**cahier des charges**

**MAKON Maniym ma**

**NAJMI Mehdi**

**MVE MEYE BEKOUROU Sankara**

**IOINASCU Félicia**

# **Génie logiciel et Projet GLPOO**

* MAKON MANYIM MA
* MVE MEYE BEKOUROU Sankara
* NAJMI Mehdi
* Professeur: Ioinascu Félicia

Table des matières

[I. CADRE DU PROJET 2](#_Toc70976977)

[1. Presentation de l’équipe 2](#_Toc70976978)

[2. Enjeux et objectifs 2](#_Toc70976979)

[3. Livrables 2](#_Toc70976980)

[4. Planning prévisionnel 2](#_Toc70976981)

[II. SPECIFICATIONS 3](#_Toc70976982)

[1. Exigences fonctionnelles et non-fonctionnelles 3](#_Toc70976983)

[2. Diagramme de cas d’utilisation 4](#_Toc70976984)

[3. Diagrammes de sequence 4](#_Toc70976985)

[4. Backlog 7](#_Toc70976986)

[III. CONCEPTION 9](#_Toc70976987)

[1. Diagramme UML de package, de composant, de déploiement et de classe 9](#_Toc70976988)

[2. Analyse de la conception selon les principes SOLID avec argumentaire 12](#_Toc70976989)

[3. : Un lien vers le projet sous GitHub et les identifiants des étudiant 13](#_Toc70976990)

# CADRE DU PROJET

## Presentation de l’équipe

Notre équipe est composée de trois étudiants de classe 31 qui sont

Mehdi NAJMI.

MAKON MANYIM MA

Sankara MVE MEYE BEKOUROU

## Enjeux et objectifs

L’objectif du projet génie logiciel est de nous faire travailler de manière collaborative sur un sujet donné, en mettant à l’œuvre les notions du cours, en améliorant nos compétences en POO et Java, et en produisant tous les artefacts d’un processus de développement de projet.

Toutes les notions vues en cours sont mises en pratique dans ce projet afin de comprendre leur utilité.

## Livrables

Comme livrable, il est attendu une application fonctionnelle. Celle-ci sera basée sur l’application fournie par le professeur et notre but est de l’améliorer.

En implémentons tous les différents éléments de modification que nous avons présenté dans ce cahier de charge, nous faisons évoluer l’application.

## Planning prévisionnel

Toutes les modifications logicielles que nous devrons appliquées au logiciel seront opérationnelles au 3 mai 2021.

Nous avons déterminé toutes les exigences et spécifications nécessaires qui nous aideraient à la modification de l’application Music.

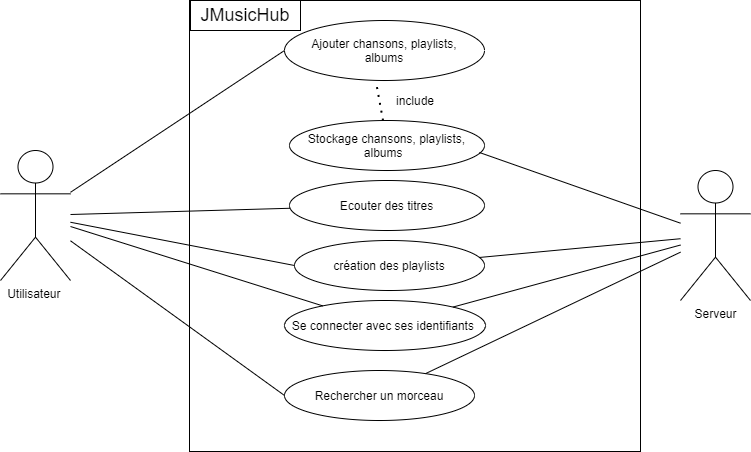
Nous avons également conçu tous les diagrammes utiles à la modification des logiciels.

# SPECIFICATIONS

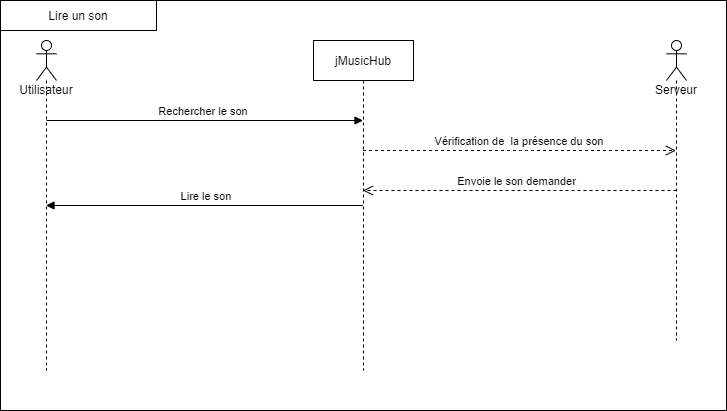
## Exigences fonctionnelles et non-fonctionnelles

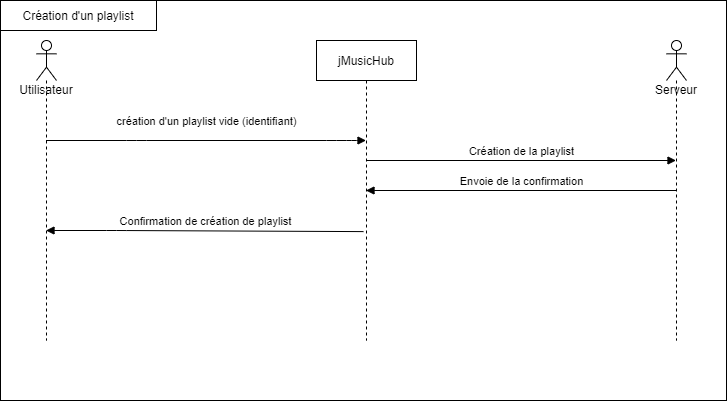
|  |  |
| --- | --- |
| **Exigences Fonctionnelles** | **Exigences Non-Fonctionnelles** |
| Ajouter des morceaux à un album. | Travail uniquement en Java. |
| Lire aléatoirement des morceaux. | Utilisation de l'application sur les OS Windows, Linux, et IOS. |
| Sélection des chansons d'un album. | L'application ne fonctionne que dans la console. |
| Sélection des chansons d'un album. |  |
| Affichage des chansons rangés par date de sortie, par genre, par playlist, par auteur (livre). |  |
| Rajout d'une nouvelle chanson, album. |  |
| Rajout d'une chanson existante à un album. |  |
| Rajout d'un livre audio. |  |
| Création d'une nouvelle Playlist à partir de chansons et livres audio existants. |  |
| Sauvegarde des Playlist, des albums, des chansons et des livres audios dans les fichiers XML respectifs. |  |

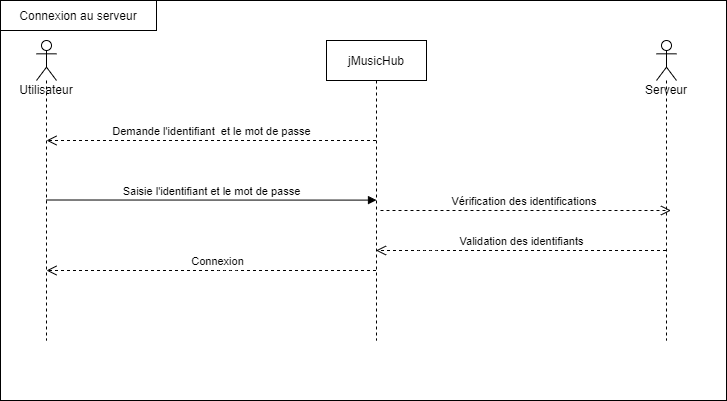
## Diagramme de cas d’utilisation

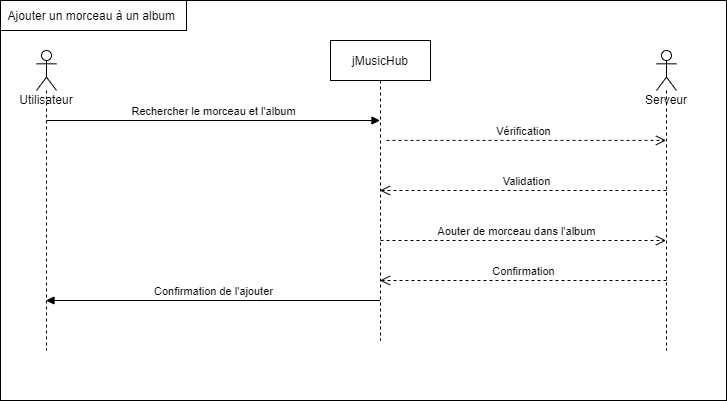


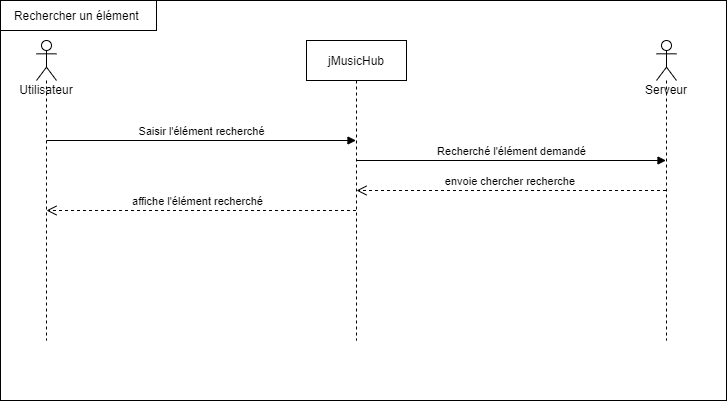
## Diagrammes de sequence





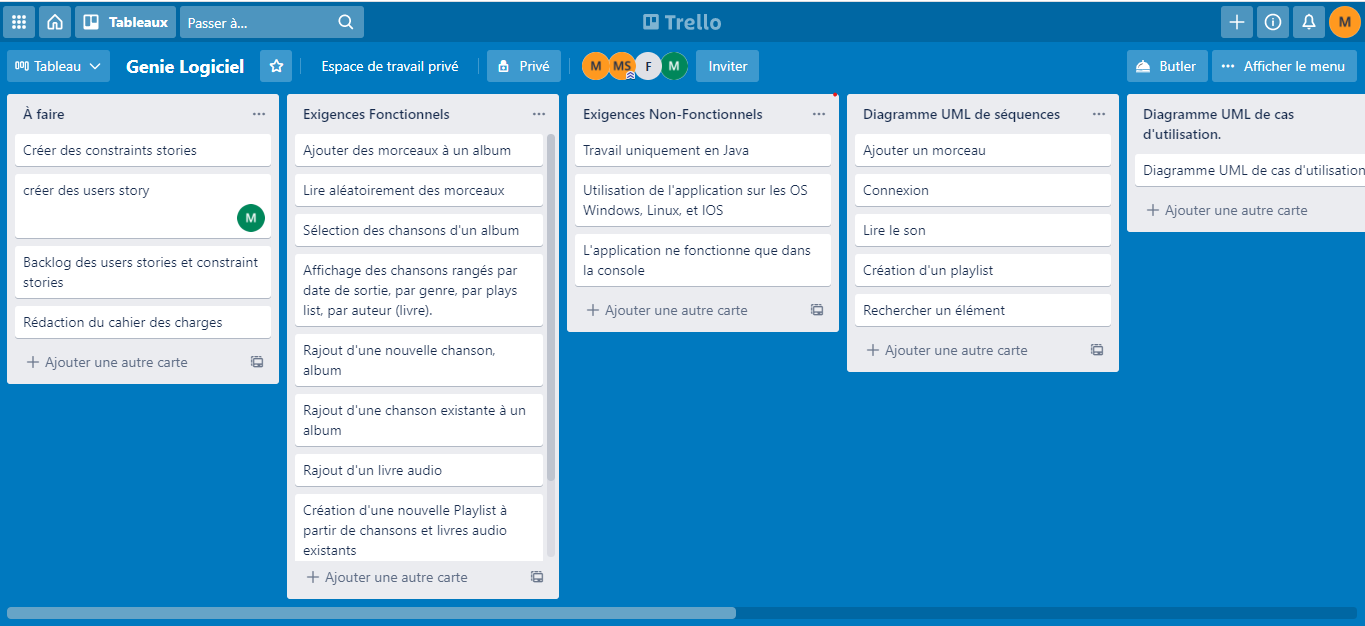




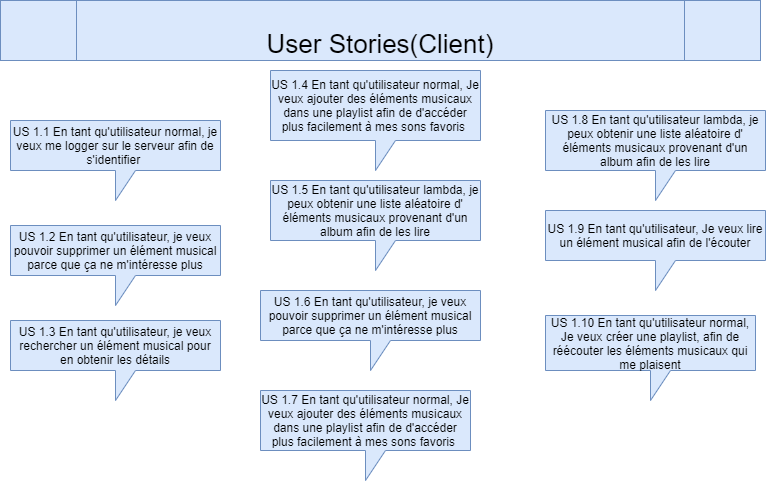


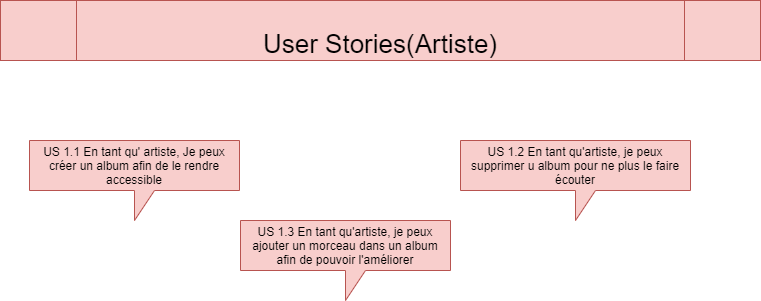
## Backlog

On a utilisé Trello comment outil de gestion de projet inspiré par la méthode Kanban.



Le lien vers le projet Trello <https://trello.com/b/DJ8JyETG/genie-logiciel>





# CONCEPTION

## Diagramme UML de package, de composant, de déploiement et de classe

Diagramme UML de classe

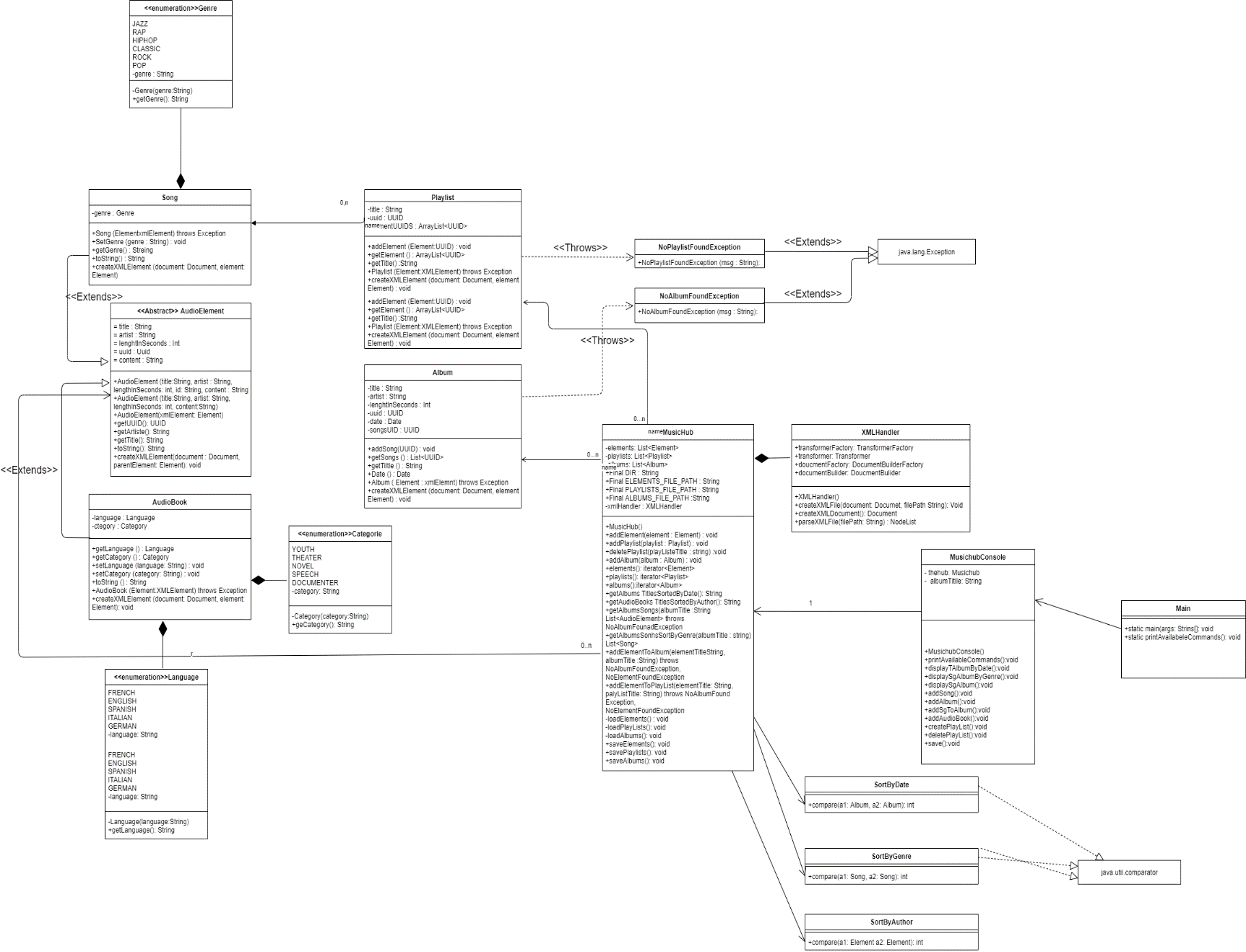


Diagramme UML de composant

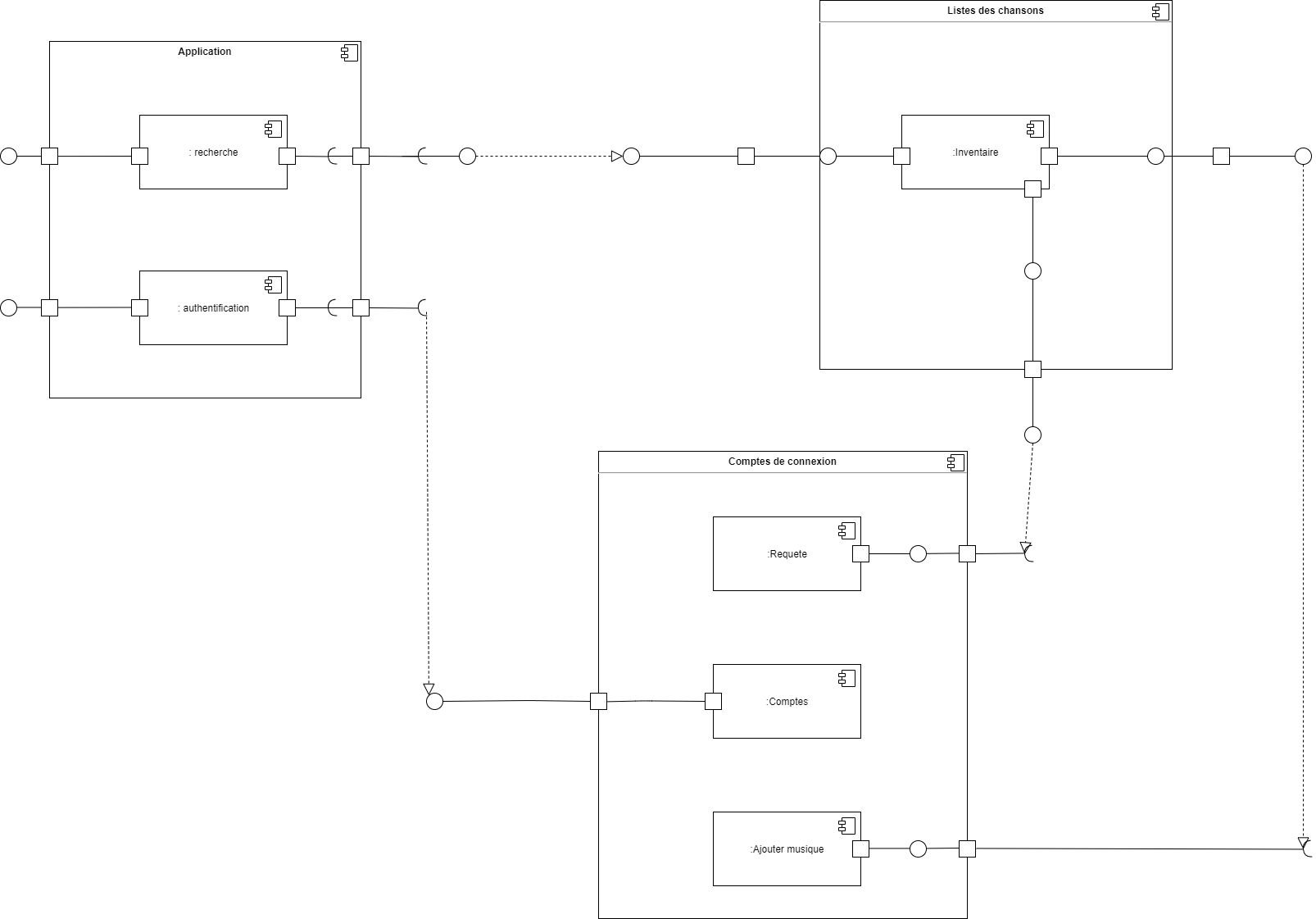


Diagramme UML de déploiement

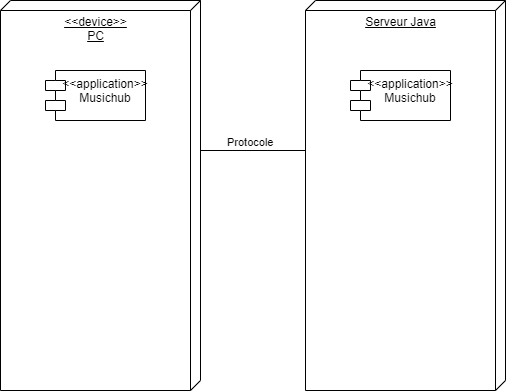
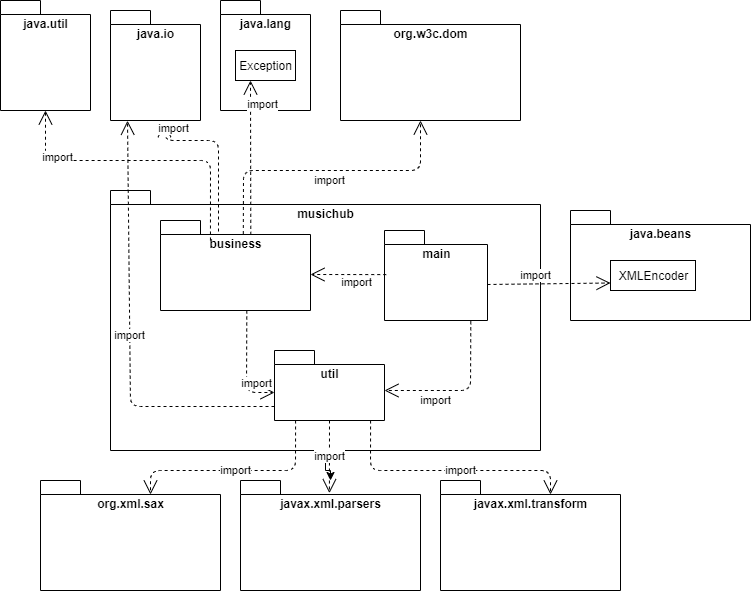


Diagramme de package



## Analyse de la conception selon les principes SOLID avec argumentaire

1. Ouvert/Fermé

Nous avons déterminé des classes et méthodes publiques qui peuvent hériter d’autres fonctionnalités, mais les symboles sont privés pour ne pas perturber le fonctionnement de notre logiciel.

« Toutes les entités logicielles doivent être ouvertes à l’extension, mais fermées à la modification »

1. Responsabilité unique

Nous pensons que la caractéristique de responsabilité unique est bien honorée puisque nous avons découpé le projet en fonctionnalité que nous avons adapté par classe. Nous avons un tas d’éléments musicaux : albums, playlist, morceaux. Ils sont chacun représentés par une classe séparée lorsqu’on ajoute des morceaux, playlists sou albums au serveur.

De même pour les relations entre le client et le serveur, les méthodes sont encapsulées par un Request manager dans le but de permettre l’envoi des commandes du client au serveur.

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

1. Ségrégation des interfaces

Pour ce caractère, le logiciel IntelliJ gouverne cela en nous proposant de supprimer les interfaces qui ne sont pas utilisés. Par ailleurs nous n’avons eu en recours que très rarement à l’implémentation d’une interface.

« Une classe ne doit jamais être forcée à implémenter une interface qu’elle ne l’utilise pas ou une méthode qui n’a pas de sens pour elle »

1. Inversion des dépendances

Le classe Song comme peut le témoigner le diagramme UML de classes ne Dépend que de l’abstraction AudioElement.

« Les entités doivent dépendre uniquement des abstractions »

## : Un lien vers le projet sous GitHub et les identifiants des étudiant

Le lien : <https://github.com/EdMkn/Projet_glpoo_musichub/tree/main>

Les identifiants : NAJMI MEHDI 🡪 Najmi19

MAKON MANYIM MA 🡪 [EdMkn](https://github.com/EdMkn)

MVE MEYE BEKOUROU SANKARA 🡪 mvesankara

