# Gestionnaire de playlists

Rapport de projet – LINFO1212

Olivia Alvares
Romain Carlier
Laurane Castiaux



UCLouvain

Année académique 2020 – 2021

# Table des matières

# Introduction

Le site web que nous avons décidé de développer pour notre projet final est un gestionnaire de playlists de musiques. Bien que le concept de gestionnaire de playlists puisse être facilement étendu ou transposé à d'autres types de médias (partage de liens vers diverses ressources sur un sujet, partage de vidéos de vulgarisation par thème, etc.) nous avons choisi de nous limiter au partage de musiques pour donner à notre application un objectif clair.

Les playlists sont visibles par tout le monde depuis la page d'accueil. Seuls les membres connectés peuvent créer de nouvelles playlists. Seul le propriétaire d'une playlist peut la modifier ou la supprimer. Pour cela il lui faut accéder à la page *account* qui rassemble l'ensemble de ses playlists.

En cliquant sur le titre d'une playlist depuis la page d'accueil, la page *account* ou une page utilisateur (page reprenant toutes les playlists d'un utilisateur et qui est accessible en cliquant sur le nom d'un utilisateur dans la liste de playlists affichée sur la homepage), on accède à la page de cette playlist. On peut y voir sa description complète, ainsi que toutes les musiques qui la composent. Pour toutes les chansons qui proviennent de Youtube, Vimeo, Dailymotion ou SoundCloud, il est possible de les écouter dans un mini-player en cliquant sur le bouton "play". Si toutes les chansons proviennent de Youtube, elles sont jouées automatiquement les unes après les autres. Pour des chansons provenant d'autres sources, un lien vers la chanson est disponible; il faut alors cliquer sur le lien pour ouvrir la chanson dans une nouvelle page, hors de notre site.

Il est possible de faire une recherche simple ou avancée afin de trier les playlists. Les genres peuvent être triés directement sur la page d'accueil, sans devoir passer par la recherche. Il est également possible de trier les résultats selon ordre croissant ou décroissant du titre, du créateur de la playlist, de la description ou des dates de modification et de création en cliquant sur le titre des colonnes correspondantes.

Notre site permet donc de créer et partager en un seul endroit des playlists comprenant des musiques provenant de différentes sources sans avoir à créer de compte sur les multiples plateformes. De plus Vimeo ne permet pas la création de playlists si ce n'est de vidéos dont leurs utilisateurs sont propriétaires ce qui rend notre application d'autant plus pertinente pour cette plateforme.

# Objectifs

Nos objectifs initiaux étaient :

- gérer des comptes utilisateur
- crypter les mots de passe
- gérer les bases de données (utilisateurs et playlists)
- gérer les permissions (ajout, modification et suppression de playlists uniquement si connecté)
- implémenter une recherche simple et une recherche avancée
- filtrer les playlists par genre

A ces objectifs, nous avons ajouté :

- un mini-player imbriqué dans notre site (pour les principales plateformes gratuites de musique en ligne)
- un tri selon différents attributs des playlists
- un tri selon différents attributs des chanson au sein d'une playlist

# Architecture du système

## Arborescence du projet

A la racine de notre projet se trouvent notre *README*, le programme principal de notre serveur (*server.js*) et les fichiers json qui gèrent nos packages. Il s'y trouvent également quatre dossiers : __tests__, node_modules, modules et static.
Le dossier __tests__ contient le fichier *jtest.tests.js* qui est le programme où nous avons écrit nos tests. Le dossier node_modules contient les modules externes que nous avons utilisés (voir section "librairies" ci-dessous). Le dossier module contient le fichier *playlist_youtube.js* qui gère l'interaction avec l'API de Youtube. Le dossier statique contient toutes nos pages html et css ainsi qu'un dossier img, contenant les images que nous avons utilisé et un dossier fonts, contenant les polices d'écriture.

## Librairies

Tout comme dans le projet préparatoire, nous avons utilisé les librairies *express*, *consolidate*, *body-parser*, *mongodb*, *https*, *http*, *fs*, *express-session* et *bcrypt*. Nous avons utilisé en plus *get-video-id* et *xmlhttprequest*.

**get-video-id** est une librairie que nous avons utilisée pour pouvoir facilement extraire l'identifiant de l'URL des vidéos provenant de youtube et de vimeo, celui-ci étant nécessaire à l'intégration de la vidéo à notre site ainsi que pour récupérer le titre de la vidéo.

**xmlhttprequest** est une librairie que nous avons utilisée pour faire une http request synchronisée afin de pouvoir récupérer le titre des vidéos provenant de youtube, vimeo et dailymotion.

# Qualité du code

## Guide de style

Pour améliorer la lisibilité de notre code ainsi que sa cohérence, nous avons pris les décisions suivantes :

- indentation en javascript de 4 espaces
- indentation en html et css de 2 espaces
- noms de variables sans abréviation, en minuscule, avec un underscore pour séparer les mots s'il y en a plusieurs
- noms de fonctions en camelCase
- regroupements des éléments similaires

## Revues de codes

Nous avons décidé de nommer les responsables suivants:

- html / css : Laurane
- javascript / tests : Romain
- base de donnée / recherche : Olivia

Mais nous avons tous travaillé sur toutes les parties de notre projet. Quant à la relecture du code, étant donné que notre site est assez petit, nous avons décidé de relire l'ensemble du projet, à trois, une fois le site (presque) terminé.

# Base de données

## MongoDB Atlas

Afin de gérer notre base de données, nous avons choisi d'utiliser MongoDB Atlas qui, développé par les créateurs de MongoDB, utilise le même modèle de documents JSON que ce dernier. Atlas présente deux avantages importants : tout d'abord, notre base de données est centralisée sur le cloud et peut facilement être gérée via l'interface utilisateur Atlas; ensuite, la fonction de recherche Atlas Search permet une recherche plus performante que la recherche native de MongoDB qui utilise un indice "text" unique. En effet, Atlas Search, qui se base sur le moteur de recherche opensource Apache Lucene, permet:

- la création de plusieurs indices
- de définir au moment de la requête l'élément sur lequel doit se baser la recherche
- une recherche sur différents types de données (notamment: texte, dates, objectID)
- d'utiliser une recherche approximative ("fuzzy search", cf. description de la fonction de recherche ci-après) qui apporte une certaine tolérance aux fautes de frappes et inclus des mots similaires à ceux recherchés

## Description de la base de données

Nous avons décidé de séparer notre base de données en deux collections distinctes. La première est consacrée aux utilisateurs et la seconde aux playlists et à leur contenu.

La première collection, nommée "users" est composée de tous les comptes utilisateur. Chaque utilisateur a un username, un mot de passe et une adresse email. Deux utilisateurs ne peuvent avoir le même username, ce qui est vérifié lors de la création de compte. Bien entendu, les mots de passe des utilisateurs ne sont pas en "clair" dans la base de données mais sont cryptés par le module bcrypt (voir description sécurité ci-dessous).

La seconde collection, nommée "playlists" est composée de toutes les playlists enregistrées par les utilisateurs. Voici les différents attributs d'une playlist :

- title : le titre de la playlist
- description : la description de la playlist
- creator : l'username du créateur de la playlist
- creation_date : la date de création de la playlist
- modification_date : la date de la plus récente modification
- genres : un array avec tous les genres de la playlist
- songs : un array d'objets "song" (description d'un objet song ci-dessous)
- color : la couleur du logo de la playlist

- theme : le thème du logo en fonction de la couleur choisie
- song_titles : un array reprenant les titres de toutes les chansons de la playlist

Description d'un objet song :
- url : l'url de la chanson
- date : la date d'ajout de la chanson dans la playlist
- vid_id : l'id de la chanson (si le site source en propose un)
- vid_title : le titre de la chanson
- source : le site duquel vient la chanson
- embedded_video : le lien pour afficher une iframe qui joue la chanson
- play_button : un string qui apparaît quand on passe la souris sur le bouton play

# Fonctions de recherche

Nous avons implémenté deux fonctions de recherche : une recherche simple et une recherche avancée.

La recherche simple va rechercher tous les mots entrés par l'utilisateur dans les champs *description, creator, title*, *song_titles* et *genres*. Nous avons utilisé la méthode collection.aggregate() qui retourne un score unique en fonction de la recherche des mots-clés dans chaque champ spécifié. Nous avons utilisé la pipeline $search uniquement sur des champs "text". Nous lui avons appliqué les valeurs par défaut de "fuzzy", c'est-à-dire que la recherche va inclure dans les résultats les mots qui sont à une distance de Levenshtein de 2, mais en se limitant à 50 variantes.

La recherche avancée se déroule en deux parties: d'abord une recherche par mots-clés, puis un filtre selon les dates.
La recherche par mots-clés utilise "compound" qui permet de spécifier plusieurs recherches différentes à faire en même temps. L'utilisateur peut entrer des mots différents pour les champs *description, creator, title, genres* et *song_titles.* La recherche se fait à chaque fois dans le champ correspondant. Plus il y a de champs qui correspondent à la recherche, mieux le résultat sera classé. A chaque fois, un champ "text" est utilisé avec les paramètres par défaut de "fuzzy", comme expliqué précédemment. Si l'utilisateur n'a entré aucun mots-clés, toutes les playlists sont retournées à ce stade.
Les playlists sélectionnées sont ensuite filtrées selon les dates de modification et/ou de création entrées par l'utilisateur.

Pour les deux recherches, les résultats sont affichés par ordre décroissant de pertinence.

# Sécurité

## https

Afin de sécuriser l'utilisation de notre site web, le serveur utilise par défaut le protocole "https" qui utilise une clé préalablement créée afin de sécuriser la connexion. Ceci étant dit, il est possible de lancer notre serveur en utilisant le protocole "http" en ajoutant "http" en argument à la ligne de commande pour lancer le serveur. L'utilisation du protocole "http" non sécurisé s'est avérée nécessaire pour réaliser les tests.

### bcrypt

Pour améliorer la sécurité de notre site web, nous avons fait le choix d'utiliser la librairie "bcrypt". Celle-ci nous permet de facilement hasher un mot de passe lorsqu'un utilisateur créer un compte pour ne pas le stocker en "clair" dans la base de donnée. Dès lors, lorsqu'un utilisateur essaie de se connecter à son compte, nous rappelons le module pour qu'il "décrypte" le mot de passe afin de vérifier que celui-ci est correct. C'était pour nous une fonctionnalité basique très importante à avoir afin que les mots de passe ne voyagent pas lisiblement entre le serveur et la base de données.

# Tests

Dans notre groupe, le responsable des tests était Romain. Le but de nos tests étaient de vérifier que notre site fonctionnait bien pour de simples tâches telles que :

- Se créer un compte
- Se connecter au site après avoir créer son compte
- Se déconnecter
- Créer une nouvelle playlist
- Vérifier que cette playlist ait été ajoutée
- Tester la fonctionnalité de recherche

Tous les tests que nous avons écrits passent sans le moindre souci.
Nous avons travaillé sur les tests une fois que le site était globalement terminé. Cela nous semblait plus cohérent de travailler avec une version semi-finale de notre site pour faire les tests que de les faire au fur et à mesure. Nos scénarios de tests sont basiques, un compte est créé avec un username généré aléatoirement à chaque fois.

# Perspectives

Pour l'instant, notre site permet de jouer directement les musiques fournies par les plus grands sites de streaming de musique gratuit, mais il en existe d'autres que nous voudrions également pouvoir inclure.
NB: nous avons envisagé de partager des liens provenant de Spotify et Deezer mais ceux-ci nécessitent la connexion à un compte pour permettre la lecture de musiques.

La lecture automatique actuelle se sert de l'API de Youtube. Nous voudrions trouver un moyen de jouer les chansons les unes à la suite des autres pour les autres sites également.

Les titres ne sont disponibles que pour les vidéos provenant de Youtube, Vimeo et Dailymotion, il serait intéressant de pouvoir récupérer ceux de SoundCloud également mais l'API n'accepte pas de nouvelles demandes d'applications à l'heure actuelle. Pour SoundCloud comme pour les liens provenant d'autres sources, il serait intéressant de trouver une alternative pour récupérer le titre de la vidéo ou bien d'implémenter la possibilité pour l'utilisateur d'ajouter manuellement le titre.

# Remarques

Pour l’affichage des dates, nous avons d’abord voulu utiliser les Locales, mais il y avait des problèmes d’affichages chez certains d’entre nous, nous avons donc trouvé plus sûr de générer les strings des dates manuellement.

Nous avons également remarqué que notre base de données hébergée sur Atlas n’est pas accessible lorsqu’on est connecté sur le réseau de l’UCLouvain.

Sur l'un de nos ordinateurs, le bouton "play" et le défilement automatique des chansons ne fonctionnent pas sur firefox. Cela fonctionne sur les autres ordinateurs ou autres moteurs de recherches, cela doit donc être un bug indépendant de notre site.