esiea

RAPPORT DE PROJET

Projet GLPOO

Equipe:

Chidiac Bryan

Leopold Arnaud

Miconnet Sandrine

Prospa Florence

Groupe:

Classe 3A-33

Professeur:

Mme Ionescu

3A - S2 2021

Sommaire

I- L'organisation du projet

Notre équipe est composée de 4 étudiants de la classe 33:

_ CHIDIAC Bryan

_ LEOPOLD Arnaud

_ MICONNET Sandrine

_ PROSPA Florence

Nous avons décidé de diviser le travail en fonction du travail demandé, d'un côté les tâches de gestion du projet avec l'ensemble des rendus "papiers" et de l'autre les tâches techniques avec l'écriture du code du programme. Néanmoins, chaque élève a eu des productions à réaliser dans l'ensemble des 2 tâches décrites afin de s'améliorer et d'appliquer le cours.

Bryan avait pour rôle de s'occuper des implémentations des patrons. Il a également réalisé l'UML de séquence, les US et leurs AC et a fini l'implémentation du MVC.
Arnaud avait pour rôle de parvenir à lire les fichiers MP3 sur notre client. Il a également réalisé l'UML des cas d'utilisation, les US et leurs UA.
Sandrine avait pour rôle de réaliser le squelette du patron de conception, de faire le script Maven et de réaliser les tests unitaires. Elle s'est aussi occupée d'écrire le rapport, de faire l'UML de déploiement, de composants et les CS.
Florence avait pour rôle d'adapter le music Hub du S1 avec la gestion client/serveur demandé pour ce projet, elle a codé les interfaces et les classes dédiées à ce point. Elle a également réalisé l'UML de package, de classe, les exigences fonctionnelles et non-fonctionnelles.

II- La spécification

1. Liste des exigences fonctionnelles et non-fonctionnelles

ID	Nom	Type	Description
SY_REQ_1	Fonctionnalité de base	F	Le MusicHub permettra d'écouter le contenu musical
SY_REQ_2	Portabilité	NF	L'application console est seulement disponible sur PC
SY_REQ_3	Connexion serveur	F	Le client pourra se connecter au serveur
SY_REQ_4	Mise à jour	F	L'application mettra à jour automatiquement du contenu sur le serveur
SY_REQ_5	Mise à Jour	F	L'application permettra la mise à jour des données du serveur affichées sur le client via l'utilisation d'une commande précise du client
SY_REQ_6	Affichage	F	L'application permettra l'affichage des listes des chansons, des albums et des playlists disponibles sur le serveur sur l'écran du client
SY_REQ_7	Création	F	Le client pourra créer des albums, chansons, ou des playlists sur le serveur
SY_REQ_8	Ajout	F	Le client pourra ajouter ses propres musiques à l'application
SY_REQ_9	Modification	F	Le client pourra rajouter des éléments audio à des albums/ playlists déjà existants

SY_REQ_10	Ecoute	F	L'application permettra d'écouter les morceaux de musique sur le terminal du client
SY_REQ_11	Interopérabilité	NF	L'application sera développée exclusivement en Java
SY_REQ_12	Stockage	NF	Les données ne seront pas persistantes sur le client
SY_REQ_13	Interopérabilité	NF	Tous les fichiers de données audios seront stockées dans des fichiers XML
SY_REQ_14	Gestion d'erreurs	NF	L'application disposera d'un système de journalisation des erreurs
SY_REQ_15	Signalements	F	L'application côté serveur, permettra l'affichage dans un fichier texte des erreurs survenues lors de l'exécution avec l'horodatage de celles-ci
SY_REQ_16	Aide	F	L'application proposera un menu d'aide pour l'utilisateur
SY_REQ_17	Options	F	L'application proposera un menu d'options pour changer la vitesse d'écoute et la qualité du son
SY_REQ_18	Modification	F	L'application permettra au client de supprimer et/ ou de modifier le contenu du MusicHub
SY_REQ_19	Précision	NF	L'application permettra au client de voir 100% du contenu du MusicHub lors de la connexion

2. Le diagramme UML des cas d'utilisation, de séquence

Diagramme UML des cas d'utilisation:

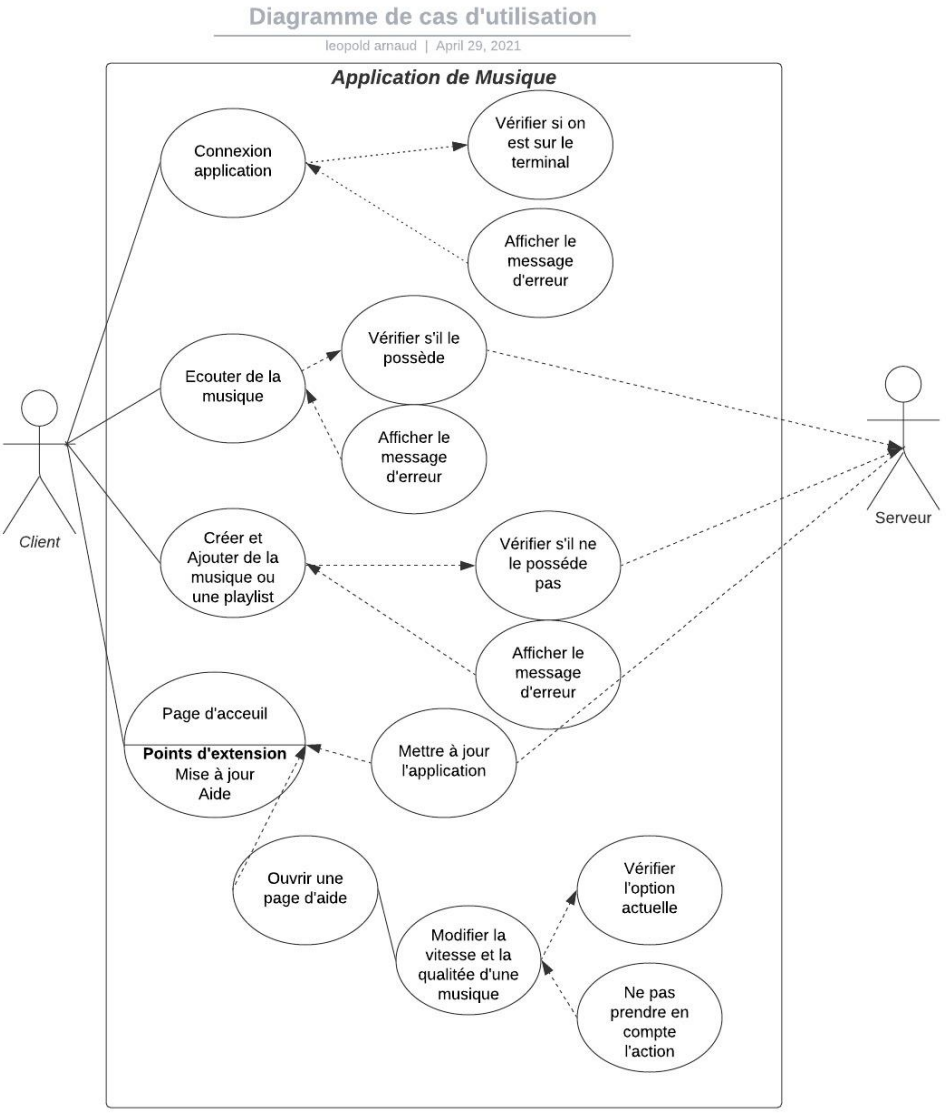
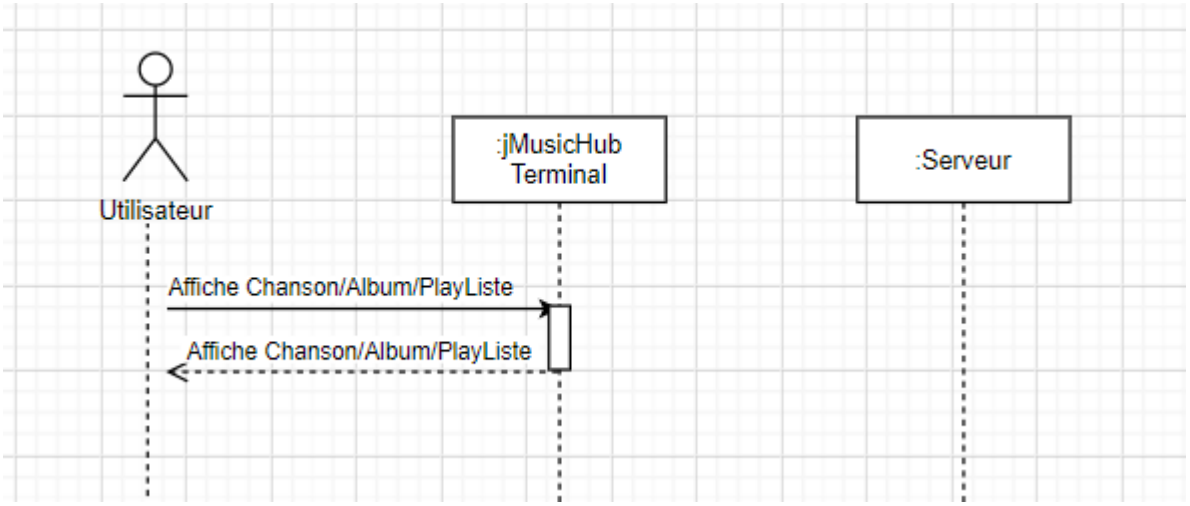
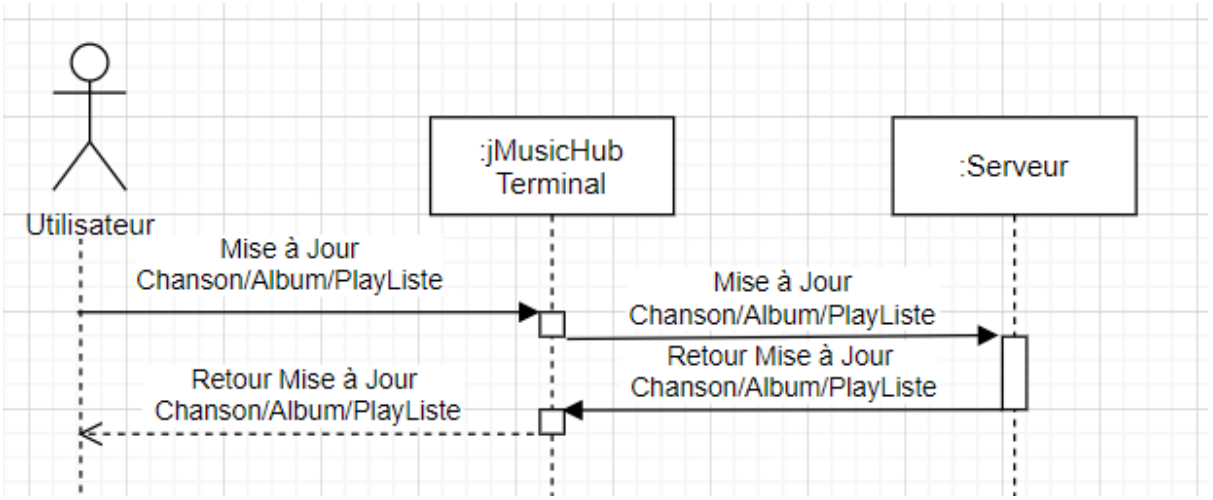


Diagramme de séquences:

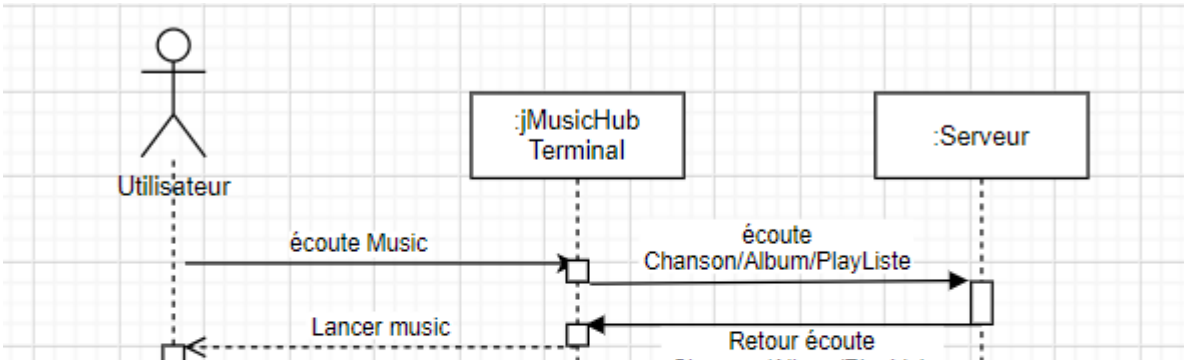
Affichage Chanson/Album/PlayList:



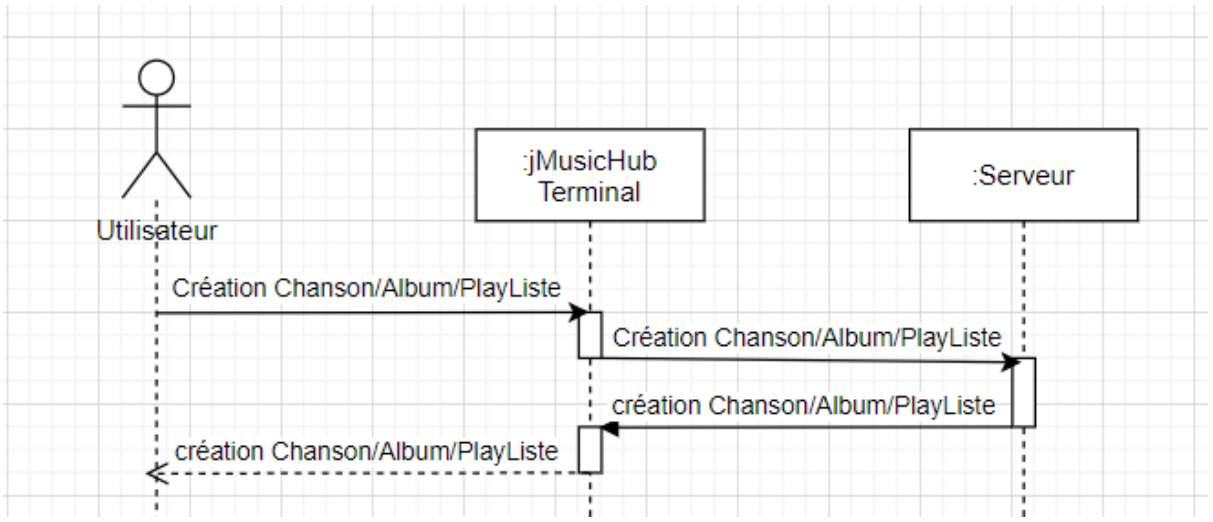
Mise à jour:



Ecoute Music:



Création Chanson/Album/PlayListe:



3. Le backlog des User Stories et Constraint Stories

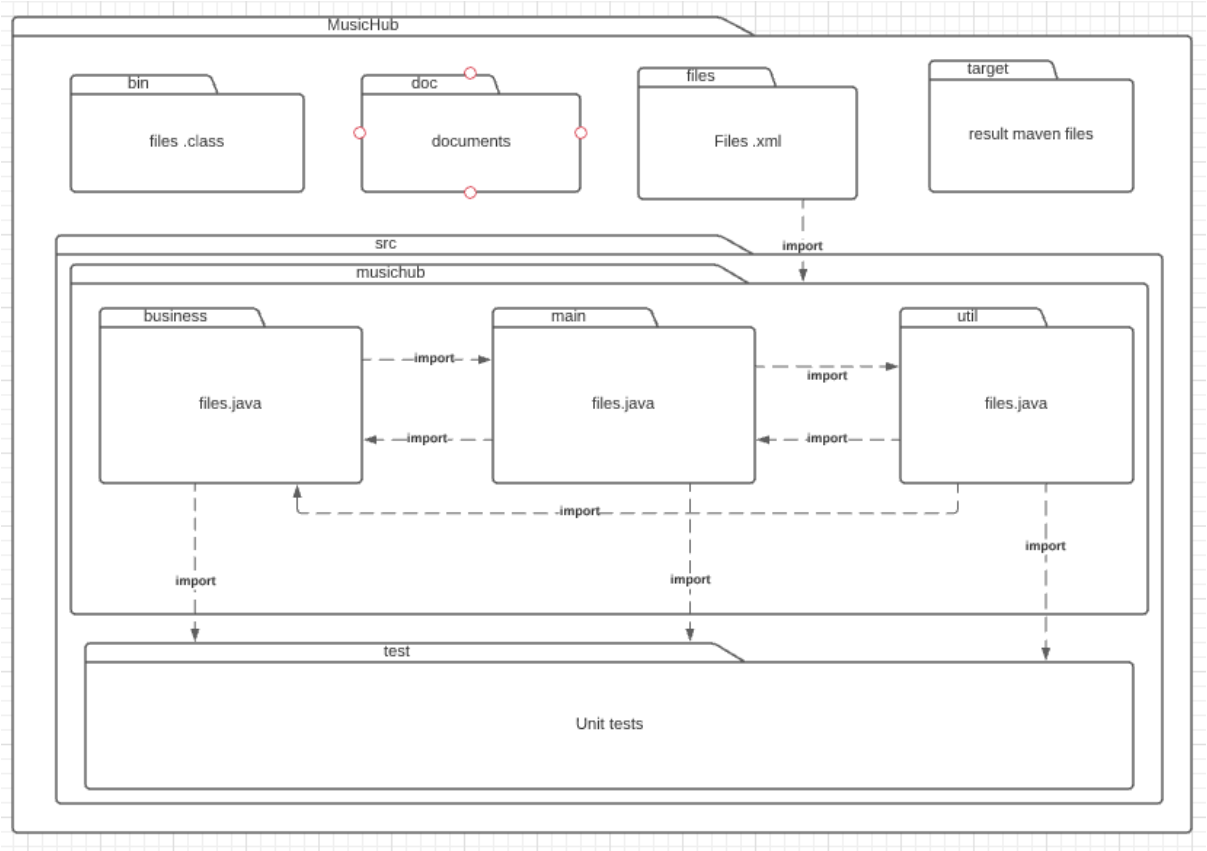
Voici le lien vers notre Trello afin de consulter nos différentes User Stories ainsi que les Constraint Stories accompagnées de leurs critères d'acceptances :

https://trello.com/b/L4wyM1UR/backlog

III- La conception

1. Les 4 diagrammes UML: package, composants, déploiement, classes

Diagramme UML de package:



Dans le package “musichub”:

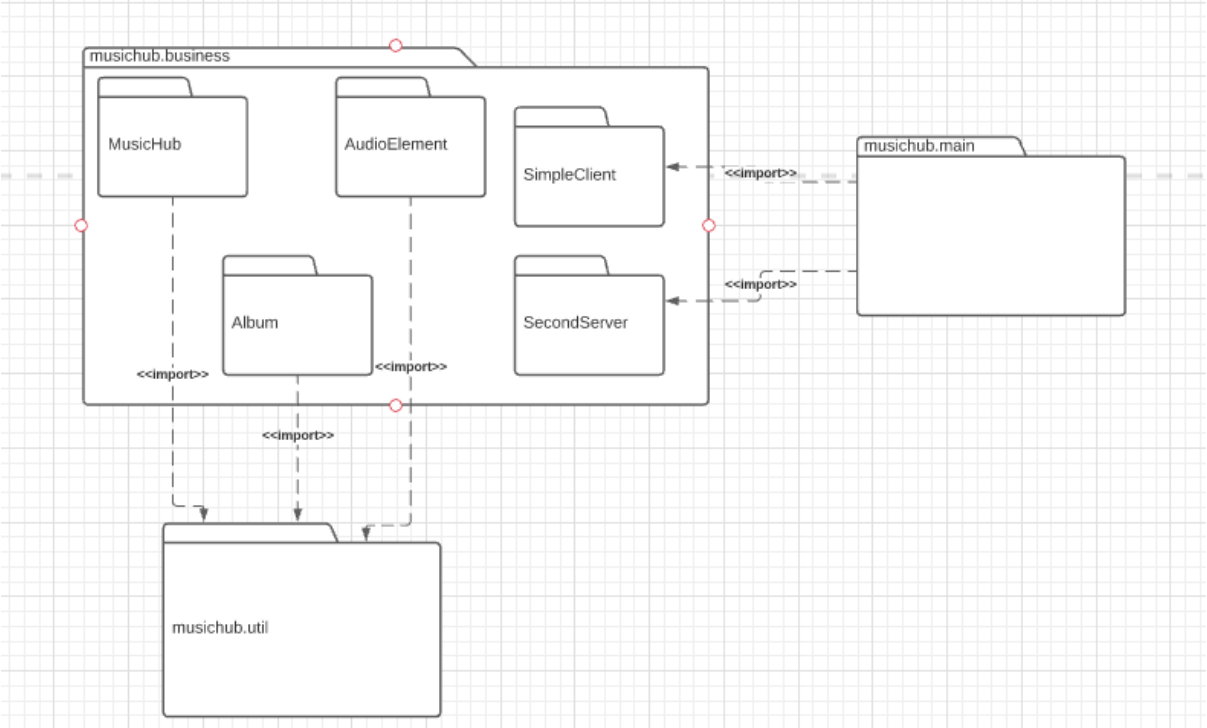


Diagramme UML de composants:

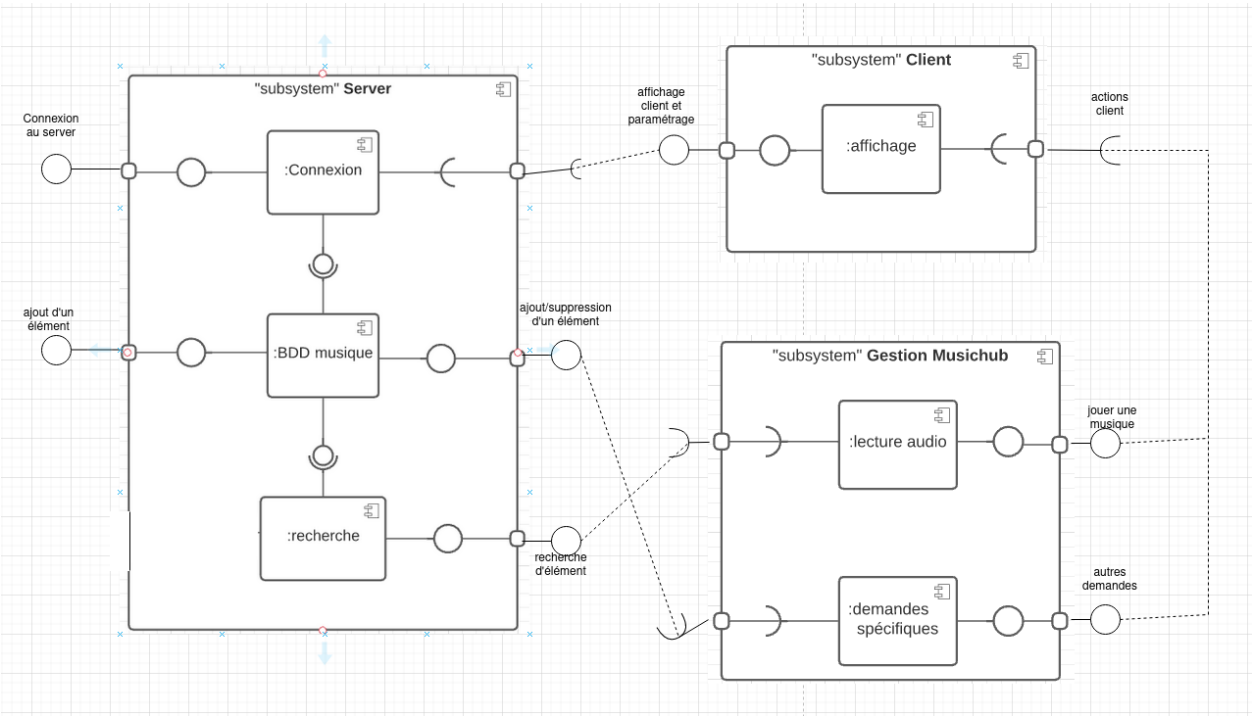


Diagramme UML de déploiement:

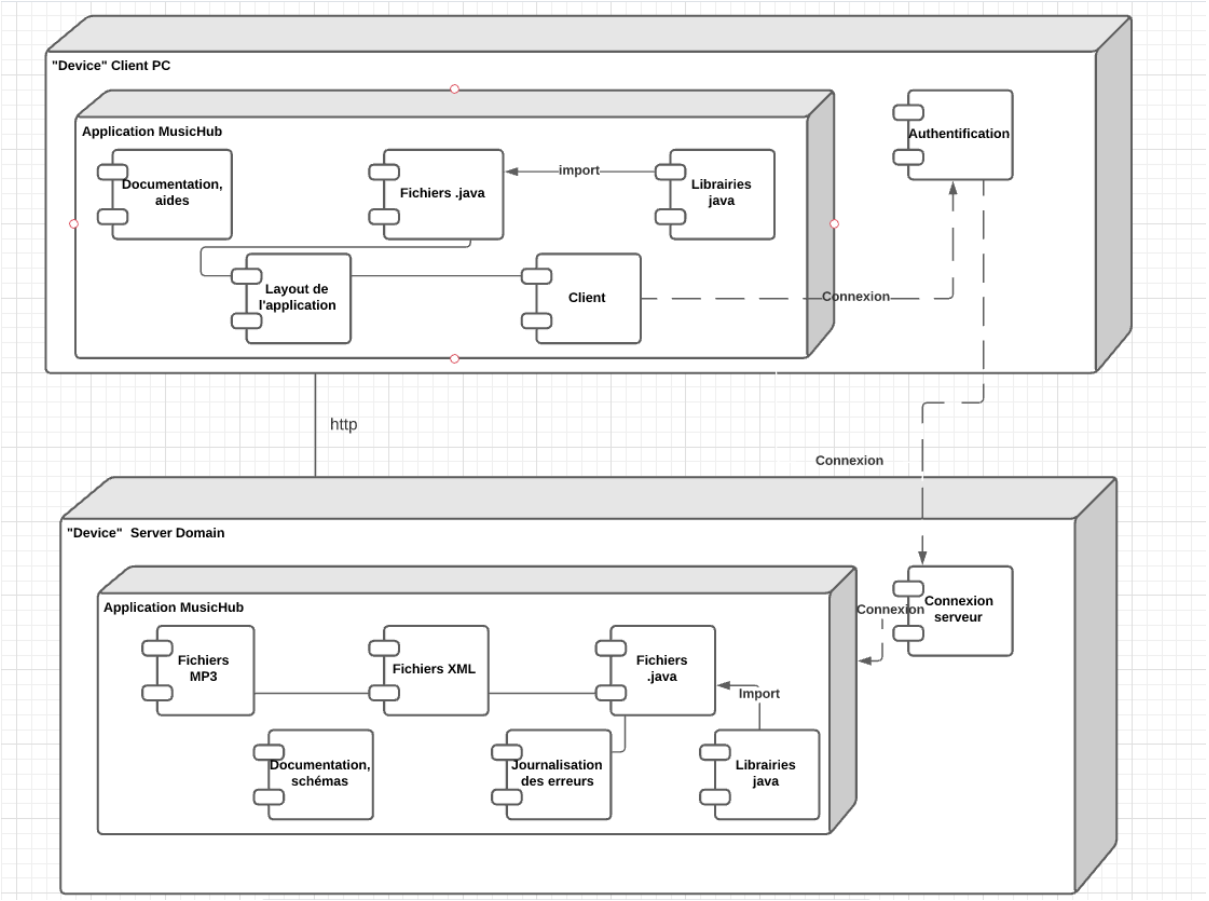
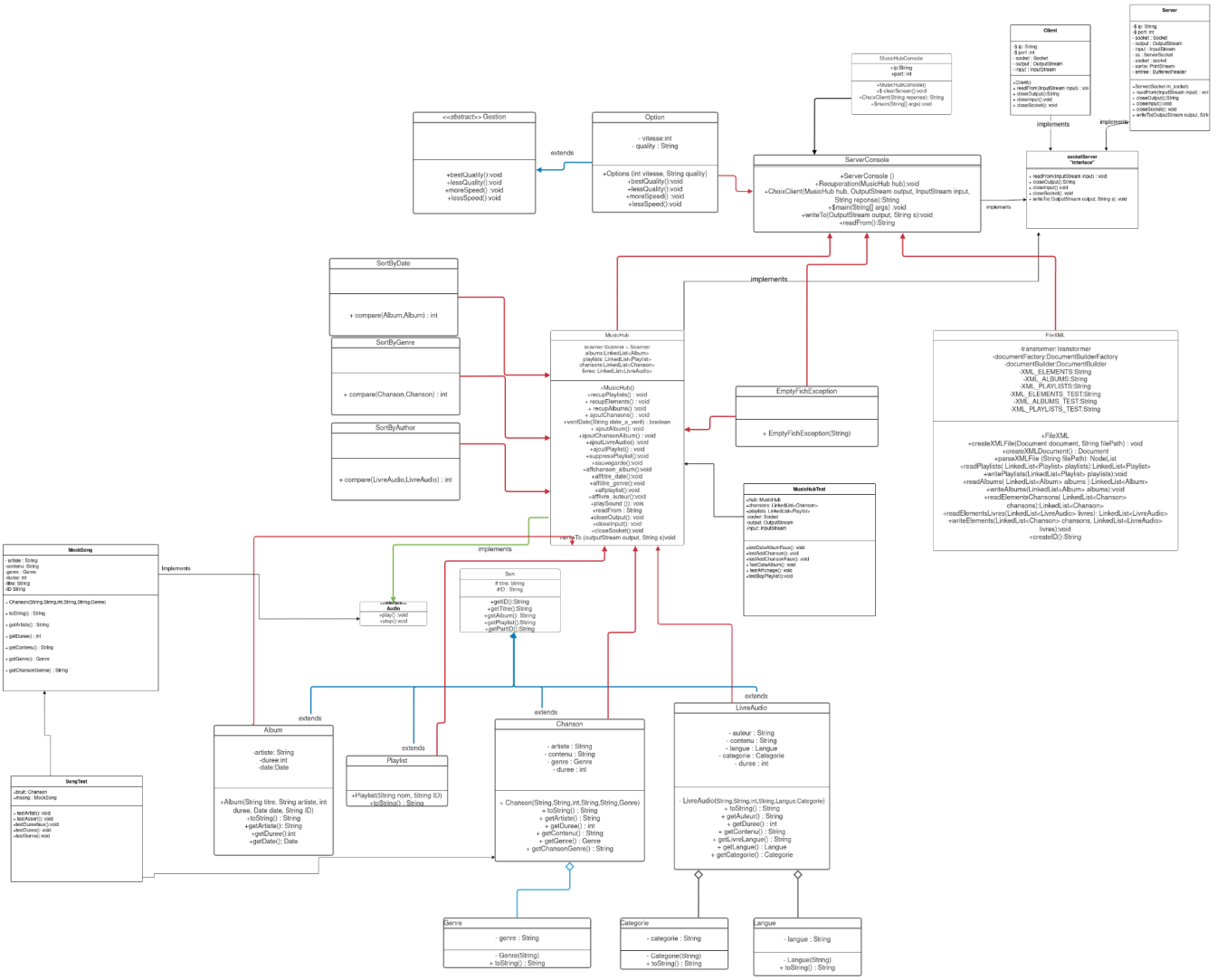


Diagramme UML de classes:



2. Les patrons de conception utilisés

Nous avons décidé d'utiliser un patron MVC car nous avons besoin de ne pas dupliquer du code sur nos interfaces client et serveur pour la gestion de la base de données audio. Grâce à ce patron, une modification du nombre de chansons ou d'albums dans notre BDD n'impactera pas directement notre interface graphique. Le client pourra mettre à jour son côté de l'application -le terminal- sans que le serveur en soit impacté. Nous avons donc bien un découpage entre les

informations internes du serveur et la façon dont celles-ci sont représentées sur le client, ce qui est la définition même du patron MVC.

De même, en respectant les principes SOLID, nous pouvons également dire que notre projet respecte le patron Stratégie puisque que nous encapsulons nos aspects qui ne varient pas, nous utilisons des interfaces et on améliore la flexibilité du programme en évitant trop d'héritages. Notre algorithme varie alors indépendamment du nombre de clients qui peuvent l'utiliser.

3. Une analyse de la conception selon les principes SOLID

- *Responsabilité unique*

Chaque module, classe, ou méthode doit être responsable d'une seule partie de la fonctionnalité que le logiciel fourni, et cette responsabilité devrait être entièrement encapsulée par la classe, le module ou la méthode.

Nous avons fait différentes interfaces pour chaque fonctionnalité du code : une interface pour gérer les accès client et serveur, une pour les éléments audios et une pour le MVC. Nous avons donc bien séparé les fonctionnalités en différents éléments et encapsulé ceux-ci . Ainsi, un changement dans la façon de récupérer les chansons n'affectera pas la façon de les afficher sur le terminal. De même, un changement sur le client n'affectera pas le serveur puisque les fonctions seulement utilisées par le client ne se retrouvent pas dans l'interface commune au serveur et à ce client.

- Ouvert/Fermé

Toutes les entités logicielles doivent être ouvertes à l'extension, mais fermées à la modification.

Nous utilisons des classes mères abstraites que l'on étend dans les classes filles où rajoutons des comportements. Nous pouvons étendre les différentes classes mères mais les éléments internes sont private ou protected ce qui empêche la modification des entités. Par exemple, pour le serveur, nous avons une interface socketServer qui est

implémentée dans nos classes client et serveur. Il serait donc plus simple de pouvoir ajouter des clients sans toucher au code présent. Ainsi, cette abstraction ferme la modification de la classe mère mais permet de modifier ses filles : nous respectons bien ce principe.

- Substitution de Liskov

Si S est un sous-type de T, alors les objets de type T peuvent être remplacés avec des objets de type S sans altérer aucune des propriétés du programme.

Nous n'utilisons pas la classe mère Audio telle quelle mais ses classes filles Chansons, LivreAudio, Playlist ou Albums dans notre main, soit notre façade. Nous remplaçons donc l'appel de la classe mère par les classes filles et déclarons des objets filles dans notre façade. Nous utilisons alors les objets audios des classes filles lors d'une référence à une classe mère.

- Ségrégation des interfaces

Une classe ne doit jamais être forcée à implémenter une interface qu'elle n'utilise pas ou une méthode qui n'a pas de sens pour elle.

Pour l'instant, nos interfaces pour générer les éléments audios sont assez grosses donc nous sommes à la limite du respect de ce principe mais du côté de la gestion client/serveur, nous respectons ce principe car nous avons de petites interfaces qui sont ciblées à un usage précis. Dans l'ensemble, nous pouvons dire que nous avons réalisé des interfaces qui sont spécifiques à un usage précis car nous avons plusieurs interfaces pour gérer les différentes parties de notre application. Néanmoins, il ne faudrait pas rajouter plus de lignes de codes dans ces interfaces audios sinon elles ne respectent plus ce principe car elles créeraient trop de fonctionnalités perdant de sa spécificitée.

- Inversion des dépendances

Les entités doivent dépendre uniquement des abstractions.

Les objets que nous utilisons dépendent des abstractions (classes abstraites et interfaces) de notre application. Dans ces abstractions, nous voyons qu'elles ne dépendent pas des détails et seulement des classes bas-niveau lorsqu'il s'agit de classes supérieures. Par exemple,

on ne retrouve pas les détails des définitions du genre de la chanson dans l'interface Audio et on ne les retrouve pas dans la classe mère Son qui est supérieure à Chansons.

IV- Le lien du projet GitHub

Voici le lien vers notre projet déposé sur Github :

https://github.com/Sandypn67/Projet-GLPOO.git

Répertoire de travail:

https://github.com/Arkani66/M_P_JMusicHub.git

Voici la correspondance nom - pseudo :

Chidiac Bryan : Kuaterzo

Leopold Arnaud : arnaudteaching

Miconnet Sandrine : Sandypn67

Prospa Florence : Arkani66

V- Les tests

1. Les critères right-BICEP utilisés

Rappel des critères right-BICEP:

• *right: Est-ce que les résultats sont corrects ?*

• *B: Est-ce que les résultats sont corrects aux limites ?*

• *I: Est-ce que nous pouvons vérifier les relations inverses?*

•*C: Est-ce que nous pouvons faire une vérification des résultats en utilisant d'autres moyens?*

• *E: Est-ce que nous pouvons forcer l'apparition des conditions d'erreur?*

• *P: Est-ce que les exigences de performance sont-elles respectées?*

Pour la classe MusicHub:

Right : lancement de l'affichage d'un menu et de la chanson "The Beatles - Hey Jude" -> l'affichage est correct et on a accès aux fichiers des éléments

B: Ajout d'un élément/ album avec des valeurs boguées (caractères non supportés) ou valeurs vides.

I : Recherche dans la base de données de l'élément "The Beatles - Hey Jude" pour savoir si elle a bien été insérée

C : Vérification des résultats en utilisant l'ancienne version de l'appli (que le côté client)

P: On crée plusieurs playlists pour voir si le programme peut en créer plusieurs d'un coup.

```
[INFO] --- maven-surefire-plugin:2.22.0:test (default-test) @ musicHubLibrary ---
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running test.MusicHubTest
               Affichage en cours ...
Mois incorrect ! 56
Création de la Chanson :
Création de la Chanson :
[ERROR] Tests run: 6, Failures: 2, Errors: 0, Skipped: 0, Time elapsed: 0.047 s <<< FAILURE! - in test.MusicHubTest
[ERROR] testBcpPlaylist  Time elapsed: 0.023 s  <<< FAILURE!
org.opentest4j.AssertionFailedError: expected: <true> but was: <false>
        at test.MusicHubTest.testBcpPlaylist(MusicHubTest.java:122)

[ERROR] testAddChansonFaux  Time elapsed: 0.003 s  <<< FAILURE!
org.opentest4j.AssertionFailedError: expected: <true> but was: <false>
        at test.MusicHubTest.testAddChansonFaux(MusicHubTest.java:56)
```

Avant le débogage des tests

Pour la classe Chanson:

Right : lancement du music hub client -> si l'interface affiche les chansons avec le nom de l'artiste suivi du titre

B: Test de la durée avec un type incorrect

C : Essayer de retrouver une chanson directement dans le fichier .xml

E: Utilisation d'un mock objet de Chanson pour tester l'apparition d'erreurs avec une chanson donnée

P: On teste la performance en créant plusieurs affectation de genre à la chanson

2. La liste des tests d'acceptance exécutés, la liste des bugs trouvés

Tests d'acceptance:

_ US 1.1 : Test sur l'affichage des différents éléments audios sur l'écran du client, on arrive à avoir sur la vue du client tous les éléments du serveur : réussi.

_ US 1.10 : Test sur le package des éléments avec maven, les directories sont créés : réussi.

_ CS 1.1 : Test sur la compilation des éléments avec maven, sur Linux et Windows : réussi.

_ US 1.0 : Test sur le lancement du serveur puis du client : réussi.

Listes bugs :

ID	001
Constraint/User Story	US 1.6
Description	Le terminal n'arrive pas à se connecter au serveur lorsqu'on lance le ./runServer.sh
Résultat	réussi
Sévérité	S1
Priorité	P1

Ce bug, en raison de sa grande priorité et de sa forte sévérité à été retravaillé afin de rendre notre projet fonctionnel.

ID	002
Constraint/User Story	US 1.10
Description	Les mains s'attendaient l'un l'autre créant une boucle infinie
Résultat	réussi
Sévérité	S1
Priorité	P2

Ce bug, en raison de sa priorité et de sa sévérité à été retravaillé : On décale les différents "read" avec des "write" afin de ne pas faire une boucle infinie entre les 2 mains.

ID	003

Constraint/User Story	US 1.10
Description	Les erreurs ne pouvaient pas être mise dans des fichiers de doc car le chemin maven n'était pas le bon
Résultat	réussi
Sévérité	S2
Priorité	P2

Ce bug, en raison de sa priorité et de sa sévérité à été documenté via internet "<testSourceDirectory>${basedir}/src/test/java</testSourceDirectory>" et nous l'avons résolu.

ID	004
Constraint/User Story	US 1.6
Description	Les classes Client et Serveur telles qu'elle sont désuètes
Résultat	réussi
Sévérité	S4
Priorité	P4

Ce bug, est plus un problème dans l'optimisation du code car ils ne sont plus nécessaire au bon lancement du code.

ID	004
Constraint/User Story	US 1.0
Description	Nous ne pouvons pas créer plusieurs clients
Résultat	pas réussi
Sévérité	S3

Priorité	P4

Nous avons décidé de nous concentrer sur le fonctionnement de l'appli avec un seul client plutôt que plusieurs.

Un autre problème rencontré fut la taille des musiques en format .wav. Ainsi il nous a été impossible de mettre les musiques en entier, nous avons du en mettre peu et les tronquer pour diminuer leur poids.