ENSSAT LANNION

UNIVERSITÉ DE RENNES 1

Rapport technique - Projet JEE

Étudiants :
(groupe 8)
Erwan Garreau
Kelig Jouquan-Mary
Baptiste Martin
Lilian Savona

Sommaire

I - Introduction

A) Contexte

Ce rapport technique a pour but de présenter le projet développé au cours du module Programmation Web Client/Serveur (JEE) de l'unité d'enseignement Algorithmique et Programmation.
L'objectif de ce projet est de pouvoir mettre en pratique au travers d'une activité concrète les connaissances acquises au cours des cours magistraux et des séances de travaux pratiques.

B) Objectif

Notre équipe a pour mission de créer une plateforme web de streaming musical avec le langage de programmation Java. Pour réaliser ce projet, nous nous basons sur le travail préparatoire réalisé au cours du module de modélisation et méthode de conception.

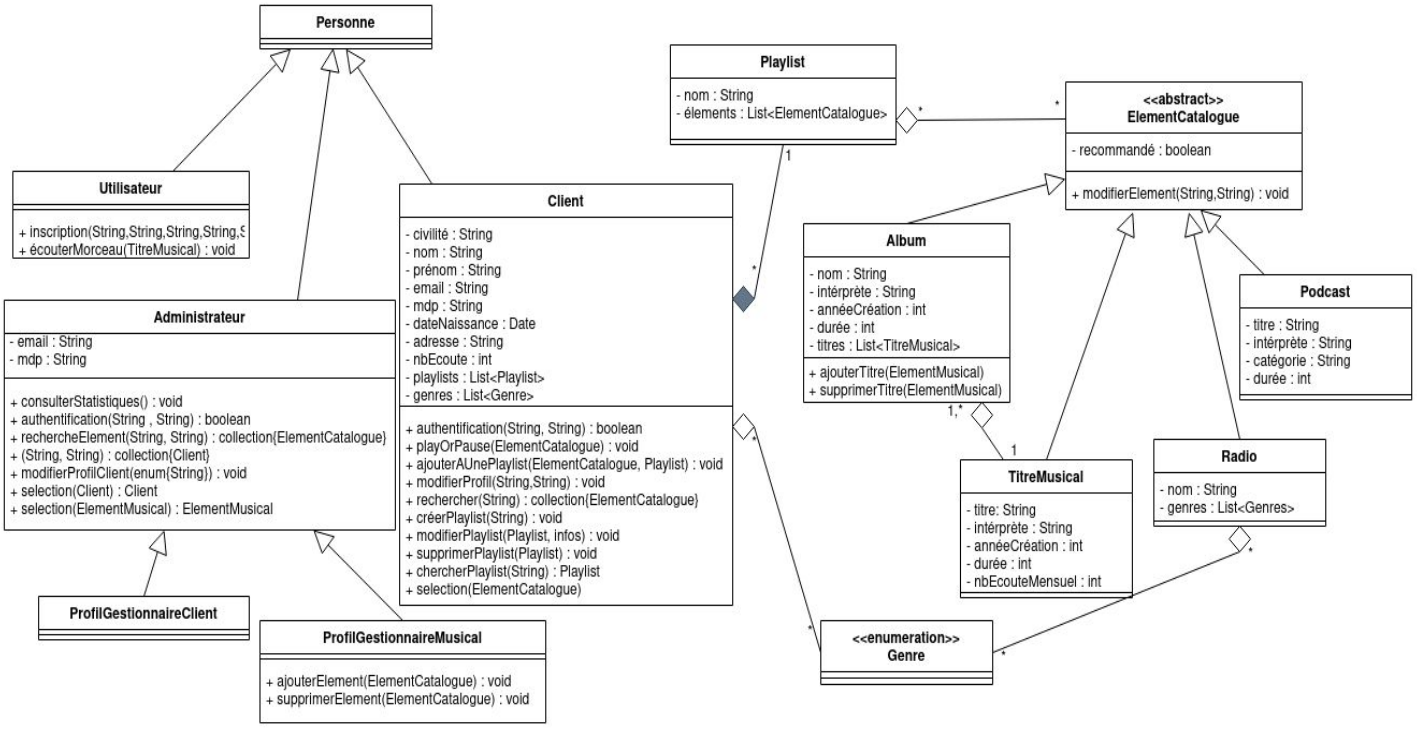
Dans ce rapport, nous observerons les différents aspects techniques de notre projet tels que les diagrammes de classes, le modèle des données développé avec SQL Workbench et les implémentations réalisées.

II- Diagramme de classes

Nous avons réalisé ce diagramme au cours du projet de modélisation associé. Nous avons basé en grande partie notre schéma relationnel et nos classes d'Objets java à partir de celui-ci, tout en mettant à jour certains aspects de ce diagramme lorsque cela nous semblait pertinent, en ajoutant une table d'interprètes par exemple.

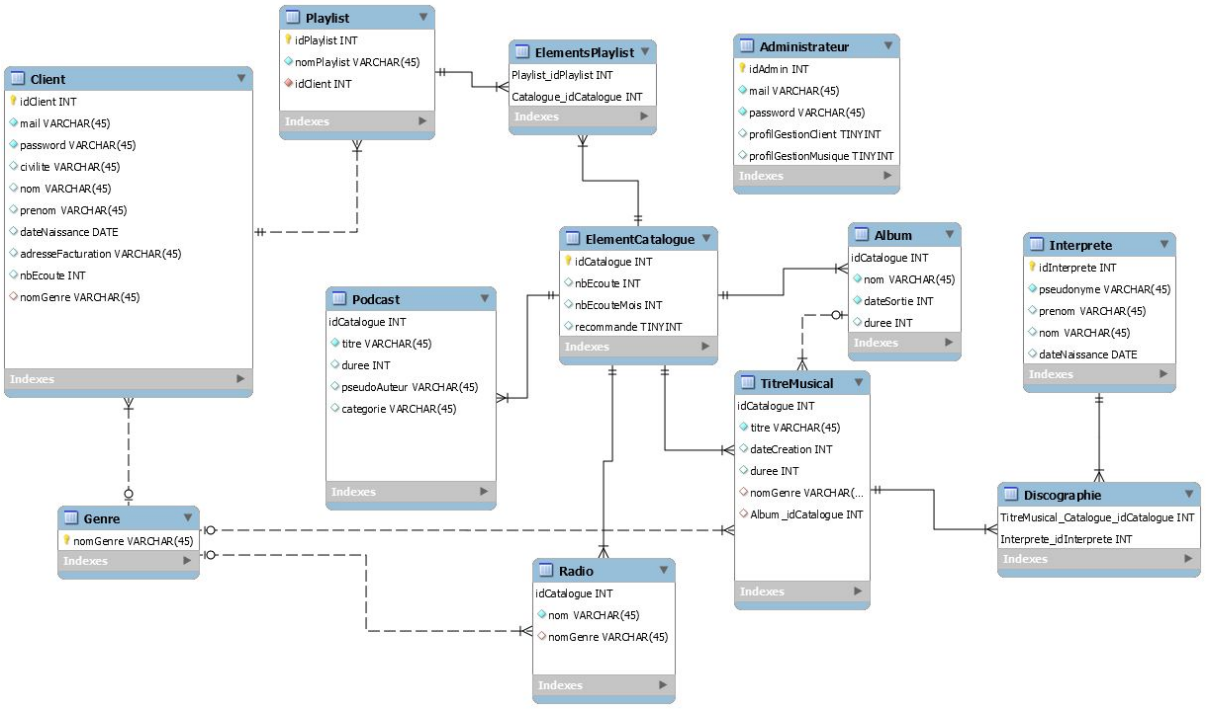
Les fonctionnalités disponibles dans chaque Acteur sont décrites plus précisément dans la partie **VI - B) Architecture Globale - Fonctionnalités**

Ce diagramme est également disponible en pièce jointe dans les livrables.



III - Modèle de données

A) Schéma relationnel



Ce Schéma Relationnel est également disponible en pièce jointe dans les livrables.

Dans la majorité des tables, un attribut correspond à l'id. Il correspond à la clef primaire et est incrémenté automatiquement. Il garantit l'unicité des éléments de ces tables.

- **Acteurs :**

Il existe quatre acteurs dans notre application : l'Utilisateur, le Client, l'Admin de gestion de musiques et l'Admin de gestion de clients.

Bien que nous n'ayons pas chiffré les mots de passe, il nous a paru pertinent de séparer la table Administrateur des autres acteurs pour avoir la possibilité de sécuriser plus facilement ou isoler autre part cette partie de la BDD.

Les tables Client et Administrateur correspondent aux informations des acteurs enregistrés dans la base de données. Lorsqu'un utilisateur non connecté navigue dans l'application, on ne conserve pas d'informations sur lui dans la base de données.

Deux attributs booléens sont présents dans la table Administrateur. Ils permettent de différencier le gestionnaire de musique du gestionnaire de clients. Il était possible de les différencier avec un unique booléen, mais nous avons préféré ajouter un attribut, dans le cas d'ajout de rôles futurs chez les administrateurs.

Nous considérons que l'administrateur et le client n'ont pas de relation d'héritage. Nous partons du principe que l'Administrateur s'occupe principalement de maintenir et gérer les données de l'application, il ne nous a donc pas paru pertinent de lui proposer de renseigner des informations non nécessaires, ou de lui donner accès à la gestion de playlist.

En revanche, la table client contient tous les attributs correspondant au client dans l'énoncé des besoins, dont l'attribut "*nbEcoute*" qui permet de faire des mesures statistiques sur le nombre de musiques écoutées par un Client sur l'application. Un Client peut créer plusieurs Playlists, mais chaque Playlist appartient à un unique Client, il existe donc une dépendance fonctionnelle 1:N entre la table Client et la table Playlist. Chaque client a un unique style de musique préféré dans notre conception, donc la relation entre la table Client et la table Genre est de type N:1.

- **Autres objets :**

Il existe quatre types différents d'objets dans les éléments du Catalogue, représentés par les tables Podcast, Radio, TitreMusical et Album. Tous sont des spécifications d'un élément du catalogue et en héritent. C'est pourquoi la clef primaire de chacune de ces tables est une clef étrangère pointant sur la clef primaire de la table ElementCatalogue. Ces éléments du catalogue partagent les attributs "*nbEcoute*", "*nbEcouteMois*" et "*recommande*".

La table Radio et la table TitreMusical possèdent toutes les deux un attribut genre. Cette attribut est un élément appartenant à une énumération, il ne s'agit pas d'un champ d'écriture libre, donc nous avons ajouté une clef étrangère pointant sur le l'attribut "*nomGenre*" qui correspond à un nombre limité de Genres.

Chaque TitreMusical peut appartenir à un unique Album, mais un Album peut contenir plusieurs titres, la relation entre ces deux tables est donc de type 1:N, et il existe une clef étrangère dans la table TitreMusical correspondant à la clef primaire de la table Album.

Lors de l'ajout d'un titre à un Album, la durée présente dans l'Album est mise à jour également. Il était possible de calculer cette durée à partir de la liste des Titres de l'Album, comme nous le faisons pour retrouver les interprètes d'un Album (Il s'agit des interprètes ayant réalisé les titres), mais nous avons préféré rendre cette donnée accessible dans la table Album de manière immédiate.

La relation entre une Playlist et un élément du Catalogue est de type N:N car une Playlist peut contenir plusieurs éléments du Catalogue et un élément du Catalogue peut appartenir à plusieurs Playlists. Il est donc nécessaire de créer une nouvelle table correspondant à l'association entre une Playlist et un ElementCatalogue. Nous la nommons ici ElementsPlaylist. Cette table a pour clef primaire le couple de clefs étrangères faisant référence aux deux tables précédentes.

De la même manière que pour la relation entre une Playlist et le Catalogue, un Interprète peut réaliser plusieurs Titres, et chaque Titre peut être composé par plusieurs interprètes dans notre modèle. Nous avons donc réalisé la table Discographie pour représenter les associations entre ces deux tables.

Les tables entre lesquelles il y existe une contrainte d'intégrité forte, donc les contraintes de clef étrangère sur une clef primaire sont supprimées en cascade, afin de garantir l'intégrité des données.

Par exemple, il ne paraît pas pertinent de conserver la Discographie d'un interprète supprimé, les Playlists d'un Client supprimé, ou le TitreMusical dont l'id correspond à un élément du Catalogue supprimé.

- Relation entre les tables et le code Java :

Nous avons mis à jour nos classes Java pour qu'elles correspondent au Schéma Relationnel. Les attributs présents dans les classes en Java sont tous présents dans les tables associées.

B) Vues, Fonctions stockées et procédures stockées

Pour accéder aux données présentes dans les tables et les modifier, nous avons fait le choix d'utiliser des procédures stockées. Cela permet de réaliser des transactions en s'assurant de l'intégrité des données en cas de problèmes, mais aussi de ne pas faire apparaître la structure de notre base de données dans le code en Java, car toutes les requêtes sont faites par des procédures.

Voici une liste des différents outils utilisés pour interagir avec la base de donnée

1. Vues

vue_morceaux_populaires : Affiche les 10 Titres les plus écoutés au cours du mois
vue_recommandations : Affiche les Titres recommandés par un administrateur.
(Note : cette fonctionnalité n'est pas encore implémentée côté Java)
vue_top_utilisateurs_ecoutes : Affiche les 10 Clients ayant écouté le plus de morceaux.

2. Fonctions stockées

nouveau_"objet" : Création de l'Objet correspondant aux attributs donnés en paramètre, renvoie l'id de l'Objet créé en cas de succès, et rien en cas d'échec.

Fonctions disponibles :
nouveau_admin
nouveau_album
nouveau_client
nouveau_client_complet
nouveau_interprete
nouveau_interprete_complet
nouveau_titre
nouvelle_playlist

3. Procédures stockées

supprimer_"objet" : Suppression de l'Objet correspondant à l'id donné en paramètre s'il existe.

authentification_admin : Sélectionne la ligne correspondant à l'Administrateur donné en paramètre si elle existe.
authentification_client : Sélectionne la ligne correspondant au Client donné en paramètre s'il existe.

existe_"acteur" : Permet de vérifier que l'acteur n'existe pas déjà dans la table avant de le créer et d'empêcher les doublons de comptes (acteurs possibles : Client, adminClient, adminMusique).
get"Objet" : Sélectionne les informations correspondant à l'id donné en paramètre (Objets possibles : Client, TitreMusical, Album, Interprete, Playlist,).

association_"objet1"_"objet2" : Établit la relation entre deux tables.
dissociation_"objet1"_"objet2" : Supprime la relation entre deux tables.

majDureeAlbum : Appelé lors de l'ajout ou la suppression d'un titre dans un Album, pour mettre à jour la durée de l'album.

modifier_"objet" : Update sur les champs de l'objet dont l'id est passé en paramètre (Objets possibles : client, admin, titre, album, interprète, playlist).

rechercherPar"Attribut""Objet" : Sélectionne tous les résultats dont l'Attribut de l'Objet correspond au moins partiellement au mot clef recherché.
Exemple : *rechercherParPseudoInterprete("Edith")* sélectionne au moins la ligne correspondant à Edith Piaf.

recommander : Met à true l'attribut "recommande" de l'ElementCatalogue (Non implémenté en Java)
recommander_annuler : Met à false l'attribut "recommande" de l'ElementCatalogue (Non implémenté en Java)

regarder(idClient, idCatalogue) : Incrémente de 1 le nombre de vues du Client et de l'ElementCatalogue donnés en paramètre.

reset_Vues_Catalogues : Réinitialise tous les attributs nbEcouteMois des éléments du catalogue.

4. Événement

Défini dans le script du modèle, *event1 est un* événement prévu chaque mois. Il réinitialise le nombre de vues au cours du mois des éléments du catalogue en appelant la procédure *reset_Vues_Catalogue().* Il est nécessaire d'avoir activé l'option
SET GLOBAL event_scheduler = 1; au préalable pour que cela fonctionne.

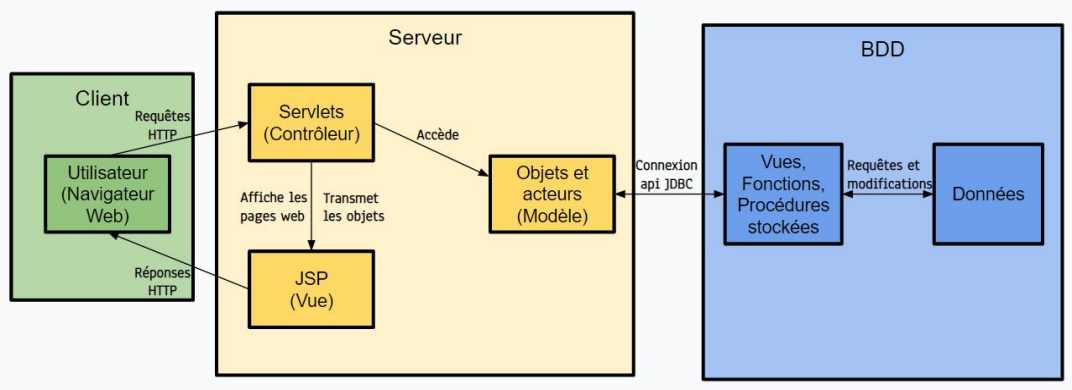
Ce modèle de données a été exploité pour réaliser les implémentations techniques répondant aux besoins formulés par l'utilisateur.

VI - Implémentations techniques

Dans cette partie, nous décrirons tout d'abord l'architecture globale de notre projet, puis nous expliciterons les fonctionnalités implémentées et l'état d'avancement du rendu, avant de présenter plus en détail certaines parties de l'architecture.

A) Architecture globale

Nous avons adopté un modèle MVC dans notre architecture globale. Pour communiquer avec notre système, l'utilisateur interagit avec des pages web en faisant des requêtes à des Servlets.



1. Communication Client/Serveur

Nous avons défini un Servlet pour traiter les requêtes de chaque acteur. Les Servlets prennent en compte les requêtes de l'utilisateur, puis affichent différentes pages jsp en fonction des requêtes de l'utilisateur.

Lorsqu'il accède au site, l'Utilisateur interagit avec le servlet UtilisateurServlet, car le rôle par défaut est celui d'utilisateur, mais lorsqu'il s'authentifie, celui-ci a la possibilité de devenir :

- Un Client : interagit alors avec ClientServlet
- Un Administrateur : interagit alors avec AdministrateurServlet. En fonction du type d'Administrateur (ProfilGestionnaireClient ou ProfilGestionnaireMusical), l'administrateur n'a pas accès aux mêmes fonctionnalités, même si certaines fonctionnalités sont héritées d'Administrateur, donc disponible pour les deux Profils.

2. Communication interne Serveur

Lorsqu'un accès à des données est nécessaire, le Servlet accède à des Objets définis dans le Modèle. Deux types d'objet sont définis :

- Acteurs : Les objets Administrateur, dont héritent les objets ProfilGestionnaireClient et ProfilGestionnaireMusical, Client et Utilisateur implémentent chacun une interface qui définit les méthodes de classe auxquelles chacun de ces objets peut accéder.
  Chaque interface définie a pour nom *"Objet"Interface*.

- Autres objets : Les objets ElementCatalogue, Album, TitreMusical, Interprete, Genre, Playlist, (Radio et Podcast non implémentés) sont des objets manipulés par les acteurs. Ils sont transmis aux Servlets qui affiche les informations demandées par l'utilisateur. Le Servlet peut également transmettre ces objets aux différentes méthodes de classe des Acteurs pour effectuer des modifications sur les données..

3. Communication Serveur/Données

La classe DBManager et l'api JDBC assurent la liaison entre les acteurs et la base de données. Lorsque la connexion est établie, des procédures stockées sont appelées, pour effectuer les opérations demandées.

Les méthodes accessibles par les différents acteurs sont détaillées plus en détail dans les fonctionnalités.

B) Fonctionnalités

Nous avons traduit les cas d'utilisation identifiés dans l'analyse des besoins par les fonctionnalités suivantes. Chaque méthode de classe correspondant à une fonctionnalité est déclarée dans l'interface de l'acteur associé.

1. Utilisateur

L'utilisateur peut :

- **Regarder un élément du catalogue**
  Méthode associée : *regarder*
  Description : Ajoute 1 au nombres de vues de l'Élément regardé
- **Créer un compte**
  Méthode associée : *creerCompte*
  Description : Création d'un nouveau Client dans la BDD, Renvoie l'objet Client si succès, null sinon
- **S'authentifier (en tant que Client)**
  Méthode associée : *authentification*
  Description : vérifie l'existence du couple mail, password dans la table Client
- **S'authentifier (en tant qu'Administrateur)**
  Méthode associée : *authentificationAdmin*
  Description : vérifie l'existence du couple mail, password ayant des droits Administrateur
- **Accéder aux titres les plus écoutés (Page d'accueil)**
  Méthode associée : *topTitresEcoutes*
  Description : renvoie la List<TitreMusical> correspondant aux 10 titres les plus écoutés

Une fois connecté, l'utilisateur a accès à de nouvelles fonctionnalités.

2. Client

Le client peut :

- **Regarder un élément du catalogue**
  Méthode associée : *regarder*
  Description : Ajoute 1 au nombres de vues de l'Élément regardé
- **Modifier ses informations**
  Méthode associée : *modifierInformations*
  Description : Met à jour les informations du client dans la BD puis dans l'objet
- **Créer une Playlist**
  Méthode associée : *creerPlaylist*
  Description : Création d'une nouvelle Playlist dans la BD
- **Supprimer une Playlist**
  Méthode associée : *supprimerPlaylist*
  Description : Retire la Playlist de la BD
- **Accéder à ses Playlists**
  Méthode associée : *getPlaylists*
  Description : Retourne la List<Playlist> du Client
- **Effectuer des recherches dans le Catalogue**
  Méthodes associées :
  rechercherParPseudoInterprete,
  rechercherParNomTitre
  rechercherParNomAlbum
  Description : renvoie la List correspondant a la recherche en paramètre
- **Accéder aux titres les plus écoutés (Page d'accueil Client)**
  Méthode associée : *topTitresEcoutes*
  Description : renvoie la List<TitreMusical> correspondant aux 10 titres les plus écoutés

3. Administrateur

Toutes les fonctionnalités présentes dans Administrateur sont accessibles par les deux types d'administrateurs.
Il peut :

- **Regarder un élément du catalogue**
  Méthode associée : *regarder*
  Description : Ajoute 1 au nombres de vues de l'Élément regardé
- **Modifier ses informations**
  Méthode associée : *modifierInformations*
  Description : Met à jour les informations du client dans la BD puis dans l'objet
- **Effectuer des recherches dans le Catalogue**
  Méthodes associées :
  rechercherParPseudoInterprete,
  rechercherParNomTitre
  rechercherParNomAlbum
  rechercherParMailClient
  Description : renvoie la List correspondant a la recherche en paramètre
- **Accéder aux titres les plus écoutés (En détail dans l'onglet Statistiques)**
  Méthode associée : *topTitresEcoutes*
  Description : renvoie la List<TitreMusical> correspondant aux 10 titres les plus écoutés
- **Accéder aux utilisateurs ayant le plus d'écoutes (Dans l'onglet Statistiques)**
  Méthode associée : *topUtilisateursEcoutes*
  Description : renvoie la List<Client> correspondant aux 10 clients ayant écouté le plus d'éléments du catalogue
- **Accéder aux titres les plus écoutés (Page d'accueil Client)**
  Méthode associée : *topTitresEcoutes*
  Description : renvoie la List<TitreMusical> correspondant aux 10 titres les plus écoutés

4. GestionnaireMusical

L'administrateur qui gère les éléments du catalogue peut également :

- **Créer un Interprète**
  Méthode associée : *creerInterprete*
  Description : Ajoute l'interprète dans la BD
- **Modifier un Interprète**
  Méthode associée : *modifierInterprete*
  Description : Modifie l'interprète donné dans la BD
- **Supprimer un Interprète**
  Méthode associée : *supprimerInterprete*
  Description : Retire l'interprète de la BD

- **Créer un TitreMusical**
  Méthode associée : *creerTitre*
  Description : Ajoute le titre dans la BD
- **Modifier un TitreMusical**
  Méthode associée : *modifierTitre*
  Description : Modifie le titre donné dans la BD
- **Supprimer un TitreMusical**
  Méthode associée : *supprimerTitre*
  Description : Retire le titre de la BD
- **Ajouter un titre au répertoire d'un interprète**
  Méthode associée : *ajouterDiscographie*
  Description : Ajoute le titre en paramètre à la discographie de l'interprète en paramètre
- **Supprimer un titre du répertoire d'un interprète**
  Méthode associée : *retirerDiscographie*
  Description : Retire le titre en paramètre de la discographie de l'interprète en paramètre

- **Créer un Album**
  Méthode associée : *creerAlbum*
  Description : Ajoute l'album dans la BD
- **Modifier un Album**
  Méthode associée : *modifierAlbum*
  Description : Modifie l'album donné dans la BD
- **Supprimer un Album**
  Méthode associée : *supprimerAlbum*
  Description : Retire l'album de la BD

- **Ajouter un TitreMusical à un Album**
  Méthode associée : *ajoutTitreAlbum*
  Description : Ajoute le titre en paramètre à l'ensemble des titres de l'Album
- **Supprimer un TitreMusical d'un Album**
  Méthode associée : *suppressionTitreAlbum*
  Description :Ajoute le titre en paramètre de l'ensemble des titres de l'Album

5. GestionnaireClient

L'administrateur qui gère les comptes clients peut également :

- **Supprimer un profil Client**
  Méthode associée : *supprimerClient*
  Description : Retire le Client de la BD
- **Modifier un profil Client**
  Méthode associée : *modifierInformationsClient*
  Description : Met à jour les informations du client dans la BD

6. Fonctionnalités non implémentées

- **Accéder aux Recommandations**
  Description : La méthode *recommandationsDuMoment* renvoie bien les recommandations, mais nous n'avons pas ajouté cette fonctionnalité sur le web car nous n'avons pas ajouté la possibilité aux administrateurs GestionnaireMusical d'ajouter ou retirer un titre des recommandations dans l'interface.

- **Ajouter un titre aux recommandations**
  Description : Les méthodes *recommander* et *recommander_annuler* sont implémentées, mais nous ne les utilisons pas dans notre affichage web.
- **Client : Modifier une Playlist**
  Description : Méthodes d'objet implémentées non appelées par le Servlet :
  changerNomPlaylist, ajouterTitrePlaylist, retirerTitrePlaylist
- **Administrateur Musical :**
  - **Créer/Modifier/Supprimer Radio**
  - **Créer/Modifier/Supprimer Podcast**

  Ces fonctionnalités n'ont pas été implémentées par manque de temps.

Maintenant que nous connaissons l'architecture globale du projet et les fonctionnalités implémentées, regardons plus en détail quelques aspects techniques de notre implémentation.

C) Architecture détaillée

Dans cette partie, nous détaillons l'implémentation des SERVLET et des JSP ainsi que leurs communications.

Les vues et servlets sont réparties en trois catégories, une par type d'internautes, l'utilisateur (internaute non connecté), le client (internaute connecté) et l'administrateur.
Ainsi, on retrouve pour les servlets un servlet par type d'internaute et pour les vues, un dossier pour chaque type d'internaute.

1. Détail des Servlets

Dans un servlet, on retrouve une méthode *doPost()* et une méthode *doGet().*
On fait appel à doPost() lorsqu'on fait une requête depuis un formulaire et doGet() sinon.

Chaque méthodes effectuent au minimum ces opérations ;

Retrouver ce que le client demande
doGet()
Récuperation de l'URL
String servletPath = request.getServletPath();
doPost()
Récuperation d'un input caché
String action = request.getParameter("action");/

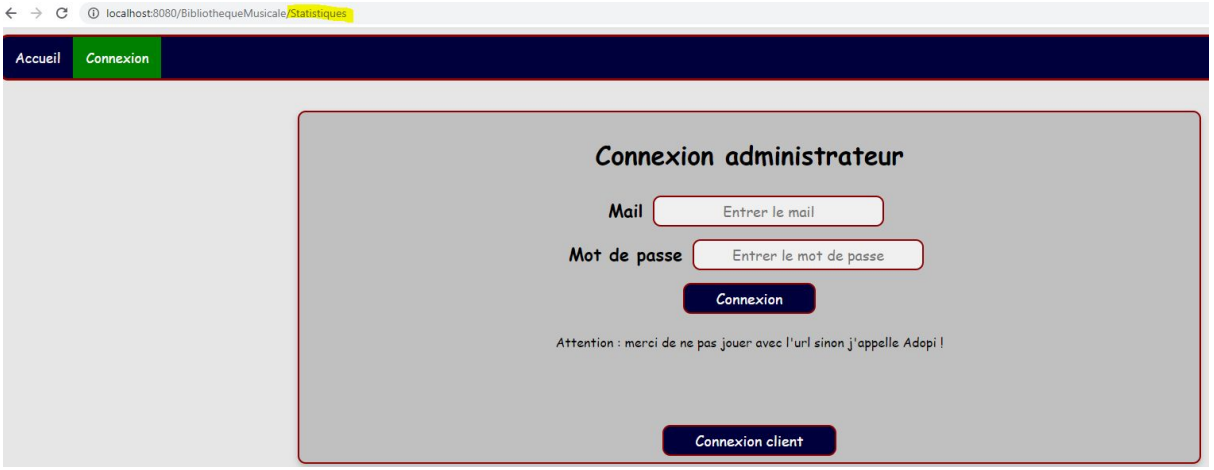
Traitement de la requête en fonction de la provenance
doGet()
if (servletPath.equals("/[...]"))
doPost()
if (action.equals("[...]"))

Affection vue
vue = "/JSP/[...]/[...].jsp";

Affichage de la vue
RequestDispatcher rd = getServletContext().getRequestDispatcher(vue);
rd.forward(request, response);

On utilise les attributs pour transférer des objets entre le servlet et le jsp et des paramètres si un String suffit.

Gestion des droits



L'AdministrateurServlet possède une variable utilisateur initialisé et un administrateur déclaré à sa création.
On fait une requête à la base de donnée pour vérifier le couple mail mot de passe.

administrateur = utilisateur.authentificationAdmin(mail, motDePasse);

Ainsi, le serveur sauvegarde l'objet administrateur tant que l'on travaille sur le servlet.
On vérifiera par la suite que le servlet a son objet administrateur correctement initialisé avant de traiter les requêtes.

Le ClientSevlet fonctionne de la même manière.

2. Détail des JSP

Héritage
Chaque fichier JSP importe les ressources CSS et JS de cette manière :

```
<head>
    <%@include file="../head.jsp" %>
</head>
```

Puis, il inclut la barre de navigation et le footer ainsi :

```
<body>
    <%@include file="../header.jsp" %>

    <%@include file="../footer.jsp" %>
</body>
```

Remarque
Les formulaires sont liés à leurs servlet par un input caché :

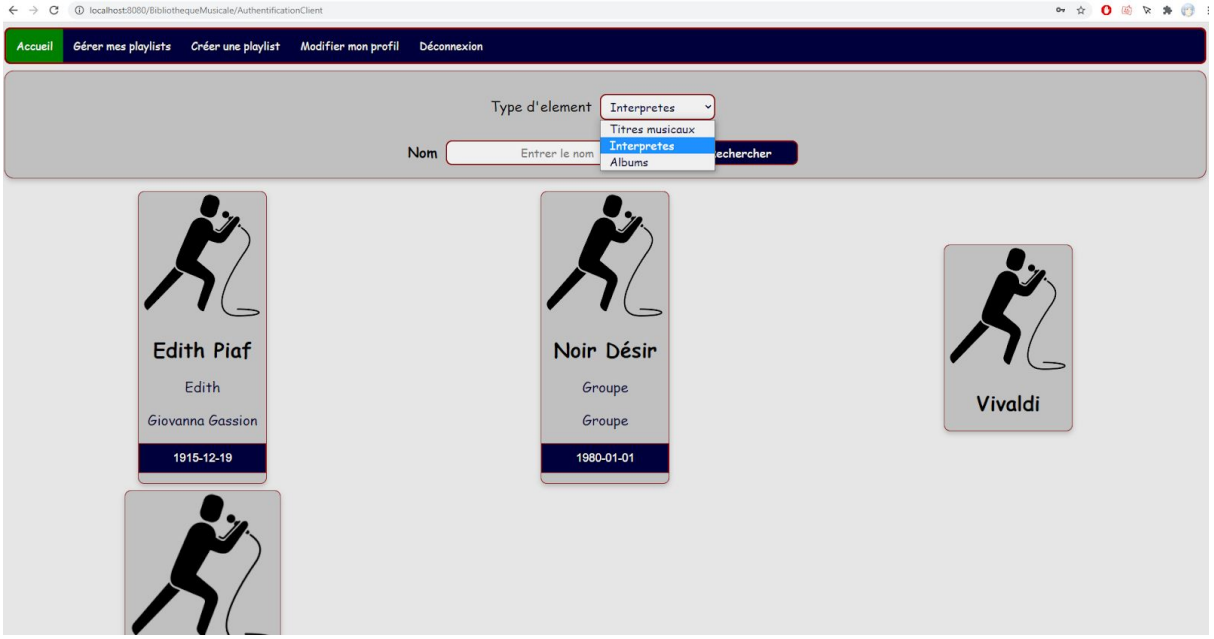
```
<input name="action" type="hidden" value="[...]">
```

La barre de navigation est mise à jour selon les droits du client grâces aux attributs renseignés par le serlvet (isClient, isAdministrateur). De plus, elle est actualisé avec la variable navBarActive.

```
<% boolean isClient = (boolean)request.getAttribute("isClient"); %>
<% boolean isAdministrateur = (boolean)request.getAttribute("isAdministrateur"); %>
<% String navBarActive = (String)request.getAttribute("nav-bar-active"); %>

<!-- navbar -->
<div class="topnav">
    <!-- client -->
    <% if(isClient==true){ %>
        <div id="myDIV">
            <a class="aref
            <% if(navBarActive!=null){
                if(navBarActive.equals("AccueilClient")){
                    out.print("active");
                }
            }else{
                out.print("active");
            }%>" href=AccueilClient>
                Accueil
            </a>
```

Extrait de header.jsp

L’affichage dynamique de certaines pages est permis grâce à du JS. Par exemple, lorsqu’on choisi une valeur dans la select box de l’accueil client, la page affiche les types d’éléments du catalogues demandés (titre, album ou interprète). Ceci est réalisé grâce à la fonction JS *changerTypeElementsAffichesRecherche().* Cette dernière affiche la balise souhaitée et cache les autres.




```
<!-- select box -->
<p>
<% String TypeElement = (String)request.getAttribute("TypeElement"); %>
    <label for="cars">Type d'element</label>
  <select id="TypeElement" onChange="changerTypeElementsAffichesRecherche()" name="TypeElement">
    <option value="Titres musicaux"
```

Extrait de accueilClient.jsp

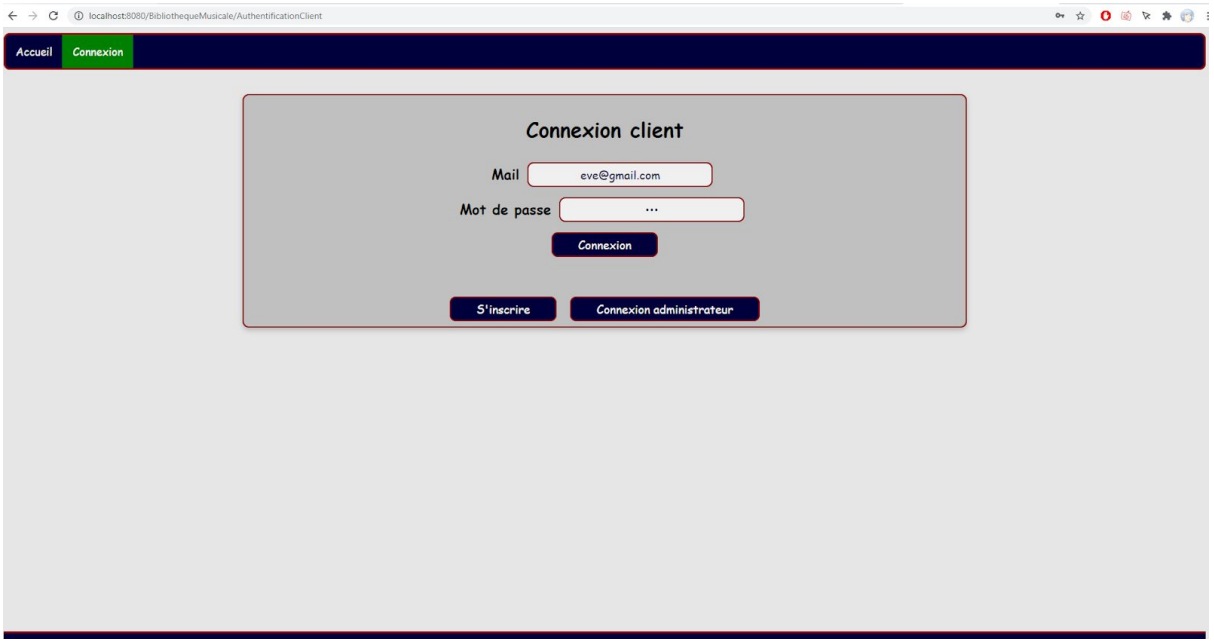
```
function changerTypeElementsAffichesRecherche(){
    TypeElement = document.getElementById("TypeElement").value;

    switch (TypeElement) {
      case 'Titres musicaux':
        document.getElementById("catalogueInterpretes").style.display = "none";
        document.getElementById("catalogueAlbums").style.display = "none";
        document.getElementById("catalogueTitresMusicaux").style.display = "grid";
        break;
```

Extrait de script.js

3. Exemple concret - connexion d'un client.

Ici, on suit l'avancement d'une requête HTTP à travers notre code.

Un utilisateur souhaite se connecter.
Il clique sur l'onglet connexion.
Il arrive ensuite sur une page demandant son mail et son mot de passe.

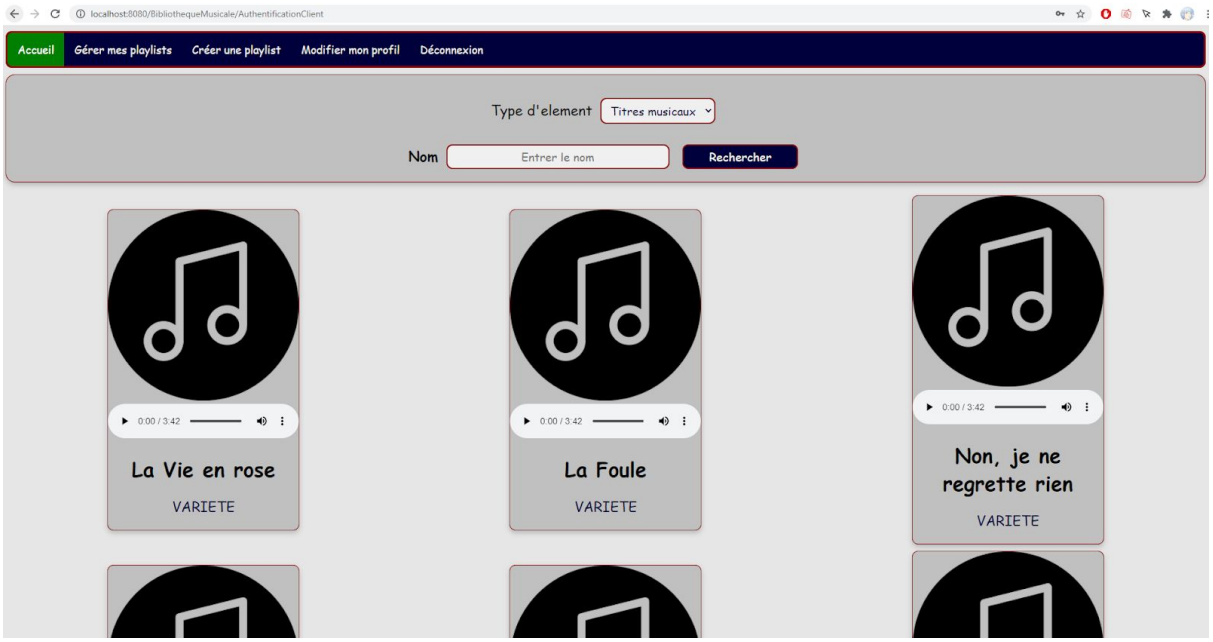



```
<!-- formulaire de connexion -->
<div class="connexion">
    <form method="POST">
        <input name="action" type="hidden" value="AuthentificationClient">

        <h1>Connexion client</h1>

        <label><b>Mail</b></label>
        <input type="text" placeholder="Entrer le mail" name="mail" required><br>

        <label><b>Mot de passe</b></label>
        <input type="password" placeholder="Entrer le mot de passe" name="password" required><br>

        <input type="submit" class="button" id='submit' value='Connexion' >
    </form>
```

Extrait de Utilisateur.Authentification.jsp

Il rentre ses informations et clique sur valider.

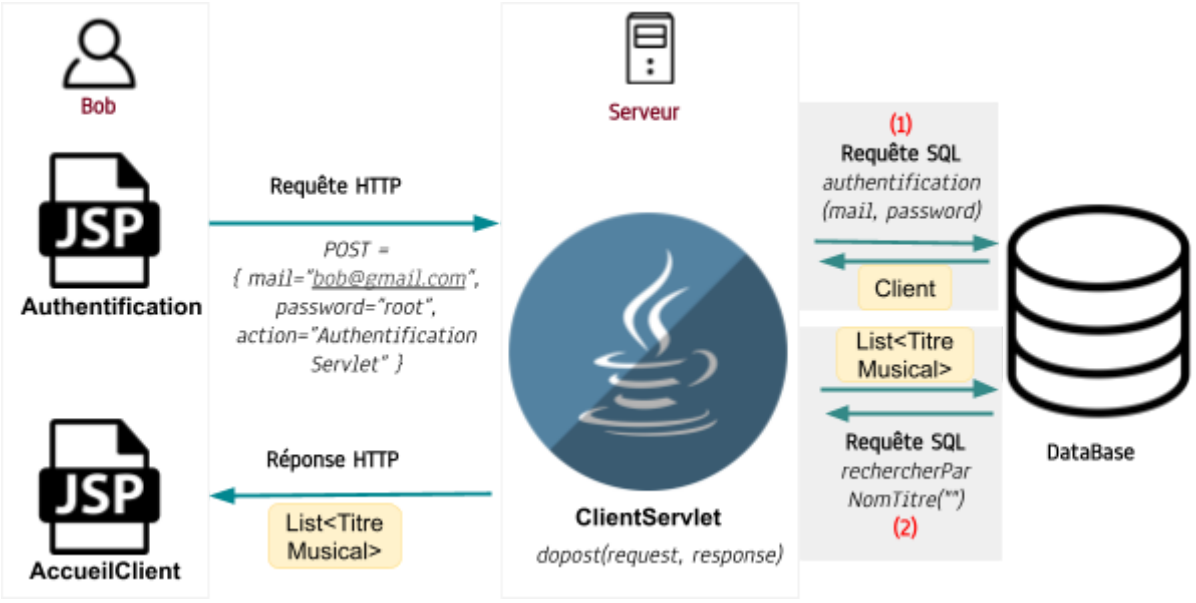
Le serveur envoie une requête à la base de donnée avec les identifiants puis les vérifie.
Il affecte différents attributs nécessaires à l'affichage de la page suivante.
Il affecte la prochaine vue, AccueilClient.jsp.




```
if(action.equals("AuthentificationClient")) {
    //requï¿½te ï¿½ la BDD
    client = utilisateur.authentification(mail, motDePasse);

    if(client!=null) {
        request.setAttribute("nav-bar-active", "AccueilClient");
        //affectation paramï¿½tres ï¿½ la vue
        request.setAttribute("isClient", true);

        // requeteBDD
        List<TitreMusical> titresMusicaux = client.rechercherParNomTitre("");
        List<Interprete> interpretes = client.rechercherParPseudoInterprete("");
        List<Album> albums = client.rechercherParNomAlbum("");

        // envoie de parametres a la vue
        request.setAttribute("titresMusicaux", titresMusicaux);
        request.setAttribute("interpretes", interpretes);
        request.setAttribute("albums", albums);
        request.setAttribute("isClient", true);

        //choix de la vue
        vue = "/JSP/Client/AccueilClient.jsp";
```

Extrait du ClientServlet>doPost()

On peut résumer ces événement à travers le schéma suivant :



Connexion client

Conclusion

L'architecture et l'implémentation proposées permettent de répondre aux besoins formulés dans la problématique.

Certains besoins sur lesquels nous n'avons pas mis notre priorité doivent encore être réalisés, mais dans l'ensemble la plupart d'entre eux trouvent leur solution dans cette réponse.

Ce projet a permis de confronter concrètement les connaissances issues du cours à une situation réellement plausible, et nous a permis de constater qu'il peut toujours y avoir des imprévus, et qu'il faut savoir s'adapter à tout type de situation.

Pour finir, la résolution du problème dans un environnement réel semble beaucoup plus ouverte, et il est beaucoup moins aisé de trouver une solution parfaite, contrairement au cadre scolaire.

Certains choix d'implémentation peuvent en effet avoir des avantages, mais aussi des inconvénients, comme par exemple l'implémentation de fonctions stockées, qui permettent de s'assurer la cohérence de toutes les requêtes, mais qui rendent le code côté données plus lourd.