Livrable JEE

Spiritual Music

Team Spiritual

18 mars 2021

Table des matières

1 Introduction

Pour ce projet, l'équipe Spiritual est composée de : Alex Fulbert, Anthony Ingels, Eliott Thomas et Julien Thomas.

L'objectif de ce projet est de développer partiellement un site de streaming musical. On ne s'occupe pas ici de la gestion du contenu multimédia. Nous implémenterons les fonctionnalités suivantes :

- Parcours du catalogue musical (Titres,Albums) par les membres et administrateurs
- Recherche par mot clé
- Gestion par les administrateurs du contenu du catalogue et des données clients
- Inscription en tant que membre
- Gestion des playlists personnelles par les membres
- Hachage des mots de passe pour plus de sécurité

La conception du projet s'appuie sur les éléments du projet UML : diagrammes de classes, diagramme de cas d'utilisation. Nous développerons le site sur le modèle MVC(model, view, controller/servlet), sans frameworks, contenu par un serveur Tomcat 9.0.

2 Choix Techniques

2.1 Utilisation d'XHR

Nous avons utilisé à l'aide du langage javaScript l'objet httpRequest XHR afin de réaliser des requêtes asynchrone avec le serveur. En ce sens, cela permet d'envoyer des requête à la Servlets concernée en lui indiquant ou non des paramètres dans l'URL. Ainsi, la servlet produit une réponse HTML qui sera récupérée côté client et affichée dans une balise spécifique dans le DOM (Représente le document affiché par une structure en arbre, comportant des nœuds). De ce fait, on évite de recharger entièrement la page à chaque requête, ce qui est plus agréable pour l'utilisateur.

2.2 Utilisation d'XML

Comme mentionné dans le cours, nous avons favorisé l'utilisation du `@WebServlet("/nomDeLaPage")` plutôt que de l'écrire dans le fichier `web.xml` comme il a été demandé de le faire en TP.

2.3 Utilisation de la session

Nous avons décidé d'utiliser la session pour transmettre des informations complexes, comme les objets membres et visiteurs. En effet, à travers la requête nous pouvions simplement transmettre des chaînes de caractères, ce qui devient rapidement lourd si l'on souhaite transmettre de nombreuses informations.

2.4 MD5

Il nous a semblé important de ne pas laisser les mots de passe en clair dans la base de données. Pour cette raison nous avons haché les mots de passe avec l'algorithme MD5. Notre idée initiale était de hacher les mots de passe de manière beaucoup plus robuste, en utilisant l'algorithme SHA-2 et un salt. Malheureusement, suite à un manque de temps, nous avons décidé de faire une première version du hachage des mots de passe en MD5.

3 Fonctionnalités du site

3.1 Page du visiteur

- Un visiteur n'a accès qu'à la playlist publique, les fonctions de tri et de recherche ne parcourent que les éléments de la playlist publique.
- Il peut aussi créer un compte pour devenir membre.

3.2 Page du membre

- Le membre a accès à la playlist publique sur la page principale
- Il peut parcourir l'ensemble du catalogue à l'aide des boutons, les menus déroulants et de la barre de recherche
- Il peut gérer ses playlists personnelles
- Il consulter et modifier les informations de son compte (changer le mot de passe, l'adresse, l'email ...).

3.3 Page Administrateur membre

- L'administrateur de membres a accès à l'ensemble des fonctionnalités de membre.
- Il peut également supprimer un membre ou modifier les informations d'un compte.

3.4 Page Administrateur musical

- L'administrateur musical a accès à l'ensemble des fonctionnalités de membre
- Il peut également modifier, ajouter ou supprimer un élément du catalogue

3.5 Fonctionnalité Bonus : bloquer et débloquer un compte

Si un utilisateur tente de se connecter à un compte et se trompe de mot de passe, il a été décidé de bloquer le compte pour une heure s'il y a 3 tentatives frauduleuses.

3.6 Fonctionnalité Bonus : Hachage des mots de passe

Nous avons décidé de hacher les mots de passe des utilisateurs pour éviter de les stocker en clair dans la base de données. Nous avons choisi la technique de chiffrement MD5 pour sa facilité d'implémentation. Ainsi, quand un membre s'inscrit sur le site, c'est le haché de son mot de passe qui est stocké dans la base de données. Lorsque le membre souhaite se connecter par la suite, il peut simplement rentrer son mot de passe, et l'on compare le haché du mot de passe rentré avec le haché présent dans la base de données.

4 Base de données

Les informations structurées du catalogue et des membres sont stockées dans la base de données. Nous allons expliquer comment les playlists personnelles, publiques et les albums sont implémentés.

4.1 Titres

La table *Titres* contient l'ensemble des titres du catalogue ainsi que leurs différents attributs, dont l'attribut *refAlbum* qui est égal à 0 si le titre associé n'appartient pas à un album, ou égal à l'id de l'album de référence présent dans la table *Albums*. Ainsi on peut retrouver par une requête simple l'album associé à un titre ou les titres composants un album. L'attribut *inPb* est égal à 1 si le titre est présent dans la playlist publique qui sera donc affichée dans les titres du moment.

4.2 Albums

Comme pour *Titres*, la table *Albums* possède l'attribut *inPb* égal à 1 si l'album doit apparaître dans les titres du moment. Comme expliqué pour la table *Titres*, l'attribut id d'un album permet de faire référence à celui-ci pour déterminer les titres qui lui appartiennent.

4.3 lienAlbumTitres

Pour faciliter le travail de recherche "Quel titre appartient à cet album ? A quel album appartient ce titre ?", la table *lienAlbumTitres* est constituée des attributs *idTitre* et *idAlbum* où chaque occurrence associe l'id du titre à l'id de l'album auquel il appartient.

4.4 PlaylistPerso

Une playlist personnelle ne doit être accessible que par le membre qui l'a créée. La table *PlaylistPerso* contient l'id et le nom de toutes les playlists personnelles. Trois autres tables sont créés :

- *lienPlaylistPersoMembre* : cette table associe l'id des playlists personnelles à l'id du membre qui les possèdent. Ainsi le membre a accès à la liste des ses playlists.
- *lienPlaylistPersoTitres* : cette table associe l'id du titre à l'id de la playlist personnelle qui le contient.
- *lienPlaylistPersoAlbums* : cette table associe l'id de l'album à l'id de la playlist personnelle qui le contient. Ces deux dernières tables permettent de déterminer le contenu d'une playlist personnelle.

4.5 membres

Cette table contient les informations de tous les membres et administrateurs. Les attributs *adminCompte* et *adminMusique* prennent la valeur 1 si le membre est un administrateur du domaine musique ou du domaine catalogue. Ils donnent accès aux fonctionnalités propres aux administrateurs.

Note : Nous n'avons au final pas utilisé la base de données fournie en début de projet. En effet, suite à l'incendie des serveurs OVH, nous avons mis en place une base de données"AlwaysData". Nous avons travaillé avec celle-ci depuis et n'avons pas souhaité revenir à une version trop antérieure de la base de données quand le backup a été fourni.

5 Conclusion

Pour conclure, il a été possible de mettre en place la plupart des fonctionnalités demandées par le sujet : créer un compte, modifier ce compte, accéder à l'application en tant que visiteur ou membre, pouvoir gérer des titres et des albums etc etc. Nous avons toutefois été limité par le temps . Finalement nous avons réussi à implémenter un système de hachage des mots de passe. Dans une version future de cette application nous souhaiterions pouvoir rajouter le système d'xp et de succès que nous avions prévu de faire lors de la partie UML. Nous souhaiterions également pouvoir améliorer et homogénéiser les interfaces de l'application. Il aurait également été intéressant de se pencher sur la mise en place de HTTPS. Enfin, il serait évidemment souhaitable de pouvoir travailler avec de vrais morceaux de musique pour plus d'immersion. Nous sommes néanmoins satisfait du résultat final de l'application considérant le temps accordé et la période de travail.