pour plus d'informations concernant l'utilisation de l'application, référez-vous au manuel d'utilisation.

2 Mise en place

Pour pouvoir réaliser tout projet, il est préférable de définir certains points et règles afin que le déroulement du projet se fasse de la manière la plus cadrée et optimale possible. Nous avons donc séparé les différentes tâches.

- Interface : il s'agit d'un des points centraux de notre application. Elle sera réalisé M. Baehler Simon.
- Lecteur audio : cela prend en compte la gestion des différents événements du lecteur (play, pause, etc…). Il sera réalisé par M. Moret Jérôme.
- Lecteur vidéo : Même chose que le lecteur audio seulement pour les films et les séries. Il sera réalisé par M. Berney Léonard.
- Synchronisation : la synchronisation fera appel à une partie réseau, et surtout la mise en place d'un protocole sommaire. Elle sera réalisée par M. Roubaty Anthony.
- Base de données : elle permettra de stocker diverses informations de configuration, et aussi de stocker les informations relatives à la bibliothèque de médias. Elle sera réalisée par M. Purro Jan
- API audio : elle permettra de récupérer diverses informations sur les morceaux de musique. Elle sera réalisée par M. Roubaty Anthony
- API vidéo : idem que pour l'API audio, mais adapté pour les films et les séries. Elle sera réalisée par M. Purro Jan.

Si deux personnes ont 2 tâches différentes à réaliser, cela vient du fait que la somme de travail varie d'une tâche à l'autre. Certaines tâches, comme la rédaction de la documentation, sont considérer comme une tâche commune, il est du devoir de la personne responsable d'une tâche de documenter son travail dans le rapport.

2.1 Gestion des fichiers

Pour pouvoir travailler le plus efficacement possible au cours de projet, nous avons utilisé la plateforme Github. Cette dernière nous permet de garder une trace de tous les changements, autant au niveau du code que du rapport.

Le répertoire Git étant public, il est accessible à cette adresse.

https://github.com/snup482/PDG_Project

3 Analyse

Au cours de ce chapitre, nous allons expliquer les différentes recherches qui ont été mené. Cela afin de mieux comprendre les choix finaux.

3.1 Fonctionnement de base

D'après le cadre défini du projet

3.2 Base de données

Dans notre application, la base de données a pour but de stocker certaines informations nécessaires au bon fonctionnement de celle-ci.

En effet, notre application permettant de chercher des informations concernant les différents fichiers de la collection de l'utilisateur, lorsque celle-ci ne sont pas présentes. Il serait inutilement lourd de rechercher ces informations à chaque fois, d'où l'intérêt de pouvoir stocker ces informations.

Notre application offre également la possibilité de synchroniser la lecture des médias avec une autre personne. Plutôt que de rentrer à chaque fois les informations (un nom et l'adresse IP) d'une autre personne, nous offrons également la possibilité de se rappeler des contacts. À nouveau il est nécessaire de pouvoir stocker ces informations.

L'utilisation d'une base de données paraît être une bonne solution pour stocker ces informations, particulièrement en ce qui concerne les médias qui peuvent contenir beaucoup d'informations.

Nous avons choisi SQLite comme moteur de base de données. Le principal argument en sa faveur étant qu'il ne requiert pas un serveur mais est directement intégré à l'application sous la forme d'un fichier unique. De plus il est accessible par le langage SQL, qui nous est bien connu.

Pour pouvoir interagir avec la base de données depuis notre application nous utilisons JDBC.

3.3 Collection

Le principe de notre application est de fournir un lecteur multimédia qui permette également d'afficher des informations sur les médias de l'utilisateur.

Nous avons donc décidé que l'utilisateur possèdera une « collection » qui contiendra les fichiers qu'il souhaite que notre application gère.

Comme nous souhaitions nous concentrer sur les aspects de lectures des médias, de récupération des données et de l'interface graphique, nous avons décidé que la gestion des fichiers serait relativement simple.

La collection de l'utilisateur n'est donc de fait, qu'un simple dossier dans lequel il dépose les fichiers audio et vidéo directement (pas dans des sous-dossiers). Les séries, sont des dossiers, dont le nom doit correspondre à la série, contenant les épisodes de cette série.

Notre application s'occupe principalement de lire ces fichiers et d'afficher leurs métadonnées. Il n'est donc pas vraiment possible de supprimer des fichiers, ou de les modifier depuis l'interface de notre application.

Du point de vue de l'utilisateur, le but est qu'il puisse indiquer où se situe sa collection et de pouvoir « scanner » celle-ci afin d'obtenir les informations concernant les fichiers qu'elle contient.

Une fois qu'un fichier a été ajouté à la base de données, il est ignoré lors des scans suivant.

Si un fichier est supprimé de la collection, il est également supprimé de la base de données.

3.4 API web

Les APIs web sont utilisées dans l'application pour retrouver les différentes informations sur les médias exportés. Il faudra en utiliser deux différentes, une pour les films, et une pour les musiques.

Concernant les films, le but était de fournir des informations au sujet des fichiers vidéo de la collection de l'utilisateur. Nous pensions utiliser les informations du site IMDb, qui fournit des informations quasi-exhaustives sur la plupart des films et séries existantes. Il s'agissait de trouver une API qui permette d'extraire ces informations.

Il existe une API fournie par IMDb, mais elle n'est pas libre et n'est apparemment pas documentée.

Nous nous sommes donc penchés sur OMDb (Open Movie Database), qui offre une API permettant de retirer les informations d'IMDb. L'avantage étant que cette API est libre et gratuite, même s'il est possède quelques limitations au niveau des recherches qui peuvent être effectuées, la principale concernant les titres des films, qui doivent être le titre original.

La possibilité de pouvoir obtenir les informations au sujet des films et séries étant assurée, il fallait également s'occuper des métadonnées présentes dans les fichiers eux-mêmes.

Le principal défi était de pouvoir facilement extraire ces métadonnées depuis tous les types de fichiers supporté par notre application. Après avoir recherché plusieurs moyens d'extraire ces métadonnées il a fallu nous rendre à l'évidence que ce ne serait pas une tâche facile. En effet, si certains formats autorisent les métadonnées, celles-ci n'obéissent pas forcément à un format bien défini (mkv et mp4). D'autre format ne possèdent pas vraiment de métadonnées (le format avi pour ne pas le citer).

Après avoir étudié plusieurs alternatives nous avons opté pour la simplicité. La libraire VLCJ fourni des classes permettant d'extraire les métadonnées des fichiers vidéo de manière facile. Il n'y a bien entendu pas de miracle et certaines métadonnées peuvent ne pas être extraites, mais quel que soit la solution adoptée, ce genre de problème se serait posé. De plus VLCJ, possédait l'avantage d'être de toute façon utilisé dans notre application.

Et maintenant concernant l'API choisi pour les musiques, l'idée de base été de pouvoir reconnaître un fichier audio juste le lisant. Nous nous sommes donc penchés sur des algorithmes d'« acoustic fingerprint ».

Le principe de cet algorithme est assez simple. Cela consiste à détecter, sur base d'un échantillon, les différentes caractérise d'un morceau (tempo, etc…), et ensuite de recherche dans une base de données si cela correspond à un fichier déjà connu. Le problème principal de cette méthode c'est justement la dernière partie. Car posséder une telle base de données n'est pas une chose facile.

Après plusieurs heures recherches, nous avons trouvé différents outils.

Musicbrainz : site mettant à disposition un large catalogue gratuit de musique. Mais attention ce catalogue ne contient que les informations textes, et la pochette. De plus, il ne propose pas un moyen de reconnaître un fichier audio.

Echonest : Entreprise mettant à disposition différent outils concernent les fichiers audio. Notamment un système de « fingerprint ». Malheureusement, ces dernières ne mettent pas leur base de données à disposition gratuitement (par contre la librairie est disponible gratuitement).

Audiotag : site permettant de faire une reconnaissance en ligne de musique. Malheureusement il n'existe pas d'API.

En d'autres termes, vous l'aurez compris, il n'existe actuellement pas d'entreprise mettant à disposition une telle base de données gratuitement.

Il faut maintenant passer au plan B, c'est-à-dire, reconnaître un fichier audio grâce à ses métadonnées.

Pour faire cela, il faut pouvoir les lire. A première vue, cela semble facile, mais elle dépende du format du fichier. Et comme nous avons dit, dans le cahier des charges, que nous supportons plusieurs types de format, il faut être capable de tout lire sans avoir une librairie par format.

Pour réussir cela, nous avons utilisé la librairie jaudiotagger qui permet de lire les métadonnées de tous les formats qui nous intéresse.

A partir de là, il nous faut encore un moyen de récupérer les informations d'une musique. Pour cela, il existe une API, qui a déjà fait ces preuves, l'API de Spotify. Il faut juste trouver un moyen de l'intégrer efficacement dans notre application.

Après un peu de recherche, nous sommes tombés sur le répertoire Github suivant : https://github.com/thelinmichael/spotify-web-api-java

Ce dernier met à disposition une librairie pour faciliter l'utilisateur de l'API, pas d'appel HTTP à faire.

Donc si on résume, nous allons utiliser jaudiotagger pour lire les différentes métadonnées des fichiers audio, et ensuite, nous allons utiliser l'API Spotify (à travers la librairie trouvée) pour récupérer toutes les informations qui nous intéresse (pochette album, etc…).

3.5 Lecteur vidéo

Java ayant des capacités limitées concernant la lecture de fichiers multimédias, nous nous sommes tournés vers une librairie externe pour accomplir cette tâche. Nous avons choisi la librairie vlcj qui fournit des bindings vers la librairie native « libvlc ». Cette libraire permet de décoder la plupart des formats audio et vidéo sans avoir à se soucier d'installer des codecs.

Le lecteur vidéo utilise Swing pour l'affichage de la vidéo et des contrôles. La raison pour laquelle nous utilisons Swing et non JavaFx est que pour bénéficier de l'accélération graphique il est nécessaire d'utiliser des composants AWT « heavyweight » dont l'équivalent n'est pas disponible en JavaFx. Après une première tentative d'implémentation en JavaFx, on a bien pu vérifier qu'il fallait utiliser l'accélération graphique, car il n'était pas possible de lire une vidéo de manière fluide.

A la base le lecteur vidéo offrait un « overlay » pour le contrôle de la vidéo. Pour réaliser une telle interface, il fallait créer une deuxième fenêtre superposée au lecteur vidéo. Cette fenêtre doit avoir une position synchronisée avec celle du lecteur. Le problème avec cette méthode est qu'au moment de passer le lecteur vidéo en plein écran, il arrivait que le gestionnaire de fenêtre du système d'exploitation décide de mettre l' « overlay » soit en arrière-plan, soit en plein écran. Pour résoudre ce problème, nous avons intégré le menu de contrôle directement dans la fenêtre principale du lecteur.

La gestion de la synchronisation s'effectue d'une manière similaire à celle du lecteur audio.

Remarque : Un composant « heavyweight » est un composant qui utilise l'API graphique fournie par le système d'exploitation.

3.6 Lecteur audio

Pour concevoir un lecteur audio, il est nécessaire d'utiliser un Framework efficace permettant la lecture d'un grand nombre de formats.

Nous avons décidé d'utiliser **vlcj** qui est un Framework Java offrant l'intégration d'une instance de VLC dans notre projet. L'avantage de cette approche est que nous découplons la partie visuelle de la partie fonctionnelle. Nous codons notre propre fenêtre pour le lecteur, celle-ci utilisant les services de VLC pour l'étape finale qu'est la lecture du média. Rappelons que VLC supporte, pour la musique, les formats suivants : MPEG Layer ½, **MP3 - MPEG Layer 3**, AAC - MPEG-4 part3, Vorbis, AC3 - A/52 (Dolby Digital), E-AC-3 (Dolby Digital Plus), MLP / TrueHD">3, DTS, WMA ½, WMA 3, **FLAC**, ALAC, Speex, Musepack / MPC, ATRAC 3, **Wavpack**, Mod (.s3m, .it, .mod), TrueAudio (TTA), APE (Monkey Audio), Real Audio, Alaw/µlaw, AMR (3GPP), MIDI, LPCM, ADPCM, QCELP, DV Audio, QDM2/QDMC (QuickTime), MACE.

L'instance de VLC est jointe dans les ressources de notre projet. Nous nous sommes basé sur une instance 64 bits car nous utilisons la JDK 1.8 64 bits de Java pour la compilation de notre projet.

D'une fois que nous avons notre partie fonctionnelle, nous pouvons passer à la partie visuelle. Généralement en Java il est difficile de réaliser une belle interface car on doit bien souvent s'appuyer sur la bibliothèque **Swing** ou **AWT** qui peine à sortir un design actuel.

Pour ce projet, nous nous sommes lancé le défi de réaliser un design flat et pour cela nous avons utilisé **JavaFX,** nouvelle technologie d'Oracle promettant du renouveau en termes d'interface.

Pour la conception d'un lecteur audio, il suffit d'utiliser une liste, des boutons Play / Suivant / Précédent et une barre de progression.

3.7 Interface

A cause de l'importance de l'interface, nous avons choisi de diviser ce chapitre en sous-chapitre en prenant une vue par chapitre.

3.7.1 Introduction

L'interface a été réalisé au moyen de la librairie graphique FXML et grâce au logiciel « SceneBuilder » pour une construction WYSIWYG de nos vues. L'avantage principal de faire nos interfaces graphiques avec FXML et SceneBuiler est que nous allons utiliser du css (css légèrement différent que celui qu'on utilise pour les Web), nous allons donc pouvoir réaliser des interfaces relativement facilement, et qui plus est, des interfaces esthétiquement bonnes. Le problème en revanche avec FXML, est qu'il est relativement jeun et certaines fonctionnalités ne sont pas disponible ou mal fonctionnelle, notamment la lecture de vidéo qui a dû, elle être faite en Swing.

Dans cette partie nous allons détailler chaque vue du programme ainsi que le comparer avec les mockups réalisés sur Photoshop au début du projet.

D'un point de vue global, l'interface n'a que très peu changée, elle est restée très proche de ce qui avait été désigné au début. La plupart des modifications apportées dans le but de simplifier l'interface et d'éviter la redondance, ces points seront détaillé plus bas quand nous aborderons les comparaisons des vue une à une.

3.7.2 Accueil

Nous allons commencer par détailler la vue d'accueil de notre programme, la première image représente l'interface désignée sur Photoshop, la seconde l'interface réelle. Globalement les deux vues sont similaires, dans les deux cas nous avons notre menu en haut à gauche (nous l'appellerons root) ce menu sera présent tout au long de la navigation. Nous avons également le menu d'accueil qui reprend le menu du haut sans le bouton d'accès à l'accueil. L'unique différence entre l'interface fait sur Photoshop et celle réalisé est le style css des boutons. La raison à cela est dû au fait que nous avions pris des css déjà existant pour ne pas nous attardé sur quelque chose de si futile, de plus les css trouvés allaient bien avec notre interface.

Figure 1 : MockUp de l'accueil

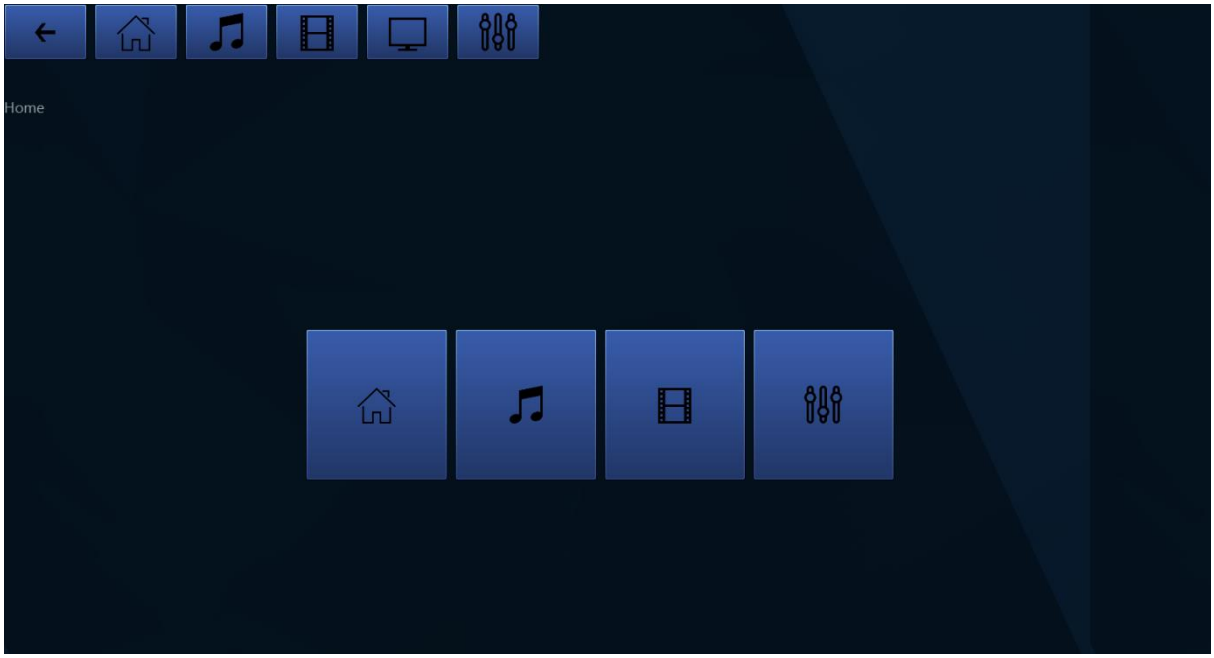


Figure 2 : Réalisation de l'accueil

3.7.3 Vue film

Nous allons désormais passer à la vue des films, cette vue a été considérablement simplifié, lors de la réalisation sur Photoshop nous avions eu les yeux plus grand que le ventre et nous avons réalisé une maquette avec une interface relativement poussée, nous nous sommes très vite rendu compte qu'une telle interface allait être trop chronophage et nous avons donc dû faire des concessions et réalisé une interface plus simple. Nous avons opté pour un simple tableau listant les différents films.

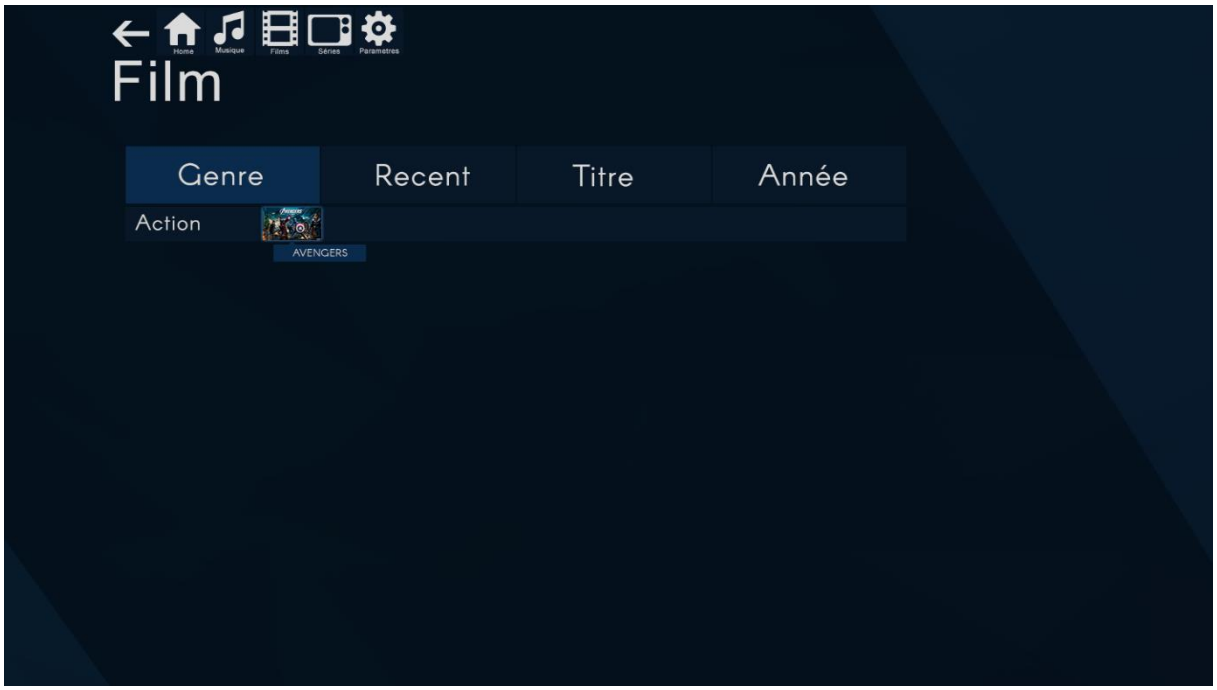


Figure 3 : MockUp vue film

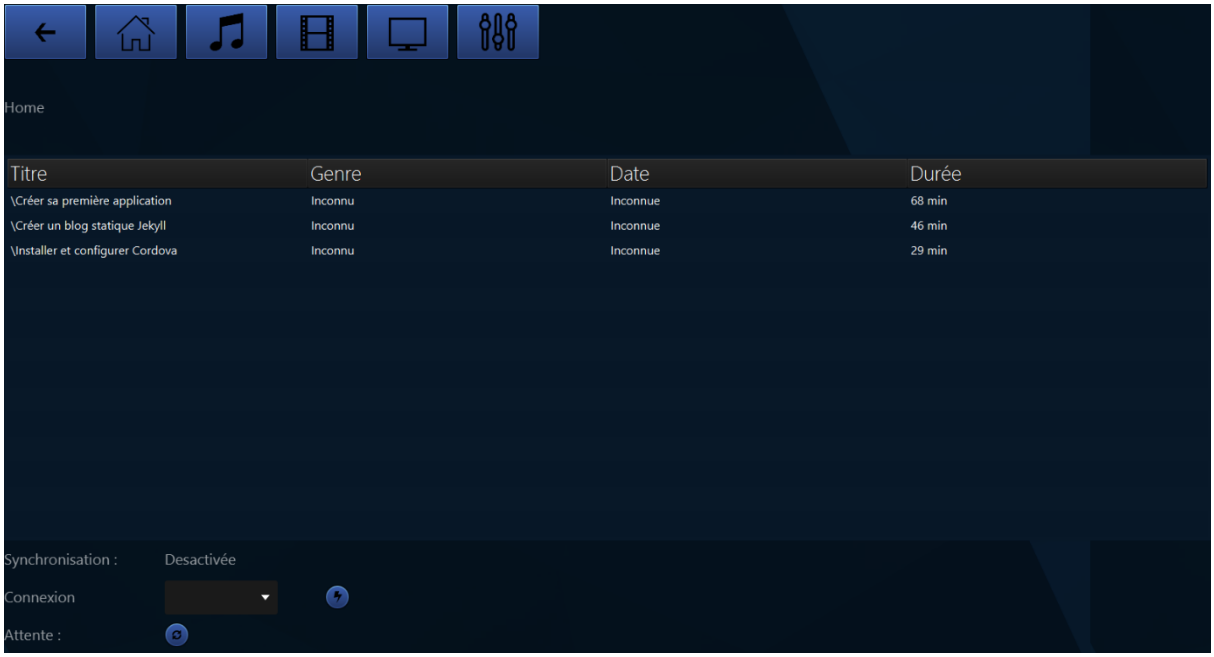


Figure 4 : vue des films

3.7.4 Vue film détaillée

Pour la vue détaillée pour les films nous avons fait quelque chose de relativement similaire à ce que nous avions fait comme maquette, nous avons simplement supprimé les onglets casting et critiques. Nous les avons jugés peu pertinentes.

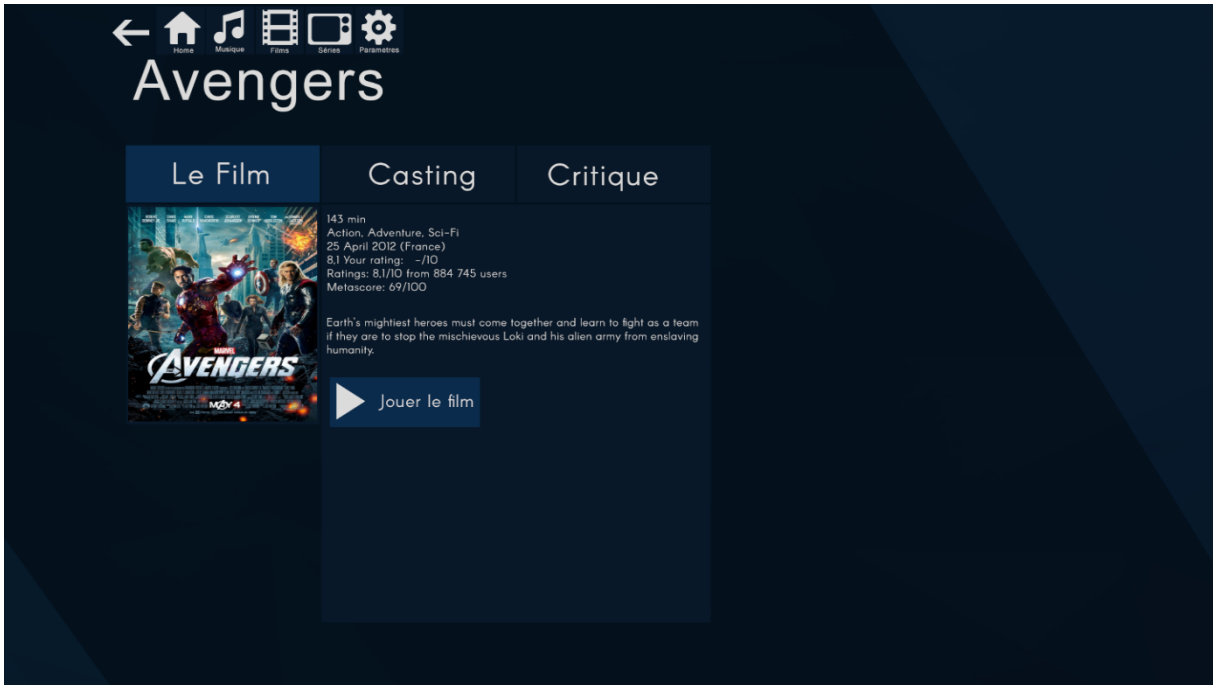


Figure 5 MockUp Film détaillée

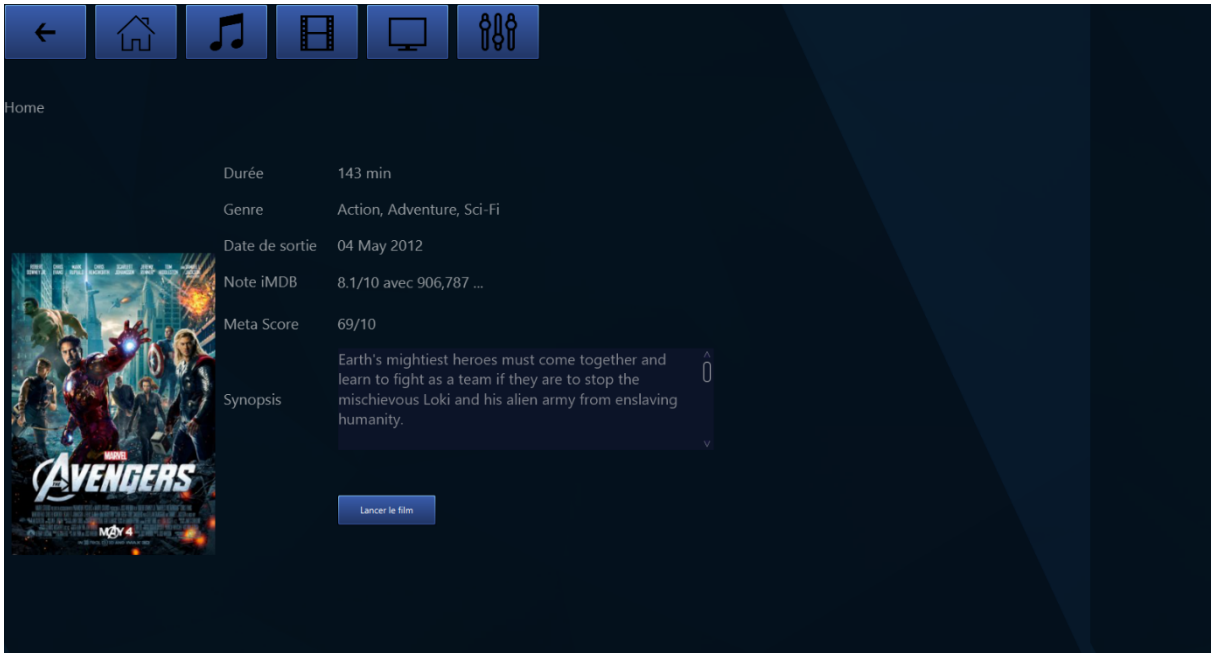


Figure 6 vue Film détaillé

3.7.5 Vue musique

La vue musique à elle aussi été grandement simplifié, le fait que dans un tableau nous pouvions trier nos musique en cliquant sur les têtes de colonne (Titre, Artiste, Album etc…) nous avons jugé inutile de faire ces onglets, et tout comme ces homologues « film » et « série » son interface réalisée sur Photoshop était là aussi beaucoup trop chronophage. Nous n'avions aussi pas prévu d'espace pour la partie synchronisation, ce qui nous a pousser vers une restructuration complète de cette vue. Nous avons donc pris la décision de merger la partie navigation et la partie player

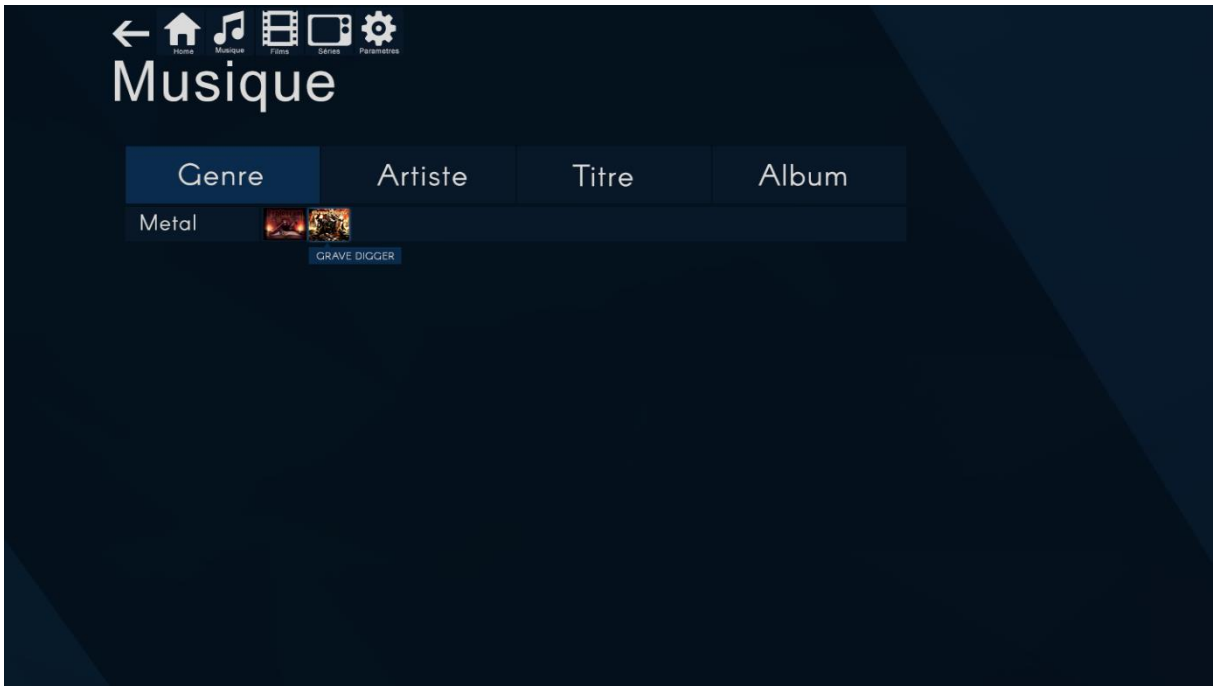


Figure 7 MockUp musique

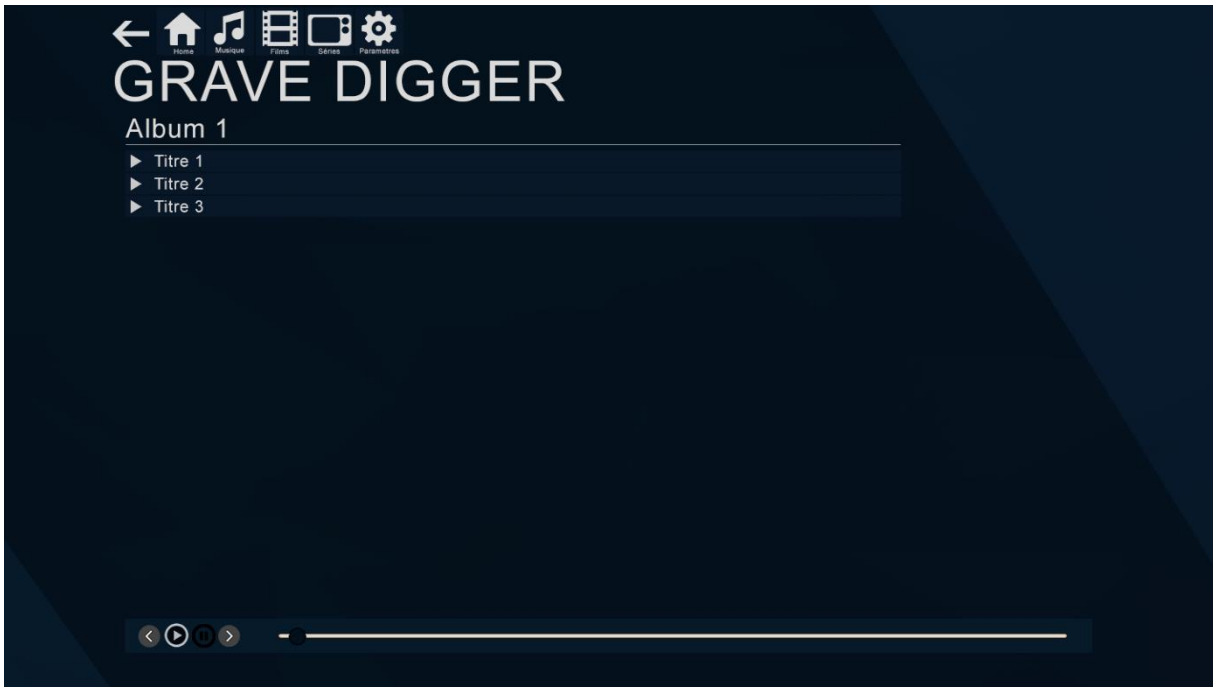


Figure 8 MockUp player

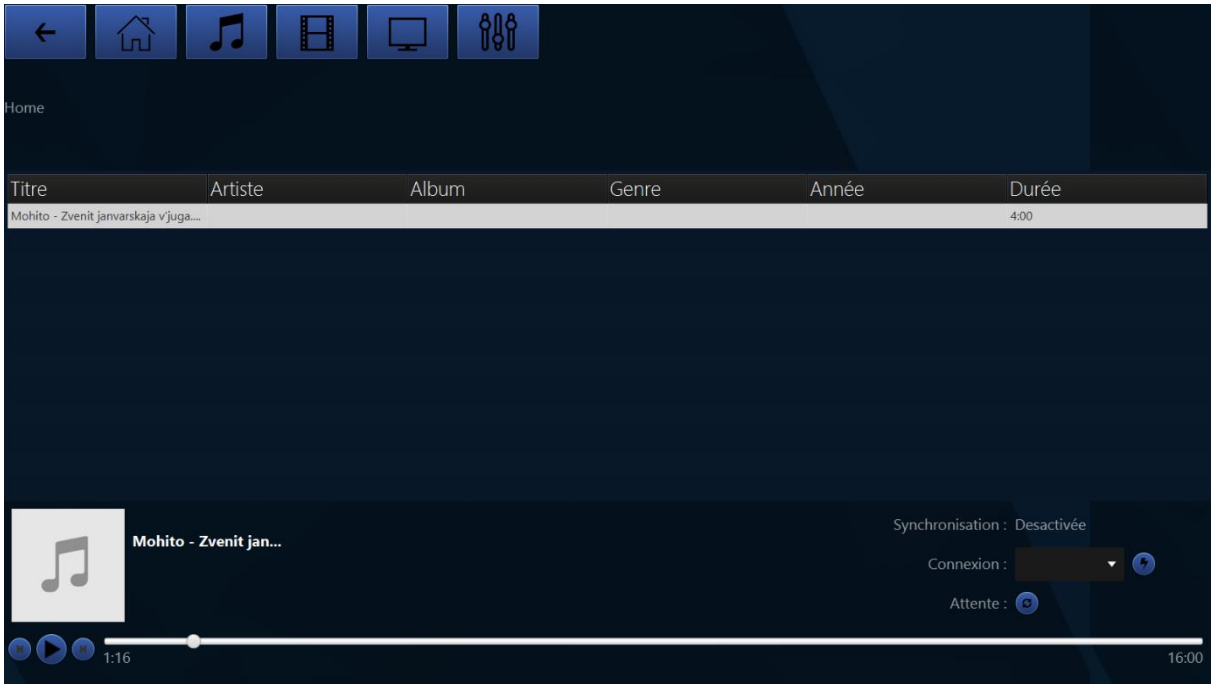


Figure 9 Player réalisé

3.7.6 Vue série

Tout comme les films et la musique la vue des séries a été simplifiée à un simple tableau rempli par le nom des séries que nous possédons (un simple dossier avec un nom correct suffi pour récupérer les informations relatives à la série)

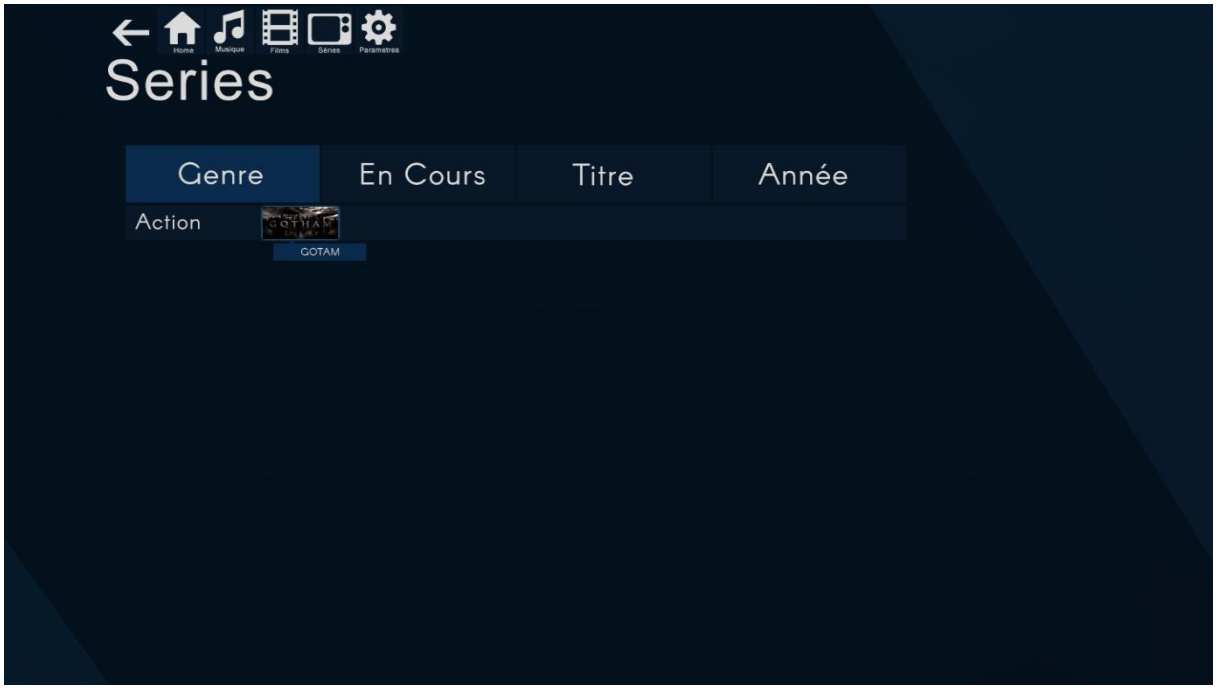


Figure 10 : MockUp série

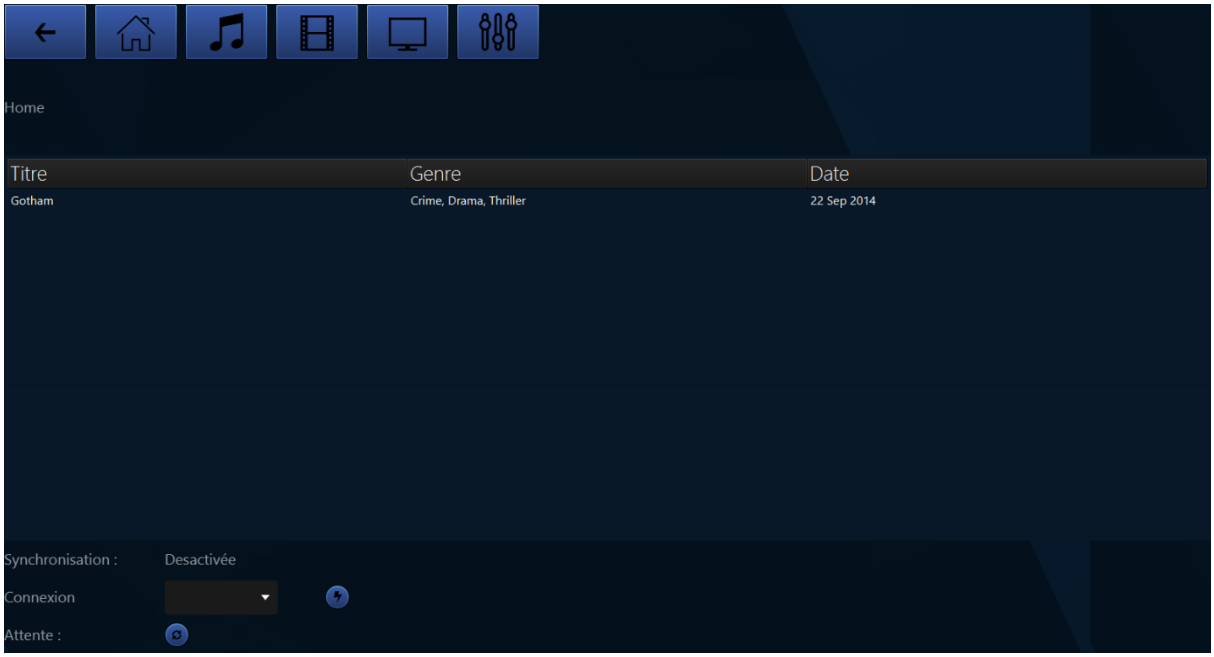


Figure 11 : Vue des séries

3.7.7 Vue série détaillée

Tout comme la partie détaillée sur le film, la partie détaillée des séries et plutôt proche de l'originale. Les onglets casting et critique ont été supprimer et un onglet « épisode » a été ajouté.

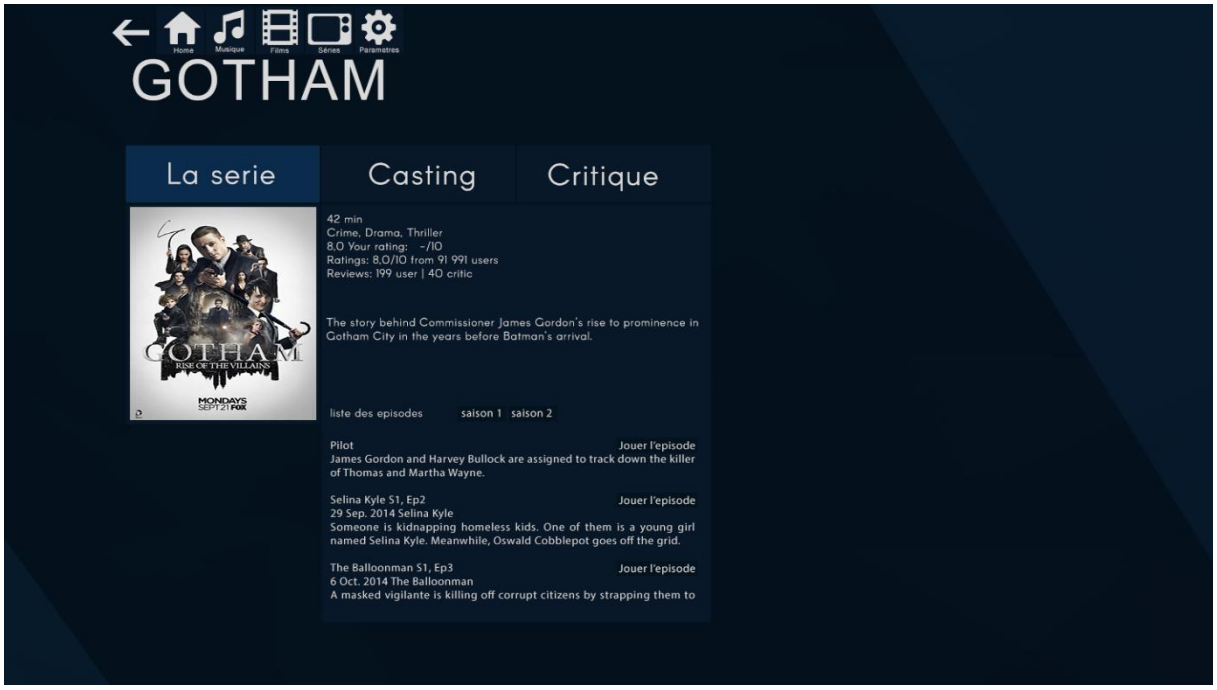
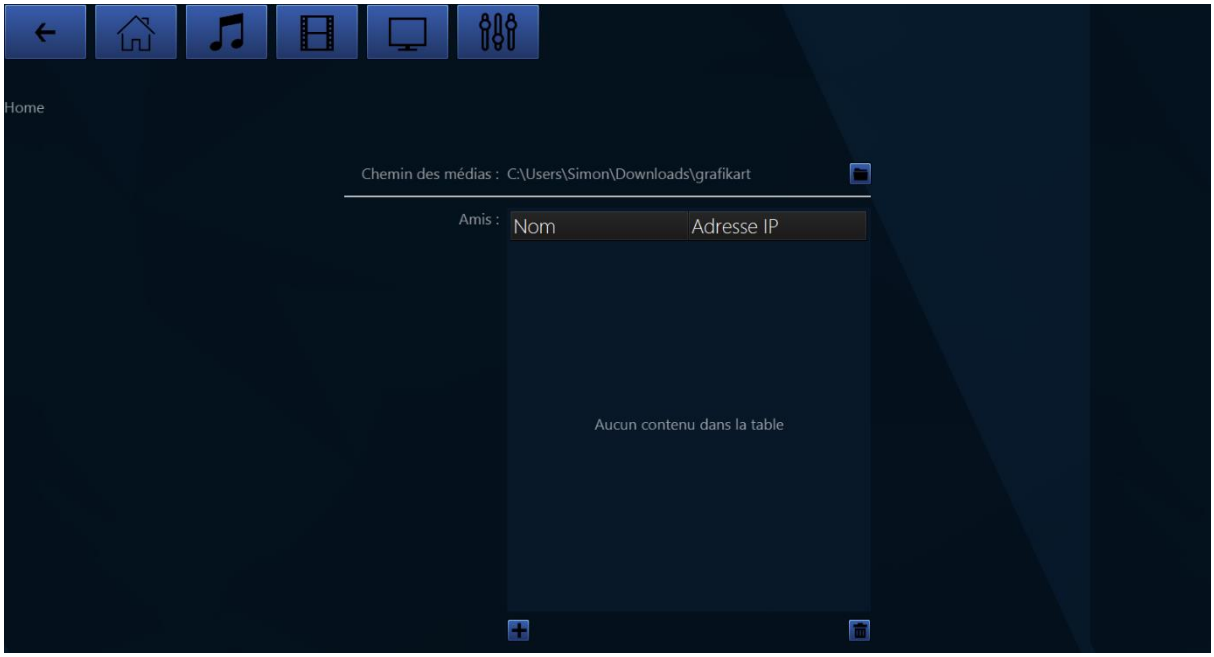


Figure 12 : MockUp série détaillée

Figure 13 vue Film détaillé

3.7.8 Settings

Cette dernière vue ne figurait pas dans nos mockUp, car au début du projet nous ne savions pas vraiment ce que nous allions y mettre et surtout comment. Nous avions une vague idée des éléments à y mettre, comme les répertoires de musique, film et vidéo.



3.7.9 Note finale

Pour terminer ce chapitre nous pouvons dire que la réalisation d'une interface sur un outil telle que Photoshop, puis la reproduire via du code n'est pas une mince affaire, surtout quand nous avons les yeux plus grand que le ventre et nous réalisons un mockUp bien travaillé et où l'on a plus pensé artistiquement que pratiquement.

Dans l'ensemble nous avons quand même pu avoir une interface convenable et agréable à utiliser même si elle ne diffère en pas mal de points de l'originale.

4 Réalisation

Après avoir expliqué nos différents choix, nous allons aborder la conception en elle-même avec la structure du code.

4.1 Base de données

L'utilisation d'une base de données paraît être une bonne solution pour stocker ces informations, particulièrement en ce qui concerne les médias qui peuvent contenir beaucoup d'informations.

Nous avons choisi SQLite comme moteur de base de données. Le principal argument en sa faveur étant qu'il ne requiert pas un serveur mais est directement intégré à l'application sous la forme d'un fichier unique. De plus il est accessible par le langage SQL, qui nous est bien connu.

Pour pouvoir interagir avec la base de données depuis notre application nous utilisons JDBC.

Tout ce qui concerne la base de données se trouve dans le paquet SQLite.

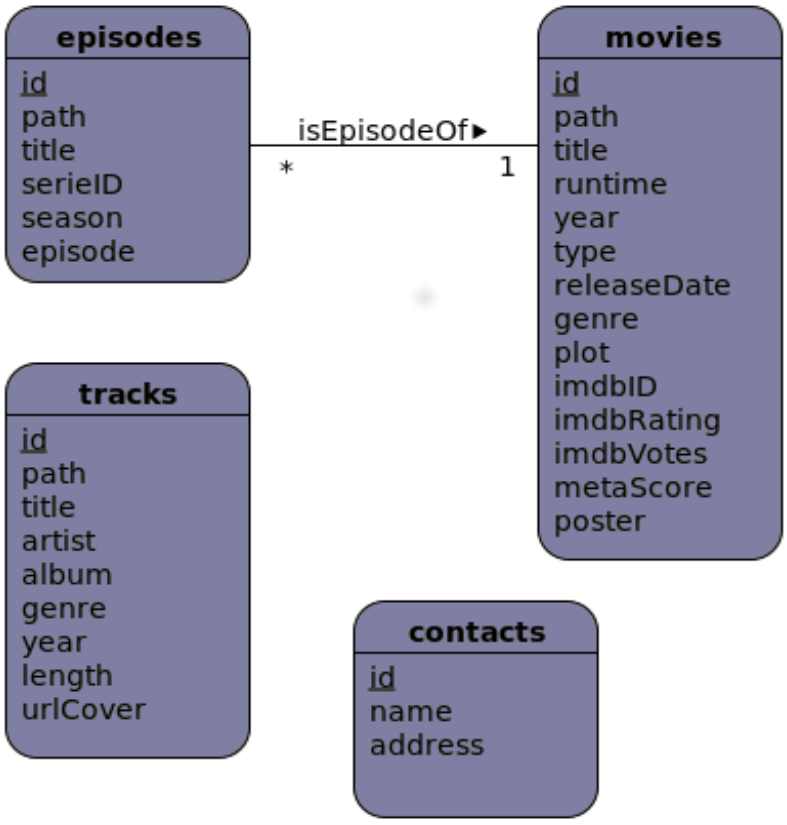


Figure 14 : Schéma entité associations de la base de données

4.1.1 Table contacts

Cette table contient tout simplement les contacts de l'utilisateur et ne possède que deux colonnes en plus de l'identifiant : le nom du contact et son adresse IP.

4.1.2 Table tracks

Cette table contient les informations concernant les fichiers audio présents dans la collection de l'utilisateur. Elle contient les colonnes suivantes :

- id : qui est simplement l'identifiant d'un enregistrement.
- path : correspond au chemin du fichier correspondant aux informations.
- title : correspond au titre du morceau de musique.
- artist : correspond au nom de l'artiste jouant le morceau.
- album : correspond au nom de l'album dont le morceau fait partie.
- genre : correspond au genre de musique du morceau.
- year : correspond à l'année de sortie du morceau.
- length : correspond la durée du morceau
- urlCover : correspond à l'url d'une image de la pochette.

Il est bien entendu tout à fait possible que certains de ces champs soient vides, à l'exception de l'id, du titre et du « path ».

4.1.3 Table movies

Cette table contient les informations aux sujets des fichiers vidéo. Il est à noter que les informations contenues dans cette table peuvent concerner soit une vidéo « solitaire » (un film ou une vidéo quelconque) soit une série. Dans le second cas, il ne s'agit pas à proprement parler d'informations concernant un fichier vidéo mais d'informations concernant une série, qui sont les mêmes pour tous les épisodes de cette série.

Comme nous tentons d'obtenir les informations d'IMDb, certains champs sont directement liés avec cette base de données.

Cette table contient les colonnes suivantes :

- id : qui est simplement l'identifiant d'un enregistrement.
- path : correspond au chemin du fichier correspondant aux informations ou au dossier contenant les épisodes de la série correspondant aux informations.
- title : correspond au titre de la vidéo ou de la série.
- runtime : correspond à la durée, en minutes, de la vidéo ou des épisodes de la série.
- year : correspond à l'année ou la vidéo a été publiée.
- type : indique le type de l'enregistrement vidéo (movie) ou série (series).
- releaseDate : correspond à la date de sortie de la vidéo ou de la série.
- genre : le ou les genre(s) de la vidéo ou de la série.
- plot : un résumé de l'intrigue de la vidéo ou de la série.
- imdbID : identifiant du film ou de la série sur IMDb.
- imdbRating : note donnée au film ou à la férie par les utilisateurs d'IMDb.
- imdbVotes : nombre de votes dont la note est la moyenne.
- metaScore : méta-score du film ou de la série.

- poster : url d'une image représentant le poster du film ou de la série.

Il est toujours possible que la plupart des informations ne soient pas disponibles. Les seuls champs qui ne peuvent pas être nuls sont : l'id, le path, le title, le runtime et le type.

4.1.4 Table épisodes

Cette contient les épisodes des différentes séries contenus dans la collection de l'utilisateur. Elle contient les colonnes suivantes.

- id : qui est simplement l'identifiant d'un enregistrement.
- path : correspond au chemin du fichier correspondant aux informations.
- title : titre de l'épisode.
- serieID : id de la série dont l'épisode fait partie. Cette id est celui de la série dans la base de données de l'application.
- season : indique la saison dont l'épisode fait partie.
- episode : indique le numéro de l'épisode dans la saison.

Tous les champs peuvent être nuls à l'exception de l'id, de path, de title et de serieID.

4.1.5 Paquet sqlite

Ce paquet contient toutes les classes qui interagissent avec la base de données.

- La classe SQLiteConnector qui permet de se connecter à la base de donnée et également de créer les tables s'il elles n'existent pas déjà. Elle permet également de réinitialiser le contenu de la collection.
- La classe ContactManager qui s'occupe de gérer les contacts (ajouts, suppression, modifications et obtentions des contacts).
- La classe MovieManager qui s'occupe de gérer les movies (film et séries) ainsi que les épisodes.
- La classe TrackManager qui s'occupe de gérer les tracks.
- La classe Contact qui est une classe qui représente un contact.

4.2 Collection

La gestion de la collection, en dehors de la base de données, ait effectuée par la classe LibraryManager qui se trouve dans le paquet utils du projet.

Le principe de cette classe est simple. Elle possède une méthode statique qui, à partir d'une racine passée en paramètre, liste tous les fichiers et sous-dossier et les parcours un à un.

Dans le cas de fichiers on vérifie s'il s'agit d'un fichier audio ou vidéo valide. Si c'est le cas, et après avoir vérifié que le fichier n'est pas déjà connu de la base de données, les métadonnées du fichier sont extraites et les informations supplémentaires sont récupérées à travers les apis sons et vidéos.

Dans le cas d'un sous-dossier, celui-ci est considéré comme une série, dont le titre correspond au nom du sous-dossier. On vérifie tout d'abord si la série n'est pas connue de la base de données puis, si ce n'est pas le cas, on tente alors d'obtenir des informations sur la série à travers l'api vidéo. La série est ensuite ajoutée à la base de données.

Le sous-dossier est ensuite parcouru à la recherche de fichiers vidéo, qui seront ajoutés comme épisodes de la série, s'il n'était pas déjà connu de la base de données. Seuls les métadonnées déjà présentes dans les fichiers sont extraites, à cause de la difficulté à récupérer les informations concernant un épisode à travers l'api vidéo.

Concernant la suppression des fichiers, lorsqu'un fichier est détecté comme ayant été supprimé de la collection, les informations correspondantes sont retirées de la base de données. La difficulté était de détecté que le fichier avait été supprimé. Cela est fait au lancement de l'application, lors de la récupération de la liste des films, séries et musiques, afin d'éviter que les fichiers supprimés ne soient affichés dans l'interface de l'application. Toutefois, la suppression d'un fichier en cours d'utilisation de l'application reste problématique.

4.3 API Vidéo

Concernant l'aspect vidéo le traitement de l'aspect métadonnées est différent de celui de l'API web. Effectivement, l'extraction des métadonnées est faite relativement facilement en faisant appel à une classe de la libraire VLCJ. Il n'y a donc pas grand-chose à en dire. L'extraction de ces métadonnées est faite directement lors du scan des fichiers de la collection (classe LibraryManager).

Concernant l'API web, plusieurs classes ont été créées :

- OMDbClient : cette classe est celle qui envoi les requêtes vers l'API de OMDb et également celle qui parcours les réponses à ces requêtes.
- SearchQuery : cette classe représente une requête. Les différents paramètres de la requête peuvent être modifié à travers cette classe avant qu'elle ne soit envoyée à l'API d'OMDb en utilisant la class OMDb.
- MovieDataGetter : cette classe abstrait les requêtes faîte à OMDb. Elle se charge elle-même de créer les requêtes et de faire les appels aux méthodes d'OMDbClient.
- MovieInfos : cette classe contient l'ensemble des données concernant un film ou une série. C'est à l'intérieur d'objet de cette classe que sont stockées les informations obtenues sur OMDb. Cette classe est celle qui est utilisées pour transmettre les informations entre l'API vidéo, la base de données et les contrôleurs des vues films et séries.
- SearchResult : cette classe représente le résultat d'une recherche sur OMDb. Elle devait permettre de laisser l'utilisateur choisir entre les différents résultats d'une recherche en cas d'ambiguïtés (films portant le même nom). Toutefois, comme rien n'avait été prévu dans l'interface à cette fin, cette classe n'est finalement pas utilisée dans l'application.
- Season : cette classe représente une saison d'une série, avec notamment une liste d'épisodes.
- Episode : cette classe représente un épisode d'une saison et était typiquement contenu dans un objet de type « Season ». Le but de cette classe et de « Season » était de pouvoir lister les saisons et épisodes d'une série, même ceux à paraître. Toutefois, comme il n'était pas possible de garantir que les épisodes possédés par l'utilisateur seraient reconnus comme tels, nous avons remis à plus tard l'utilisation de ces classes et avons mis la priorité sur l'affichage des épisodes à disposition de l'utilisateur.

Une grande difficulté qui a été rencontrée est le fait qu'il n'est pas facile d'obtenir des informations sur toutes les vidéos. En effet, il suffit que le fichier soit dénué de métadonnées et que le nom de ce fichier ne corresponde pas exactement au nom du film (ou que le nom du dossier ne corresponde pas à la série), pour qu'aucune information ne soit disponible.

De plus, il n'est pas possible de rechercher directement des informations sur OMDb à propos d'un épisode uniquement à partir de son nom. Cela nous a beaucoup handicapé sur la possibilité d'afficher les épisodes d'une saison, puisque, si un fichier appartenant à une série mais dont la saison et le numéro, voir même le nom, ne peuvent être identifiés, est présent nous ne pourrions pas l'afficher dans l'interface, si celle-ci se base sur les saisons et épisodes obtenus sur OMDb. C'est la raison pour laquelle nous avons dû abandonner cette fonctionnalité.

4.3.1 Fonctionnement de l'API d'OMDb

L'API d'OMDb est une API REST. Il suffit donc d'effectuer des requêtes HTTP de type GET, en précisant les paramètres de recherches dans l'URL, pour pouvoir obtenir une réponse de l'API. Les réponses peuvent être au format JSON ou XML. Nous avons choisi, par préférence personnelle, le format JSON.

On peut faire principalement trois types de recherches :

- Les recherches par titres, plus large, qui permettent de récupérer une liste de films ou séries (avec, pour chacun d'entre, des informations limitées) dont le nom correspond plus ou moins au titre passé en paramètre de la recherche.
- Les recherches d'un titre précis. Seul un film (ou une série) dont le titre correspond exactement au paramètre de recherche sera retournés, avec toutes les informations disponibles.
- Les recherches par identifiant. Il s'agit de recherches selon l'identifiant du film ou de la série sur IMDb. SI l'identifiant est correct, les informations de la série ou du film sont retournées. Il est à noter que cette recherche permet également, dans le cas d'une série, de préciser une saison, ce qui retournera les informations de cette saison (principalement les épisodes qui la composent). Toutefois il n'y a pas (encore) de moyen d'obtenir directement la liste des saisons.

Lorsqu'une recherche ne donne rien, un JSON contenant un champ Error est retourné.

L'URL de base de l'API est *http://omdbapi.com/* les paramètres sont précisés selon le format

?<nom du paramètre>=<valeur du paramètre>&<nom du paramètre>=<valeur du paramètre>

Les principaux paramètres sont :

- t pour faire une recherche selon un titre précis, qui est le paramètre.
- s pour faire une recherche de titres correspondant au paramètre.
- id pour faire une recherche selon l'id passé en paramètre.
- type pour préciser le type de contenu recherché (film ou série, il existe aussi le type épisode, mais il ne fonctionne pas pour l'instant).
- r pour préciser le type de réponse (JSON ou XML).
- y pour préciser l'année de sortie du contenu recherché.

4.4 API web

Concernant l'implémentation des API, nous avons pris le choix de créer un package « api » avec 2 sous-package « sound » et « video ». Les différentes dépendances, nécessaire pour le fonctionnement des API, sont ajoutées dans le fichier Maven.

Concernant la partie son, nous avons créé 2 classes. La première classe « GetSoundInfo » qui contient une méthode statique qui prend en paramètre le chemin vers le fichier audio. Cette dernière va lire les métadonnées, et ensuite en se basant sur le nom du morceau, on va faire une recherche sur l'API Spotify. Une fois le résultat trouvé, la méthode va nous retourner un objet de type « TrackInfos » qui contient toutes les informations trouvées sur Spotify, avec notamment l'URL de la pochette.

Nous avons appliqué la stratégie suivante pour la définition des tags d'une musique quelconque :

- On récupère dans tous les cas la longueur des médias
- On récupère le titre de la musique par tag ID3
  - Si on ne le trouve pas, on affichera simplement le nom du fichier mp3 comme titre à l'affichage
- On récupère un maximum de tags ID3 (artiste, album, genre, année)
  - Pour chaque tag qu'on ne trouve pas, on tente une récupération via l'API Spotify
- Finalement, on récupère une image de la pochette de l'album via l'API Spotify

4.5 Synchronisation

Pour réaliser la synchronisation de médias, il faut faire attention à 2 points principalement :

- L'exécution des commandes sur le lecteur
- Le passage des commandes via le réseau

Pour passer les commandes via le réseau, nous allons utiliser TCP, car c'est le moyen le plus rapide et le plus simple de mettre en œuvre une communication réseau.

Dans notre application, nous avons implémenté cela sous format d'une seule classe et d'une interface. La classe singleton « SyncManager » permet de créer un serveur TCP, et ensuite d'utiliser le protocole mis en œuvre dans notre application. Tandis que l'interface « SyncHandler » est utilisé pour exécuté les commandes reçus.

Voici maintenant le protocole de communication que nous avons implémenté pour la synchronisation des médias (un « #@ » sépare la commande du paramètre):

- **begin#@NomDuMedia** : permet de donner le nom du fichier afin de comparer et d'informer l'utilisateur le cas où c'est différent.
- **pause** : indique que l'utilisateur a appuyé sur pause.
- **play** : demande au lecteur de lire le média.
- **setAt#@Seconde** : demande au lecteur de se positionner à N seconde.
- **bye** : fin de la synchronisation.

A partir de ces 2 fichiers créés, il s'agit de les « lier » aux différents lecteurs (audio et vidéo), Et cela fonctionne.

Aux niveaux de la robustesse, étant donné que nous utilisons la pile TCP, on est assuré de recevoir les messages dans l'ordre.

4.6 Lecteur vidéo

Le lecteur vidéo se repose sur deux classes principales : la classe « Player » et la classe « ControlPanel ».

La classe « Player » est un « singleton » fournissant des méthodes permettant de lire une vidéo et la mettre en pause. Il y a également des méthodes permettant d'avancer ou de reculer la vidéo de deux secondes ainsi que de position le lecteur à un temps donné. Chacune de ces méthodes intègre également la possibilité d'envoyer des événements utilisés par le gestionnaire de synchronisation. Le « ControlPanel » est un élément Swing contrôlant directement le lecteur au moyen des méthodes mentionnées précédemment.

4.7 Lecteur audio

Voici la structure, ainsi que la technologie que nous avons utilisée pour créer le lecteur audio.

4.7.1 Technologies

Le lecteur audio a été réalisé à l'aide de JavaFX en respectant le modèle de programmation **MVC**. La *vue* est caractérisée par le fichier fxml décrivant les composants de la fenêtre, leurs positions, leurs tailles, leurs contraintes ainsi que d'éventuels id. Le *contrôleur* est représenté par une classe interagissant avec les composants de la fenêtre. Le *modèle* est défini par une classe Java simple, appelée aussi POJO, représentant une musique elle-même caractérisée par ses tags (titre, artiste, album, …) et autres informations utiles.

4.7.2 Structure

Son implémentation se trouve dans le package **music**.

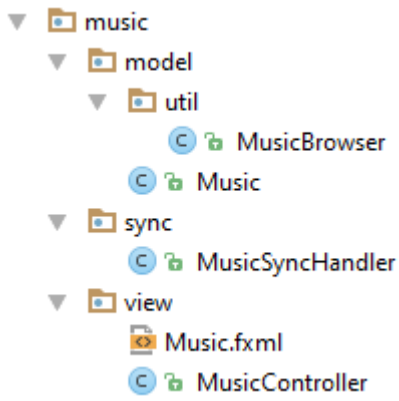


Figure 15 : Package music

Nous avons à l'intérieur un package **model** fournissant la classe modèle Music.java ainsi qu'un autre package **util** offrant une classe MusicBrowser.java permettant la récupération de musique depuis la base de données puis la conversion vers notre format modèle.

Nous avons un package **sync** et sa classe MusicSyncHandler.java implémentant l'interface SyncHandler.java détaillée dans le chapitre 0. Cette classe recevra toutes les actions effectuées par l'ami distant lors d'une synchronisation.

Nous avons un package **view** possédant le fichier fxml pour la vue et la classe pour le contrôleur MusicController.java. Nous aurions pu séparer le contrôleur de la vue en l'insérant dans un autre package mais cela aurait empêché le SceneBuilder de le détecter directement.

4.7.3 Programmation

Lors de la création du lecteur audio, nous avons dû utiliser plusieurs concepts, et librairies pour mener cette partie à bien.

4.7.3.1 vlcj

Pour la réalisation d'un lecteur audio se basant sur vlcj, nous avons besoin en premier lieu d'informer le système de la location de **VLC.**

Comme VLC est intégré directement dans nos ressources, nous pouvons lui passer directement le chemin à l'aide de la commande suivante :

```
NativeLibrary.addSearchPath(RuntimeUtil.getLibVlcLibraryName(), NATIVE_LIBRARY_SEARCH_PATH);
```

Celle-ci cherchera dans le dossier NATIVE_LIBRARY_SEARCH_PATH la présence d'une instance VLC.

Dans un second temps, nous avons besoin des deux classes suivantes :

- uk.co.caprica.vlcj.component.AudioMediaPlayerComponent

Classe de base à instancier encapsulant le lecteur audio. Tous les détails d'implémentation telle que la création de factory sont intégrés.

- uk.co.caprica.vlcj.player.MediaPlayer

Spécification pour un composant lecteur multimédia.

Afin d'avoir un lecteur audio prêt en arrière-plan, il est nécessaire d'effectuer 2 étapes :

- Instancier un AudioMediaPlayerComponent

```
playerComponent = new AudioMediaPlayerComponent();
```

- Récupérer la référence sur le MediaPlayer embarqué

```
player = playerComponent.getMediaPlayer();
```

Une fois le lecteur prêt en arrière-plan nous n'avons plus qu'à lui passer différentes commandes au travers son interface MediaPlayer tel que :

- `playMedia(String mrl, String... mediaOptions)`

Lecture du média X.

- `pause()`

Mise en pause du lecteur.

- `play()`

Lecture du lecteur en pause.

- `setTime(long time)`

Saut à un certain moment dans la musique.

Enfin, pour réagir à certains états du lecteur, il est encore intéressant d'écouter les évènements du lecteur médias tel que :

- `error(MediaPlayer mediaPlayer)`

Une erreur est arrivée.

- `finished(MediaPlayer mediaPlayer)`

Le média a fini d'être lu.

- `playing(MediaPlayer mediaPlayer)`

Le média a commencé à être lu.

- `timeChanged(MediaPlayer mediaPlayer, long newTime)`

Le temps du play-back a changé.

4.7.3.2 Concurrence des threads avec le thread de l'interface

Lorsque plusieurs threads accèdent en même temps à un composant présent dans le thread de l'interface graphique, le programme plante à l'exécution.

Pour résoudre ce problème il est nécessaire d'utiliser le code suivant :

```
Platform.runLater(() -> { });
```

Figure 16 : Implémentation vide

Où l'on donne l'implémentation (ici Anonyme) d'un Runnable effectuant l'action désirée.

Le système se chargera d'effectuer dans l'ordre les actions sur le thread de l'interface.

4.7.3.3 Synchronisation

Pour indiquer au lecteur audio que le système est synchronisé un simple booléen a été utilisé.

Lors de la connexion à un ami ou lors de l'attente d'un ami, nous avons utilisé une stratégie garantissant que le système reste utilisable pendant la connexion et évitant tout inter-blocage.

Lors de l'appui sur le bouton pour la connexion et pour l'attente, le système exécute un nouveau thread s'occupant d'appeler la méthode correspondante dans le SyncManager soit respectivement connect et accept. Ces deux méthodes attendent sur des messages (bloquant) il retourne à la fin true ou false selon la réussite ou l'échec de l'opération.

En parallèle, le système déclenche un compte à rebours de 30s à la fin duquel, une vérification si le thread est toujours en cours sera effectué. Dans le cas où le thread s'exécuterait encore, celui-ci est instantanément tué sinon cela veut dire que l'opération de connexion s'est réalisée avec succès.

4.7.3.3.1 Accept

Dans le cas d'un accept, on attend en premier lieu sur une connexion TCP. Dès qu'on reçoit une connexion, on renvoi un *OK* indiquant à cette personne qu'on a reçu sa connexion. A son tour il nous répond *OK* s'il est toujours de la partie. A ce moment-là, un dialogue apparait nous informant qu'une certaine personne souhaite se connecter. Nous pouvons alors accepter ou refuser l'invitation. Enfin dans le cas où l'on aurait accepté, on attend encore la réponse de la personne en question avant de confirmer la bonne connexion avec un return **true** et la création d'un Worker.

4.7.3.3.2 Connect

Dans le cas d'un connect, on se connecte d'abord à la personne souhaitée en instanciant un nouveau Socket (connexion TCP). Puis, on attend sur le *OK* de la personne. Lorsqu'on a reçu le OK on renvoi un nouveau OK. A ce moment-là, on attend sur la réponse au dialogue de la personne, on attend donc sur un accept ou un deny. Dans le cas d'un deny, on retourne **false** et la connexion échoue. Dans le cas d'un accept, on envoie un Accept à la personne souhaitée et on retourne **true**.

Afin de mieux comprendre le mécanisme, voici un schéma résumant une connexion à une personne en attente :

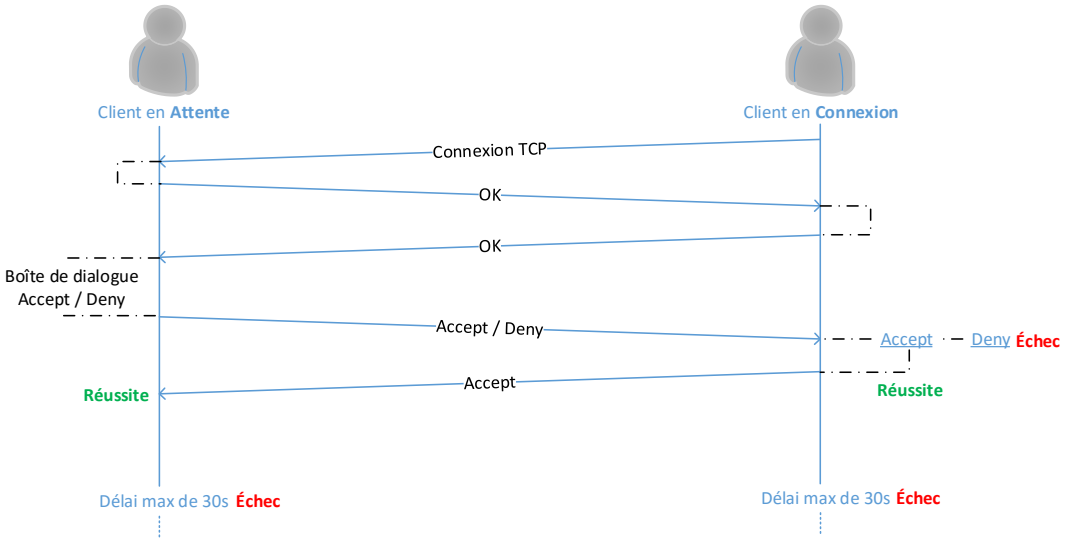


Figure 17 : Schéma mécanisme synchronisation musique

5 Tests Réalisés

Voici la liste des tests que nous avons réalisés afin de vérifier les fonctionnalités de notre application. Cela a été réalisé à la main sans utiliser de librairie particulière.

Test	Accès des vues à partir du menu « home ».
Comportement attendu	Il est possible d'accéder aux différentes vues depuis le menu « home ».
Résultat du test	Réussi
Remarque	Aucune

Test	Accès des vues à partir de la barre de navigation.
Comportement attendu	Il est possible d'accéder aux différentes vues en utilisant la barre de navigation en haut de l'interface.
Résultat du test	Réussi
Remarque	Aucune

Test	Ajouts des fichiers dans la collection.
Comportement attendu	Les différents fichiers contenus dans la collection de l'utilisateur sont ajoutés dans la base de données lorsque la collection est scannée.
Résultat du test	Réussi
Remarque	La vérification a été faite à l'aide d'un outil de visualisation de base de données.

Test	Suppression d'un fichier de la collection.
Comportement attendu	Lorsqu'un fichier est retiré de la collection, il est supprimé de la base de données lors du lancement de l'application.
Résultat du test	Réussi
Remarque	La vérification a été faite à l'aide d'un outil de visualisation de base de données.

Test	Suppression d'un fichier de la collection en cours d'utilisation.
Comportement attendu	Lorsqu'un fichier est retiré de la collection durant l'utilisation de l'application et que l'utilisateur tente ensuite de le jouer, l'utilisateur est informé de l'impossibilité de lire le média.
Résultat du test	Problème
Remarque	Le programme plante. Aucune vérification n'est faite à ce niveau.

Test	Affichage des musiques.
Comportement attendu	Les fichiers audio présents dans la collection sont visibles dans la vue « Musiques ».
Résultat du test	Réussi
Remarque	Aucune

Test	Affichage des films.
Comportement attendu	Les fichiers vidéos présents dans la collection sont visibles dans la vue «Films».
Résultat du test	Réussi
Remarque	Aucune

Test	Affichage des séries.
Comportement attendu	Les séries présentes dans la collection sont visibles dans la vue «Séries».
Résultat du test	Réussi
Remarque	Aucune

Test	Affichage des informations d'un film.
Comportement attendu	Les informations d'un film sont affichées lorsque l'utilisateur double-clique sur l'un d'eux dans la vue « Films ».
Résultat du test	Réussi
Remarque	Aucune

Test	Affichage des informations d'une série.
Comportement attendu	Les informations d'une série sont affichées lorsque l'utilisateur double-clique sur l'une d'elles dans la vue « Séries ». Les épisodes de cette série s'affichent dans l'onglet « Épisodes ».
Résultat du test	Réussi
Remarque	Aucune

Test	Lecture d'une musique.
Comportement attendu	Il est possible de lire une musique en double-cliquant sur l'une d'elles dans la vue « Musique ».
Résultat du test	Réussi
Remarque	La lecture de musique commence à lire la musique.

Test	Lecture d'une vidéo
Comportement attendu	Il est possible de lire un film en cliquant sur le bouton « Lancer Film ».
Résultat du test	Réussi
Remarque	Le lecteur vidéo se lance et commence à lire la vidéo. Toutefois, il n'est pas possible de quitter le plein écran ou de stopper la vidéo en cours de lecture (obligatoire d'attendre la fin).

Test	Lecture d'un épisode d'une série
Comportement attendu	Il est possible de lire un épisode d'une série en double-cliquant sur l'un d'entre eux dans l'onglet « Épisodes » de la vue d'une série.
Résultat du test	Réussi
Remarque	Le lecteur vidéo se lance et commence à lire l'épisode.

Test	Synchronisation d'une musique.
Comportement attendu	Il est possible de se synchroniser avec un autre utilisateur pour lire une musique.
Résultat du test	Partiellement réussi
Remarque	Il est possible de se synchroniser. Les musiques se lisent en même temps et si un des utilisateurs déplace le curseur le déplacement se fait chez l'autre utilisateur. Toutefois, il n'y a pas de gestion d'erreurs.

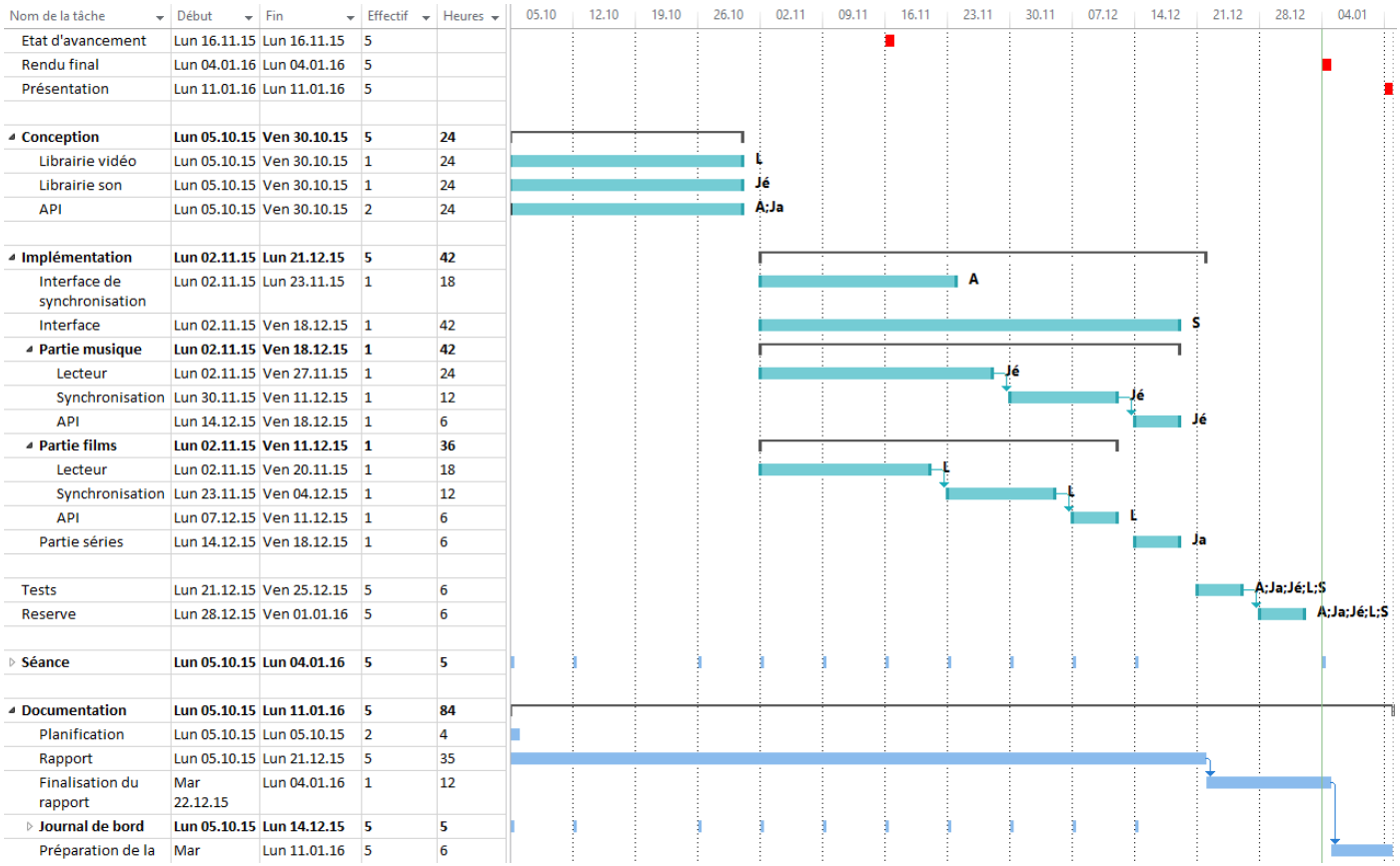
Test	Synchronisation d'un film.
Comportement attendu	Il est possible de se synchroniser avec un autre utilisateur pour lire un film.
Résultat du test	Partiellement réussi
Remarque	L'interface utilisateur ne fournit pas de feedback. La lecture est synchronisée, mais les erreurs ne sont pas gérées.

Test	Synchronisation d'un épisode.
Comportement attendu	Il est possible de se synchroniser avec un autre utilisateur pour lire un épisode.
Résultat du test	Pas réalisé
Remarque	Pas encore de synchronisation au niveau des épisodes.

6 Planification

La planification recense les différents diagrammes de Gantt que nous avons utilisé lors de la gestion du projet.

6.1 Planification initiale



Nom de la tâche	Début	Fin	Effectif	Heures
Etat d'avancement	Lun 16.11.15	Lun 16.11.15	5	
Rendu final	Lun 04.01.16	Lun 04.01.16	5	
Présentation	Lun 11.01.16	Lun 11.01.16	5	
Conception	**Lun 05.10.15**	**Ven 30.10.15**	**5**	**24**
Librairie vidéo	Lun 05.10.15	Ven 30.10.15	1	24
Librairie son	Lun 05.10.15	Ven 30.10.15	1	24
API	Lun 05.10.15	Ven 30.10.15	2	24
Implémentation	**Lun 02.11.15**	**Lun 21.12.15**	**5**	**42**
Interface de synchronisation	Lun 02.11.15	Lun 23.11.15	1	18
Interface	Lun 02.11.15	Ven 18.12.15	1	42
Partie musique	**Lun 02.11.15**	**Ven 18.12.15**	**1**	**42**
Lecteur	Lun 02.11.15	Ven 27.11.15	1	24
Synchronisation	Lun 30.11.15	Ven 11.12.15	1	12
API	Lun 14.12.15	Ven 18.12.15	1	6
Partie films	**Lun 02.11.15**	**Ven 11.12.15**	**1**	**36**
Lecteur	Lun 02.11.15	Ven 20.11.15	1	18
Synchronisation	Lun 23.11.15	Ven 04.12.15	1	12
API	Lun 07.12.15	Ven 11.12.15	1	6
Partie séries	Lun 14.12.15	Ven 18.12.15	1	6
Tests	Lun 21.12.15	Ven 25.12.15	5	6
Reserve	Lun 28.12.15	Ven 01.01.16	5	6
Séance	**Lun 05.10.15**	**Lun 04.01.16**	**5**	**5**
Documentation	**Lun 05.10.15**	**Lun 11.01.16**	**5**	**84**
Planification	Lun 05.10.15	Lun 05.10.15	2	4
Rapport	Lun 05.10.15	Lun 21.12.15	5	35
Finalisation du rapport	Mar 22.12.15	Lun 04.01.16	1	12
Journal de bord	**Lun 05.10.15**	**Lun 14.12.15**	**5**	**5**
Préparation de la	Mar	Lun 11.01.16	5	6

Nous avons structuré les différentes tâches de telle manière à avoir une partie de conception, à travers laquelle nous allons mener divers étude de faisabilité afin de pouvoir identifier les problèmes à l'avance, pour mieux les gérer.

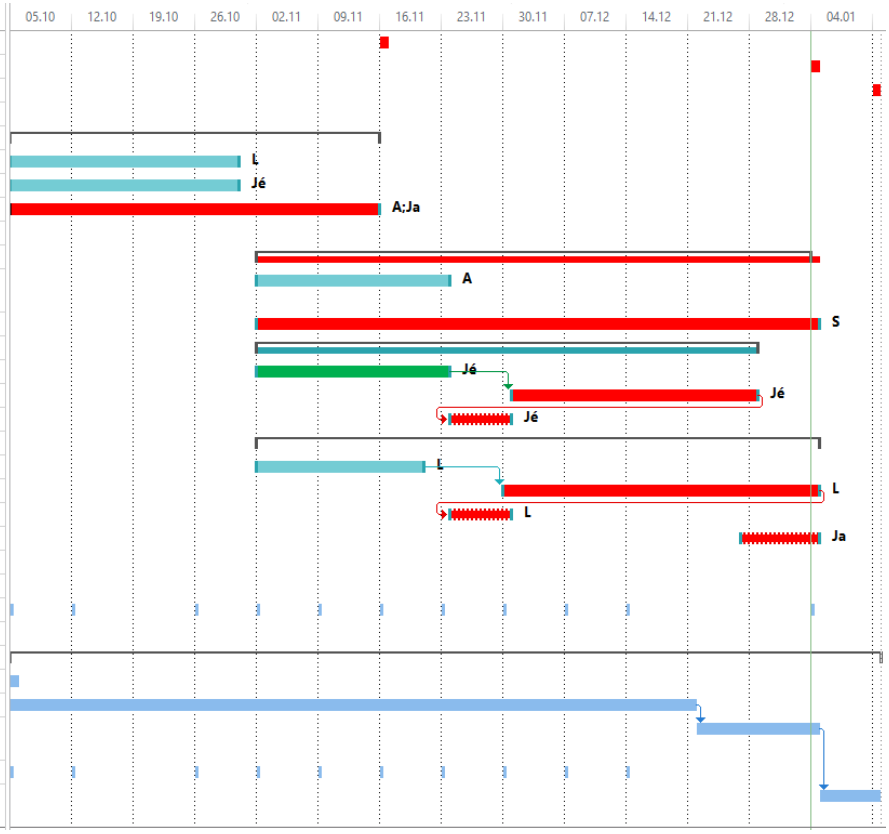
A partir du mois de Novembre, nous rentrons dans la phase d'implémentation dans laquelle nous allons utiliser les différentes méthodes testées pendant la première phase, afin de produire l'application finale.

Au cours du projet, nous avons placé des tâches récurrentes qui doivent être exécuté chaque semaine, comme notamment la séance hebdomadaire et la rédaction de la documentation.

Nous avons aussi ajouté en rouge les différentes dates importantes du projet (état d'avancement, rendu final, et la présentation).

6.2 Planification finale

Nom de la tâche	Début	Fin	Effectif	Heures
Etat d'avancement	Lun 16.11.15	Lun 16.11.15	5	
Rendu final	Lun 04.01.16	Lun 04.01.16	5	
Présentation	Lun 11.01.16	Lun 11.01.16	5	
Conception	**Lun 05.10.15**	**Dim 15.11.15**	**5**	**36**
Librairie vidéo	Lun 05.10.15	Ven 30.10.15	1	24
Librairie son	Lun 05.10.15	Ven 30.10.15	1	24
API	Lun 05.10.15	Dim 15.11.15	2	36
Implémentation	**Lun 02.11.15**	**Lun 04.01.16**	**5**	**54**
Interface de synchronisation	Lun 02.11.15	Lun 23.11.15	1	18
Interface	Lun 02.11.15	Lun 04.01.16	1	54
Partie musique	**Lun 02.11.15**	**Lun 28.12.15**	**1**	**54**
Lecteur	Lun 02.11.15	Lun 23.11.15	1	18
Synchronisation	Mar 01.12.15	Lun 28.12.15	1	28
API	Mar 24.11.15	Lun 30.11.15	1	6
Partie films	**Lun 02.11.15**	**Lun 04.01.16**	**1**	**54**
Lecteur	Lun 02.11.15	Ven 20.11.15	1	18
Synchronisation	Lun 30.11.15	Lun 04.01.16	1	31
API	Mar 24.11.15	Lun 30.11.15	1	6
Partie séries	Dim 27.12.15	Lun 04.01.16	1	6
Séance	**Lun 05.10.15**	**Lun 04.01.16**	**5**	**5**
Documentation	**Lun 05.10.15**	**Lun 11.01.16**	**5**	**84**
Planification	Lun 05.10.15	Lun 05.10.15	2	4
Rapport	Lun 05.10.15	Lun 21.12.15	5	35
Finalisation du rapport	Mar 22.12.15	Lun 04.01.16	1	12
Journal de bord	**Lun 05.10.15**	**Lun 14.12.15**	**5**	**5**
Préparation de la défense	Mar 05.01.16	Lun 11.01.16	5	6



Comme on peut le voir, la planification a été passablement remaniée.

Les tâches en rouge correspondent aux tâches où du retard a été pris, inversement celles en vert ont été réalisées plus vite que prévu. Les tâches hachées ont subi un déplacement dans le temps.

Suite aux différents retards pris sur certaines tâches, nous avons perdu la phase de tests et de réserve que nous avions planifiée initialement. Dans tous les cas, la tâche « synchronisation » a pris plus de temps qu'on ne le pensait. Les tâches « API » ont subi un déplacement dans le temps. Effectivement, elles étaient à la base prévues après « Synchronisation » seulement vu l'ampleur de ces tâches, l'intégration des API a été réalisée directement après l'implémentation de base des lecteurs.

La partie séries a également été déplacée dans le temps. Placé à la base à la semaine du 21 décembre, elle a finalement été réalisée pendant les vacances de Noël.

7 Conclusion

Nous avons réussi a effectué un travail relativement bon dans l'ensemble, en effet, nous avons pu réaliser tout ce que nous désirions : lecteur de vidéos et de musiques, gestion d'une collection, obtentions de métadonnées et synchronisation au niveau de la lecture des films et des musiques, le tout dans une interface relativement proche de ce qui était prévu.

Nous sommes également content d'avoir pu découvrir une technologie que nous ne connaissions pas : JavaFX, même si nous avons dû abandonner l'idée de réaliser un lecteur vidéo avec elle. Il est toujours plaisant de découvrir de nouvelles choses, surtout lorsqu'elle comble un vide, en effet les interfaces réalisées à l'aide de Swing ne sont pas toujours très belles.

Toutefois nous avons rencontrés un gros problème au niveau de l'organisation de notre travail. Chaque membre du groupe a eu tendance à travailler de son côté. Lorsqu'il a fallu mettre en commun notre travail, nous avons rencontrés de nombreuses difficultés, devant parfois refaire une partie du code.

Ceci a eu pour conséquence une finition du projet qui laisse quelque peu à désirer, avec notamment une synchronisation fonctionnelle, mais sans gestion d'erreurs et indisponible pour les séries.

Nous nous sommes montrés un peu trop confiant, sous-estimant fortement la difficulté de mettre en commun notre travail. Nous avons également eut quelques problèmes de communications, n'étant pas toujours au courant de ce que les autres avaient réalisés concrètement, nous avons eu quelques mauvaises surprises.

Malgré ces difficultés nous sommes satisfaits du résultat final.

7.1 Points non-réalisés

Tous les points ont été au moins partiellement réalisés. Il existe encore quelques problèmes (voir section Problèmes Connus), surtout concernant la synchronisation. Certains points n'ont également pas put être réalisé exactement comme prévu :

- Les métadonnées des vidéos ne sont pas toujours toutes extraites. Cela dépend des fichiers, et est une limitation de la solution adoptée pour extraire ces métadonnées.
- La vue série est moins fournie que ce que nous avions prévus, mais présente toutefois l'essentiel : les informations de la série et la liste des épisodes possédés par l'utilisateur.
- Nous n'avons pas pu déployer notre application sur Linux. Nous avons rencontré des problèmes pour linker la librairie VLCJ dans l'exécutable final.
- Gestion d'erreur lors de la synchronisation. Par exemple, lorsqu'un client se déconnecte sans utiliser notre interface.
- Impossible de quitter une vidéo avant la fin. De ce fait, on est obligé d'attendre ou d'avancer la vidéo à la fin.
- Synchronisation lors de la lecture de série. Par manque de temps nous n'avons pas eu le temps d'implémenter cette fonctionnalité, malgré la présence de l'interface.

7.2 Problèmes connus

Il existe encore quelques problèmes au niveau de notre application.

- Une fois que la synchronisation est enclenchée dans le mode vidéo, il n'est pas possible de la stopper. À partir de ce moment, les deux utilisateurs seront synchronisés jusqu'à la terminaison du programme.
- Si un fichier est supprimé de la collection lorsque le programme est lancé, cela peut poser des problèmes si l'utilisateur tente de lancer ce fichier depuis l'interface.
- Si l'application est quittée en cours de lecture d'une vidéo ou d'une musique, celle-ci continuera de jouer même après que l'application se soit fermée.

7.3 Proposition d'améliorations

Voici une liste des quelques propositions d'amélioration possible pour notre projet.

- Streamer les médias (musiques et vidéos) lors d'une synchronisation, ainsi seul un utilisateur a besoin de posséder le fichier.
- Plus de contrôles au niveau des lecteurs (volume, …)
- Pouvoir faire des playlists, particulièrement pour les musiques.
- Fonctions de recherches des médias.
- Amélioration de la collection. En effet, à l'heure actuelle elle assez contraignante et peu pratique.

8 Table des illustrations

9 Signatures

Ce document imprimé est validé et certifié par les auteurs de Flat 5, daté du 4 Janvier 2016.

Baehler Simon

Moret Jérôme

Purro Jan

Berney Léonard

Roubaty Anthony

10 Annexes

Les annexes regroupent les documents qui sont externes au rapport, mais que nous avons utilisés afin de réussir ce projet.

10.1 Journal de travail

Le journal de travail contient le travail effectué par les différents membres du groupe au cours de projet. Le format est différent que nous n'avons pas généralisé le Template.

10.1.1 Moret Jérôme

---

26 octobre – 1 novembre

- Essai avec l'API VLCj pour le cas de la lecture de musique
- Recherche d'informations sur la programmation à l'aide de JavaFX 8
  - Utilisation du modèle MVC
  - Conception d'interface graphique à l'aide du SceneBuilder 2.0

---

2 novembre – 8 novembre

- Conception du lecteur audio Flat 5
  - Play/Pause
  - Double click to play
  - Get local music files

---

9 novembre – 15 novembre

- Conception du lecteur audio Flat 5
  - Previous/Next
  - Récupération des tags id3 et affichage dans des colonnes de tableau
  - Implémentation des temps de début et fin ainsi que du slider
- Conception de la présentation PowerPoint pour l'état intermédiaire

---

16 novembre – 22 novembre

- Structuration du code et commentaires
- Réglage des derniers problèmes
  - « Seeking » qui plante une fois sur deux

        - Prenais en compte la valeur de la lecture automatique au lieu du déplacement
    - Redimensionnement de la fenêtre peut aller jusqu'à écraser les différents composants du BorderLayout

---

23 novembre – 29 novembre

- Intégration de l'API Spotify
    - Définition de la politique de récupération des informations d'un fichier audio
- Ajout d'un nouveau Panel affichant la lecture en cours (photo de l'album, titre, artiste, album)

---

30 novembre – 06 novembre

- Généralisation du mécanisme de récupération des informations sur une musique

---

07 décembre – 13 décembre

- Implémentation du mécanisme de synchronisation

---

14 décembre – 23 décembre

- Corrections de petits problèmes concernant le mécanisme de synchronisation

---

24 décembre

- Configuration de la fenêtre settings
    - Gestion des amis (contacts, nom & adresse IP)
        - Ajout
        - Modification
        - Suppression
    - Couplage avec la base de données SQLite
    - Gestion du chemin des médias
        - Explorateur de dossier

28 décembre

- Fin de l'intégration de l'interface de synchronisation
- Tests rapides du système

28 décembre – 04 janvier

- Écriture du rapport

10.1.2 Baehler Simon

Lundi 12 Octobre 2015

	description	Problèmes rencontrés
Prise en main de JavaFX	Découverte de l'environnement JavaFX et des différents outils mis à disposition. Recherche de tutoriels	-

Lundi 26 Octobre 2015

	description	Problèmes rencontrés
Prise en main de JavaFX et suivie de tutoriel	Découverte de l'environnement JavaFX et des différents outils mis à disposition. Recherche de tutoriels. Début du tuto suivant : http://code.makery.ch/library/javafx-8-tutorial/	-
Création de la class PTableColumn	Cette classe permet de faire des tableaux avec des tailles en pourcent, option n'étant pas offertes par l'élément tableClomn de base de javaFX	-

Lundi 2 Novembre 2015

	description	Problèmes rencontrés
Implémentation du début de la gui	Attaque de la GUI, il y a désormais un menu en haut servant de rootlayout, c-a-d que cette partie sera toujours la	-

Dimanche 8 Novembre 2015

	description	Problèmes rencontrés
Action sur les bouton/navigation	Début de la navigation sur le projet	**Une exception se lève quand nous utilisons les boutons du rootlayout**

Lundi 9 Novembre 2015

	description	Problèmes rencontrés
Action sur les bouton/navigation	Merge des projets de Jerome avec celui de Simon + attaque du css de la partie player	

Lundi 16 Novembre 2015

	description	Problèmes rencontrés
Action sur les bouton/navigation	Implémentation du bouton précédent	

Lundi 23 Novembre 2015

	description	Problèmes rencontrés
Action sur les bouton/navigation	La navbar du haut fonctionne correctement. Il y a désormais un problème quand nous retournons sur la home page	

Lundi 30 Novembre 2015

	description	Problèmes rencontrés

Action sur les bouton/navigation	Correction du problème de retour home, désormais nous créons une seul foi le loader, de plus quand nous accédons à la musique nous créons une seul instance du loader	

Lundi 7 Decembre 2015

	description	Problèmes rencontrés
Vue	Création de la vue des paramètres et tentative de mettre le place la sélection d'un dossier	**Sélection d'un dossier non fonctionnelle**

Lundi 14 Decembre 2015

	description	Problèmes rencontrés
Vue	Création des vue d'informations de film et de série + vue de listing des films et séries	**Affichage à amélioré**

Samedi 26 Decembre 2015

	description	Problèmes rencontrés
Vue	Navigation de test, il est possible de naviguer dans le programme, il n'y a cependant aucune information	**Affichage à amélioré**

Samedi 2 Janvier 2016

	description	Problèmes rencontrés

Vue	Amélioration des vue, merge du travail de Jan, ajout des informations des films dans les vue des films, partie graphique pour la synchronisation des films, modification des images des boutons, modification du css pour les vue	

Dimanche 3 Janvier *2016*

	description	**Problèmes rencontrés**
Vue	Clean du code, documentation	

Lundi 4 Janvier 2016

	description	**Problèmes rencontrés**
Vue	Clean du code, documentation	

10.1.3 Roubaty Anthony

Octobre 2015

26 - Test de diverse APIs pour lire les métadatas des MP3

- Test de l'API Spotify

- Implémentation de classes permettant de gérer les MP3

Novembre 2015

2 - Création du squelette du rapport finale

- Rédaction du rapport

9 - Synchronisation : client / serveur TCP

- Création du SyncManager

- Implémentation dans le projet final

- Javadoc pour la partie TCP

16 - Tests et corrections de bug dans SyncManager

- Slide API et partie TCP

23 - Implémentation d'un contrôleur central

30 - Documentation

Décembre 2015

7 - Mise en place de la synchronisation du lecteur audio

14 - Correction de bugs lors de la synchronisation

- Documentation

Janvier 2015

4 - Finalisation des tests de synchronisation

- Documentation

10.1.4 Purro Jan

Semaine du 12.10.2015

- Recherche d'API pour récupérer les informations de films et séries sur IMDb. Trouver et tester OMDb (application web qui permet de faire des requêtes REST pour obtenir des infos au format JSON.

Semaine du 19.10.2015

- Vacances

Semaine du 26.10.2015

- Implémentation des premières requêtes vers OMDb depuis Java pour tester l'API.
- Création de l'architecture des classes de l'API vidéo.
- Implémentation partielle des requêtes nécessaire à l'API.

Semaine du 02.11.2015

- Fin de l'implémentation des requêtes ver OMDb.
- Implémentation du parsing des réponses d'OMDb (utilisation de la librairie Gson pour parcourir le JSON).
- Recherche d'information sur les métadonnées des fichiers vidéo utilisés

Semaine du 09.11.2015

- Ajout de la gestion de la récupération des informations des séries vers OMDb.
- Test de plusieurs classes permettant de récupérer les métadonnées des fichiers
- vidéos.
- Préparation de la présentation intermédiaire.

Semaine du 16.11.2015

- Rédaction de la documentation concernant l'api vidéo.
- Poursuite des recherches concernant les métadonnées. Finalement le choix se retenu sera celui de l'utilisation de la bibliothèque VLCJ, qui a l'avantage d'être déjà utilisée sur le projet.

Semaine du 23.11.2015

- Ajout de la base de données contenant les informations des contacts et des fichiers de la collection.

Semaine du 30.11.2015

- Mise en place du scan des fichiers de la collection, récupération des métadonnées et des informations à travers les API et ajout des données dans la base de données.

Semaine du 07.12.2015

- Prise en main de JavaFX afin de pouvoir travailler sur les vues films et séries.

Semaine du 14.12.2015

- Réalisation des contrôleurs des vues films et séries.

Semaine du 21.12.2015

- Vacances

Semaine du 28.12.2015

- Finalisation du projet, mise en commun du code, corrections de bugs.
- Finalisation de la documentation du projet.

10.1.5 Berney Léonard

Semaine du 12.10.2015

- Recherche d'une API permettant la lecture de vidéos avec Java.

Semaine du 19.10.2015

- Vacances

Semaine du 26.10.2015

- Découverte de vlcj. Lecture de la documentation et prise en main de la librairie grâce au tutorial.

Semaine du 02.11.2015

- Début de la conception d'un lecteur vidéo.

Semaine du 09.11.2015

- Création d'un lecteur vidéo avec JavaFx et constatation que la technologie n'est pas appropriée.

Semaine du 16.11.2015

- Création d'un overlay pour les contrôles du lecteur vidéo.

Semaine du 23.11.2015

- Constatation qu'un overlay n'est pas une bonne option et qu'il est plus approprié d'intégrer les contrôles directement dans la fenêtre du lecteur.

Semaine du 30.11.2015

- Implémentation de des contrôles du lecteur vidéo.

Semaine du 07.12.2015

- Conception du protocole de synchronisation.

Semaine du 14.12.2015

- Implémentation de la synchronisation.

Semaine du 21.12.2015

- Vacances

Semaine du 28.12.2015

- Finalisation du projet, mise en commun du code, corrections de bugs.
- Finalisation de la documentation du projet.

10.2 Cahier des charges

Voir les documents qui suivent le rapport.

10.3 Manuel d'utilisation

Voir les documents qui suivent le rapport.