|  |  |
| --- | --- |
|  | **2015-16** |
|  | HEIG-VD, section TIC  Auteurs: Baehler Simon,  Moret Jérôme,  Purro Jan,  Berney Léonard &  Roubaty Anthony  Date du rendu finale du projet : 4 janvier 2016  Professeur: René Rentsch |

C:\Git\PDG_Project\Rapport\img\logoHeigVd.png

|  |
| --- |
| **[Documentation Flat5]** |
| Rapport du projet de semestre nommé Flat5 |

Flat 5

Table des matières

[1 Introduction 2](#_Toc434240037)

[1.1 Choix de projet 2](#_Toc434240038)

[1.2 Objectif de Flat 5 2](#_Toc434240039)

[1.3 Utilisation de l’application 2](#_Toc434240040)

[2 Mise en place 3](#_Toc434240041)

[2.1 Gestion des fichiers 3](#_Toc434240042)

[3 Analyse 3](#_Toc434240043)

[3.1 Fonctionnement de base 3](#_Toc434240044)

[3.2 Base de données 3](#_Toc434240045)

[3.3 Lecteur vidéo 3](#_Toc434240046)

[3.4 Lecteur audio 3](#_Toc434240047)

[3.5 API web 3](#_Toc434240048)

[3.6 Interface 4](#_Toc434240049)

[4 Réalisation 5](#_Toc434240050)

[4.1 Lecteur vidéo 5](#_Toc434240051)

[4.2 Lecteur audio 5](#_Toc434240052)

[4.3 API web 5](#_Toc434240053)

[4.4 Synchronisation 5](#_Toc434240054)

[5 Points non-réalisés 5](#_Toc434240055)

[6 Problèmes connus 5](#_Toc434240056)

[7 Planification 5](#_Toc434240057)

[7.1 Planification initial 5](#_Toc434240058)

[7.2 Re-planification 5](#_Toc434240059)

[8 Conclusion 5](#_Toc434240060)

[9 Table des illustrations 5](#_Toc434240061)

[10 Signatures 6](#_Toc434240062)

[11 Annexes 8](#_Toc434240063)

[11.1 Journal de travail 8](#_Toc434240064)

[11.2 Cahier des charges 8](#_Toc434240065)

[11.3 Manuel d’utilisation 8](#_Toc434240066)

# 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.

## Choix de projet

A la différence avec PRO, dans le cours PDG nous n’avons pas de restriction de langage. La seul restriction imposé 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 si 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 facilité 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.

## 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.

## 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.

# Mise en place

TODO : comment s’est séparer le boulot ! (par groupe, etc…)

## 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>

# Analyse

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

## Fonctionnement de base

## Base de données

## Lecteur vidéo

## Lecteur audio

## API web

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

Pour les films, TODO FILM

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 tombé sur la repertoire 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…).

## Interface

# Réalisation

## Lecteur vidéo

## Lecteur audio

## 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 trouvés sur Spotify, avec notamment l’URL de la pochette.

## 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 « 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 comparé et d’informer l’utilisateur le cas où c’est différent.
* **pause** : indique que l’utilisateur a appuyé sur pause.
* **playAt-#Seconde** : demande au lecteur de lire le média à partir de N seconde.
* **setAt-#Seconde** : demande au lecteur de se positionner à N seconde.
* **bye** : fin de la synchronistion

# Points non-réalisés

# Problèmes connus

# Planification

## Planification initial

## Re-planification

# Conclusion

# Table des illustrations

# Signatures

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

Baehler Simon

Moret Jérôme

Purro Jan

Berney Léonard

Roubaty Anthony

# Annexes

## Journal de travail

## Cahier des charges

## Manuel d’utilisation