Groovify

Contexte

Beaucoup sont ceux ayant du mal à faire le lien entre leurs discographies physiques (CD…) et digitales (MP3…). Ordonner sa bibliothèque audio peut parfois être difficile, surtout si cette bibliothèque possède de nombreux titres. A titre d'exemple personnel, ma bibliothèque spotify possède 936 titres et sans les diverses fonctionnalités permettant de les ordonner, il serait très difficile de s'y retrouver.

L'objectif est ici de réaliser une application permettant de parcourir un catalogue de musiques triées par Artiste, Album etc… Cela permettra aux utilisateurs d'organiser leurs bibliothèques de musiques. Cette application est fortement inspirée de logiciels permettant d'écouter de la musique depuis des fichiers (Groove music, Media player, iTunes, …) ainsi que de logiciels permettant d'écouter de la musique depuis internet (Spotify, Deezer…).

Les fonctionnalités principales (donc développées en premier) de l'application seront de pouvoir parcourir une liste de musiques en ayant accès aux Artistes, Albums et Titres. En allant sur les artistes, il sera possible d'accéder à une petite biographie et d'accéder à leurs albums et titres.

Si le temps nous le permet, nous aimerions mettre de nombreuses fonctionnalitées tel que celle d'inclure un lecteur de fichier audio (avec possibilité de mettre en pause, reprendre la lecture d'un morceau et de passer un titre), concevoir des playlists, avoir des favoris, pouvoir utiliser une fonction de recherche et charger les morceaux depuis internet pour ne pas avoir à tout stocker en local.

Le public visé par cette application est extrêmement divers. Par rapport aux utilisateurs de services avec abonnement, notre application aura surement un public de personnes ayant plus leurs propres sources de musique, ayant par exemple une large collection de musique, une connexion internet ne permettant pas le streaming de musique ou étant attiré par la gratuité de ce service.

Personas

Tom

Age : 19
Métier : étudiant

Tom étudie souvent sur son ordinateur, c'est pourquoi il aime écouter de la musique. En ligne ou hors-ligne rien de l'arrête. Avec ses nombreux titres, il aime ecouter sa musique avec casque et enceintes.

Objectifs :
- Trier toutes les musiques qu'il possède.
- Ne pas payer pour ce service.

Besoins :
- Créer ses playlists sans publicité, hors-connection.
- Ranger toutes les musiques qu'il a téléchargé sur son ordinateur.

Frustration :
- Trop de musique sur son ordinateur et aucun de logiciel gratuit pour les trier.

Elene

Age : 29
Métier : Ingénieur Civile

Elle est mère de 2 enfants et est impliquée dans son travail. Elle apprécie particulièrement écouter ses playlists le matin en fonction de ses envies et lors de ses trajets professionnels. Elle partage principalement ses playlists avec ses amis ou ses collègues et écoute leurs playlists. Elle apprécie particulièrement les nouveautés et les albums de ses artistes préférés.

Objectifs :
- Possibilité d'écouter de la musique sur son téléphone ou son ordinateur.

Besoins :
- Créer ses propres playlists
- Ecouter ses artistes préfères

Frustration :
- Les applications modernes ont trops de fonctionnalitées.

Vincent

Age : 36
Métier : Ingénieur audiovisuel

Vincent aime animer des soirées et possède toujours la meilleure playlist peut importe la situation. Grand audiophile depuis qu'il est petit, il collectionne les titres dans sa bibliotèque.

Objectifs :
- Créer de meilleures playlist pour lui-même et les autres.
- Gérer sa liste de musiques pour toujours avoir la musique idéale pour ses activités.

Besoins :
- Accès a un large catalogue de musiques de divers artistes.
- Capacités de gestion de Bibliothéque efficaces, déplacement des chansons entre les listes de lecture et suppression.

Frustration :
- Il est compliqué de gérer ses playlists ou de modifier sa liste de lecture.
- Avoir une grande sélection d'artiste est intéressant mais si on ne s'y retrouve pas cela n'a plus aucun intérêt.

Ergonomie et accessibilité.

L’ergonomie est une partie importante dans la conception d’une application. Une bonne application se doit d’être non seulement agréable à regarder mais aussi instinctive et accessible à un maximum de personnes.

L’application sera entièrement en thème sombre, similaire à des applications comme Discord ou Spotify. Ce type d’affichage est non seulement plus agréable pour les yeux la nuit mais permet aussi d'économiser de l'énergie avec les écrans de type OLED.

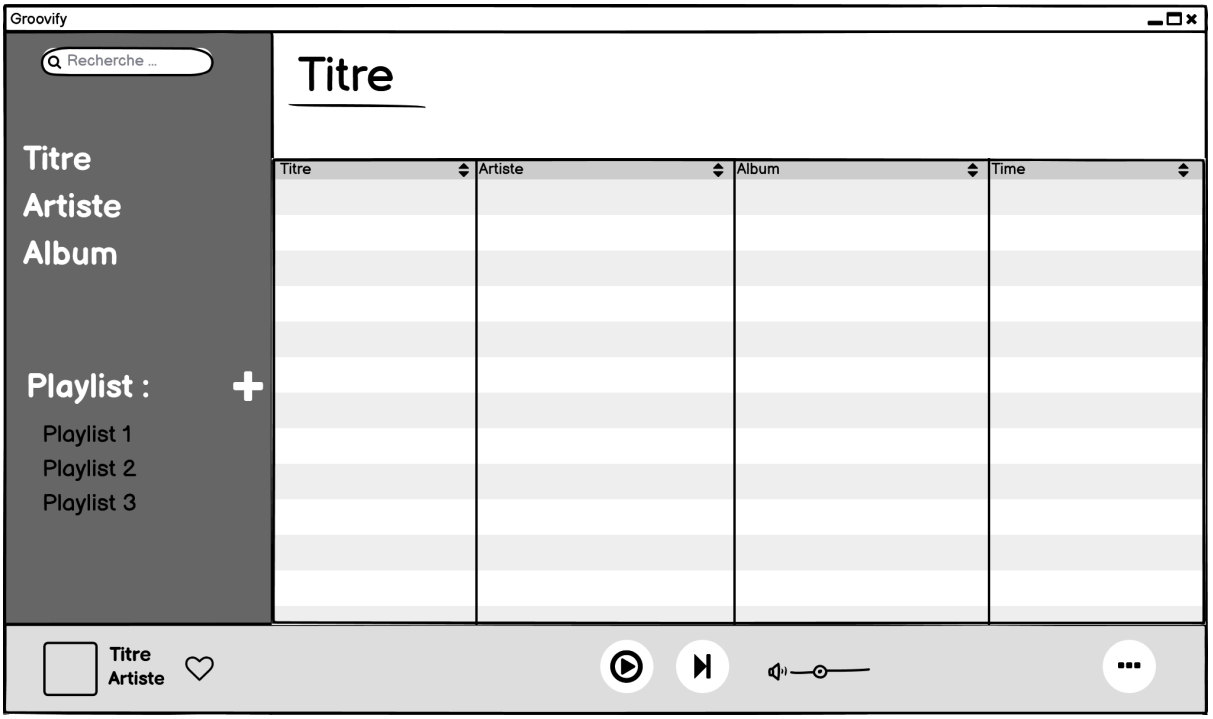
L’interface de l’application sera similaire à la plupart des services similaires, une application ayant une excellente ergonomie, le but n'étant pas ici de réinventer la roue. L’interface se veut sobre, minimisant ayant ainsi une navigation simple et minimaliste.

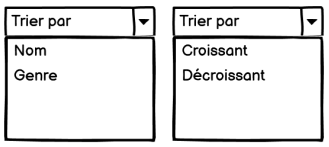
La police choisie sera Milliard ou Roboto, ces polices étant toutes les deux sobres et efficaces, claires et belles à la fois.

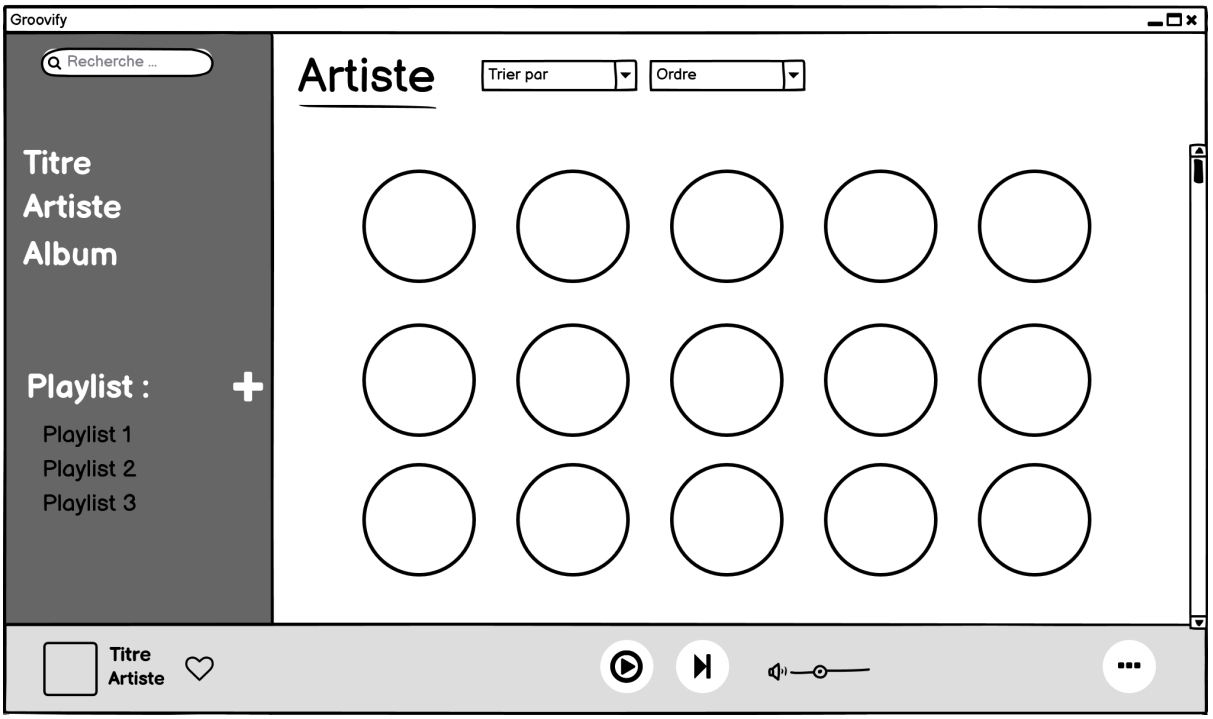
(exemple de la police - Roboto Medium.)

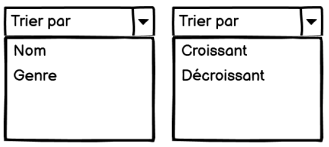
Pour l’accessibilité, nous utilisons une gamme de couleurs de sorte à ce que même avec un daltonisme achromatopsique (absence totale de perception des couleurs, la vision se fait en nuance de gris), il est possible de distinguer chaque élément de l’interface.

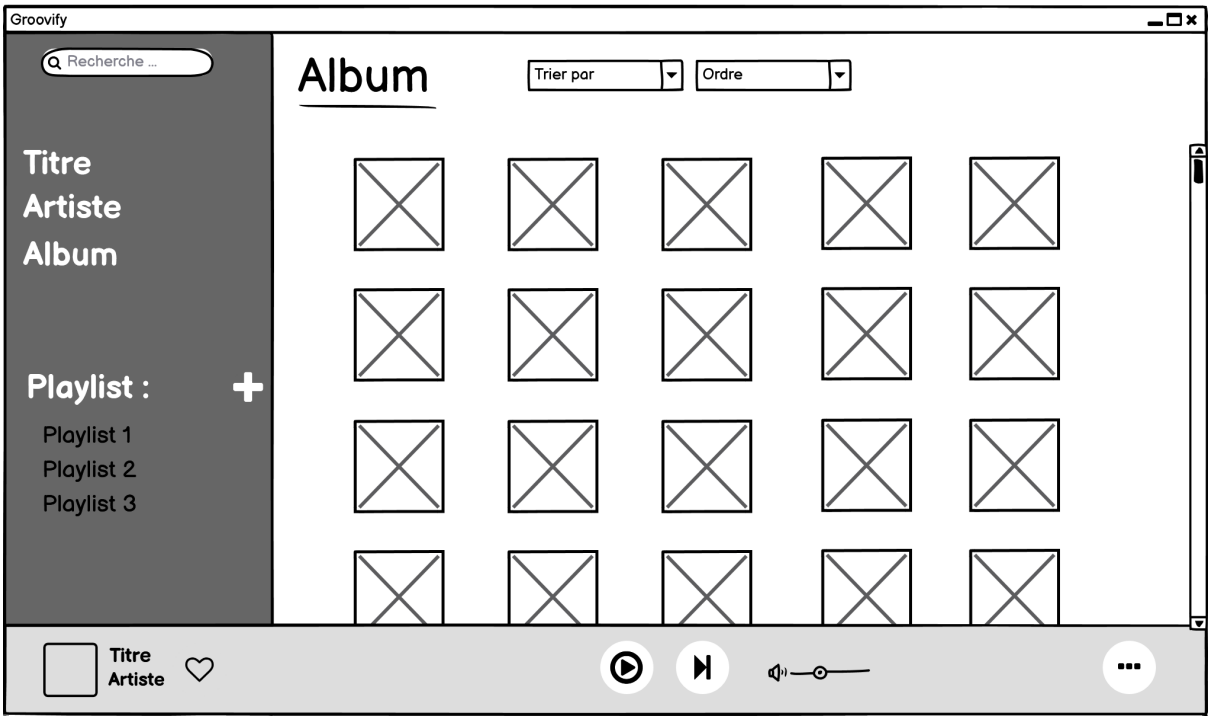
Wireframe

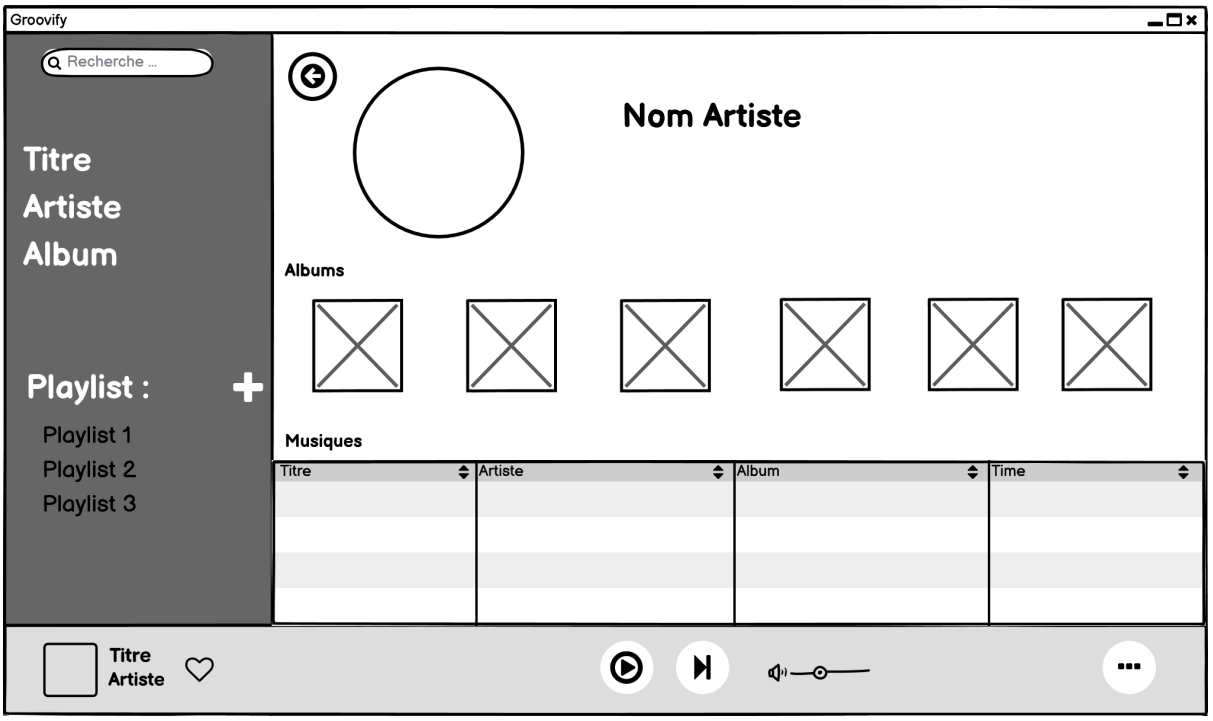

Groovify
Recherche ...
Titre
Titre
Artiste
Album
Playlist :
Playlist 1
Playlist 2
Playlist 3
Titre
Artiste
Titre
Artiste
Album
Time
Titre
Artiste

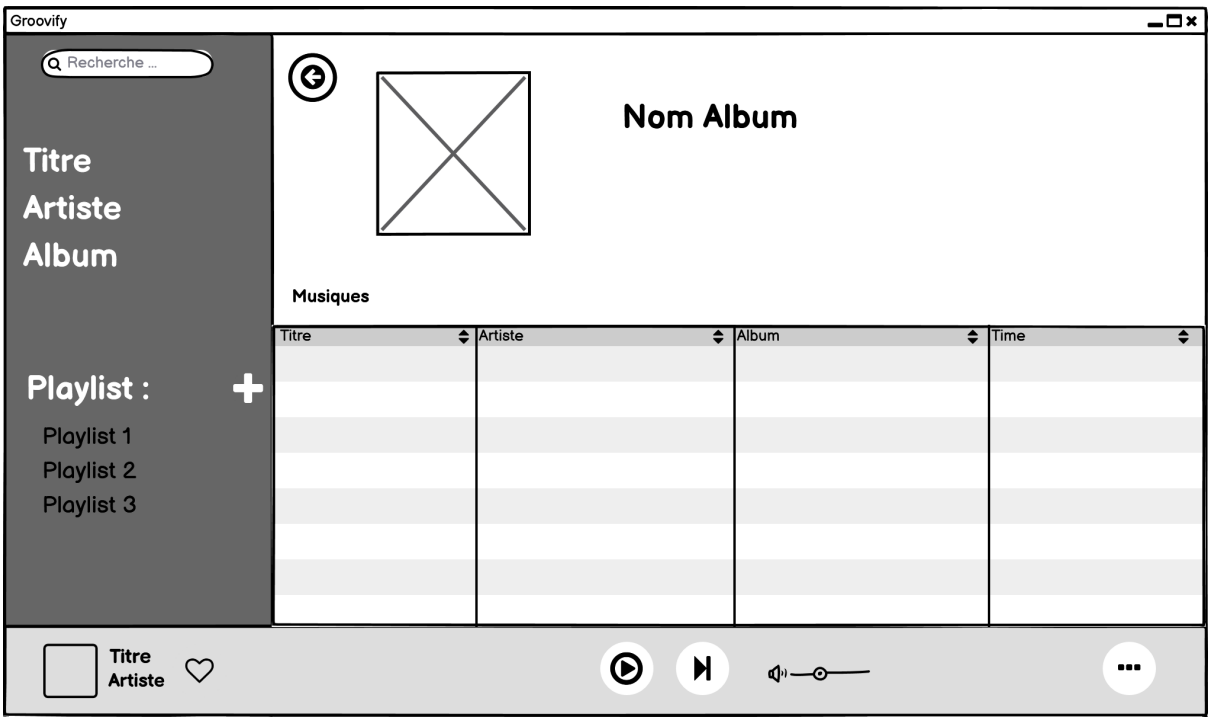

Trier par
Nom
Genre
Trier par
Croissant
Décroissant

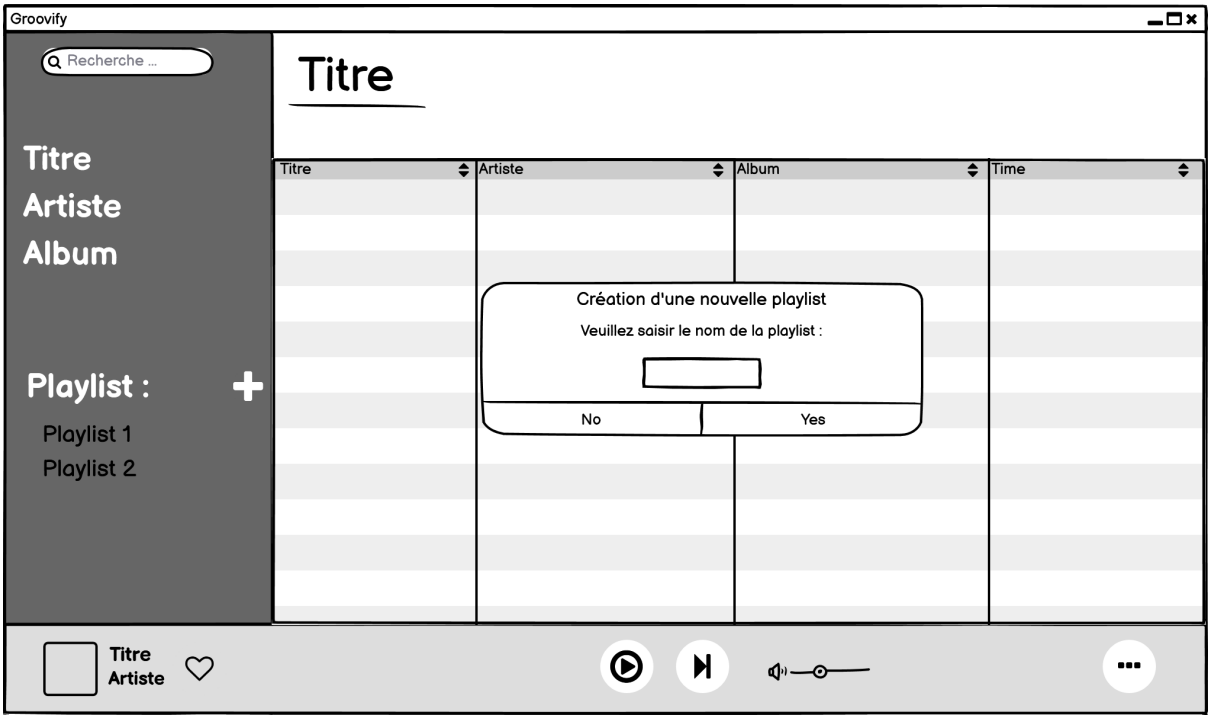

Groovify
Recherche ...
Artiste
Trier par
Ordre
Titre
Artiste
Album
Playlist :
Playlist 1
Playlist 2
Playlist 3
Titre
Artiste

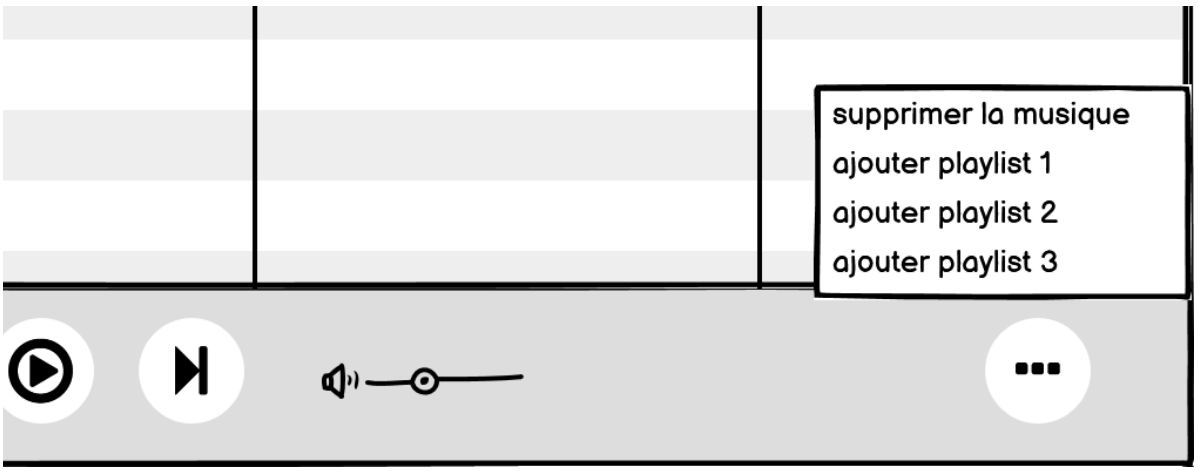

Trier par
Nom
Genre
Trier par
Croissant
Décroissant


Groovify
Recherche ...
Album
Trier par
Ordre
Titre
Artiste
Album
Playlist :
Playlist 1
Playlist 2
Playlist 3
Titre
Artiste


Groovify
Recherche ...
Nom Artiste
Titre
Artiste
Album
Albums
Playlist :
Playlist 1
Playlist 2
Playlist 3
Musiques
Titre
Artiste
Album
Time
Titre
Artiste


Groovify
Recherche ...
Titre
Artiste
Album
Playlist :
Playlist 1
Playlist 2
Playlist 3
Nom Album
Musiques
Titre
Artiste
Album
Time
Titre
Artiste


Groovify
Recherche ...
Titre
Titre
Artiste
Album
Playlist :
Playlist 1
Playlist 2
Titre
Artiste
Album
Time
Création d'une nouvelle playlist
Veuillez saisir le nom de la playlist :
No
Yes
Titre
Artiste


supprimer la musique
ajouter playlist 1
ajouter playlist 2
ajouter playlist 3

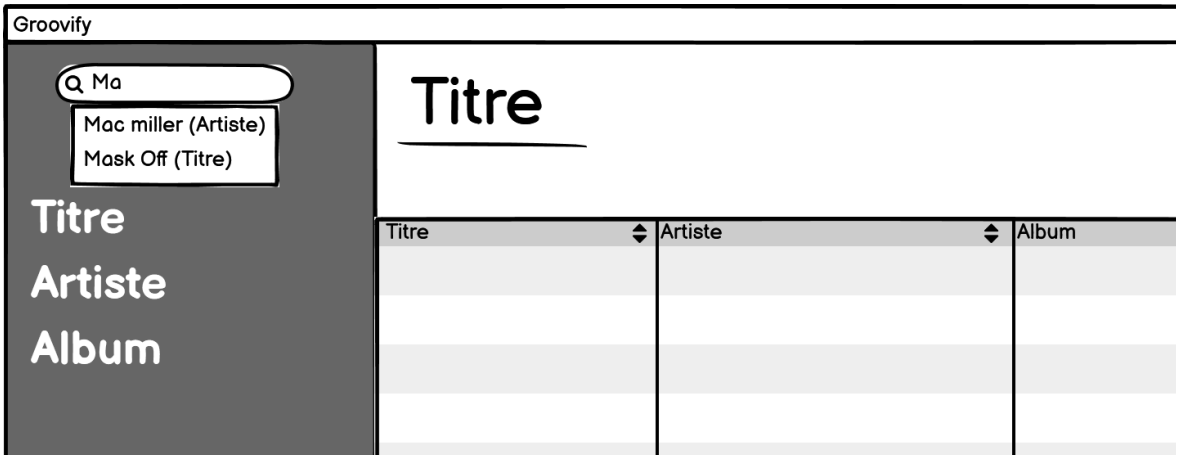
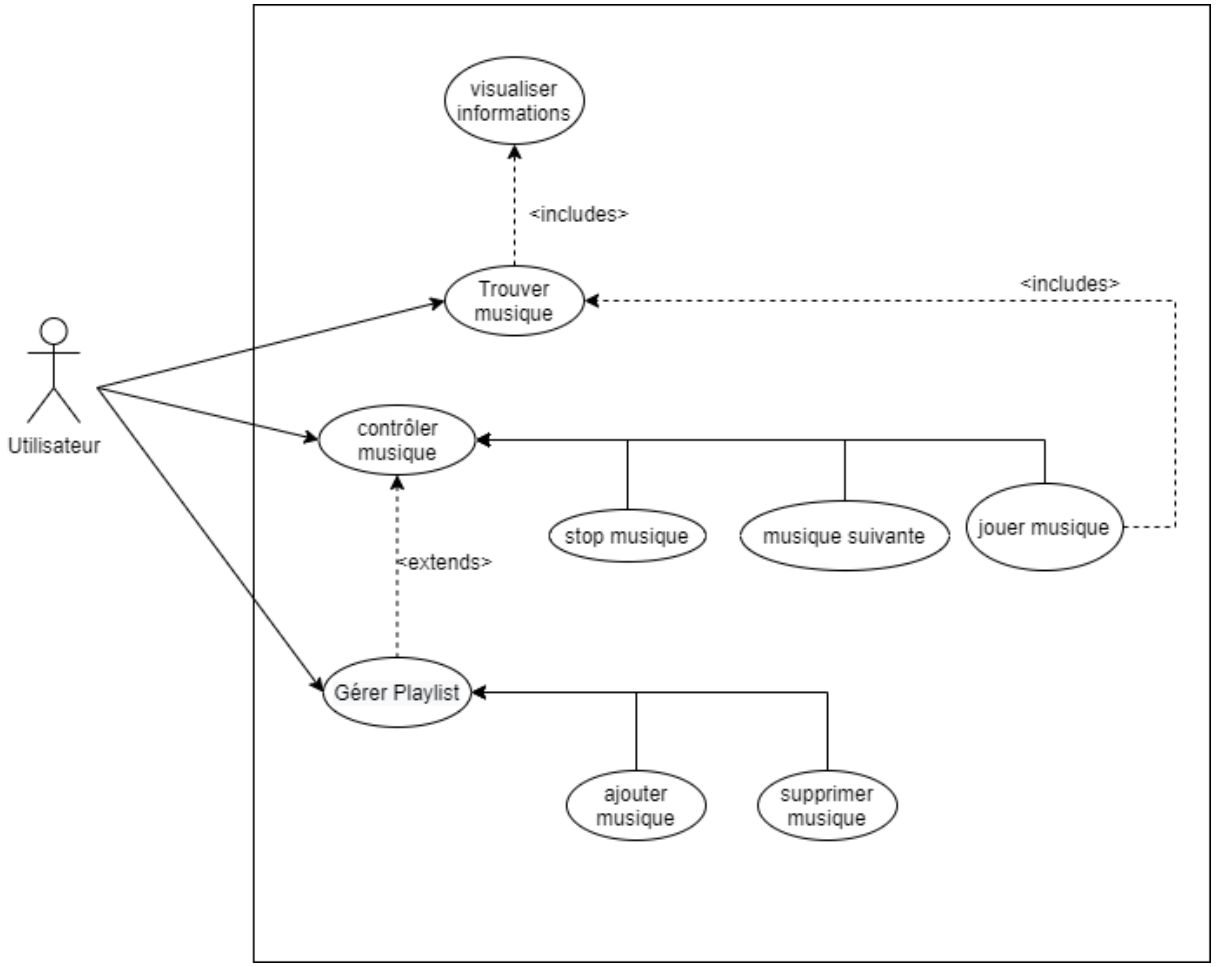

Groovify
Ma
Mac miller (Artiste)
Mask Off (Titre)
Titre
Artiste
Album
Titre
Titre
Artiste
Album

Diagramme d’utilisation



Cas “Trouver musique” :

Nom :	Trouver musique
Objectif :	Trouver une musique spécifique
Acteur principal :	Utilisateur
Acteur secondaires	Aucun.

Conditions initiales	Aucune.
Scénario d’utilisation	L’utilisateur cherche un Titre/Artiste/Album disponible. Il entre donc le nom de ce qu’il cherche et le sélectionne.
Conditions de fin	L’utilisateur sélectionne un résultat ou appuie sur entrée ce qui lui donne le premier résultat.

Cas "Contrôler musique” :

Nom :	Contrôler musique
Objectif :	Pouvoir choisir la musique jouée ainsi que de l'arrêter.
Acteur principal :	Utilisateur
Acteur secondaires	Aucun.

Conditions initiales	Aucune.
Scénario d’utilisation	L’utilisateur cherche à changer la musique. Il peut en choisir une depuis l’interface principale ou contrôler la lecture depuis
Conditions de fin	Aucune.

Cas "Gérer playlist”:

Nom :	Gérer playlist
Objectif :	Pouvoir choisir quels titres mettre dans une playlist.
Acteur principal :	Utilisateur
Acteur secondaires	Aucun.

Conditions initiales	Aucune.
Scénario d’utilisation	L’utilisateur peut créer une playlist. Ensuite, il peut choisir d’ajouter des titres à cette playlist et les voir dans une forme similaire à l'affichage des titres.
Conditions de fin	Aucune.

Cas “Visualiser informations” :

Nom :	Visualiser informations.

Objectif :	Afficher les informations sur l'artiste (Discographie, Biographie…) et les Albums (liste des titres).
Acteur principal :	Utilisateur
Acteur secondaires	Aucun.

Conditions initiales	Aucune.
Scénario d'utilisation	L'utilisateur sélectionne un album ou artiste. Le logiciel affiche alors la page correspondante.
Conditions de fin	Aucune.

Cas "stop musique" :

Nom :	stop musique.
Objectif :	Arrêter la musique en cours de lecture.
Acteur principal :	Utilisateur
Acteur secondaires	Aucun.

Conditions initiales	Une musique est en cours de lecture.
Scénario d'utilisation	L'utilisateur clique sur Pause, la musique en cours de lecture s'arrête alors.
Conditions de fin	La musique a été arrêtée.

Cas "musique suivante" :

Nom :	musique suivante.
Objectif :	Passer au titre suivant.
Acteur principal :	Utilisateur
Acteur secondaires	Aucun.

Conditions initiales	Aucune.

Scénario d’utilisation	L’utilisateur souhaite passer au titre suivant, il clique donc sur le bouton Suivant.
Conditions de fin	La musique est passée.

Cas “jouer musique” :

Nom :	jouer musique
Objectif :	L’utilisateur cherche à jouer une musique.
Acteur principal :	Utilisateur
Acteur secondaires	Aucun.

Conditions initiales	Aucune.
Scénario d’utilisation	L’utilisateur souhaite lancer la lecture d’un titre, il clique donc sur le titre pour lancer la lecture.
Conditions de fin	Aucune.

Cas “ajouter musique” :

Nom :	ajouter musique
Objectif :	Ajouter un titre à une playlist.
Acteur principal :	Utilisateur
Acteur secondaires	Aucun.

Conditions initiales	Avoir au moins une playlist dans laquelle ajouter un titre.
Scénario d’utilisation	Avec les options du titre en écoute, il est possible de l’ajouter à une playlist ou alors de faire clique droit -> ajouter playlist x.
Conditions de fin	Le titre a été ajouté.

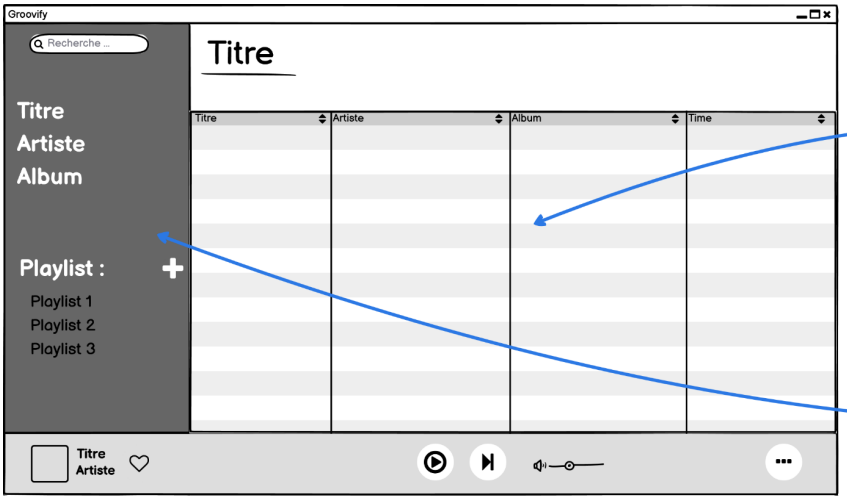
Cas “supprimer musique” :

Nom :	supprimer musique

Objectif :	Supprimer une musique d’une playlist
Acteur principal :	Utilisateur
Acteur secondaires	Aucun.

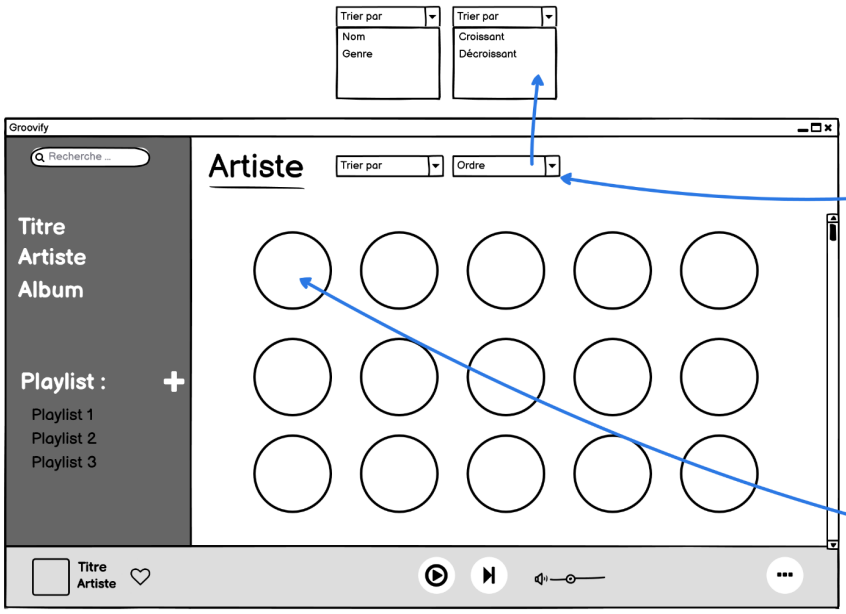
Conditions initiales	La playlist sélectionnée contient au moins une chanson (à supprimer).
Scénario d’utilisation	L’utilisateur veut supprimer un titre d’une playlist. Il va donc sur la playlist et fait clique droit -> supprimer.
Conditions de fin	La musique a été supprimée de la playlist.

Storyboard



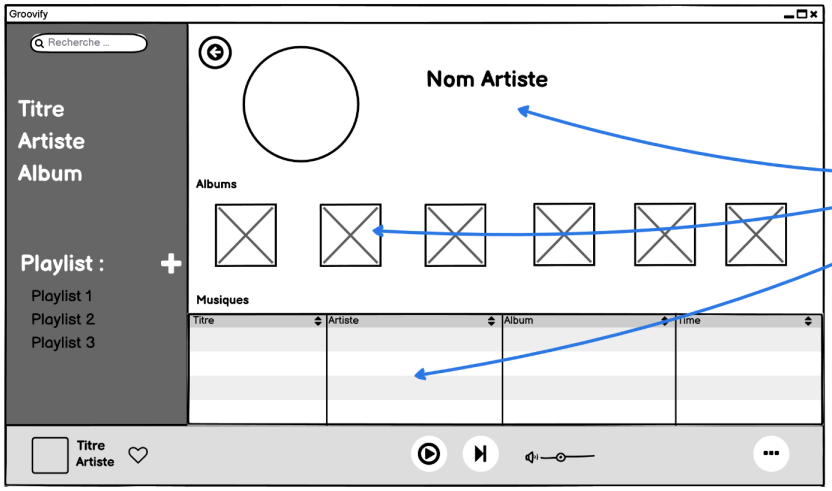
Lorsque l'utilisateur ouvre l'application, il voit apparaitre la liste des musiques chargés par l'application. En cliquant sur une des musiques il peut ainsi l'écouter.

En utilisant le menu sur la gauche,il peut ainsi arriver à différentes parties de l'application, qui chacune permettent d'utiliser de nouvelles fonctionnalitées.

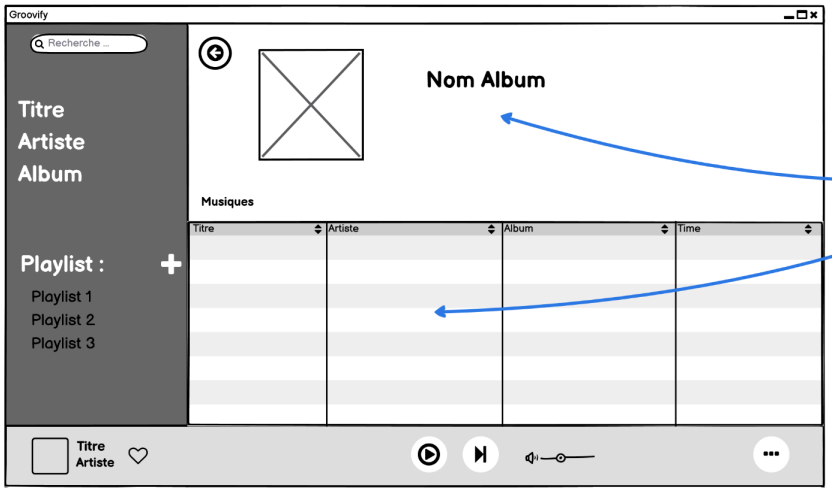


Après avoir sélectionné Artiste dans le menu, l'utilisateur a la possibilité d'observer les artistes qui ont été chargés par l'application ainsi que de les trier

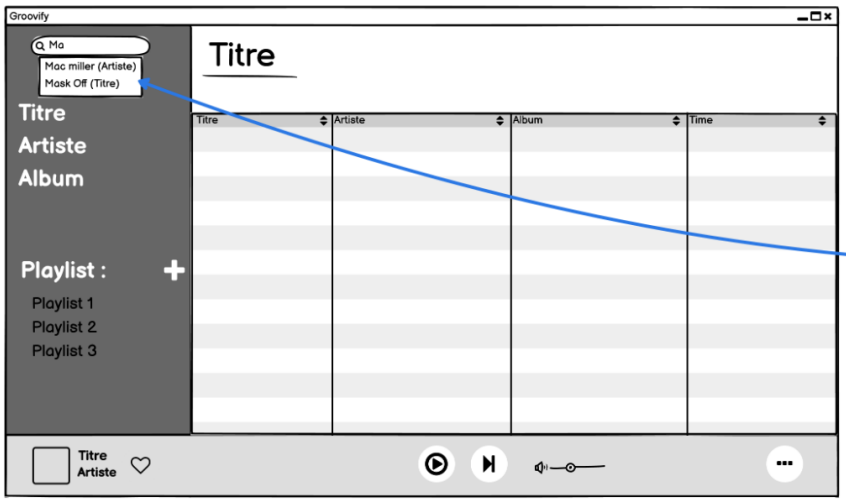
L'utilisateur a ainsi la possibilité de choisir un artiste pour obtenir plus d'information sur celui-ci.



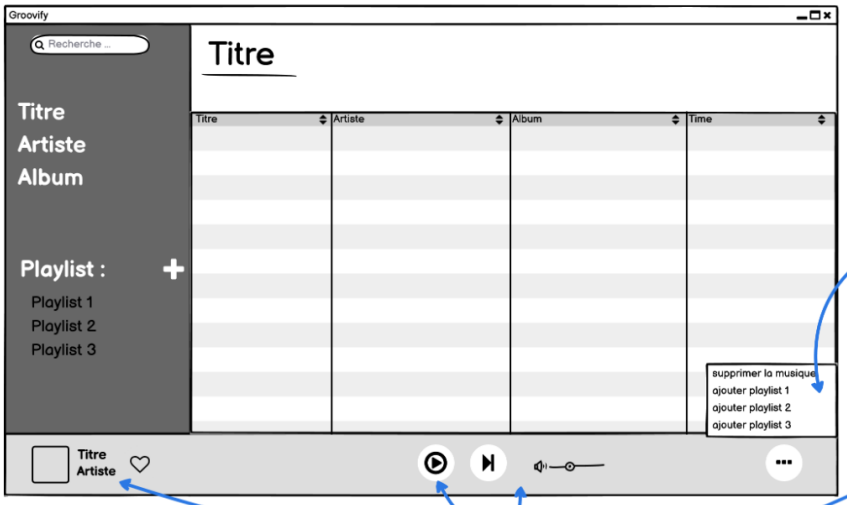
L'utilisateur peut obtenir des informations sur l'artiste mais aussi savoir quelles musiques et albums sont disponible sur l'application.



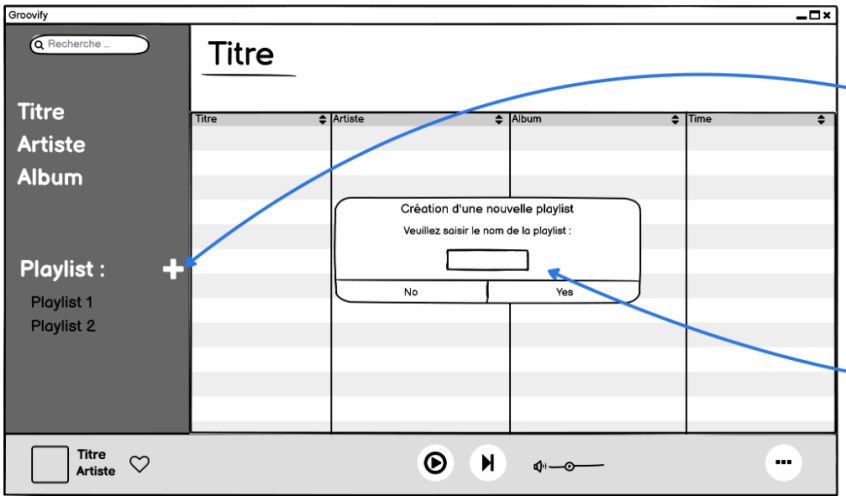
S'il sélectionne un album, il peut comme avec l'artiste obtenir des informations sur l'album et sélectionner la musique qu'il souhaite écouter.



Grâce au système de recherche, le client a la possibilité de rechercher directement son artiste ou sa musique préféré. Les résultats apparaitront juste en dessous, où il pourra sélectionner le résultat.



Une fois la musique sélectionnée, il sera possible à l'aide de la barre du bas de la contrôler, d'obtenir des informations sur la musique ainsi que de la supprimer ou de l'ajouter a une playlist existante.

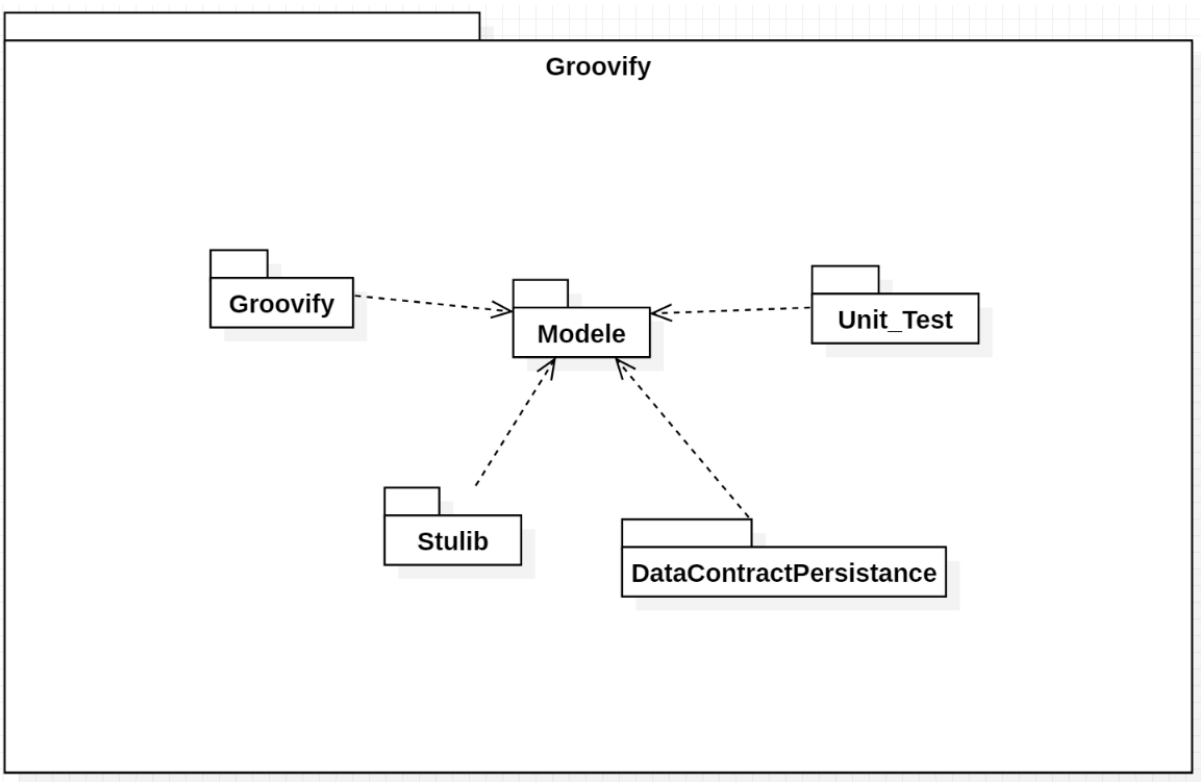


Pour que l'utilisateur crée une nouvelle playlist, il devra utiliser le + à côté du menu de la Playlist.

A ce moment une fenêtre apparaitra lui demandant de choisir un nom pour ça nouvelle playlist.

Diagramme de Paquetage

a) Diagramme



b) Explication

- Appli, package qui contient toutes les vues et le binding de l'application:

    - Son rôle : Gérer toutes les « entités » présentes dans l'application (comme les films ou séries, acteurs y figurant ou même les utilisateurs) en comportant des classes instanciables qui permettent de les concrétiser.

    - Son contexte : Il s'agit du premier assembly qu'on ait eu à créer et pour cause, sans vues, il n'y a pas d'application ni même de liaison à faire avec le code-behind. C'est un assembly clé du projet.

- Modèle, package comprenant toutes les classes :

    - Son rôle : Gérer toutes les « entités » présentes dans l'application (comme les Musiques, Albums, Artistes, Playlists) en comportant des classes instanciables qui permettent de les concrétiser.

    - Son contexte : Cet assembly trouve également une position clé dans le projet car il permet à l'application de prendre vie en faisant fonctionner les vues et en concrétisant les entités vues dans le XAML, en leur attribuant des classes et des propriétés.

- Unit_Test, où l'on trouve tous les tests unitaires des classes de Modèles :

    - Son rôle : Tester toutes les classes, propriétés et méthodes de l'assembly Modèle indépendamment. On trouve dans les fichiers de cet assembly des tentatives de création de bug lors de l'instanciation de classes en tous genres pour essayer de sécuriser au maximum l'utilisation de l'application.

    - Son contexte : Cet assembly a été créé en même temps que Modèle. Son rôle est bien moins important que le reste des autres assembly, mais au cours du développement de l'application, il a joué un rôle crucial dans la programmation de l'application.

- Stulib, package permettant la simulation de la persistance :

    - Son rôle : Créer l'entièreté des instances nécessaires au bon remplissage de l'application, avec de fausses données.

    - Son contexte : Ce projet a été créé pour faire persister les données à travers les différentes Vues et Bindings de l'application. Ainsi nous avons pu utiliser notre application avec de fausses données faciles à modifier.

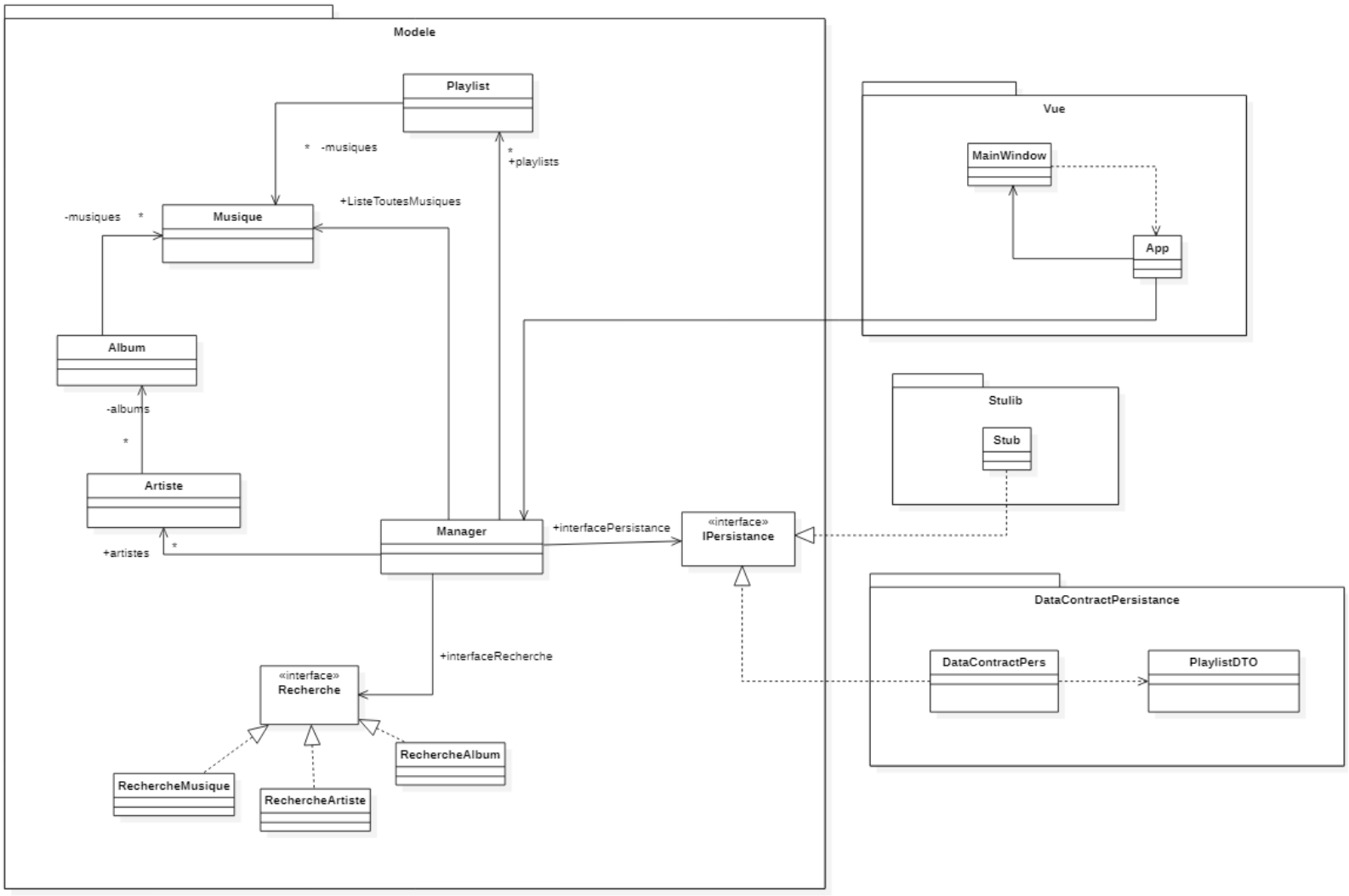
- DataContractPersistance,

    - Son rôle est de permettre la persistance des données modifiées par l’utilisateur sur les Playlists (créer, modifier, ajouter).

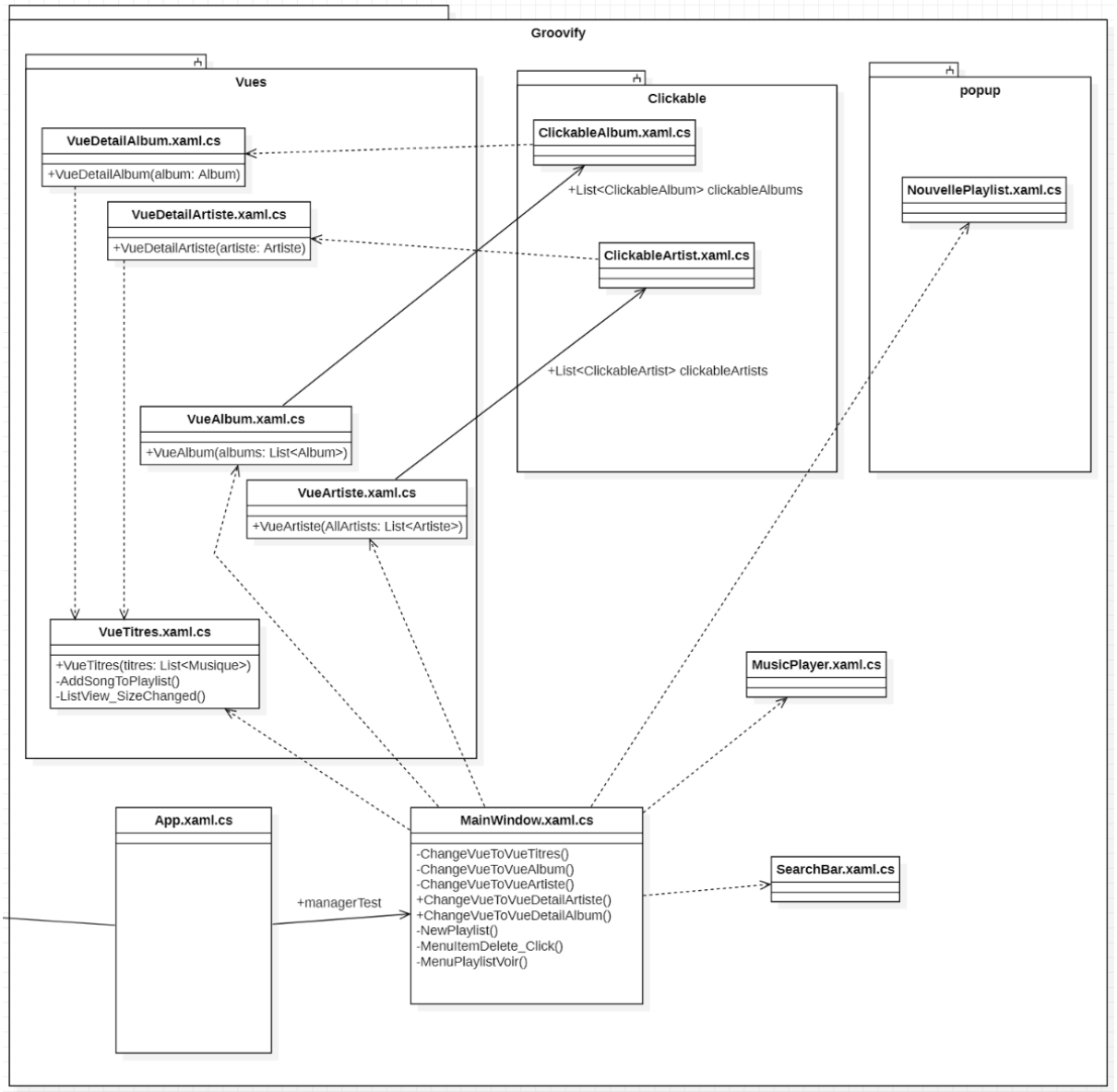
    - Son contexte : Ce projet a été créé pour pouvoir lire et écrire les données des playlists créées par le client à l'aide du Binding de l’application. Ainsi la persistance des playlists est possible et nous ne perdons plus les informations à la fermeture de l’application.

Diagramme de Classe

1) Diagramme version simplifiée



2) Groovify



Vues

Les “vues” sont les UserControls utilisés pour l’affichage en élément principal des données (Titres, Artistes…).
La VueTitres est aussi utilisée comme partie des VueDetail pour y afficher des listes de titres et pour afficher les playlists. Elle prend en entrée une liste de titres à afficher. La structure avec laquelle sont stockées les données permet de facilement accéder à tous les titres d’un artiste ou d’un album. VueTitres contient aussi une méthode permettant de gérer les cliques droit sur les titres pour les utiliser pour ajouter le titre à une playlist et une méthode liée à l’affichage de cette vue (taille horizontale dynamique).

Les VueDetail prennent en entrée la classe qui leur correspond pour l'afficher (la VueDetailArtiste affiche notamment la liste des albums et des titres produits par un artiste spécifique).
Les VueArtiste et VueAlbum prennent en entrée la liste de tous les artistes et la liste de tous les albums.

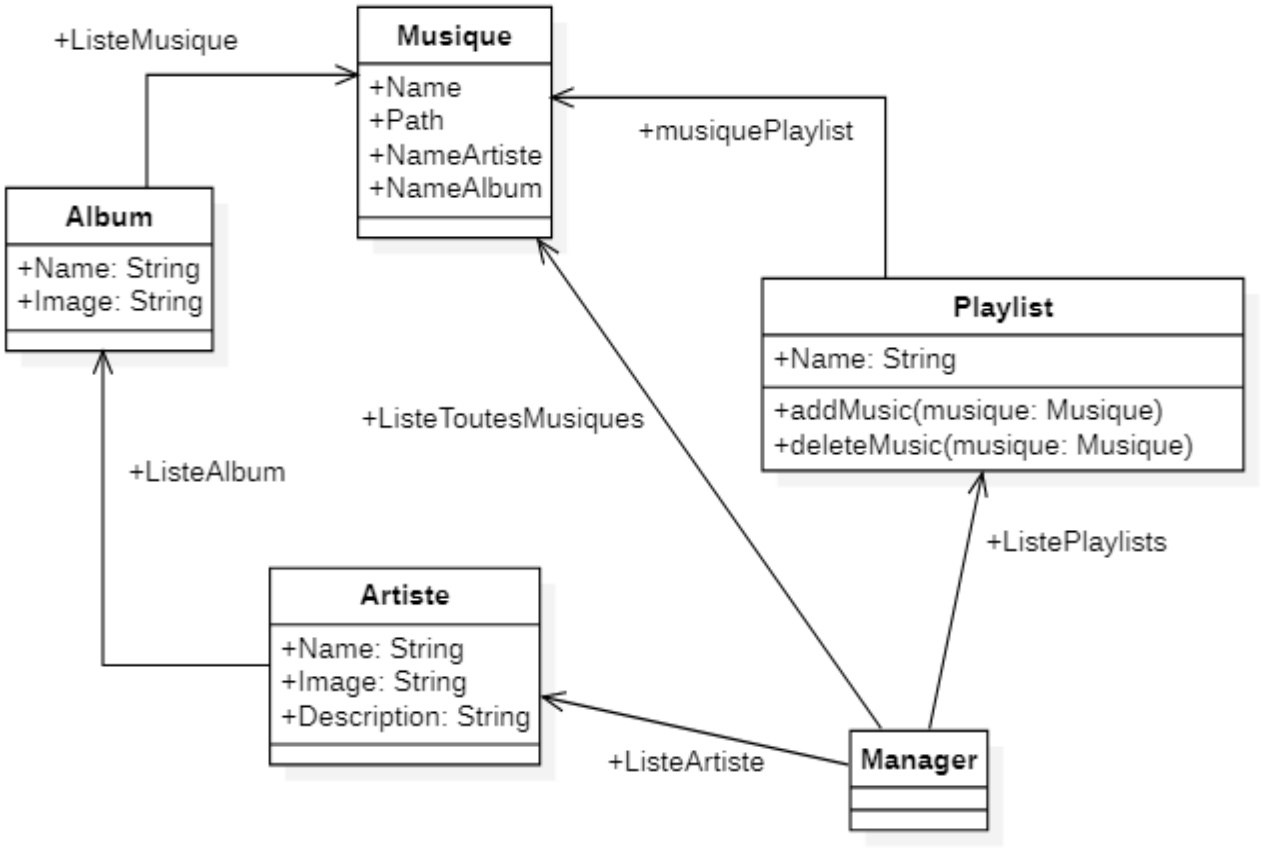
Clickable

Les éléments listés dans clickable peuvent à partir d'une classe Album ou Artiste donner un élément cliquable avec l'image et le titre de l'artiste/album qui, si cliqué, conduit vers une VueDetail de l'artiste/album concerné.

Popup

Quand le symbole + à côté de la liste de playlist est cliqué, une fenêtre popup est ouverte dans laquelle il est possible de choisir le nom de la nouvelle playlist. Il y est possible d'annuler/de valider la création de la nouvelle playlist. Si une playlist avec le même nom existe déjà, elle ne sera pas créée.

3) Classe : Musique / Artiste / Album / Playlist



Ces classes ont pour but de stocker les différentes informations (des Musiques, Artiste, Album et Playlist) donc tout ce que l'on pourrait attendre d'une œuvre musicale mais aussi des informations telle que le chemin du dossier et de l'image de couverture de l'artiste ou encore de l'album.

- Propriétés :
    - Musique
        - String Name;
        - String Path;
        - String NameArtiste;
        - String NameAlbum;

- Album
  - String Name;
  - String Image;
  - List<Musique> ListeMusique;
    Liste des musiques de l'album.
- Artiste
  - String Name;
    Nom de l'artiste
  - String Description;
    Description de l'artiste
  - String Image;
    Chemin de l'image de l'artiste
  - List<Album> ListeAlbum;
    Liste d'Album appartenant à Artiste

- Playlist
  - String Name;
    Nom de la playlist
  - List<Musique> musiquePlaylist;
    Comporte la liste des musiques de la playlist

- Méthodes :

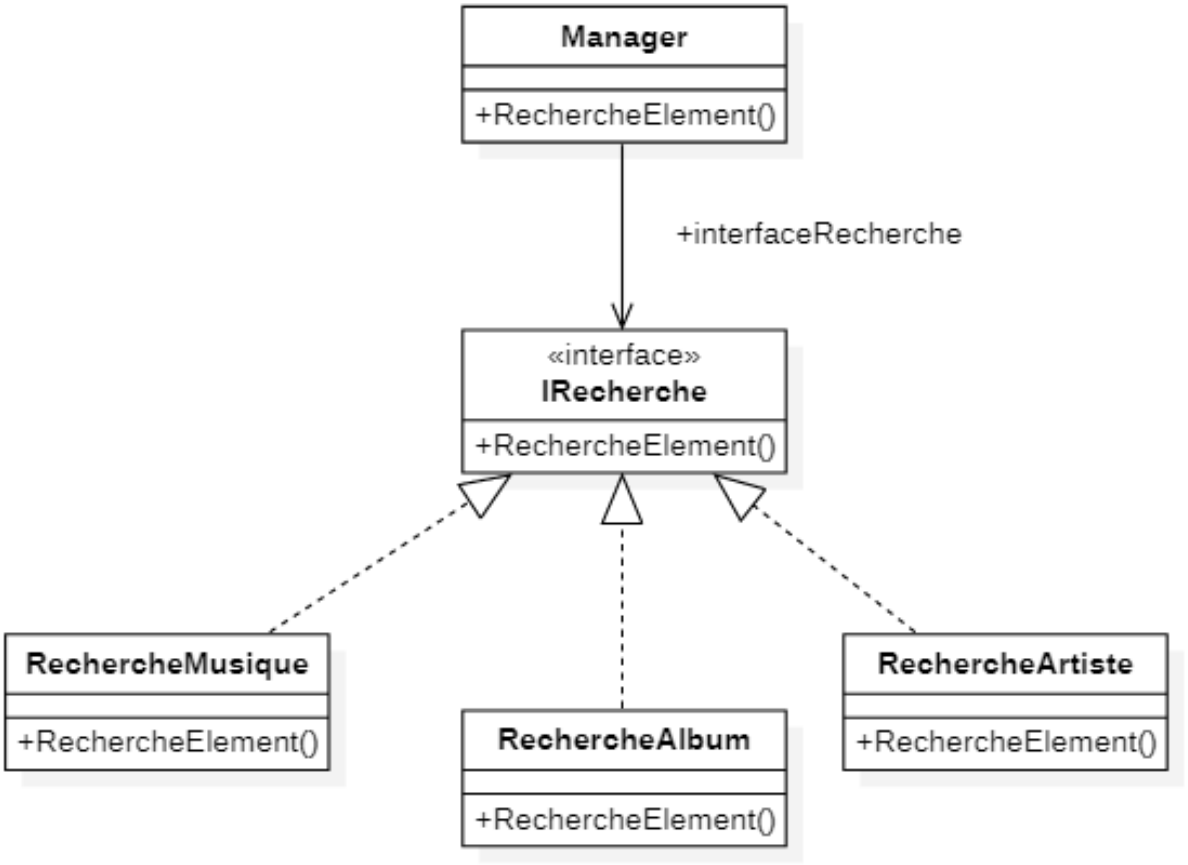
  - Playlist:
    - void addMusic(Musique musique)

      Ajoute une musique dans la liste de la Playlist

    - void deleteMusic(Musique musique)

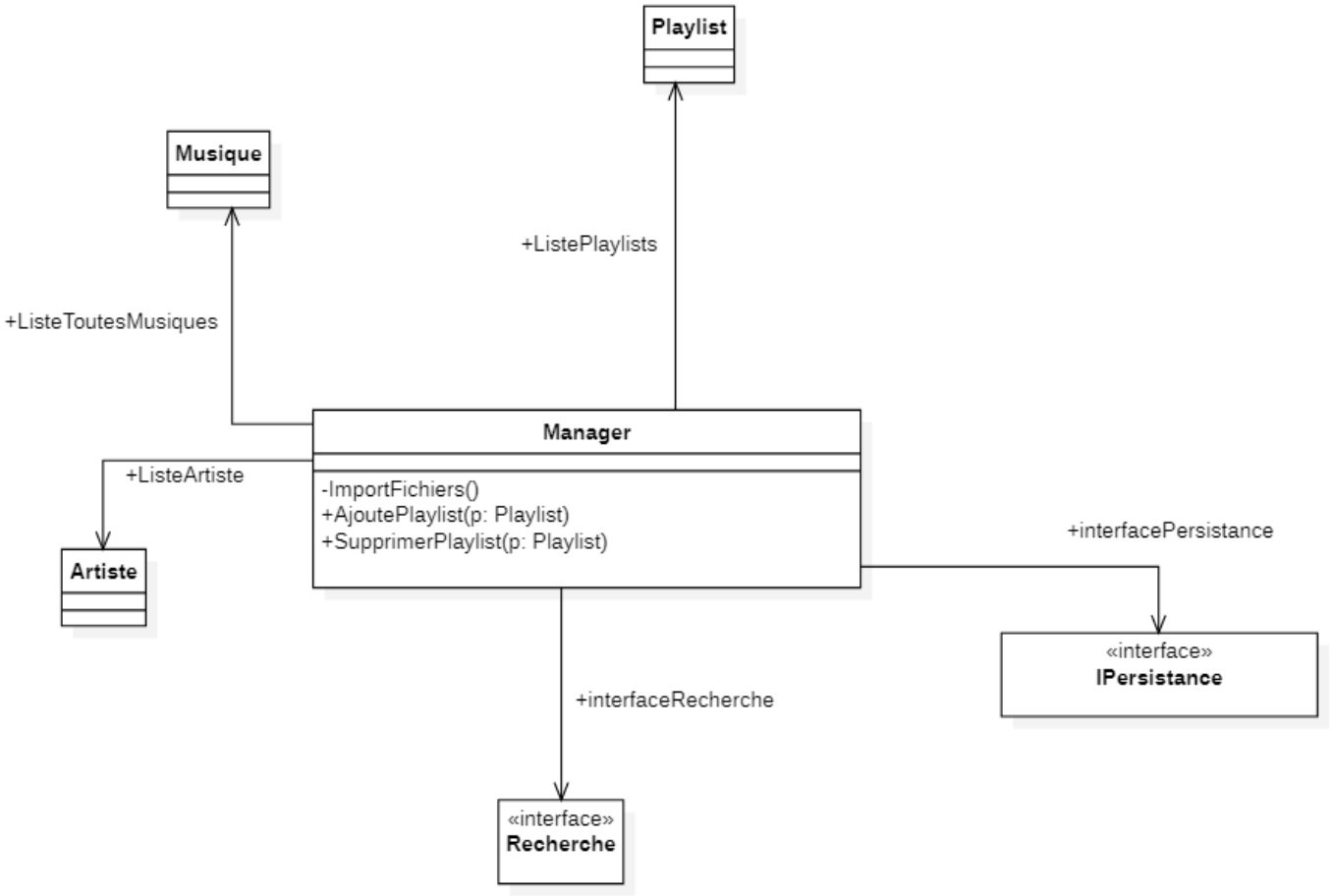
      Supprime une Musique dans la liste de la Playlist

4) Recherche



Dans le package Modèle, une interface IRecherche, comportant trois méthodes à implémenter, c'est trois méthodes correspondes a la recherche d'un objet Artiste, Album, Musique. Cette interface permet la recherche d'un élément dans les différentes listes du package Modèle de manières indépendantes.

5) Manager



On trouve dans la classe Manager deux attributs de deux types interfaces :

- interfacePersistance du type IPersistance : une instance d'une classe fille de l'interface qui constitue notre persistance que nous détaillerons plus bas.

- interfaceRecherche du type IRecherche : une instance d'une classe fille de l'interface qui permet la recherche des éléments (Artiste, Album, Musique) dans la mémoire

Propriétés :

- List<Musique> ListeToutesMusiques;
  Cette liste contient toutes les musiques les musiques en mémoire.
- List<Artiste> ListeArtiste;
  Cette liste contient tous les Artistes en mémoire
- ObservableCollection<Playlist> ListePlaylists;
  Cette collection contient toutes les playlists créées par l'utilisateur.

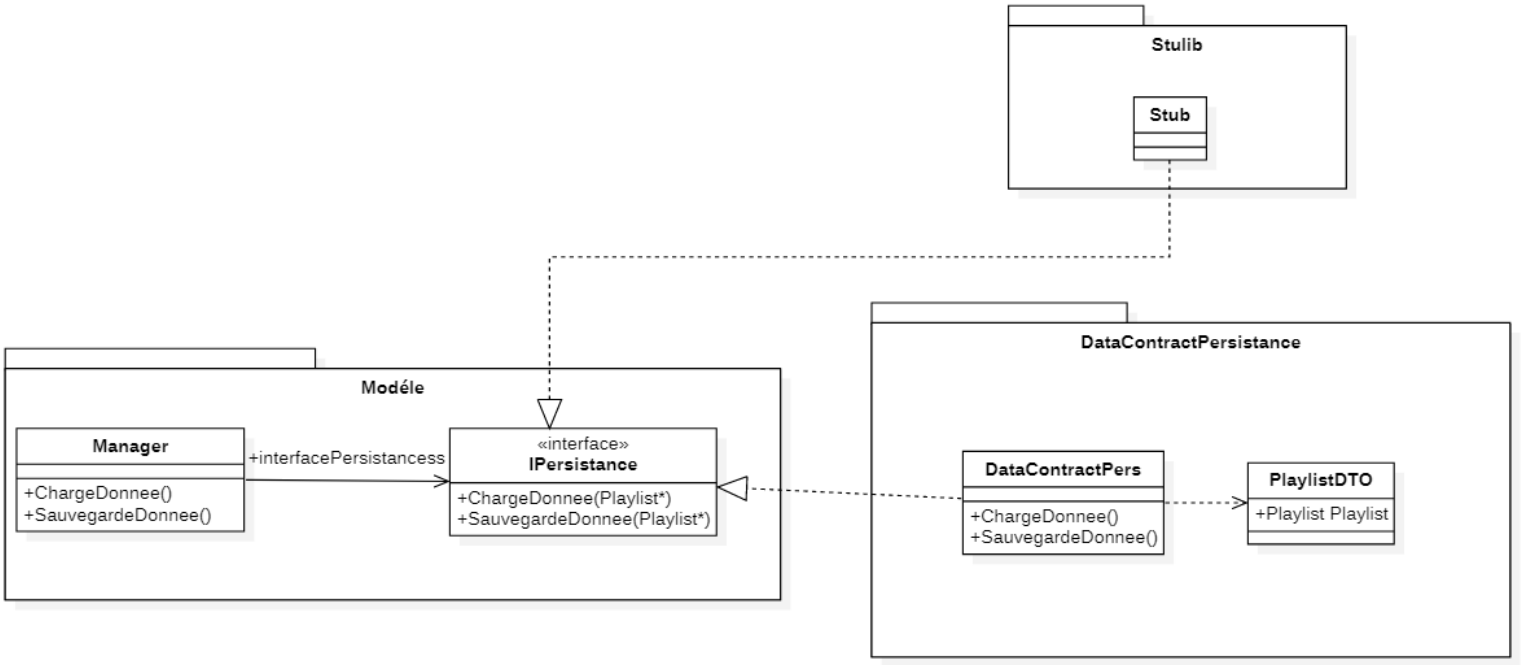
Méthodes :

- void ImportFichiers();
  Permet de récupérer les données des Artistes, Album et Musique à partir de l'arborescence spécifiquement créé pour l'application.
- bool AjoutePlaylist(Playlist p);
  Ajoute une nouvelle Playlist
- bool SupprimerPlaylist(Playlist p);
  Supprime une Playlist

Nous avons choisis que pour toutes les informations concernant les Artistes, Albums et Musiques, nous allions les stocker en utilisant une arborescence de fichier. Avec cette arborescence, il est facile pour un utilisateur d'ajouter à la main des Artistes … Cependant cette méthode ne m'est pas à l'écart de mauvaise manipulation et donc risque de gérer des problèmes à notre application. Ce qui nous laisse la possibilité de faire évoluer notre application, notre code, en remplaçant ce procédé par un système de base de données et ainsi étendre la persistance déjà existante pour les Playlists.

La classe Manager, permet de contrôler entièrement l'application, elle permet de servir d'intermédiaire avec toutes les autres classes. Que ce soit pour modifier des musiques, …, Playlist, ou encore pour la recherche, la persistance. Toutes les actions qui entraînent le Binding, la liaison entre le Back-end et le Front-end, tout se passe dans la classe Manager.

6) Persistance



Dans le package Modèle, une interface IPersistance, comportant deux méthodes à implémenter : Charger et Sauvegarder. Cette interface fait la liaison entre l'application, les Vues et la persistance des données (des Playlists) en utilisant le format JSON.

Comme expliqué dans la première partie, nous avons deux packages destinés à la persistance des données.

Parmi eux, on trouve :

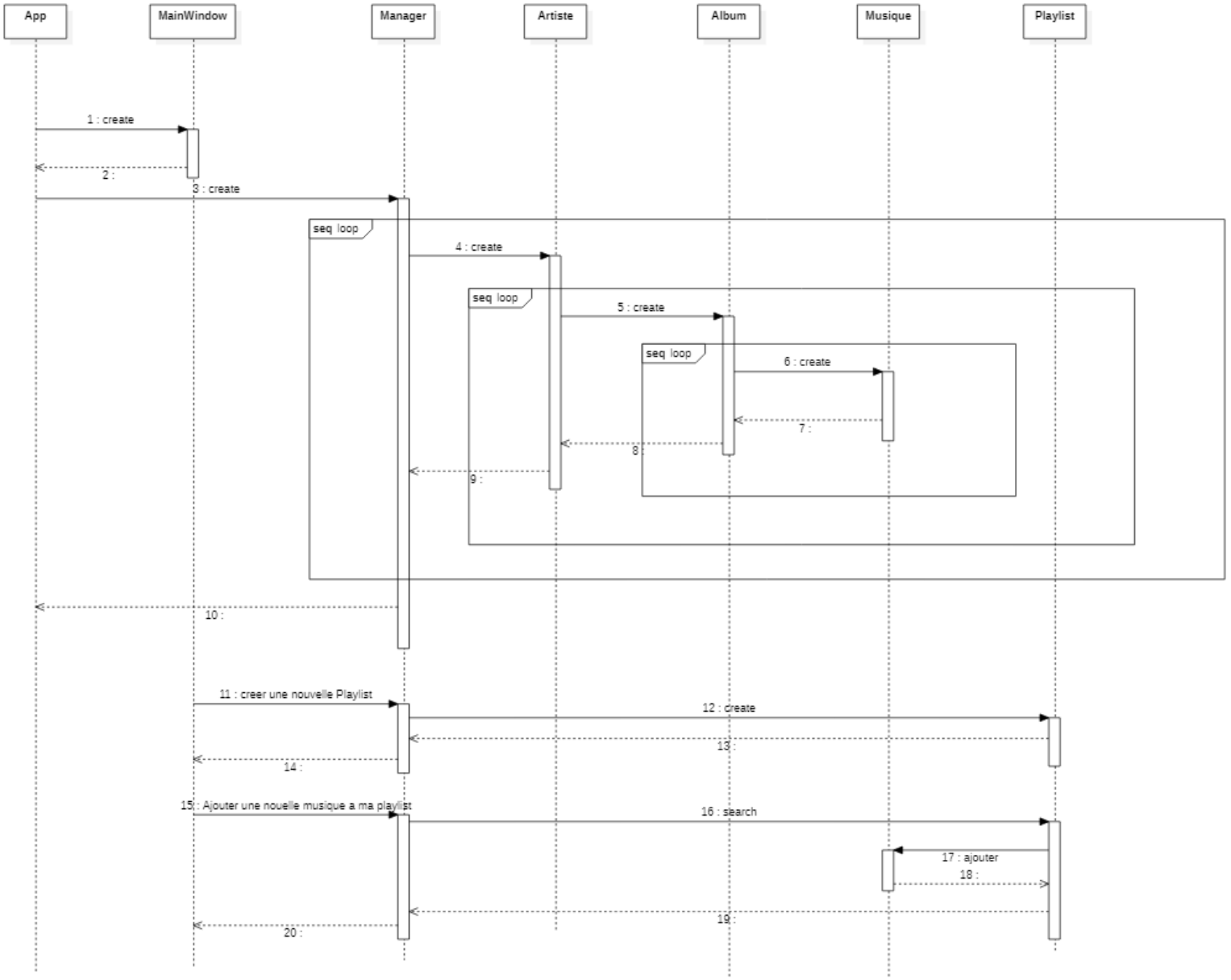
- Stulib : Contient le fichier, désormais inutile, Stub qui, jusqu'à la création d'une persistance, servait à remplir l'application de fausses données que nous avons rentrées afin de travailler sur le Data Binding et autres évènements et liaisons. Cette classe hérite de l'interface IPersistance.

- DataContractPersistance : Est le package qui contient la classe qui va permettre la sauvegarde des données ainsi que le chargement a partir d'un format de fichier bien précis, dans notre cas, sera du JSON. Ainsi pour que les données puissent passer de notre application (objet) au bon format de fichier, on a recours au classe DTO qui permettent de faciliter la conversion entre les deux.

Grâce à cette persistance, nous sommes capables de garder les modifications faites par l'utilisateur sur son application. Par la suite, si nous souhaitons améliorer notre application, il nous sera possible d'étendre cette persistance pour les Artistes, Album et Musique, et ainsi éviter la création d'une arborescence de dossier pour garder ces données et utiliser un système de Base de Donnée plus simple.

Diagramme de séquence

a) Diagramme



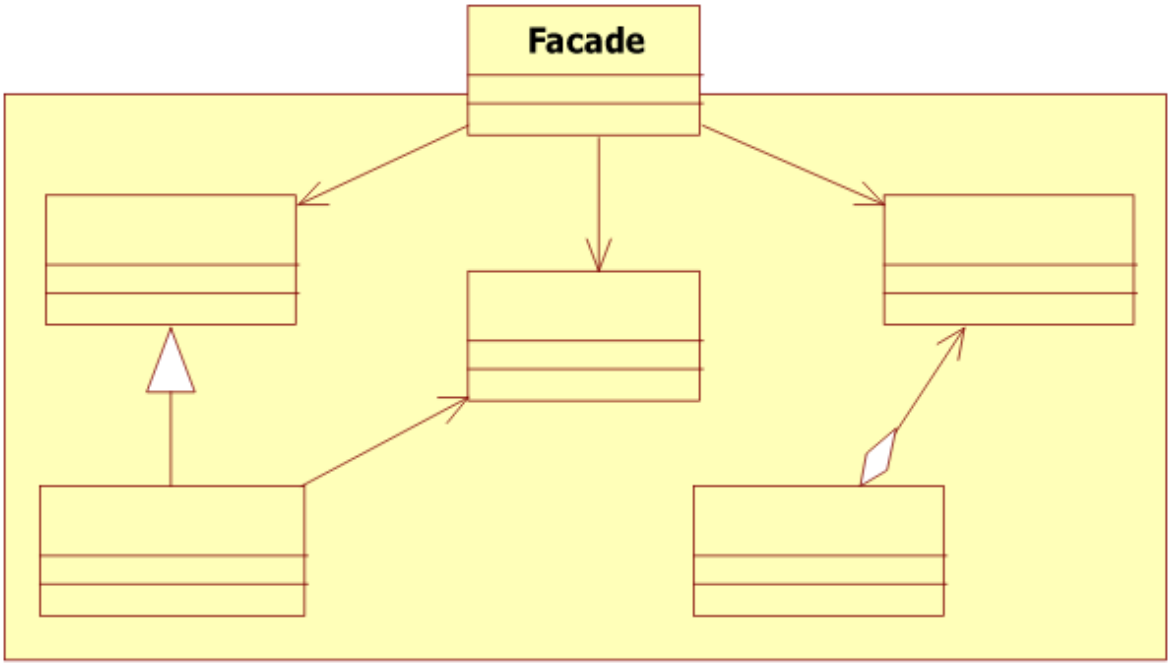
b) Explication

Au lancement de l'App, une instance de la classe Manager est créée. Dans ce manager sont créées toutes les classes qui contiennent les données des musiques (Artiste, Album et Titres). Ces classes sont ensuite automatiquement remplies à partir du contenu du dossier Musiques présent dans les fichiers de l'application. Elles sont donc ensuite accessibles en important le manager déclaré dans App.

Nous avons également représenté la création de Nouvelle playlist ainsi que l'ajout d'une nouvelle musique à une playlist. Pour la création d'une nouvelle playlist, on vérifie que le nom de cette playlist n'existe pas déjà. Dans ce cas la la playlist n'est pas créée. Cette playlist est ensuite enregistrée dans la liste de playlists contenue dans le manager qui permet notamment de les afficher sur l'interface principale. Pour ajouter une nouvelle musique à une playlist on récupère la chanson cliquée et le nom de l'option choisie dans le menu déroulant pour ensuite chercher la playlist correspondante et y ajouter la chanson sélectionnée.

Explication des patrons

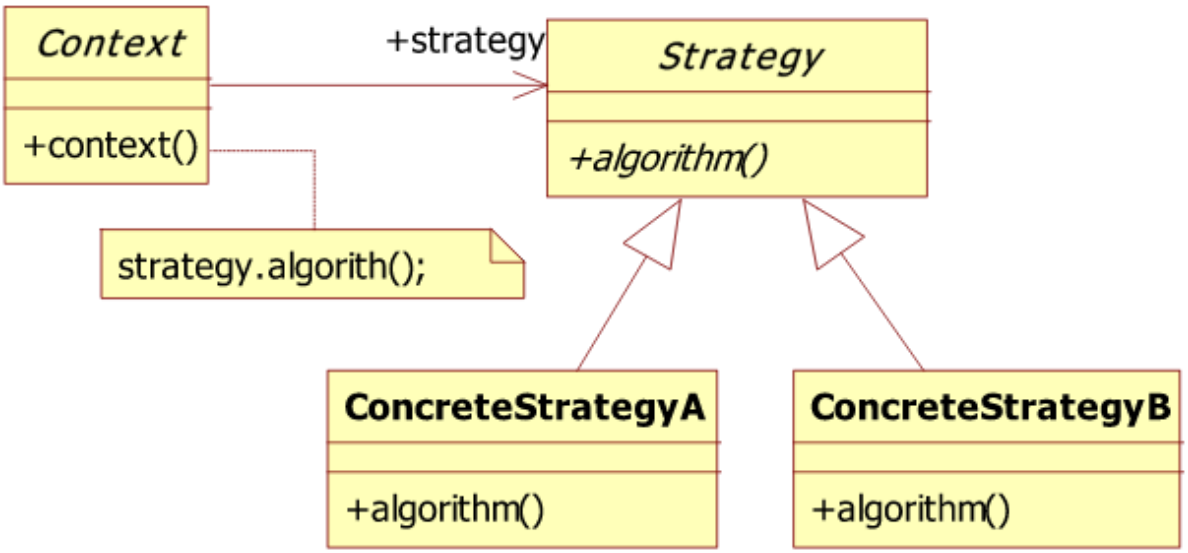
Patron de Conception :



- **Façade :**

Le principe de fonctionnement d'une façade est simplement d'offrir une interface unifiée au client pour unir les sous-systèmes potentiellement complexes d'une application. Et ainsi, elle sert à notre projet à simplifier l'interaction avec des sous-systèmes qui se compliquent au fur et à mesure que le projet avance.

Cette façade offre une certaine propreté du code. Cela permet de fournir une interface simple (Manager) du sous-système (Modèle). Il permet ainsi de rendre plus lisible et plus facile l'utilisation de la bibliothèque de classe. Il permet aussi d'encapsuler les différentes classes en maîtrisant les accès pour éviter une utilisation non prévue. Cela réduit aussi les dépendances ce qui offre une maintenance et une évolution plus simple.



- **Stratégie :**

Le patron stratégie permet de sélectionner des algorithmes à la volée, au cours de l'exécution du programme. Le patron de conception stratégie est utile pour des situations où il est nécessaire de permuter dynamiquement les algorithmes utilisés dans une application. Le patron stratégie est prévu pour fournir le moyen de définir une famille d'algorithmes, encapsuler chacun d'eux en tant qu'objet, et les rendre interchangeables. Ce patron laisse les algorithmes changer indépendamment des clients qui les emploient.

Ce patron nous permet de gérer le code de façon propre et permet en cas de maintenance, de garder les mêmes méthodes, mais donne la possibilité d'avoir des algorithmes différents et donc de pouvoir faire évoluer (améliorer) le code sans mettre en cause l'intégrité de l'application.