

Bessel Kobeissi

Florian Morin

Raphaël Ptit Haddad

## **Rapport GLPOO**

### **1. Liste étudiant et travaux effectués**

Dans un premier temps nous avons conceptualisé l'application, pour cela BESSEL KOBEISSI a conçu des design patterns que nous pourrions appliquer au code. Nous avons principalement retenu la Facade, la Factory ainsi que l'adaptateur. Il a également veillé tout au long du projet à respecter lui-même, mais également ses collègues, les principes SOLID et BICEP.

Pendant ce temps FLORIAN MORIN s'est chargé de réaliser un fichier pom.xml ainsi qu'une arborescence au projet afin de faire fonctionner MAVEN. Il s'est également occupé d'intégrer les dependency lorsqu'un membre de l'équipe en avait besoin et de les intégrer au projet. De plus il s'est occupé des Test unitaire avec leurs critères d'acceptance et du GITHUB. Concernant le code à proprement parler il a réalisé la lecture d'une musique ainsi que la mise à jour automatique des informations sur le serveur. Enfin il a réalisé pratiquement seules les deux classes Client et Server utile à l'interaction entre les 2 entités.

Enfin RAPHAEL PTIT HADDAD a réalisé l'organisation du projet, à savoir les Backlog Stories, des différents diagrammes UML ainsi que de la liste des exigences fonctionnelles. Par rapport au code, il a aidé BESSEL dans la mise en pratique des principes solides en conceptualisant certaines interactions entre les classes. Il également travaillait sur la réalisation du main et du Thread nécessaire à l'écoute audio.

Chaque personne a travaillé sur sa propre documentation de son code.

### **2. Liste des exigences fonctionnels et non fonctionnels**

#### **Exigences fonctionnelles :**

EF_1	L'utilisateur aura un menu où chaque fonctionnalité de l'application sera inscrite
EF_2	L'utilisateur pourra mettre à jour ses données via une commande sans obligation d'utilisation
EF_3	L'utilisateur pourra ajouter un élément à un album
EF_4	L'utilisateur pourra créer une playlist à partir des Eléments déjà disponible
EF_5	L'utilisateur pourra supprimer une playlist et un album
EF_6	L'utilisateur pourra écouter un élément à partir du menu
EF_7	L'utilisateur pourra mettre en pause et reprendre un morceau en cours d'écoute
EF_8	L'utilisateur pourra accélérer ou reculer de 30secs la musique
EF_9	Après qu'une commande est terminée son exécution, l'utilisateur reviens au menu principal et est notifié visuellement
EF_10	Durant la lecture d'un morceau, l'utilisateur voit son avancé temporel dans celle-ci

### **Exigences non fonctionnelles :**

ENF_1	La communication entre le serveur et le client se fera ponctuellement
ENF_2	Le serveur peut recevoir plusieurs connexions durant son exécution
ENF_1	Les communications du serveur peuvent provenir de plusieurs clients durant son exécution
ENF_3	Le client stockera localement le fichier qu'il écoute durant son utilisation et sera supprimer dès qu'il n'en aura plus besoin
ENF_4	Les classes qui servent au stockage des informations seront identiques entre le client et le serveur
ENF_5	L'affichage du menu ainsi que des listes seront colorés pour permettre leur visibilité
ENF_6	Le serveur fonctionnera en local « localhost »
ENF_7	Il n'y a pas de dépendance entre le lancement du client et du serveur, les deux peuvent être arrêté ou lancé indépendamment
ENF_8	Le système de lecture audio et d'interface utilisateur doivent être indépendant durant la lecture d'un fichier audio
ENF_9	Les erreurs et différentes étapes de fonctionnement du client ou serveur doivent figurer dans un fichier de log

### **3.Diagramme UML :**

Les diagrammes des cas d'utilisation et de séquence se trouvent dans le dossier Diagrammes sous des noms explicite selon leurs fonctions.

### **4.Backlog, User Stories, Constraint Stories**

Lien vers le trello : [Organisation | Trello](#)

Pour s'organiser concrètement dans ce projet nous avons donné à RAPHAEL la charge de product owner, il s'est occupé de nous donner à chacun nos charges, pour cela nous avons séparé le travail comme dit précédemment. La répartition du travail que nous avons faite, ne nous a pas demandé de dépendance entre nous et nous avons donc pu évoluer à chacun à son rythme. Nous avons fait 3 réunions visibles sur Trello qui nous ont permis de mettre à jour la réalisation de chacun.

Concrètement nous avons donné à chacun l'ensemble de ses tâches, qu'elles soient relatives à l'organisation du projet ou bien au code, puis nous avons laissé chacun travaillé naturellement sur sa partie. Nous faisons de manière non-officielle des points entre nous pour évaluer l'avancée de chacun et discuter de nos projets sans que cela soit particulièrement consigné. Cependant nous avons consigné les grandes étapes au sein du Trello durant les quelques réunions que nous avons faite.

### **5. Diagrammes UML :**

Les diagrammes de Package, Composant, Déploiement, Décomposition en module, Classes de l'application se trouvent dans le dossier Diagrammes, sous les noms correspondant à leur fonction.

## **6. Patrons utilisés**

Dans ce projet nous avons utilisé 3 patrons. Le premier et le plus évident est la façade, il a pour objectif de cacher le fonctionnement des listes et simplifier leur utilisation, ainsi le menu est dépendant de la façade et non plus des différentes listes que nous utilisons dans le projet.

Le second est la factory sous 2 formes, nous les avons utilisés dans le cadre d'une abstraction au travers d'interface. Nos interfaces sont utilisées pour créer des playlists ainsi que des albums notamment, nous avons donc voulu simplifier l'implémentation d'autres type de stockage comme des Vinyle ou Best-of par exemple. Ainsi si nous ajoutons un type de stockage nous n'aurons qu'à modifier le code d'une factory afin de permettre la création d'un nouveau type de contenu.

Le dernier patron utilisé est le singleton qui correspond à une variable mutable au travers des applications, cette classe a pour objectif de transmettre des informations entre les méthodes. Etant donné que nous ne transmettons les informations qu'une fois et de manière synchrone nous avons jugé judicieux d'appliquer un singleton à cette classe.

## **7. Analyse selon les principes SOLID**

Afin de montrer l'application des principes SOLID nous expliquerons d'abord les classes communes entre les 2 principales applications (serveur/ client) et puis nous verrons les classes réservées à chacune d'elles.

### **Responsabilité Unique :**

Afin de séparer les responsabilités correctement nous avons établi une liste de responsabilité issue de besoin et nous avons conçu chaque classe afin de correspondre au besoin ainsi :

Responsabilité	Classe
Manipulation d'une Chanson	Chanson
Manipulation d'une liste de Chanson	ChansonVolatile
Manipulation d'un Livre Audio	LivreAudio
Manipulation d'une liste de livres audios	LivreAudioVolatile
Manipulation d'un album	Album
Manipulation d'une Playlist	Playlist
Manipulation d'une liste d'album	AlbumVolatile
Manipulation d'une liste de Playlist	PlaylistVolatile
Echange de messages	Client/Serveur (selon le package)
Ecriture des XML	(Uniquement sur le serveur) Une classe destinée à chaque type d'information à écrire
Récupérer les informations de fichiers physique	AutomaticUpdate
(Serveur)Envoi du fichier que le client veut écouter	PlaySound
(Client)Ecoute du fichier souhaité	PlaySound

Afin d'appliquer les principes Ouverts/Fermés que nous verrons après, nous avons regroupé des fonctionnalités dans diverses interfaces :

Responsabilité	Classe
Manipuler des fichiers audios	Stockage
Manipuler des listes de fichiers audios	StockageVolatile
Manipuler des listes de listes de fichiers audios	StockageMaster

Ainsi chaque Stockage/StockageVolatile/StockageMaster a le même comportement, stocker un ArrayList, et une série de méthodes permettant d'interagir avec ses listes. De plus leur construction n'est pas dépendante d'autres classes et donc chaque liste à une certaine responsabilité, manipuler sa liste. Pour la communication entre client et serveur toute échange de donnée se fait depuis les classes Client ou le Serveur ainsi chacune à l'unique responsabilité d'échanger différents types de données.

En ce qui concerne les autres classes chacune à également une responsabilité unique et parfaitement établie. Par exemple nous avons eu besoin de lire les fichiers physiques afin d'obtenir les informations relatives au Chansons ou LivreAudio, il s'agissait donc d'un besoin qui a été traduit en une fonctionnalité au sein d'une classe. Cette classe a donc eu une responsabilité unique pour récupérer des informations des fichiers physiques.

#### **Pour récapituler les différentes responsabilités :**

Chaque liste est stockée dans une classe qui a pour unique but de servir de Façade à la liste. L'interaction Client-Serveur se fait à l'aide de 2 classes qui ont chacune pour unique objectif d'envoyer ou recevoir chaque donnée. Les autres classes utiles telles que la mise à jour automatique, la façade ou la lecture audio sont réalisés respectivement par une seule classe. Nous avons même séparé les responsabilités entre la manipulation audio et la lecture des entrées utilisateurs pour la lecture audio. En effet un Thread est ouvert grâce à une classe pour récupérer les entrées utilisateurs, et une autre classe à pour objectif de lire les fichiers audios, ainsi les 2 classes sont indépendantes en matière de responsabilité et ont un lien de ségrégation à l'aide d'une classe mutable.

#### Ouvert/Fermé :

Afin d'appliquer le principe Ouvert/fermé à notre code, nous nous sommes tout d'abord occupés des 3 principales classes de manipulations de fichier, les classes de stockage. Pour cela nous avons mis uniquement les méthodes nécessaires à la manipulation de la liste stockée dans celle-ci. Pour cela nous avons établi les fonctionnalités nécessaires dans l'utilisation de celle-ci afin de toutes les implémenter. Ainsi les différentes classes peuvent être étendues à l'aide d'une extension pour overwrite leurs méthodes et modifier la manière dont la manipulation des listes est effectuée, cependant nous avons pris soin de mettre initialement toutes les fonctionnalités de manipulation.

De plus nous avons pris soin de passer par des interfaces afin de mettre une abstraction qui permet de réécrire une classe où seront implémentées des méthodes propres au fonctionnement du reste du code. Par exemple nous avons passé le client et le serveur à travers des abstractions grâce aux interfaces afin de permettre une implémentation ou un système d'envoi de musique tout à fait différent, de même pour la lecture audio.

Pour résumer nous avons conçu chaque classe avec les informations nécessaires pour correspondre à leur responsabilité, tout en ayant appliqué des interfaces permettant de modifier les méthodes si nous remplaçons la classe très facilement.

### Substitution de Liskov :

Etant donné que chaque classe contient le strict nécessaire nous n'avons pas eu besoin de substituer les différentes classes que nous utilisons à l'aide de casts. En effet nous utilisons ce que nous avons déterminé comme essentiel pour chaque classe. Par exemple l'obtention d'un élément d'une liste ne nécessite pas de cast car les types correspondent à des interfaces.

### Ségrégation des interfaces :

Comme dit récemment nous avons tout d'abord conceptualisé chaque classe afin qu'elle est une responsabilité très unique. Par exemple nous n'avons pas conçu les interfaces pour que nous ajoutons un élément dans une liste sans pouvoir l'y retirer, ainsi l'ensemble des méthodes des classes sont liées. Nous n'avons pas eu besoin de décompenser les différentes interfaces que nous avons déjà pour ségréger leurs utilisations.

### Inversion des dépendances :

Nous avons conçu des Interfaces afin de pouvoir faire des abstractions des différentes classes. Par exemple aucune classe n'est réellement dépendante du serveur mais uniquement des méthodes appliquer dans le serveur pour connaître les outils qui permettent de communiquer. Ce principe est appliqué à toutes nos classes et donc nous avons des dépendances uniquement des interfaces qui sont conçues pour avoir le strict nécessaire, et donc sont parfaitement ouvert à l'extension.

## **8.GITHUB**

[Hikachhu/Glpoo Bessel Raph Florian \(github.com\)](https://github.com/Hikachhu/Glpoo)

## **9.Liste des tests d'acceptances exécutés, liste de bug trouvés**

User Story	Critère d'acceptance	Validation
En tant qu'utilisateur de la partie cliente Lorsque je démarre l'application Je vois l'ensemble des fonctionnalité de celle-ci	Quand un utilisateur lance l'application, il aura directement l'affichage de l'aide	Validé
En tant qu'utilisateur, au démarrage de l'application je veux pouvoir utiliser une fonctionnalité à l'aide d'une lettre	L'entrée de l'utilisateur ne doit contenir qu'une seule lettre	Validé
	Le sélecteur de fonctionnalité aura pour entrer une seule lettre	Validé
En tant qu'utilisateur, je veux pouvoir mettre à jour les informations à partir du menu	Mise à jour des données parmi les fonctionnalités accessible depuis le menu	Validé
	Une fonctionnalité permettra de mettre à jour les informations disponibles pour Validé l'utilisateur	Validé

En tant qu'utilisateur, je veux pouvoir ajouter un album ou une playlist depuis le menu	Ajout d'une fonctionnalité permettant d'ajouter un élément dans une playlist	Validé
	Ajout d'une fonctionnalité permettant d'ajouter un élément dans un album	Validé
En tant qu'utilisateur, je veux pouvoir jouer une musique depuis le menu	Accès au système d'écoute depuis le menu	Validé
En tant qu'utilisateur, lorsque je joue une musique, je veux pouvoir avant, reculer, mettre en pause et reprendre la musique.	Séparation de l'écoute en 2 parties liées, une pour la gestion audio, une pour les entrées utilisateurs	Validé
	Création d'un tunnel de communication entre les deux parties afin de les faire communiquer	Validé
En tant qu'utilisateur, lorsqu'un fichier audio est joué, je veux voir l'avancé en temps de mon écoute	Récupération des variables relative à la progression de lecture du fichier audio	Validé
	Conversion des informations pour les rendre lisibles pour l'utilisateur	Validé
En tant qu'utilisateur, lorsque j'ai terminé d'utiliser une fonctionnalité, je veux pouvoir en réutiliser une ensuite	Implémentation d'une boucle avec une condition de sortie selon l'entrée de l'utilisateur	Validé
	Une fois qu'une fonctionnalité à été utiliser, notification visuel qu'il est possible d'utiliser une autre fonctionnalité	Validé
En tant qu'utilisateur, lorsque je suis dans le menu, je veux pouvoir supprimer une playlist ou un album	La fonctionnalité de suppression pour la playlist et l'album sont disponible depuis le menu	Validé
	La méthode de suppression de playlist vérifie que nous supprimons une playlist qui existe	Validé

## **10.Critère right-BICEP des tests**

Nous avons réussi à tester la plupart des classes, cependant nous ne savions pas comment tester certaines classes dont voici la liste : Client, Serveur, PlaySound, KeyThread. En ce qui concerne les classes client et serveur, nous ne sommes pas parvenus à tester les envois de messages dans une classe de test à partir des classes fonctionnelles que nous avons. De plus pour la classe PlaySound, nous n'avons pas trouvé de moyen de tester efficacement la lecture audio à partir de la classe que nous avons. Enfin la classe KeyThread se charge directement de récupérer les entrées utilisateurs et nous n'avons pas trouvé à simuler cette fonctionnalité simplement à partir du code que nous avons.

## Les critères right-BICEP sont les suivants :

### 1. Résultat correct

Tous les tests que nous effectuons nous fournissent des résultats cohérents par rapport aux entrées que nous fournissons.

### 2. Vérifications aux limites

Lorsque nous testons les classes nous essayons d'y placer des valeurs étranges afin de vérifier si les constructeurs réagissent bien et enregistrent bien une valeur. De plus pour les suppressions nous vérifions également si l'élément existe. Ainsi nous vérifions les différentes limites possibles des classes que nous testons.

### 3. Relations inverses

Nous appliquons spécifiquement le test de relations pour le fichier XML. En effet nous vérifions si l'écriture et la lecture d'un fichier XML se compensent et donc permettant d'inverser l'effet de chacune, pour cela nous écrivons le fichier avec des informations connues, puis nous le lisons. Si la relation inverse est vérifiée, nous devons récupérer les informations initialement écrites.

### 4. Vérification par d'autre moyen

Nous pouvons rarement tester par d'autre moyens nos classes, par exemple la gestion des fichiers XML ne permet pas de tester autrement que par une relation inverse. Nous vérifions de différentes manières si les classes récupèrent les bonnes valeurs au travers des différents accesseurs à savoir, le toString() et les différents get() et ce pour différents niveaux d'inclusion de liste.

### 5. Forcer l'apparition d'erreur

Etant donné que nous vérifions les différentes entrées utilisateurs afin de ne pas créer de bug, nous ne pouvons pas forcer l'apparition de bug en dehors du domaine d'application que nous ne maîtrisons pas telles que des bugs inerrant à Windows ou au terminal lui-même.

### 6. Vérification des performances.

Les seules performances que nous pourrions tester dans ce projet viendraient de la réception des informations entre le client et le serveur ou bien de la lecture audio, en dehors de ces 2 classes nous ne pouvons pas tester de performances, et nous ne sommes pas parvenu à tester ces 2 classes.

Nous pouvons ici voir les résultats des tests que nous effectuons tout d'abord pour le serveur :

Element	Class, %	Method, % ▲	Line, %
main	0% (0/1)	0% (0/2)	0% (0/55)
util	66% (6/9)	50% (17/34)	44% (195/441)
business	82% (14/17)	74% (77/103)	44% (298/664)

Element	Class, %	Method, % ▲	Line, %
PlaySound	0% (0/1)	0% (0/2)	0% (0/22)
AutomaticUpdateXML	0% (0/1)	0% (0/5)	0% (0/84)
Serveur	0% (0/1)	0% (0/8)	0% (0/113)
MutableInt	100% (1/1)	71% (5/7)	83% (15/18)
WriteAlbumVolatile	100% (1/1)	100% (2/2)	91% (42/46)
WriteChansonVolatile	100% (1/1)	100% (2/2)	94% (34/36)
WriteLivreAudioVolatile	100% (1/1)	100% (2/2)	94% (35/37)
WritePlaylistVolatile	100% (1/1)	100% (2/2)	89% (49/55)
WriteVolatile	100% (1/1)	100% (4/4)	66% (20/30)



















Element	Class, %	Method, % ▲	Line, %
StockageMaster	0% (0/1)	0% (0/1)	0% (0/1)
StockageVolatile	0% (0/1)	0% (0/1)	0% (0/1)
FacadeMenu	0% (0/1)	0% (0/13)	0% (0/212)
PlaylistVolatile	100% (1/1)	37% (3/8)	11% (10/89)
Stockage	100% (1/1)	66% (4/6)	50% (9/18)
AlbumVolatile	100% (1/1)	75% (6/8)	38% (27/70)
ChansonVolatile	100% (1/1)	83% (5/6)	70% (19/27)
LivreAudioVolatile	100% (1/1)	83% (5/6)	75% (25/33)
FactoryOfStockageMaster	100% (1/1)	100% (1/1)	77% (7/9)
FactoryOfStockageVolatile	100% (1/1)	100% (1/1)	77% (7/9)
Categorie	100% (1/1)	100% (3/3)	100% (6/6)
Langues	100% (1/1)	100% (3/3)	100% (6/6)
Genre	100% (1/1)	100% (4/4)	100% (9/9)
Chanson	100% (1/1)	100% (9/9)	100% (35/35)
LivreAudio	100% (1/1)	100% (9/9)	100% (39/39)
Playlist	100% (1/1)	100% (10/10)	97% (36/37)
Album	100% (1/1)	100% (14/14)	100% (63/63)

Puis pour le Client :

Element	Class, % ▲	Method, %	Line, %
main	0% (0/1)	0% (0/2)	0% (0/61)
util	25% (1/4)	21% (5/23)	4% (15/340)
business	83% (15/18)	71% (73/10...	37% (230/...

Element	Class, % ▲	Method, %	Line, %
Client	0% (0/1)	0% (0/9)	0% (0/144)
KeyThread	0% (0/1)	0% (0/4)	0% (0/39)
PlaySound	0% (0/1)	0% (0/3)	0% (0/139)
MutableInt	100% (1/1)	71% (5/7)	83% (15/18)



Element	Class, %	Metho... ▲	Line, %
 StockageMaster	0% (0/1)	0% (0/1)	0% (0/1)
 StockageVolatile	0% (0/1)	0% (0/1)	0% (0/1)
 FacadeMenu	0% (0/1)	0% (0/12)	0% (0/152)
 PlaylistVolatile	100% (1/1)	25% (2/8)	10% (9/87)
 AlbumVolatile	100% (1/1)	75% (6/8)	34% (27/79)
 ChansonVolatile	100% (1/1)	83% (5/6)	70% (19/27)
 LivreAudioVolatile	100% (1/1)	83% (5/6)	75% (25/33)
 Stockage	100% (1/1)	83% (5/6)	55% (10/18)
 Chanson	100% (1/1)	87% (7/8)	41% (14/34)
 LivreAudio	100% (1/1)	88% (8/9)	41% (16/39)
 Playlist	100% (1/1)	90% (9/10)	62% (23/37)
 Album	100% (1/1)	92% (13/14)	65% (41/63)
 Adapter	100% (1/1)	100% (1/1)	100% (11/...
 FactoryOfStockageMaster	100% (1/1)	100% (1/1)	77% (7/9)
 FactoryOfStockageVolatile	100% (1/1)	100% (1/1)	77% (7/9)
 Categorie	100% (1/1)	100% (3/3)	100% (6/6)
 Langues	100% (1/1)	100% (3/3)	100% (6/6)
 Genre	100% (1/1)	100% (4/4)	100% (9/9)

## BUG

Durant le développement de l'application nous avons trouvé différents bugs de criticité variables. Le plus critique concernait la manipulation de fichier audio et le lecteur de ceux-ci. Nous avons également dû gérer les potentiels bugs lié à l'utilisation du réseau. Néanmoins nous avons réussi à traiter chaque bug et à l'empêcher notamment grâce aux exceptions, nous n'avons donc pas de bug connu dans nos applications.