

Génie Logiciel : Projet 2021

3 MAI

FERNANDES Mickaël
RABII Ahmad
NOUARI Achraf



esiea
INGÉNIEUR·E·S D'UN NUMÉRIQUE UTILE

Table des matières

I.	Liste des membres : Rôles et description.....	3
II.	Liste des exigences fonctionnelles et non fonctionnelles	4
III.	Diagramme UML des cas d'utilisation et de séquence	5
a)	Diagramme UML des cas d'utilisations.....	5
a)	Diagramme UML de séquence.....	6
IV.	Diagramme UML de package, composants, déploiement, classes	8
a)	Diagramme UML de classes.....	8
b)	Diagramme UML de composants	8
c)	Diagramme UML de déploiement	8
d)	Diagramme UML de package.....	9
V.	Analyse de la conception selon les principes SOLID	10
a.	Responsabilité unique.....	10
b.	Ouvert/Fermé.....	10
c.	Substitution de Liskov	10
d.	Ségrégation des interfaces	10
e.	Inversion des dépendances	11
VI.	Lien vers GitHub	12
VII.	Critères right-BICEP utilisés.....	12
VIII.	Tests d'acceptances et bugs trouvés	13
a)	Tests d'acceptances	13
b)	Bugs trouvés	14

I. Liste des membres : Rôles et description

a) Réalisations

NOUARI Achraf : Rédaction du rapport, liste des exigences fonctionnelles, user stories et critères d'acceptance, diagramme de package, BICEP, réalisation de l'interface utilisateur et implémentation de méthodes dans la classe RequestManager.

RABII Ahmad : Liste des exigences non-fonctionnelles, diagramme de séquence, diagramme de classes, analyse principes SOLID, liste des tests d'acceptance, JavaDOC, ajout des fonctionnalités (charger les éléments, annuler les changements...).

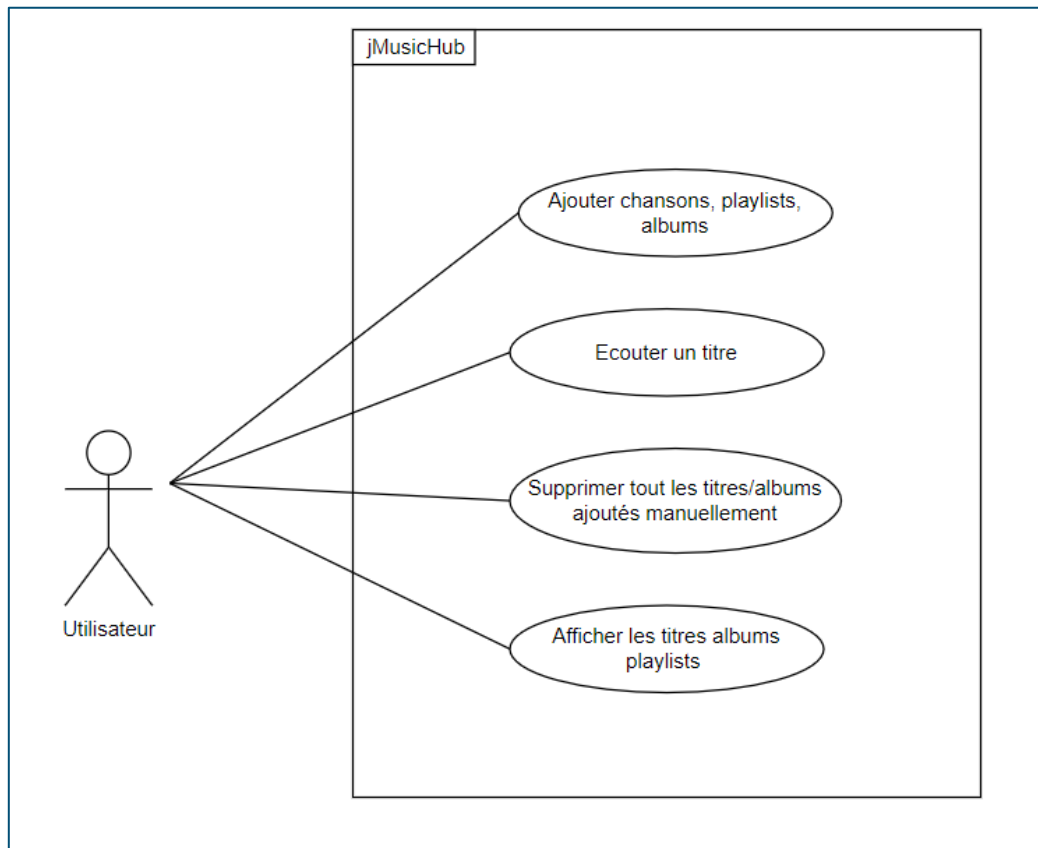
FERNANDES Mickael : Diagramme des cas d'utilisation, constraint stories, diagramme de composants et déploiement, liste des bugs trouvés, réalisation de l'architecture client-serveur ainsi que la lecture des fichiers audio.

II. Liste des exigences fonctionnelles et non fonctionnelles

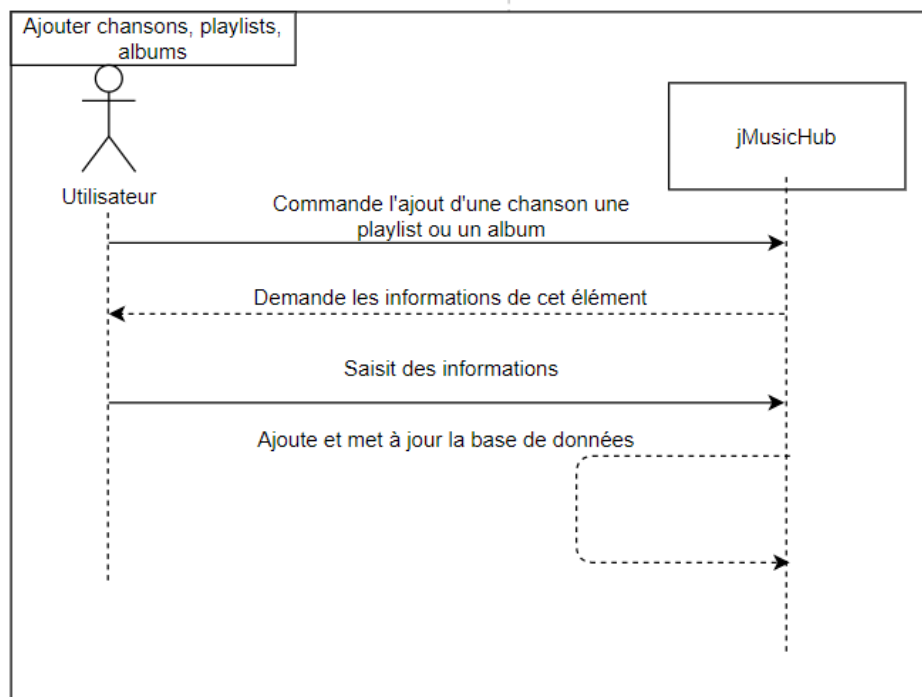
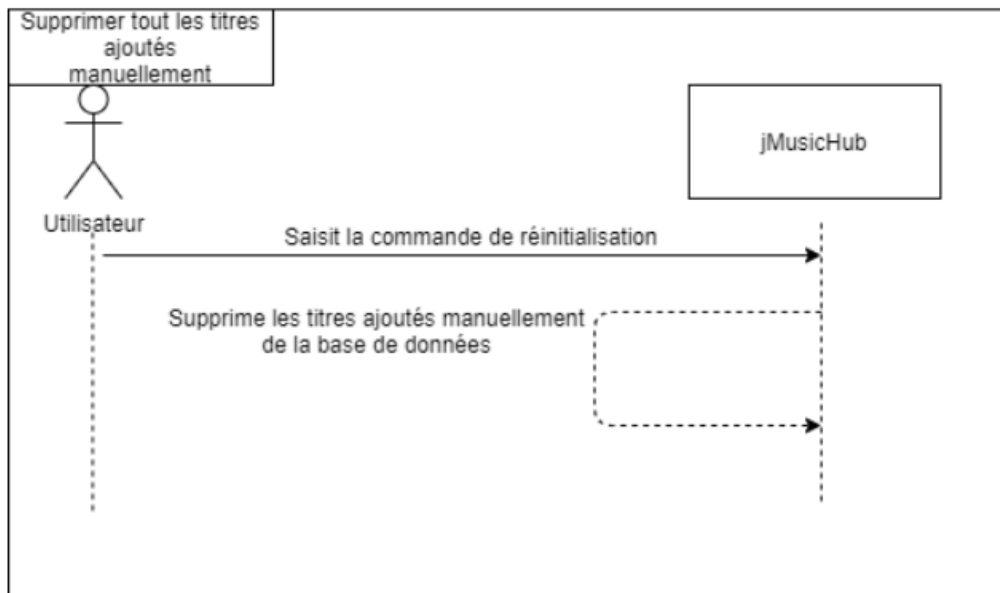
ID	Nom	Type	Description
JMUSIC_REQ_1	Fonctionnalité de base	F	Au minimum un client doit pouvoir se connecter au serveur
JMUSIC_REQ_2	Ajout	F	Le client pourra ajouter des chansons, albums ou playlists sur le serveur
JMUSIC_REQ_3	Affichage	F	Le client aura la possibilité de visualiser le contenu du serveur (chansons, albums, playlists)
JMUSIC_REQ_4	Ecoute	F	Le client aura la possibilité d'écouter les morceaux de musique
JMUSIC_REQ_5	Serveur	F	Le serveur sera mis à jour automatiquement lorsqu'un élément y est ajouté
JMUSIC_REQ_6	Mise à jour	F	Le client peut accéder aux mises à jour du serveur à tout moment.
JMUSIC_REQ_7	Gestion des erreurs	F	L'application disposera d'un système de journalisation des erreurs (avertissements, fichier lu,...) avec horodatage
JMUSIC_REQ_8	(Press to kill)	F	Le client à la possibilité d'appuyer sur une touche pour quitter l'application (en cas de bugs,...)
JMUSIC_REQ_9	Réinitialisation	F	On pourra supprimer tout les éléments ajoutés par l'utilisateur en appuyant sur un bouton.
JMUSIC_REQ_10	Unicité	NF	L'application est accessible que pour un client à la fois
JMUSIC_REQ_11	Performance	NF	L'application sera executé sur console

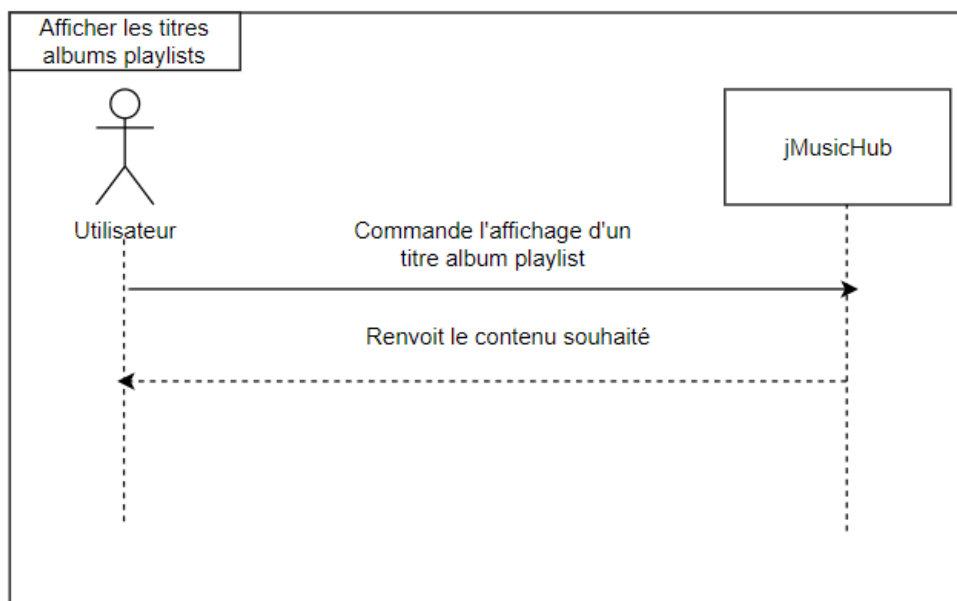
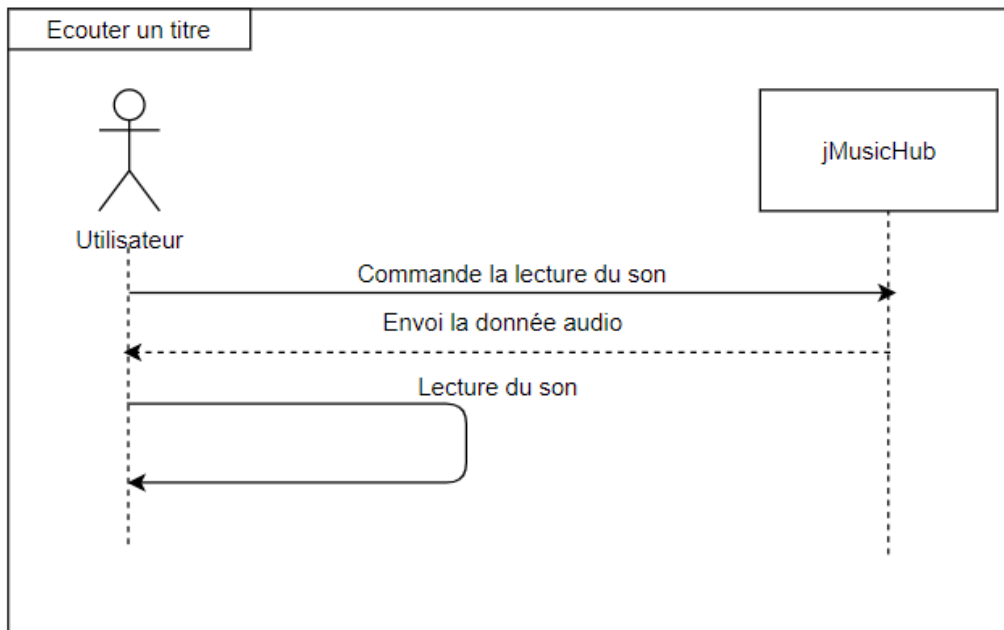
III. Diagramme UML des cas d'utilisation et de séquence

a) Diagramme UML des cas d'utilisations



a) Diagramme UML de séquence



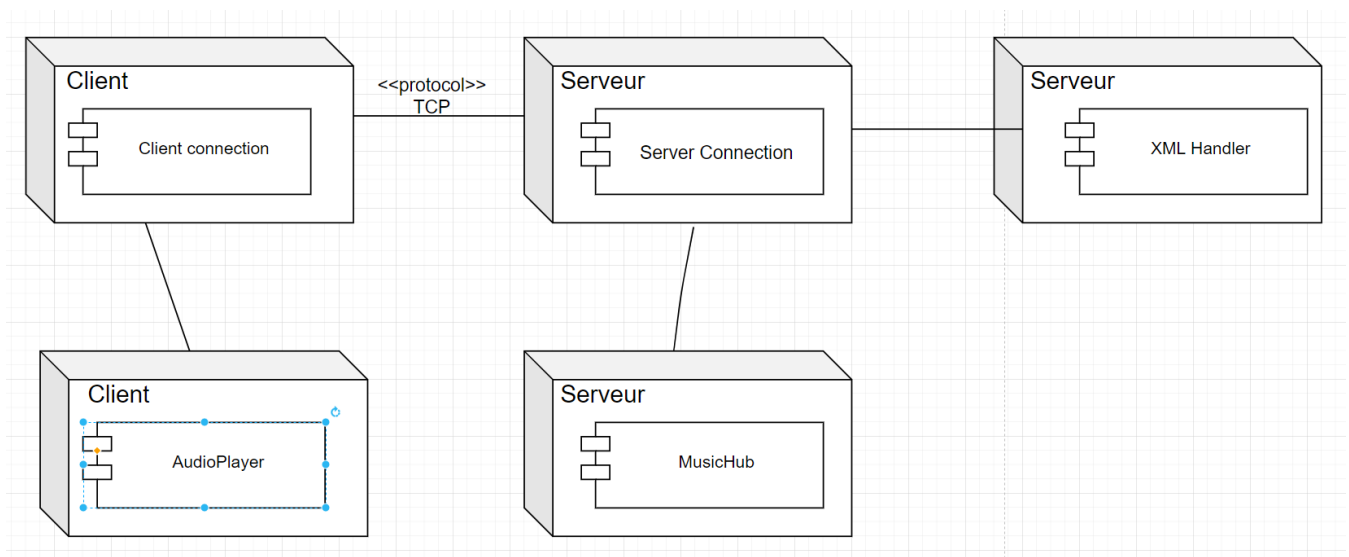


IV. Diagramme UML de package, composants, déploiement, classes

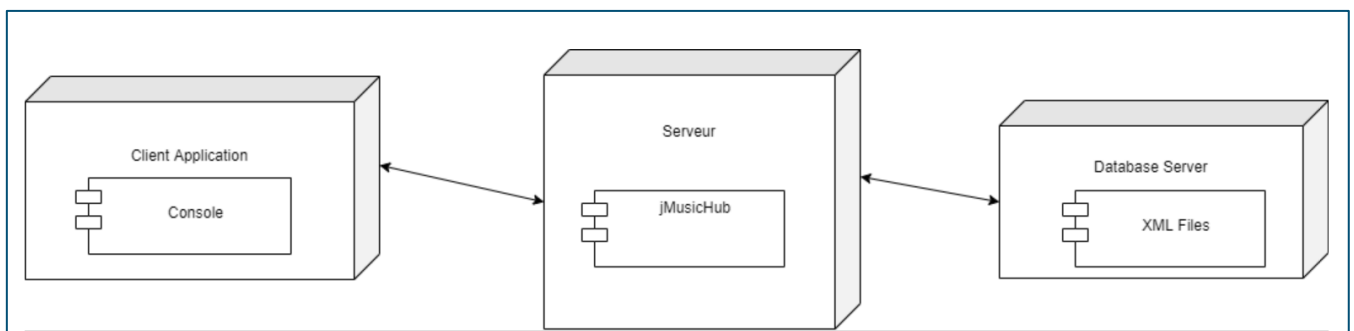
a) Diagramme UML de classes

Voir l'image UML Classe.png dans le fichier .zip

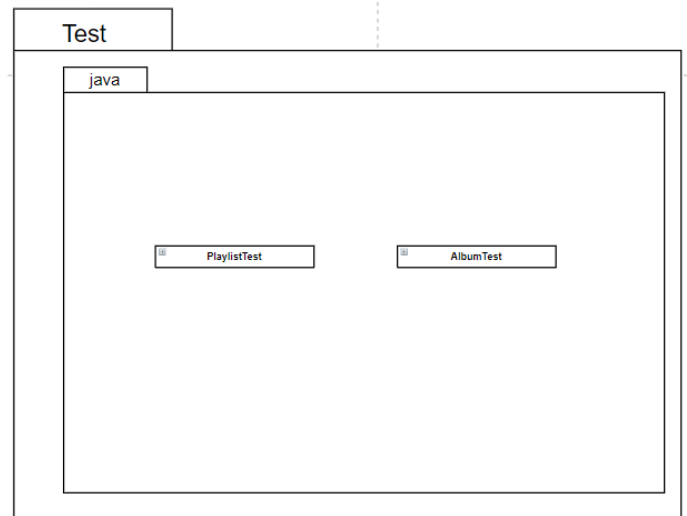
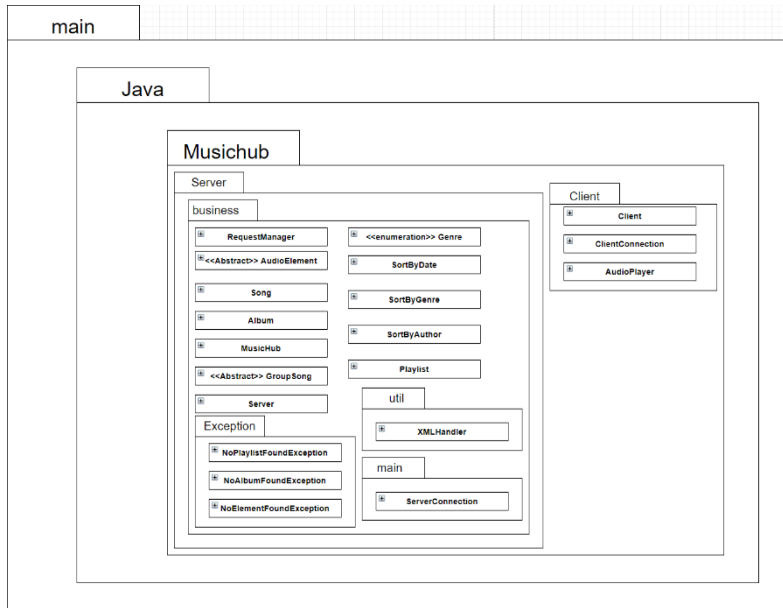
b) Diagramme UML de composants



c) Diagramme UML de déploiement



d) Diagramme UML de package



V. Analyse de la conception selon les principes SOLID

a. Responsabilité unique

« 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 »

Nous pensons que le critère de responsabilité unique est respecté puisque nous avons découpé le projet en fonctionnalités que nous avons implémenté par classe. Nous avons plusieurs éléments musicaux : albums, playlists, morceaux et ils sont chacun gérés par une classe séparée. Lorsqu'on ajoute des morceaux, playlists ou albums au serveur, les méthodes sont respectivement dans les classes de leur type, encapsulés.

De même pour les interactions entre le client et le serveur, les méthodes sont encapsulées par un RequestManager afin de permettre l'envoi des commandes du client au serveur.

⇒ **Critère respecté**

b. Ouvert/Fermé

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

Nous avons défini des classes et méthodes publiques qui peuvent hériter d'autres fonctionnalités mais les attributs sont privés pour ne pas perturber le fonctionnement de notre logiciel.

⇒ **Critère respecté**

c. 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'avons ajouté à l'exemple d'implémentation jMusicHub fourni qu'une seule classe abstraite, nous avons créé une classe abstraite Groupsong regroupant la classe album et playlist, le principe de substitution de Liskov est respecté car une instance de Groupsong peut être remplacé par une instance de playlist ou d'album.

⇒ **Critère respecté**

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

⇒ **Critère respecté**

e. Inversion des dépendances

« Les entités doivent dépendre uniquement des abstractions »

Notre classe Song comme peut le témoigner le diagramme UML de classes ne dépend que de l'abstraction AudioElement.

Les entités Albums et Playlists aussi dépendent de l'abstraction GroupSong.

⇒ **Critère respecté**

VI. Lien vers GitHub

Lien du projet

https://github.com/MickaelFrds/Projet_MusicHub_GL

Identifiants :

Fernandes Mickaël : MickaelFrds

Rabii Ahmad : a-r-z

Nouari Achraf : nouari1

VII. Critères right-BICEP utilisés

Les critères right-BICEP que nous avons utilisés sont les suivants :

- **Right** : Nous avons ajouté des playlists, albums ainsi que des morceaux puis nous avons vérifié qu'ils étaient bien ajoutés à la base de données du serveur.
- **B** : Nous avons tenté de créer des playlists avec un nom vide, un message d'erreur s'est affiché. Même chose lorsqu'on souhaite supprimer une playlist sans nom. Nous avons réalisé le même test pour album et morceaux.
- **I** : Pour vérifier les relations inverses, nous avons ajoutés puis supprimés à tour de rôle une playlist, un album et un morceau et nous avons consulté le contenu de la base de données pour vérifier qu'ils ont bien été supprimés.
- **C** : Afin de vérifier qu'une playlist/album/morceaux a été ajouté(e), il faut s'assurer qu'aucun message d'erreur n'apparaît lors de l'exécution. Le moyen de faire une vérification est d'afficher la liste des playlists/albums/morceaux
- **E** : Pour forcer les apparitions des conditions d'erreur nous avons supprimé des playlists qui n'existaient pas, le message « Playlist <nom_playlist> not found ». Il en est de même pour album et pour les morceaux.
- **P** : Nous n'avons pas pris en compte ce critère de performance dans notre conception.

VIII. Tests d'acceptances et bugs trouvés

a) Tests d'acceptances

UserStories	Critères d'acceptance
US 1.1 En tant que User je veux pouvoir ajouter des morceaux, playlists et albums à l'application musicale afin de personnaliser le contenu de mon compte.	US 1.1 - AC - -Pour ajouter un morceau, album ou playlist tout les champs doivent etre renseignés et leur types doivent correspondre aux attentes au risque de provoquer une exception. . -L'élément ajouté n'est stocké dans le serveur que si par la suite le User le confirme.
US 1.2 En tant que User je veux etre capable d'écouter un morceaux dans l'application musicale pour savoir s'il me plait ou non.	US 1.2 - AC - -L'élément à lire doit exister dans la base de données du serveur.
US 1.3 En tant que User je veux avoir la possibilité d'annuler les actions que j'ai effectuer durant ma session afin de retrouver une application non personnalisée comme au départ.	US 1.3 - AC - -L'utilisateur peut annuler les changements effectués durant la session en cours. Pour supprimer des éléments ajoutés lors d'une session précédente, il faudra manuellement supprimer les morceaux, albums, playlists
+ Ajouter une autre carte	+ Ajouter une autre carte

Test n°1 : US 1.1 :

Description : On teste les méthodes de la classe Album et Playlist en ajoutant de nouveaux éléments et en les supprimant, en supprimant des éléments inexistants,

Résultat : L'application redemande au Client de renseigner le champ vide

⇒ Réussi

Test n°2 US 1.2 :

Description : On commande la lecture d'un élément audio qui n'existe pas.

Résultat : L'application redemande au Client un nom de fichier audio qui existe.

⇒ Réussi

Test n°2 US 1.3:

Description : On ajoute un morceau, album, playlist on le charge dans la base de données puis on commande au serveur d'annuler cette modification.

On se charge d'effacer manuellement une playlist créée auparavant.

Résultat : Les données sont bien supprimées, si on renseigne un champ vide on tombe bien sur une exception qui nous demande de saisir un champ non vide

⇒ **Réussi**

b) Bugs trouvés

Bugs rencontrés	Sévérité/Priorité
Lors de la suppression d'une playlist, on peut avoir un message d'erreur qui survient rarement. Il disparaît en redémarrant le logiciel.	Sévérité : Faible Priorité : Moyenne
Lors de la création d'un album et morceau, quand on renseigne une chaîne de caractères au lieu d'un entier pour le champ « Length » : l'application nous demande de saisir un entier mais peut importe le type de cette donnée que l'on ressaisit, il y'a un message d'erreur.	Sévérité : Faible Priorité : Moyenne
La feature Stop pour AudioPlayer fonctionnait en debug mais pas en run nous avons eu aucune explication à ce problème ce qui nous a fait abandonner cette feature.	Sévérité : Faible Priorité : Faible