Rapport Technique

Projet JEE : UsTube

Jean-Hanna SALEH, Juliette DECLERCK, Erwan BOISARD, Romain BARDINET

Sommaire:

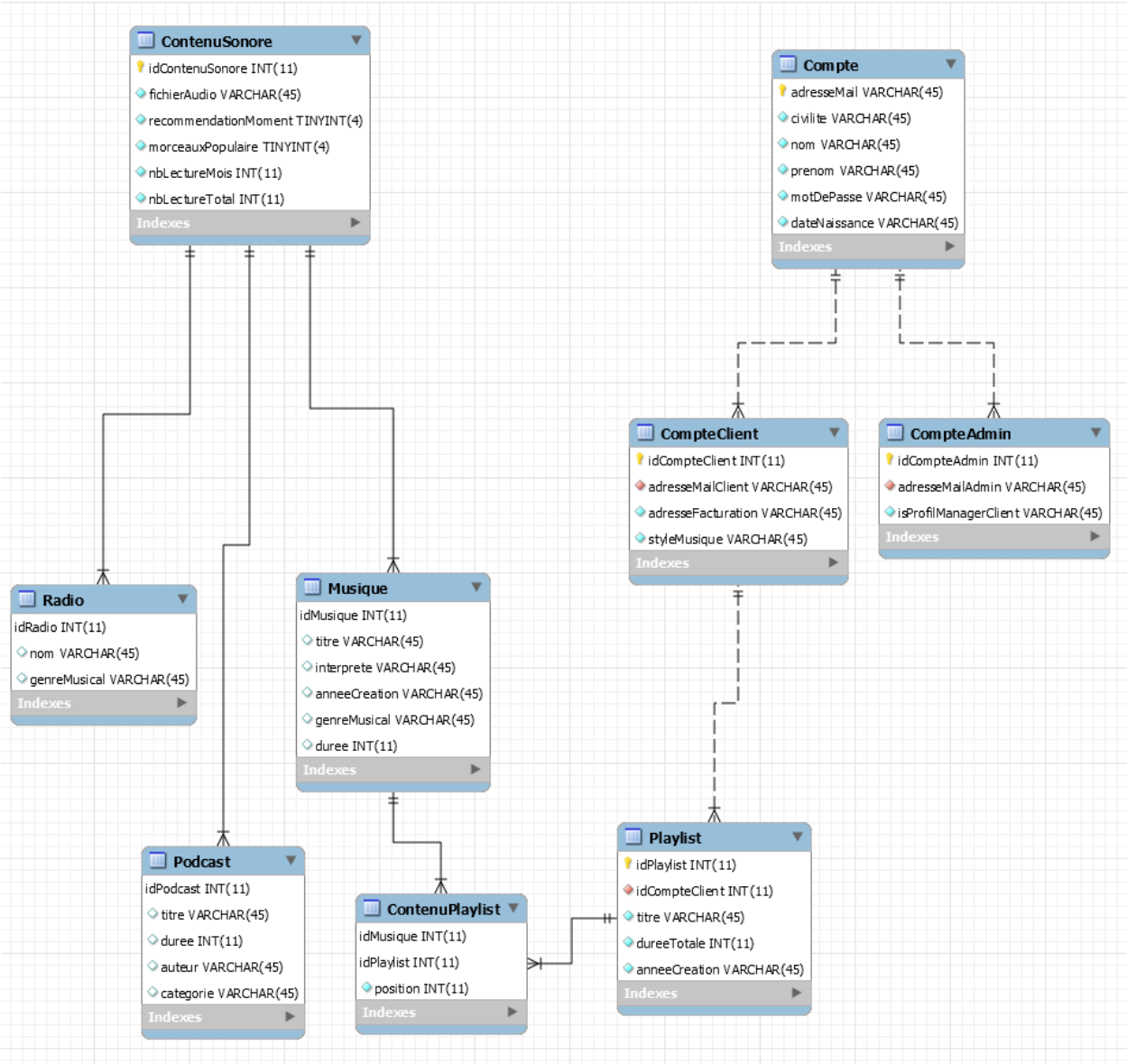
INTRODUCTION

L'objectif de ce projet est de créer une application web de streaming musical. Le site propose à tous les visiteurs du site d'écouter les morceaux les plus populaires du mois ainsi que les recommandations du moment. Pour accéder à d'avantages de fonctionnalités, un utilisateur peut se connecter à son compte ou en créer un, via la page de connexion ou d'inscription. L'utilisateur connecté peut alors accéder à l'ensemble du catalogue musical, et ajouter ses contenus sonores favoris dans ses playlists. Des utilisateurs définis comme administrateurs peuvent se connecter à leur compte administrateur pour accéder à la partie organisationnelle du site. Un administrateur de type profil gestionnaire client peut modifier les informations de n'importe quel client, tandis qu'un administrateur de type profil gestionnaire musical peut modifier les informations du catalogue musical, en créant, modifiant, ou supprimant des musiques, des radios ou des podcasts.

Ce projet est réalisé sous la forme d'une application JEE qui fonctionnera grâce à Tomcat et se connectera à une base de données MySQL. Il correspond à l'implémentation pratique d'un projet précédent d'UML consistant à l'élaboration de diagrammes sur ce problème et sur ses principaux cas d'utilisation.

MODÈLES DE DONNÉES

Le modèle de données utilisé pour ce projet se divise en deux parties principales : une partie contenant les contenus sonores et une partie contenant les Comptes (Clients/Admin). Le schéma de la base de données utilisée est le suivant :



Partie Contenu Sonore :

Tout élément de la bibliothèque est une instance de ContenuSonore, et a donc pour clé unique son idContenuSonore. Outre leur id unique, les attributs suivants sont communs à tous les éléments audios : un lien vers l'adresse du fichier audio (**fichierAudio**, type VARCHAR(45)), un booléen indiquant si l'élément fait partie des recommandations du moment (**recommendationMoment**, type TINYINT(4)), un booléen indiquant si l'élément fait partie des morceaux populaires (**morceauxPopulaire**, type TINYINT(4)), le nombre de lectures durant le mois actuel (**nbLectureMois**, type INT(11)), le nombre de lectures total (**nbLectureTotal**, type INT(11)).

Les éléments de type Radio disposent des attributs spécifiques suivants : un nom (**nom** type VARCHAR(45)), un genre musical (**genreMusical** type VARCHAR(45)).

Les éléments de type Podcast disposent des attributs spécifiques suivants : un titre (**titre** type VARCHAR(45)), une durée (**duree** type duree INT(11)), une catégorie (**categorie** type VARCHAR(45)), un auteur (**auteur** type VARCHAR(45)).

Les éléments de type Musique disposent des attributs spécifiques suivants : un titre (**titre** type VARCHAR(45)), un interprète (**interprete** type VARCHAR(45)), une année de création (**anneeCreation** type VARCHAR(45)), un genre musical (**genreMusical** type VARCHAR(45)), une durée (**duree** type duree INT(11)).

Un client peut créer plusieurs playlists. Chaque playlist a donc un ID unique en clé primaire (**idPlaylist** type INT(11)) et un idCompteClient en clé étrangère (**idCompteClient** type INT(11)). Elle contient aussi les attributs suivants : un titre (**titre** type VARCHAR(45)), une durée (**dureeTotale** type INT(11)) et une année de création (**anneeCreation** type VARCHAR(45)).

Pour stocker le contenu d'une playlist, on dispose de la table ContenuPlaylist. Ses trois attributs forment sa clé primaire : id Musique type INT(11) , idPlaylist type INT(11) et la position occupée par la musique d'id idMusique dans la playlist d'id idPlaylist (position type INT(11)). Pour accéder à la musique occupant la n ième position de la playlist idPlaylist = a, il faut donc entrer la requête suivante : SELECT **idMusique** FROM **ContenuPlaylist** WHERE **idPlaylist** = **a** AND **position** = **n**; La première position d'une playlist est l'élément en position 1 dans la base de donnée.

Partie Compte :

Tous les comptes sont des instances de la classe Compte dont la représentation dans la base de données est la table Compte, qui a pour clé primaire l'adresse email de l'utilisateur (**adresseMail** type VARCHAR(45)). Les autres attributs de cette table sont : la civilité de l'utilisateur (**civilite** type VARCHAR(45)), son nom (**nom** type VARCHAR(45)), son prénom (**prenom** type VARCHAR(45)), son mot de passe (**motDePasse** type VARCHAR(45)) et sa date de naissance (**dateNaissance** VARCHAR(45)).

Les comptes clients ont pour clé primaire un id de compte client (**idCompteClient** type INT(11)), une clé étrangère contenant l'adresse email liée au compte et permettant d'accéder aux données de la table Compte liée à ce compte client (**adresseMailClient** type VARCHAR(45)), une adresse de facturation (**adresseFacturation** type VARCHAR(45)) et un style de musique (**styleMusique** type VARCHAR(45)).

Les comptes administrateurs ont pour clé primaire un id de compte administrateur (**idCompteAdmin** type INT(11)), une clé étrangère contenant l'adresse email liée au compte et permettant d'accéder aux données de la table Compte liée à ce compte administrateur (**adresseMailAdmin** type VARCHAR(45)), et son statut de manager client (**isProfilManagerClient** type VARCHAR(45)).

EXPLICATION DES IMPLÉMENTATIONS TECHNIQUES DES DIFFÉRENTES FONCTIONNALITÉS

Les fonctionnalités décrites dans cette partie correspondent aux fonctionnalités de l'application JEE réalisée dans ce projet. Celle-ci utilise le modèle décrit à la section précédente dans une base de données MySQL.

Ces fonctionnalités peuvent être utilisées par un tiers en envoyant des requêtes web et en recevant leur réponse via un navigateur par exemple. Les requêtes sont adressées aux différentes servlets de l'application, qui utilisent des classes Java standard pour réaliser des tâches ou communiquer avec la base de données. Elles fournissent une réponse grâce à la redirection vers des pages jsp auxquelles elles donnent les informations nécessaires, et qui fournissent à l'utilisateur une interface pour utiliser l'application. Ces mécanismes sont ici détaillés pour chaque fonctionnalité principale.

- ### Création d'un compte utilisateur

La création d'un compte utilisateur est possible sur la page inscription.jsp. Pour y accéder, on clique sur le bouton **S'inscrire** de la page accueil.jsp appelée par la servlet **Accueil**, grâce à la procédure *doProcess()*, lui-même appelé par *doGet()*. Le bouton **S'inscrire** a été généré au préalable par la fonction javascript *loggedOut()* du fichier **titleBarCreation.js** parce que l'objet **client** représentant la connexion, n'a pas été mis en attribut de session. Une fois le bouton **S'inscrire** appuyé, nous sommes redirigés via un lien sur l'url **./Accueil/Inscription**. On passe ensuite par la servlet **Inscription** qui va charger la page **inscription.jsp**. Une fois sur cette page, l'utilisateur va remplir les différents champs proposés. Ces champs, présents dans un formulaire, possèdent des attributs **name.** Une fois le bouton de validation cliqué, le formulaire va envoyer ces informations à la servlet via la méthode **Post**. Par ailleurs, pour garantir le respect du format des différents champs, par exemple le fait que le mot de passe doit posséder au moins un chiffre, une lettre minuscule, une lettre majuscule et être au moins de taille 8, un écouteur d'événement **input** est ajouté à chacun d'eux. Ainsi à chaque fois qu'une touche est pressée, le script javascript va analyser l'information du champ en récupérant son identifiant. Si l'information d'au moins un champ parmi : **Adresse mail**, **Mot de passe**, **Confirmation du mot de passe** et **Date de naissance** est incorrect, alors le bouton de validation du formulaire est bloqué et un message d'erreur particulier apparaît. Dans le cas où le champ **Adresse mail** contient une adresse déjà utilisée par un autre client, en supposant que tous les champs sont correctement remplis, l'utilisateur va cliquer sur le bouton **Valider**. L'information du formulaire sera passée à la servlet **Inscription**. Pour vérifier qu'il y a bien quelque chose à traiter, un premier test est

effectué sur la valeur du paramètre **nom** correspondant au champ **Nom** du formulaire. S'il est non nul, un objet **CompteClient** est récupéré, puis est passé dans la fonction *addToDatabase()*. Cette fonction va créer un objet **clientDatabase** de la classe **ClientDatabase** qui va se charger d'effectuer une connexion avec la base de données, puis l'objet va utiliser sa méthode *addClientAccount()* pour ajouter un client.

En cas d'échec, la fonction renvoie faux. La servlet va alors mettre en attribut de requête le booléen **successSignUp** avec la valeur faux ainsi que l'objet **compteInscription** de la classe **CompteClient**. Une fois sur la page **inscription.jsp**, si l'objet **compteInscription** est non nul, un élément **div** contenant un message d'erreur est affiché. Pour éviter de perdre les informations déjà saisies par l'utilisateur, l'objet **compteInscription,** qui les contient, va être passé dans l'objet de gestion **compte** de la classe **MetaCompteClient**. Le principe est le suivant : si l'objet **compteInscription** est nul, alors en appliquant sur l'objet de gestion **compte** les accesseurs des champs, il renverra une chaîne de caractère vide. Sinon, il renvoie les champs de **compteInscription**. L'objet **compte** est alors mis dans le formulaire, au niveau du paramètre **value** avec l'accesseur correspondant au bon champ à remplir.

En cas de succès, la servlet ne met en attribut de requête que le **successSignUp**, puis nous renvoie sur la page **inscription.jsp**. Le code java présent va vérifier que le booléen est à vrai, puis afficher le message de succès contenu dans un div.

- ### Connexion à un compte utilisateur ou administrateur

La connexion à un compte utilisateur ou administrateur est possible sur la page **accueil.jsp**. Tout d'abord, on suppose que durant le premier accès à cette page, aucun objet de session n'a été créé. La page va donc appeler la fonction javascript, *loggedOut()* du fichier **titleBarCreation.js**. Cette fonction va générer le titre de l'accueil, ainsi que les boutons **Se connecter** et **S'inscrire**. En cliquant sur le bouton **Se connecter**, un écouteur d'événement sur l'objet **SignIn,** du fichier **modal.js**, va afficher un conteneur div caché appelé **connexion**. Sur ce modal, l'utilisateur va remplir son adresse mail, ainsi que son mot de passe. Le remplissage des champs du formulaire, grâce à l'option **required** des balises **input,** est obligatoire. L'utilisateur va cliquer sur le bouton **Valider** et cela va transmettre les informations du formulaire, via la méthode **POST,** vers la servlet **Accueil**. La servlet va récupérer les paramètres renseignés dans les chaînes de caractères **mail** et **password**. On vérifie que **mail** n'est pas nul, ce qui signifie bien que l'information a été envoyée depuis la page **accueil.jsp**. Une connexion avec la base de données s'ouvre, puis deux méthodes sont utilisées pour vérifier si le mail existe bien dans la base de données.

D'abord, *getCompteClient()* de la classe **CompteClient**, et *getCompteAdmin()* de la classe **CompteAdmin**. Ces méthodes renvoient respectivement une instance de **CompteClient** et de **CompteAdmin.** Par unicité du de l'adresse mail dans la table **Compte** de la base de donnée, au moins l'une des deux méthodes précédentes renvoie un compte existant. Il y a alors vérification du mot de passe entré via la méthode *isPassWord()* présente pour les deux instances, qui va comparer le mot de passe entré avec celui obtenu depuis la base de données. Si le mot de passe est correct, une session peut être créée soit pour un utilisateur, et dans ce cas la servlet nous redirige vers la page **accueil.jsp**. Cette fois, l'objet de session **client** existe, et la page va donc appeler *loggedIn()* du fichier **titleBarCreation.js**. La page d'accueil possède alors un bouton de profil, ainsi qu'un bouton pour se déconnecter. Soit la session créée est pour un administrateur, et la servlet va nous rediriger vers l'url de la servlet appropriée : **Administration/AdminProfilClient** ou **Administration/AdminGestionnaireMusical**.

Il reste ces deux cas : soit le mail ne correspond à aucun email déjà existant dans la base de données, soit le mot de passe est erroné. Dans ces deux situations, le mot de passe et le mail sont mis en attribut de requêtes et nous sommes redirigés sur la page **accueil.jsp**.

En passant sur la page accueil.jsp, les variables java **mailAddressUsed** et **passwordUsed** récupèrent ces paramètres. Comme **mailAddressUsed** est non nul, le modal est réactivé et contient les informations déjà entrées avec un message indiquant que les crédentiels entrés sont erronées.

- ### Se déconnecter d'un compte utilisateur ou administrateur

Une fois connecté, que cela soit sur un compte administrateur, ou un compte client, un bouton **Se déconnecter** est présent en haut à droite de la page. En cliquant sur ce bouton, l'utilisateur va être redirigé vers la servlet **LogOut**. Il va alors récupérer la session active et la supprimer. Enfin, nous serons redirigés vers la page **accueil.jsp**.

- Modification des informations d'un compte utilisateur

Une fois connecté sur un compte utilisateur, il existe en haut à droite de la page un bouton de profil avec le prénom et le nom du client. Si l'utilisateur clique dessus, il se fait rediriger vers la servlet **Profil**. Une fois sur la servlet, on est redirigé vers la page **profil.jsp**. Sur cette page , on récupère l'objet session représentant la session via **request**.*getSession()*. Le formulaire s'affiche, avec les informations du compte grâce à du code java inséré au niveau du paramètre **value** donnant la valeur des différents champs du compte. En utilisant le même javascript, la vérification sur les champs est exactement identique que dans le cas de la **"Création d'un compte utilisateur"**.

On notera cependant que tous les champs sont modifiables, excepté l'adresse mail qui représente l'id du compte de l'utilisateur dans la base de données.

Une fois que les champs sont considérés comme modifiés, l'utilisateur clique sur le bouton **Enregistrer mes modifications**. Le formulaire est alors renvoyé vers la servlet **Profil**. Une fois sur celui-ci, on récupère l'objet de session représentant l'utilisateur, i.e. l'instance **currentCompteClient**. On récupère également les paramètres envoyés par le formulaire. *hasChangedInformation()* va comparer les informations de **currentCompteClient** et celles envoyées par le formulaire. Si au moins un champ a été modifié, l'ancien compte client est mis à jour dans la session. De plus, on ouvre une connexion avec la base de données et on met à jour les informations de ce compte utilisateur via la méthode *modifyClientAccount()*. Le booléen **successModification** est mis en attribut de requêtes, avec la valeur "vrai". On est ensuite redirigé vers la page **profil.jsp**. Sur le page jsp, le booléen **successModification** est récupéré, puis une comparaison du booléen est effectué en bas de page. En cas de succès, un élément div est créé contenant le message de succès des modifications des informations.

- Affichage des recommandations du moment et des morceaux populaires

Les recommandations du moment et les morceaux populaires sont affichés sur la page **accueil.jsp**. Cette page est appelée par la servlet **Accueil** *doProcess()*, qui redirige vers **accueil.jsp**, en lui transmettant une liste de musique en attribut de la requête. La partie java de la page jsp lit cette liste, affiche les cinq premières musiques dans recommandations du moment et les cinq dernières dans morceaux populaires.

En effet, cette liste de musique est composée de dix éléments au maximum, et on considère qu'il y a toujours au minimum cinq recommandations du moment. On note ici que seules les musiques peuvent être marquées recommandations du moment ou morceau populaire.

Cette liste est remplie dans **Accueil** en concaténant les résultats des méthodes *getRecommendationMoment()* et *getMorceauxPopulaires()* de **CataolgueDatabase**. Ces deux fonctions se servent des méthodes de **CatalogueDatabase** *getAllBy()* et *readResulset*(). *readResulset*() lit un resultset donné si on lui précise en paramètre le type de **ContenuSonore** qu'il doit lire (musique, radio ou podcast) et retourne une liste de ces contenus sonores remplis. *getAllBy*() effectue une requête sur le **ContenuSonore** précisé en paramètre (musique, radio ou podcast). Il sélectionne tous les champs de l'objet, et ajoute un filtre en fin de requête. Ce filtre est passé en paramètre de *getAllBy*(). Dans le cas des recommandations du moment et des morceaux populaires, il vérifiait que les booléens **recommendationMoment** ou **morceauxPopulaire** étaient à vrai et que la limite de la sélection soit de cinq lignes. *getAllBy*() retourne un resultset, ce resultset est passé à *readResultset()*, qui renvoie donc une liste de musique d'au plus cinq éléments à *getRecommendationMoment()* et *getMorceauxPopulaires()*.

À chaque appel d'**Accueil**, la fonction *updateMorceauxPop()* de **CatalogueDatabase** met le booléen **morceauxPopulaire** des 5 musiques les plus écoutées à vrai. Cette fonction est appelée avant *getMorceauxPopulaire()*. Après avoir mis tous les booléens **morceauxPopulaire** à faux dans la base de donnée, elle utilise *getAllBy*() et *readResulset*() pour trouver les 5 musiques les plus lues (grâce au filtre donné à *getAllBy()*). Elle les note ensuite comme morceaux populaires puis on utilise *updateContenuSonore*() pour enregistrer ces modifications dans la base de données.

- Affichage du catalogue musical

Un affichage plus complet du catalogue est fait dans la page **exploreCat.jsp**. Cette page comporte une unique section d'affichage de **ContenuSonore**, qui peut afficher des musiques, des radios ou des podcasts. Elle est remplie grâce à un script Javascript en fin de page, en utilisant un attribut donné par la servlet **ExploreCatalogue** à la page. Cet attribut est une liste de **ContenuSonore** au format Json.

Cette liste est remplie par **ExploreCatalogue** différemment si une recherche a été faite ou non.

Si aucune recherche n'a été faite, **ExploreCatalogue** appelle la fonction *searchByGenreMusical()* de **CatalogueDatabase**. Cette fonction prend une chaîne de caractère en paramètre, et retourne une liste d'objets musiques et radios contenant cette chaîne dans leur colonne **genreMusical**. Ici cette chaîne de caractère est le style préféré de l'utilisateur connecté, car cette page est restreinte aux utilisateurs connectés. Si cette fonction ne trouve pas de contenu sonore, **ExploreCatalogue** appellera alors *getTypeCatalogue*("musique") pour remplir la liste à donner à la page jsp. Cette fonction retourne une liste d'au plus 30 musiques du catalogue. Elle utilise *readResultset()* et *getAllBy()* avec musique en paramètre de ces fonctions et une limite de 30 lignes de résultat dans le filtre.

Si une recherche a été faite, on a une chaîne de caractères à chercher dans le catalogue, et on appellera donc les fonctions : *searchAllByTitle*(), *searchByAutor*(), *searchByCategorie*() et *searchByGenreMusical*(), qui prennent en paramètre la chaîne de caractère recherchée. On ajoutera tous les résultats trouvés par ces fonctions dans une même liste qui sera transmise à la page jsp.

Les fonctions *searchAllByTitle*(), *searchByAutor*(), *searchByCategorie*() et *searchByGenreMusical*() fonctionnent de manière similaire : chacune appelle *readResultset()* et *getAllBy()* pour les trois contenus sonores possibles, musique, radio et podcast. Elles ajoutent tous les résultats trouvés à une même liste. Elles prennent pour filtre de *getAllBy*() la présence de la chaîne de caractère recherchée dans la colonne associée à leur nom (titre, auteur, catégorie, genre musical). Par exemple, *searchAllByTitle*() recherchera dans les titres des musiques et podcasts et dans le nom de la radio.

searchAllByTitle(), *searchByAutor*(), *searchByCategorie*(), *searchByGenreMusical*()

et *getTypeCatalogue*() ont aussi chacuns une limite de 30 au nombre de lignes que retourne leur recherche dans la base de données, pour ne pas surcharger les pages de trop de contenu sonore.

- ### Jouer une musique

Une musique peut être jouée depuis **accueil.jsp** ou **exploreCat.jsp**. Quand on clique sur un contenu sonore, un modal s'affiche qui nous permet de cliquer sur le bouton écouter. Si ce bouton est cliqué, l'identifiant du contenu sonore sera renvoyé à la servlet **ExploreCatalogue** ou **Accueil** et la page jsp sera rechargée avec ce contenu sonore renseigné en attribut.

Dans la servlet **Accueil**, si l'identifiant du contenu sonore à jouer est renseigné, on est sûr qu'il s'agit d'une musique qu'il faut jouer. On utilise donc la fonction *getMusique*() de **PlaylistDatabase,** fonction qui retourne une instance de Musique remplie avec les informations de la base de données associées à l'identifiant. On augmente alors les compteurs **NbLectureTotal** et **NbLectureMois** de cette musique de un à l'aide de ses getters et setters. Puis on met à jour ces informations dans la base de donnée grâce à la fonction *infoStatMAJContenuSonore*() de **CatalogueDatabase**. On remet enfin cette musique en paramètre pour la page accueil.jsp, sous format Json.

Dans la servlet **ExploreCatalogue**, si l'identifiant du contenu sonore est renseigné, un identifiant de playlist peut aussi être présent. En effet, dans **exploreCat.jsp**, les playlists d'un utilisateur et leurs musiques sont aussi affichées sur la page. Cela est fait grâce à un attribut donné par **ExploreCatalogue** à **exploreCat.jsp**, où toutes les playlists d'un utilisateur sont récupérées avec *getAllPlaylist*() de **PlaylistDatabase** et converties en tableau Json.

Si l'identifiant d'une playlist est présent, cela signifie que l'utilisateur a cliqué sur une musique depuis l'affichage des playlists. On est sûr qu'il s'agit d'une musique car seules les musiques peuvent être mises dans les playlists. On récupère alors la playlist associée dans la base de données grâce à *getPlaylistById*() de **PlaylistDatabase**. On parcourt ensuite toutes les musiques récupérées de la playlist, et pour chacune, on incrémente **NbLectureMois** et **NbLectureTotal** de un. On considère donc que toutes les musiques de la playlist sont lues une fois. On renvoie ensuite l'identifiant de la playlist à la page jsp, en plus de la musique choisie qui elle est renvoyée remplie sous format Json.

NbLectureMois et **NbLectureTotal** ne sont ici pas redondantes car **NbLectureMois** est remis à zéros tous les mois grâce à un Event stocké dans la base de

donnée : *reset_vues_mois* qui peut être trouvé à la fin du script de création de la base de donnée fournie dans les livrables : **SiteStreaming.sql**.

Si l'identifiant de la playlist n'est pas renseigné, c'est qu'on a cliqué sur un contenu sonore qui ne fait pas partie d'une playlist. Dans **exploreCat.jsp** on n'est pas sûr qu'il s'agisse d'une musique, cela peut aussi être une radio ou un podcast. On essaie donc d'abord de récupérer la musique avec *getMusique*() et l'identifiant de contenu sonore qu'on a. Si l'objet retourné est nul, c'est qu'il ne s'agit pas d'une musique. On essaie donc la fonction *getAllBy*() de **CatalogueDatabase** successivement avec podcast et radio avec l'identifiant recherché pour filtre. On prend ensuite le contenu sonore non nul retourné par l'un ou l'autre des appels, on est sûr de n'en avoir qu'un seul car l'identifiant d'un contenu sonore est unique. On met alors comme précédemment à jour le nombre de lecture du mois et de lecture totale, puis on renvoie le contenu sonore à la page jsp sous format Json.

Dans **accueil.jsp** et **exploreCat.jsp**, on a alors l'attribut correspondant à la musique écoutée sous format en Json qui est renseigné. Si c'est le cas, les fonctions Javascript *EcouterMus*() pour **accueil.jsp** dans **custom-player.js** et *playMusique*() dans le script de **exploreCat.jsp** sont appelées. Ces fonctions affichent la barre de lecture d'un fichier audio dont les contrôles ont été personnalisés. Sur cette barre de contrôle sont alors remplis le titre du contenu sonore et sa durée totale (s'il ne s'agit pas d'une radio). Le fichier audio joué est un lien vers une musique qui est toujours la même, cependant la durée jouée dépendra de notre contenu. En effet, les boutons play, pause et stop ont été réécrits pour permettre d'écrire dans les gestionnaires d'événements qu'on leur attribue. Ainsi, à l'événement du média 'ended', on associe la fonction *stopMedia1*(). On renseigne une variable **dureeMus** qui nous sert à connaître la durée de la musique à jouer. On ne le remplit pas seulement s'il s'agit d'une radio qui est jouée. Dans ce cas, on fait tourner la musique en boucle en remettant le currentTime de la musique à zéro à chaque appel de *stopMedia1*(). On agit de même tant que la durée de la musique est supérieure à la durée pendant laquelle le média a été lu. On connaît cette durée grâce à la variable **raz**, qui stocke le temps passé lorsqu'on remet la musique en boucle. On compare donc ***(raz*** + ***media.duration)*** à **dureeMus** pour savoir s'il est nécessaire de remettre la musique en boucle une fois de plus. Sinon, on arrête la musique normalement. S'il s'agit d'une musique qui est écoutée lors de la lecture d'une playlist, on passe à la musique suivante de la playlist si cette musique n'était pas la dernière de la musique.

Cependant, cela ne suffit pas à faire tourner la musique exactement autant de

temps que la durée du contenu sonore à jouer, car cette fonction n'est appelée qu'à chaque fois que le média a fini d'être lu. On ajoute donc un event listener sur l'événement 'timeupdate' du média, qui appelle la fonction *setTime*(). Cette fonction affiche le temps de lecture écoulée du média, en prenant en compte les boucles avec la variable **raz**, et vérifie à chaque appel que (**media**.**currentTime** +**raz**) >= **dureeMus**, c'est-à-dire que le temps de lecture totale de la musique n'a pas dépassé la durée du contenu qui est lu. Si c'est le cas, on arrête la lecture et on finit de lire le contenu.

Dans le cas de lecture d'une playlist dans **exploreCat.jsp**, tout se passe de façon identique, sauf qu'au lieu d'appeler *playMusique*() on appelle *playPlaylist*(). Il est alors possible de naviguer dans la playlist pour lire les autres musiques de la playlist sans avoir à recharger la page. Comme on l'a dit plus haut, à la fin d'une musique on enchaîne aussi avec la lecture de la musique suivante de la playlist. C'est pour cela qu'on avait marqué toutes les musiques d'une playlist comme lues dès qu'une musique d'une playlist était choisie. Des boutons avant et suivant permettent de lire la musique précédente ou suivante de la playlist choisie. Ces boutons appellent les fonctions Javascript *musAvant*() et *musSuivant*(). Ces fonctions appellent *playMusiqueFunction*() avec comme paramètre le numéro dans la playlist de la musique en cours de lecture, plus ou moins un selon qu'on avance ou recule dans la playlist. Ce numéro est récupéré grâce à la manière dont les playlists sont affichées sur la page. Au moment d'afficher une musique dans une playlist, dans le bouton qui l'affiche, est remplit un attribut **num** correspondant au numéro de la musique dans la playlist, numéro qu'on récupère grâce à l'identifiant de la musique qui est lui aussi renseigné pour le bouton. *playMusiqueFunction*() remet le temps de la musique à zéro, et remplace toutes les informations de lecture de la musique courante par la nouvelle choisie.

- Gestion de la playlist d'un utilisateur

La gestion de la playlist d'un utilisateur se fait depuis la page **modifierPlaylist.jsp**. Les playlists que l'utilisateur possède déjà sont affichées grâce à une liste de playlists transmise par la servlet **ModifierPlaylist** sous format Json. Lorsque l'utilisateur clique sur une de ses playlists, son contenu est affiché et la playlist est sélectionnée. L'utilisateur peut alors choisir de la renommer ou de la supprimer. Il peut aussi choisir de créer une nouvelle playlist en donnant un titre pour celle-ci.

Selon l'action faite par l'utilisateur, les paramètres de nouveau titre ou d'identifiant de la playlist sont transmis à la servlet **ModifierPlaylist**, qui agit selon l'action demandée.

Si une nouvelle playlist doit être créée, on crée une nouvelle instance de Playlist avec le titre choisi pour le client. On appelle ensuite *createPlaylist()* de **PlaylistDatabase** avec cette instance, qui crée une entrée correspondante dans la table Playlist en renseignant la date courante pour la date de création de la playlist. Rien n'est ajouté dans **ContenuPlaylist** car la nouvelle playlist est vide.

Si la playlist doit être renommée, la servlet récupère la playlist choisie dans la base de données grâce à son identifiant avec *getPlaylistById()* de **PlaylistDatabase**, puis rempli cette playlist avec le nouveau nom grâce à un setter, et enfin appelle *renamePlaylist()* de **PlaylistDatabase** pour changer le nom de la playlist dans la base de données.

Si la playlist doit être supprimée, la servlet appelle la fonction *deletePlaylist()* de **PlaylistDatabase** qui supprime la playlist dont on fournit l'identifiant dans la base de données.

- ### Ajouter une musique à une playlist

L'ajout de musique à une playlist se fait depuis **exploreCat.jsp** en cliquant sur une musique depuis la section principale d'affichage. Le modal qui s'ouvre permet alors, en plus d'écouter la musique, de l'ajouter à une playlist en sélectionnant la playlist parmi celles de l'utilisateur. Si l'on clique sur ajouter après avoir sélectionné une playlist, la page envoie la requête à la servlet **ExploreCatalogue** avec l'identifiant de la musique et de la playlist choisies en paramètre.

Si ces deux champs ne sont pas nuls, **ExploreCatalogue** appelle la fonction *addMusiquetoPlaylist()* de **PlaylistDatabase**, avec la playlist renvoyée par *getPlaylistById()*, et la musique renvoyée par *getMusique()*. *addMusiquetoPlaylist()* ajoute la musique à la playlist, puis explore la table **ContenuPlaylist** pour trouver la position de la dernière musique de cette playlist. Elle ajoute la musique choisie à **ContenuPlaylist** en la plaçant à fin de la playlist, grâce à la position récupérée plus tôt (ou à la première position si la playlist était vide). Dans **ContenuPlaylist**, les positions des musiques commencent à un, il y a donc un décalage de un entre la position d'une musique dans la base de donnée et dans la liste de musique de l'objet **Playlist** de Java.

- ### Modification du catalogue musical depuis un compte administrateur

La modification des contenus sonores du catalogue requiert la connexion à un compte administrateur comme vu dans la partie **Connexion à un compte utilisateur ou administrateur**. Une fois connecté, l'administrateur est redirigé sur la page correspondant à son type de profil. La saisie de l'url suivante **/Administration/AdminGestionnaireMusical**, va appeler la servlet **ModificationCatalogue**. Cette servlet va nous rediriger vers la page **adminModifCatalogue.jsp**. On trouve sur cette page un premier formulaire qui va permettre de choisir, via des listes déroulantes, l'action à réaliser (Ajouter, Modifier, Supprimer) ainsi que le type de contenu (Musique, Radio, Podcast). Une fois décidé, l'administrateur va cliquer sur le bouton **Valider** ce qui va permettre d'envoyer via la méthode **POST** le formulaire à la servlet **ModificationCatalogue**. Dans la servlet, on récupère les paramètres correspondants : **action** et **choixContenu** et on vérifie que la variable **action** est non **NULL**. Si c'est le cas, cela signifie que ce premier formulaire a bien été envoyé. Pour permettre la persistance des valeurs des choix précédents, les valeurs des variables **action** et **choixContenu** sont mises en attributs de requêtes et

seront récupérées sur la page jsp. Toujours dans la servlet, un booléen, indiquant que le premier formulaire a été traité, est mis à vrai.

Le fonctionnement, qui suit, sera décrit pour un seul cas de contenu sonore parmi les trois autres, c'est-à-dire la musique. Néanmoins, si le nombre et la nature des champs des deux autres types de contenus sonores sont différents, les mécanismes sont identiques.

La servlet vérifie si l'action demandée est l'ajout. Dans ce cas, nous serons redirigés vers une autre servlet **TraitementModificationCatalogue** via une url à qui on passera les paramètres correspondants à l'action et le choix du contenu. Dans cette servlet, on est redirigé vers la page **adminModifCatalogueTraitement.jsp**. On récupère en début de page les paramètres d'action et de choix du contenu. Plus bas dans le fichier, on regarde le type de contenu qui est demandé, puis l'action qui est choisie. Comme il s'agit d'un contenu de type **Musique** et qu'on souhaite ajouter, cela signifie qu'un objet **musique** vide sera créé. À partir de là, un formulaire est créé et se remplit des champs vides de cet objet. Une fois que ces champs sont complétés, l'administrateur appuie sur **Valider**. De plus, l'identifiant de la musique est automatiquement choisi par la base de données. Le formulaire est alors envoyé à la servlet **TraitementModificationCatalogue** par la méthode **POST**. Au niveau de la servlet, on récupère toutes les informations du formulaire, ainsi que l'identifiant précis du bouton de ce formulaire. Cela permet de faire la distinction entre les neufs cas qui mélangent l'ajout, la modification et la suppression avec les trois types de contenus sonores. Ici, l'identifiant du bouton est **AjouterMusiqueButton**. Au préalable, on crée un objet Musique contenant la valeur de tous les champs renseignés. Si le paramètre récupéré n'est pas nul, alors on ouvre une connexion avec la base de données, et on effectue l'insertion de l'objet musique dans la table **ContenuSonore** et **Musique** via la méthode *createContenuSonore()*. Après cela, le booléen **addSuccess** est mis à "vrai", puis est passé en attribut de requêtes. Nous sommes ensuite redirigés sur la page **adminModifCatalogueTraitement.jsp**, qui affiche un div contenant le message de succès de modification de la base de données grâce à ce booléen.

Si l'action demandée est la modification des champs d'un contenu sonore ou la suppression d'un contenu sonore, l'attribut **firstStep** est mis à "vrai" et est passé en attribut de requête. On est redirigé sur la page **adminModifCatalogue.jsp**, et grâce aux attributs mis en requête, on conserve le choix de l'action et du contenu sonore. De plus, le booléen **firstStep** permet la génération d'un formulaire de recherche du contenu sonore. On peut rechercher soit par nom, soit par identifiant du contenu. En cliquant sur

le bouton de type submit **Valider**, l'information de recherche est envoyée à la servlet **ModificationCatalogue** via la méthode **POST**. Au niveau de la servlet, on récupère le paramètre **fieldModifierSupprimer**. Étant non nulle, on récupère tous les paramètres transmis jusqu'ici et on les met en attributs de requêtes pour permettre la persistance d'informations. Un second booléen **secondStep** est ensuite mis à "vrai", puis est passé en attribut de la requête. Comme le choix du contenu correspond à la musique, une connexion avec la base de données est ouverte, puis la méthode *getMusic()* est utilisée avec les paramètres appropriés. Au niveau de la recherche SQL, par définition de la clé primaire, si un identifiant de musique est rentré, on obtient un unique résultat si la musique existe. Si un nom est recherché, grâce à l'opérateur **LIKE**, on récupère dans la liste de résultat tous les noms qui ressemblent à celui saisie précédemment. Cela permet une recherche plus aisée et efficace des contenus du catalogue. La liste de musique fournis par *getMusic()* est mise en attribut de la requête. On est redirigé vers la page **adminModifCatalogue.jsp**. Grâce au booléen **secondStep**, on entame la phase de construction du tableau de résultat. La liste des musiques recherchées est récupérée en haut de page dans **resultatListeMusique**. Il y a vérification de la longueur de la liste de résultat, car dans le cas où elle est de taille nulle, un élément div contenant un message d'erreur est affiché. Dans le cas où la taille est supérieure à zéro, le tableau des résultats se construit. Sinon un élément div apparaît en bas de page pour signaler l'absence de résultat de la recherche. Chaque ligne correspond à une musique et est remplie des informations de cette musique. Par ailleurs, une dernière colonne **Select** est ajoutée et contient un lien unique au contenu sonore. Ce lien contient l'url vers la servlet **TraitementModificationCatalogue** ainsi que les paramètres caractérisant l'action, le type de contenu et l'identifiant du contenu choisi.

L'administrateur clique alors sur le contenu qu'il désire modifier ou supprimer. Il y a redirection vers la servlet **TraitementModificationCatalogue** qui redirige à son tour sur la page **adminModifCatalogueTraitement.jsp**. Comme la variable **choixContenu** est non nulle car elle vaut la chaîne de caractères **Musique** et que l'on possède l'identifiant du contenu sonore, on va récupérer depuis la base de données l'objet **musique** correspondant via la méthode *getMusic()*. Cet objet va permettre de remplir les champs du formulaire de modification/suppression qui se construit. Il est à noter que l'identifiant des contenus sonores est non modifiable via l'attribut **disabled** présent dans les balises **select** et **label**. Ce choix vient du fait que les identifiants sont automatiquement choisis par la base de données. Une fois la modification achevée, ou que l'on désire supprimer le contenu, on appuie sur le bouton **Valider** de type submit. Comme dit plus haut, l'attribut **name** de ce bouton permet de renseigner le traitement au niveau de la servlet. Cela envoie les informations, aux champs **name** uniques à ce type

de contenu, à la servlet **TraitementModificationCatalogue**. Toutes ces informations sont récupérées dans la servlet, puis sont mises dans un objet Musique. Comme il est impossible de récupérer l'information de champs en mode **disabled** depuis le formulaire, on met dans l'objet **musique** l'identifiant passé en paramètre de l'url via **idSent**. Il y a alors deux cas, après récupération de la valeur des boutons de validation de formulaire, soit **ModifierMusiqueButton** est non nulle, soit c'est **SupprimerMusiqueButton**. Dans un cas, la base de données est mise à jour via la méthode *updateContenuSonore()*. Dans l'autre cas, la suppression du contenu se fait via la procédure *deleteContenuSonore()*. Quand la modification ou la suppression est réussie, le booléen **modifySuccess** (resp. **deleteSuccess**) est mis à vrai puis passé en attribut de requêtes. Après redirection sur la page **adminModifCatalogueTraitement.jsp**, l'un des booléens permettra la création d'un élément div avec un message de succès appropriés.

- Modification des informations d'un compte utilisateur depuis un compte administrateur

La modification des profils clients du catalogue requiert la connexion à un compte administrateur comme vu dans la partie **Connexion à un compte utilisateur ou administrateur**. Une fois connecté, l'administrateur est redirigé sur la page correspondant à son type de profil. Tout d'abord, l'url /Administration/AdminProfilClient va nous faire passer à travers la servlet **adminPageProfil**. Cette servlet va nous rediriger directement vers la page **adminResearchClient.jsp**. Sur cette page s'affiche un premier formulaire permettant de rechercher les clients présents sur la base de données selon leur nom, ou prénom ou email. L'administrateur peut, par exemple, renseigner seulement le champ **Nom** puis appuyer sur le bouton de type submit **Recherchez**. Cette action va envoyer via la méthode POST le formulaire à la servlet **adminPageProfil**. La servlet récupère les champs envoyés en formulaire, puis vérifie qu'au moins un est non **NULL**. Dans ce cas, une connexion avec la base de données est ouverte, et la méthode *searchClient()* va effectuer une requête SQL pour récupérer les clients qui correspondent plus ou moins aux valeurs des champs. Cela est possible, via l'opérateur **LIKE** qui permet la recherche de pattern. De plus, seuls des clients peuvent sortir de cette requête puisqu'on effectue une jointure symétrique entre la table **Compte** et **CompteClient**. Après que la liste de compte de client est retournée, elle est mise en attribut de requête sous le nom de **groupeUtilisateurEnvoye**. On est redirigé vers la même page jsp et on récupère la liste des comptes clients dans la variable **resultatResearch**. Plus bas dans la page, on vérifie que la variable n'est pas vide, et qu'elle est de taille non nulle. En cas de succès, un tableau dynamique, contenant les informations des champs des utilisateurs, est construit en effectuant une boucle sur chacun des éléments de la liste. Chaque ligne

du tableau correspond à un client. De plus, une dernière colonne nommée **Valider** est ajoutée contenant un lien unique au client respectif. Ce lien contient l'url menant à la servlet **adminPageProfil** avec en plus, l'information du mail du compte client respectif passé en paramètre. Si l'administrateur désire modifier les informations d'un client, il clique sur le lien **Select** correspondant à ce client. Il est alors redirigé vers la Servlet **adminPageProfil**. Cette servlet va récupérer l'email passé en paramètre dans la variable **emailSelected**. La variable étant non nulle, la variable **emailSelected** est mise en requête d'attribut et nous redirige vers une nouvelle page jsp appelée **administrationProfilClient.jsp** qui est chargée de permettre la modification des champs clients. Le fonctionnement de cette page est très similaire au cas **"Modification des informations d'un compte utilisateur"** et de la même manière, la vérification des champs des clients est exactement identique que dans le cas de la **"Création d'un compte utilisateur"**. Cela vient du fait qu'on utilise le même script **inscription.js.** Il est à noter qu'une fois la modification des informations du client est terminée, un message de succès apparaît. On peut revenir sur la page de recherche en cliquant sur le titre **Modification Profil Client** qui contient un lien non-paramétré et va ainsi rediriger vers cette page.

Il reste à parler du cas ou aucun résultat client n'est retourné à l'issue de la recherche. Toujours sur la page **adminResearchClient.jsp,** la liste **resultatResearch** est donc NULL, après comparaison, un élément **div** est créer avec un message d'erreur qui s'affiche.

- ### Blocage d'accès et redirection des mauvais utilisateurs

Il est important que seules les personnes autorisées à accéder aux pages de l'application web puissent le faire. Par exemple, un utilisateur non connecté n'a pas le droit d'accéder au page d'administration ou au catalogue. Hors, il est trivial de le faire en tapant la bonne URL. Nous avons utilisé des classes java de filtre pour empêcher ces comportements. Il y a trois filtres différents pour réglementer les trois types d'utilisateurs que nous avons. Il y a la classe **AdminFilter** qui traite les accès par des sessions administrateurs, la classe **ConnectedUserFilter** qui traite les accès par des sessions clients, la classe **ConnectedUserFilter** qui traite des accès par des utilisateurs non connectés et la classe **NonAdminFilter** qui empêche les administrateurs d'accéder à d'autres pages que celles attribuées.

Seul le fonctionnement du filtre pour les utilisateurs connectés sera décrit étant donné que le fonctionnement est très similaire pour les autres filtres. Le filtre

ConnectedUserFilter est configuré dans le fichier **web.xml** de sorte à être appliqué à l'accès de certaines servlets. Cela signifie qu'une URL menant à une servlet va lancer le filtre approprié. Dans le filtre **ConnectedUserFilter**, on passe dans la méthode *doFilter()* et on vérifie via la récupération de session et la méthode *getAttribute*() appliqué à l'objet **session**, si une session utilisateur a été enregistrée. Si c'est le cas, l'accès vers l'URL est autorisé. Sinon, il y a redirection vers la page d'accueil.

Diagramme de classe

Le diagramme de classe est présent dans l'archive des livrables.

CONCLUSION

En repartant des travaux effectués en UML, nous avons utilisé les technologies du web de JEE pour créer une application qui réponde au cahier des charges. Le projet comporte toutes les fonctionnalités requises ainsi que plusieurs fonctionnalités optionnelles comme la modification du profil du client par un administrateur.

En plus de nous familiariser avec le fonctionnement des servlets et des pages jsp, ce projet nous a ainsi permis de découvrir les mécanismes de session et de filtre des servlets, et nous a fait utiliser plus en profondeur l'HTML, le CSS et Javascript. Nous avons aussi pu revoir les mécanismes de création et de gestion de base de données, ainsi que la façon de l'utiliser depuis une application Java.

L'organisation de la structure globale du projet, allant des pages web à la gestion des contenus sonores et des comptes des utilisateurs est donc directement issue du projet UML. La description des cas d'utilisations principaux et leur déclinaison en diagrammes lors de ce projet nous avait permis de clarifier leur fonctionnement, et nous les avons suivis de façon plus ou moins proche selon les contraintes et les particularités des technologies utilisées. Pour ce qui est de la base de données, elle correspond de façon assez fidèle aux classes UML puis Java de notre application, à l'exception de la gestion des playlist qui a dû être quelque peu adaptée. Nous avons donc beaucoup profité du travail réalisé en amont de ce projet, ce qui nous a fait gagner beaucoup de temps et d'organisation. Cela nous a aussi permis de passer directement à l'implémentation des fonctionnalités importantes, et de bien nous concentrer sur la découverte et l'utilisation de ces technologies qui nous étaient inconnues.