Université Pierre Et Marie Curie

Rapport de projet

Développement d'Applications Réticulaires

Margarido Julien
Lima Gorito Félix
Nedjari Moussa

Responsable : M. Romain Demangeon

Septembre 2016 - Novembre 2016

Table des matières

1 Introduction

Dans le cadre de ce projet, il nous a été demandé de développer une application réticulaire (Web) de notre choix. Nous avons donc choisi de développer MuzikFinder. Cette application consiste en un moteur de recherche de musique. L'idée de départ de cette application est assez simple : si jamais il vous est déjà arrivé d'entendre une musique et de ne pas savoir son titre ou le nom de l'artiste l'ayant interprété, alors cette application est faite pour vous.
En effet, avec seulement quelques mots que vous avez retenu dans la chanson, MuzikFinder peut vous retrouver la musique tant recherchée. De plus, avec un système de recherche optimisée, vous obtiendrez votre réponse rapidement. La recherche repose sur la notion de "tag". Les tags sont très utilisés dans la recherche sur internet, par exemple pour des vidéos sur des plateformes de stockages et de visionnage dédiées (Youtube, DailyMotion, ...), et permet de référencer différents contenus et ainsi, pouvoir les retrouver rapidement via une recherche optimisé. Nous avons donc décidé d'utiliser cette notion de tags afin de référencer les musiques et ainsi pouvoir y accéder facilement et rapidement.

Passons maintenant à la description du projet. Dans la suite de ce rapport, nous allons vous décrire en détail les différentes techniques et technologies utilisées afin de mener à bien ce projet. Tout d'abord, nous allons présenter l'architecture générale du projet, et de la façon dont ont été agencé les différents composants de notre application.
Ensuite, nous allons présenter l'API utilisée pour ce site web. Puis nous allons nous concentrer sur la partie serveur du projet ainsi que tout le traitement fait en amont afin de satisfaire les promesses annoncées par cette application.
Nous décrirons également la partie client du projet, et tous les outils qui ont été utilisés afin de mettre en relation les traitements effectués côté serveur et les affichages côté client, sur le site web.
Enfin, nous parlerons des fonctionnalités qui auraient pu être développé en plus sur notre application.

2 Architecture du projet

Dans cette partie, nous allons nous concentrer sur l'architecture de notre application MuzikFinder. En effet, l'application disposant d'une architecture assez complexe utilisant plusieurs technologies différentes, il est bon de présenter tout ce qui a été utilisé afin de faire fonctionner de la meilleure des façons notre application.

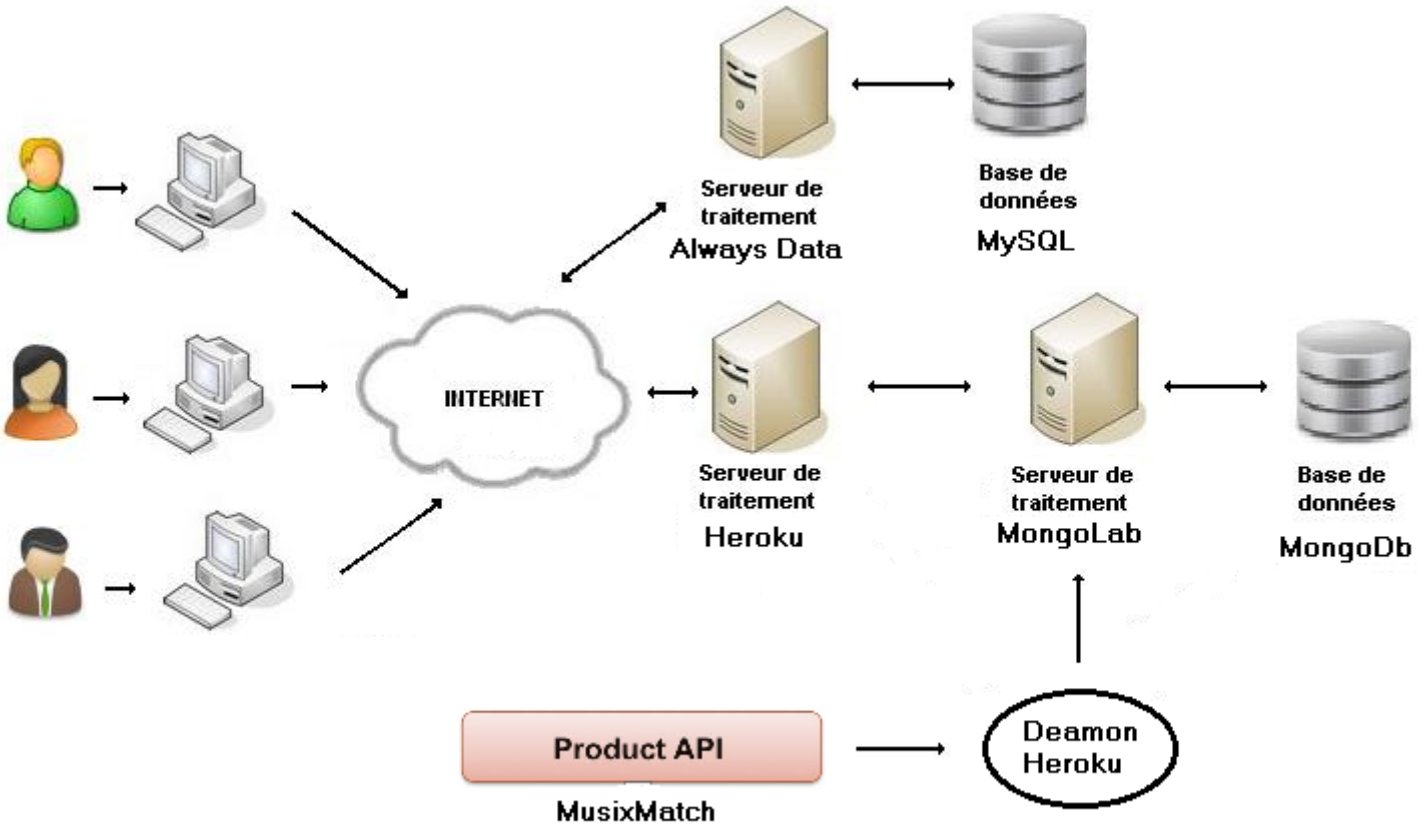


FIGURE 1 – *Architecture de l'application*

Notre application utilise deux types de bases de données *(cf. Figure 1)*. En effet, nous avons décidé d'utiliser une base NoSql (MongoDB) ainsi qu'une base de données relationnel SQL (MySQL). Ce choix a été motivé par le fait que nous avons à traiter différentes types de données. Il a été jugé préférable de stocker les informations sur les musiques dans une base MongoDB. En effet, disposant d'une grosse quantité de données à traiter et comme beaucoup de requêtes seront faites sur celles-ci, la base de données non relationnel, de part ses caractéristiques, nous est apparue comme la meilleure solution afin de stocker toutes les informations relatives aux musiques (nom, paroles, album, nom de l'artiste, genre...). Pour ce qui est de la base de données SQL, elle nous est utile afin de stocker les informations relatives aux utilisateurs. Elle nous sert également à gérer les connections et déconnections de ces derniers. Dans cette base seront stockés les noms d'utilisateurs, mot de passe, etc...
Nous allons maintenant parler de l'API utilisée dans cette application. Afin de pouvoir retrouver les informations sur les musiques, nous avons utilisée l'API REST MusixMatch. Elle nous permet d'avoir accès aux différentes informations sur les musiques telles que les paroles ou encore les artistes, à l'aide de simple requête GET et d'un token (obtenu à l'inscription). Ces données récupérées

via cette API seront stockées dans notre base MongoDB *(cf. Figure 1)*.
Dans la suite de ce rapport, nous parlerons de la façon dont s'effectue le stockage de ces données dans la base de données MongoDB et à quelle fréquence elles sont récupérées.
Ainsi, après avoir vu l'architecture globale de l'application, nous allons maintenant nous concentrer sur les différentes parties qui la compose.

3 Partie Serveur

Comme toute application web, on distingue deux parties : le serveur et le client. Le serveur nous servira à gérer toutes les données et à les traiter en conséquences suivant les demandes faites par l'utilisateur via le client. On va également y expliciter les traitements qui devront être effectués régulièrement afin de permettre une ergonomie la plus complète possible du site pour l'utilisateur.
Comme explicité précédemment, nous disposons de deux bases de données différentes : une base de données relationnel pour stocker les informations des utilisateurs et une base de données non relationnel pour tout ce qui concerne les musiques.
Le choix de ces deux types de bases de données a été motivé par la nature des données que nous avons à stocker et à gérer. En effet, l'API MusixMatch nous fournis toutes les informations nécessaires relatives aux musiques. Elle nous fournis une grande quantité de données textuelles et ces données n'ont pas vraiment de relation entre elles. L'accès à ces données se fait très régulièrement. Ainsi, stocker ces données dans une base NoSQL nous a paru efficace.
Pour ce qui est de la base de données MySql, pour le stockage des informations relatives aux utilisateurs, la raison est que nous n'avons que peu de données à stocker. En effet, seules les informations tels que le nom, prénom, adresse mail ainsi que l'historique des recherches y seront stockées. En conséquence, une base de données relationnel nous est apparu comme étant suffisante afin de gérer ces données. De plus, cela laisse la possibilité d'ajouter de nouvelles tables et de nouvelles informations liées aux utilisateurs.
Enfin, nous avons choisis d'utiliser une API externe afin de pouvoir recevoir des données régulièrement mises à jour par l'organisme la gérant et ainsi avoir une plus grande efficacité quand au service fournis à l'utilisateur de notre site.
Il est important de préciser que l'API MusixMatch nous fournit seulement 60% des paroles mais cela reste suffisant pour notre site : l'utilisateur cherchant souvent des lyrics apparaissant dans les paroles, celles-ci figurent dans la plupart des morceaux, malgré que nous n'ayons que 60% des paroles.

3.1 Architecture du serveur

Notre serveur se compose de plusieurs classes et packages. On pense notamment à tout ce qui nous permet de gérer et d'effectuer les traitements vers nos deux bases de données (nosql pour MongoDB et mysql pour la base de données relationnel). On voit également sur la *(cf. Figure 2)* la présence du package server qui contient les servlets et notre service principale. De plus, le package API nous permet de référencer notre Api MusixMatch et de pouvoir faire les appels à cette dernière afin de récupérer les musiques.

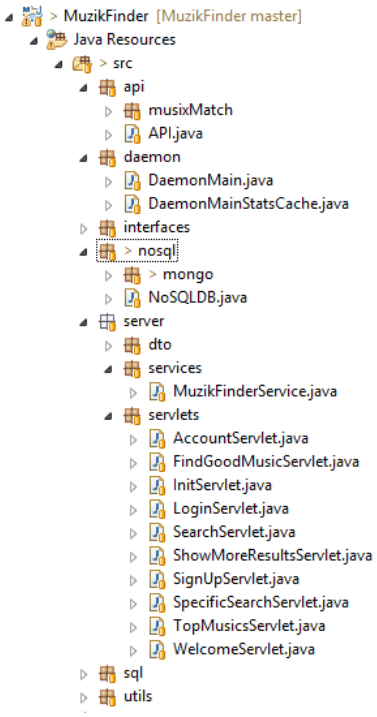


FIGURE 2 – *Architecture du serveur*

3.2 Hébergement et gestion des données

Nous allons concentrer dans cette partie sur l'insertion des données dans notre base MongoDB. Tout d'abord, il est important de préciser que l'hébergeur de notre site, Heroku, nous fournit la possibilité de lancer un **daemon**. Il s'agit d'un programme qui s'exécutera suivant une fréquence de temps prédéfinie, afin d'effectuer la tâche qui lui a été assigné.
L'un des objectifs de MuzikFinder est de fournir des résultats les plus optimaux possibles aux utilisateurs. Ceci passe donc par une mise à jour constante de nos données musicales. En conséquence, il fallait faire en sorte de toujours actualiser notre base de données. Ainsi, nous avons programmé que toutes les nuits, à 4 heures, un daemon s'exécuterais.
Celui-ci nous permet d'effectuer automatiquement des requêtes à l'API MusixMatch, pour récupérer les tops 100 de différents pays. Grâce à ce traitement, nous disposons d'une base de données regroupant plusieurs musiques de différents pays. À ce jour, notre base de données dispose de plus 4000 musiques différentes ainsi que plus de 500 albums.


```
{
    "_id": {
        "$oid": "5822709e47d238000496d1c1"
    },
    "musicId": "113810490",
    "lyrics": "Red light, green light\nEverybody take a shot\nRed light, green light\nGive me everything you got\nRed light, green light\n\nMister 305\nFlo Rida and
LunchMoney\nThree Miami boys, you know what time it is\n\nGive me the green light, oh yeah\nCause I'm ready to go\nLet's have a good time, let's go\nWhat you waiting
for?\nYou only got one life, one life\nAnd we gon' live it up\nSo give me the green light, give it to me\nCause I'm ready to go, oh, oh, oh\nWee-oh, wee-oh, wee-oh,
wee-oh, oh, oh\nCause I'm ready to go, oh, oh, oh\nWee-oh, wee-oh, wee-oh, wee-oh, oh, oh\nGive me the green light\n\nI got the green light mami and you know what that
means\nThat we could do anything, anything that you dream\nI wanna make you ooh, ah, until you scream\nAnd I practice what I preach, if you know what I mean\nI'mma
about that mami, yeah I got the key and I'mma lock that mami\nYeah, we can roll and I'mma rock that mami\nI do what I say and say what I mean, now let me jump in
between\nI'm getting loose in this thing, like OJ the Juice in this thing\nFeeling like Left Eye, boy, I burn the roof in this thing\nI got all the women getting naked,
feeling like Luke in this thing\nThink it's a game\nNow I got the green lit and the green right\nAll I need is, mami, for you to give me the green light",
    "artistId": "31993989",
    "artistName": "Pitbull feat. Flo Rida & LunchMoney Lewis",
    "albumId": "23866661",
    "albumName": "Greenlight",
    "musicName": "Greenlight",
    "language": "en",
    "spotifyId": "",
    "soundCloudId": "",
    "musicGenre": "Pop"
}
```


FIGURE 3 – *Exemple de document pour les musiques*

Nous allons maintenant parler de la notion de tags qui a été introduite précédemment. Les tags nous sont très utiles pour la recherche de musique. En effet, ils servent à référencer des musiques. La méthode de construction des tags est assez simple. On sélectionne les mots, dans les paroles des musiques, susceptibles de fournir une indication vers cette même musique. L'idée est de se débarrasser des mots fournissant peu ou trop d'informations tels que les prépositions ou les pronoms. Ainsi, dans notre base de données MongoDB, nous disposons, en plus des musiques, des tags les référençant. Nous disposons en ce moment même de plus de 14000 tags dans notre base de données.

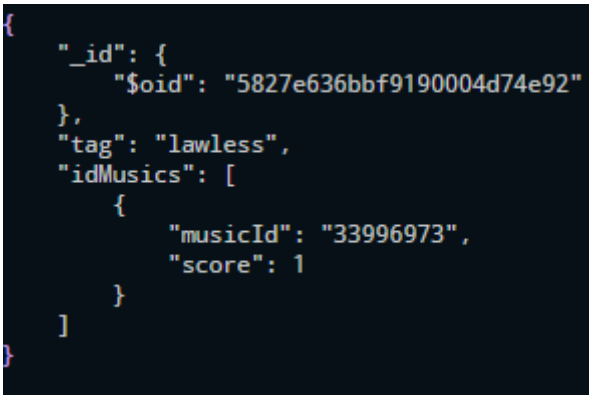


FIGURE 4 – *Exemple de document pour les tags*

Pour chacun des tags présents dans notre base de données, nous lui référençons plusieurs musiques, via leur identifiant, ainsi qu'un score correspondant au nombre de fois où le tag se répète dans les paroles de la musique en question *(cf. Figure 4)*. Cette structure des documents présents dans notre base Mongo nous sera très utile lors de la recherche d'une musique. Il est importante de souligner que le daemon, qui récupère les musiques via l'API MusixMatch, possède également la tâche de construire ces tags (en retirant les mots non désirés, à l'aide d'un Parser que nous avons créés) au fur et à mesure. Les tags étant dépendants des musiques que l'on récupère, il nous a paru judicieux d'effectuer ce traitement de cette manière. Le but est de toujours avoir les meilleurs informations possibles dans les tags pour que la recherche soit la plus correcte possible et retourne une liste de musique cohérente avec la recherche de l'utilisateur.

3.3 La recherche de musique

La recherche de musique est la fonctionnalité principale de notre site MuzikFinder. Ainsi, il a fallu l'optimiser au maximum pour offrir à l'utilisateur, à la fois un résultat cohérent avec sa recherche mais également une réponse rapide.
Dans notre cas, nous avons décomposé la recherche en deux étapes différentes. Tout d'abord, on traite les mots entrés par l'utilisateur comme une suite de mots. C'est à dire que l'on recherche directement dans les paroles des musiques (à notre dispositions dans notre base Mongo) celles qui contiennent la suite formée par ces mots. Cette recherche nous permet d'éviter des traitements inutiles. Elle repose sur les capacités de recherche très performantes de MongoDB et sur une regex que nous avons écrites (pour ne pas prendre en compte des caractères séparant ces mots, comme des virgules ou des sauts de lignes). Nous effectuons cette recherche pas uniquement sur la suite de mots exactes entré par l'utilisateur, mais sur les possibles suite de mots jusqu'à ce que l'enchaînement est une taille supérieur à 3. Si l'on traite cette recherche sur l'exemple suivant : "i know you love me", on effectuera la recherche suivante :

— i know you love me

Si cette recherche retourne un unique résultat, on arrête et l'on retourne la musique trouée. Sinon on effectue les recherches suivantes :

- i know you love
- know you love me

Si ces 2 recherches retournent un résultat, on le retournera. Sinon on effectue encore une dernière recherche :

- i know you
- know you love
- you love me

De même, si ces 3 recherches retournent un ou plusieurs résultats, ils seront retournés à l'utilisateur. Si cette recherche ne retourne aucun résultat, nous passons à un autre type de recherche, puisque nous avons atteint la limite de 3 mots que l'on a définie, pour passer à la recherche par "tag".

La recherche par "tag", quand à elle, ne s'occupe en aucun cas de l'enchaînement des mots mais uniquement de rechercher chaque mot de manière indépendante dans une collection "TAGS".
Pour chaque mots entrés par l'utilisateur, on va chercher dans la collection l'ensemble des musiques ayant ce mot dans ces paroles et récupérer le score d'un mot dans chacune de ces musiques. Le score d'un mot dans une musique correspond aux nombres d'occurrences de celui-ci dans la musique. Si un mot apparaît 8 fois dans une musique alors il aura un score de 8.
Après avoir récupéré l'ensemble des musiques contenant les mots entrés par l'utilisateur, on va classer les musiques en fonction du nombre de "tag" présent dans chaque musique.
Reprenons l'exemple de la recherche pour "i know you love me" pour mieux comprendre :

- Musique A (["i", score=0],["know", score=5],["you", score=0],["love", score=3],["me", score=0])
- Musique B (["i", score=0],["know", score=0],["you", score=0],["love", score=10],["me", score=0])
- Musique C (["i", score=0],["know", score=0],["you", score=0],["love", score=13],["me", score=0])
- Musique D (["i", score=0],["know", score=6],["you", score=0],["love", score=10],["me", score=0])

Les musiques A et D ont des mots qui reviennent 2 fois contrairement aux musiques B et C qui ne contiennent qu'un seul mot. On retournera en premier les musiques A et D, D en premier puisqu'il possède un score total supérieur à A. Enfin, on effectue le même traitement entre C et B. Ainsi, on renverra dans l'ordre : D, A, B, C

3.4 Les statistiques des recherches

Une autre fonctionnalité fournie par MuzikFinder est la mise en place de statistiques sur les recherches de musiques. Notre application collecte des données sur les recherches utilisateurs de manière anonyme afin de fournir des tops musiques par tranche d'âge et pour toutes tranches d'âges confondues. Le principe en lui même est simple. Lorsqu'un utilisateur effectue une recherche et obtient le résultat qu'il voulait, il le notifie et cela effectue automatiquement un ajout ou une mise à jour dans notre base de données MongoDB. Nous avons décidé de nous limiter à l'affichage des tops par semaine. Ainsi, chaque semaine, de nouvelles informations seront stockés et l'utilisateur aura accès aux informations les plus récentes sur les tops musiques par tranche d'âge et pour toutes tranches d'âges confondus. La structure du document stockant ces informations est assez complexe *(cf. Figure 5)*. On dispose d'un champ **weeks** permettant de distinguer les différentes semaines. On dispose également d'un champ **ageRanges** qui sert à stocker les différentes informations pour chacune des tranches d'âges que nous utilisons dans notre application web.

On dispose de 4 tranches d'âges différentes :

— de 0 à 17 ans
— de 18 à 24 ans
— de 25 à 49 ans
— les 50 ans et plus

```
{
    "_id": {
        "$oid": "5827cb51707a7d102c490d0d"
    },
    "weeks": "47-2016",
    "ageRanges": [
        {
            "age": "-25",
            "musics": [
                {
                    "musicId": "105666331",
                    "score": 1
                },
                {
                    "musicId": "114440428",
                    "score": 1
                },
                {
                    "musicId": "114440416",
                    "score": 1
                },
                {
                    "musicId": "113673904",
                    "score": 1
                },
                {
                    "musicId": "115002116",
                    "score": 2
                },
                {
                    "musicId": "115754731",
                    "score": 1
                }
            ]
```


FIGURE 5 – *Exemple de document pour les statistiques*

Ensuite, pour chacune des tranches d'âges, on dispose d'une liste de documents référençant les musiques, via leur identifiant, avec un score correspondant au nombre de fois où un utilisateur de cette tranche d'âge a indiqué que cette musique était bien celle qu'il a recherché et trouvé.

Cependant, dans l'état actuel des choses, lorsqu'un utilisateur demandera à recevoir les informations sur les tops de la semaine, il faudra effectuer un traitement côté serveur au moment de la demande de l'utilisateur pour obtenir cette information. Cela prendrait trop de temps et ne serait pas optimal pour l'utilisateur de recalculer ces tops. Par conséquent, nous avons décidé de créer une nouvelle collection dans notre base MongoDB qui servira de cache. En effet, un autre deamon est utilisé afin d'effectuer ce calcul des tops musiques de la semaine suivant les informations stockées dans la collection présentée précédemment. Régulièrement, il récupérera le contenu de notre collection stockant les statistiques et remplira la collection relatives aux tops des musiques de la semaine pour chaque tranche d'âge et pour toutes les tranches d'âges confondues. Le principe du traitement de ces informations est simple. Il consiste à trier les musiques suivant leurs scores pour chaque tranche d'âge présentes dans la première collection stockant les statistiques. De même pour le classement général des musiques.

Ainsi, lorsque l'utilisateur demandera à connaître les tops musiques de la semaine, les informations seront récupérées dans le cache des statistiques et lui seront affichées, sans calcul lourd. Grâce à ce traitement des données côté serveur, l'affichage de ces informations se fait beaucoup plus rapidement que s'il fallait, à chaque demande de l'utilisateur, retraiter les informations. Maintenant que nous avons découverts les différentes fonctionnalités que propose le site MuzikFinder, nous allons nous concentrer sur la gestion des comptes et données des utilisateurs.

FIGURE 6 – *Exemple de document pour le cache des statistiques*

3.5 Les données des utilisateurs

Comme expliqués précédemment, les données des utilisateurs sont stockées dans une base de données relationnel MySQL, hébérgée par AlwaysData. Dans cette base de données, y seront référencées les différentes données relatives aux utilisateurs, telles que les pseudos, les mots de passe, les adresses mails mais également l'historique des recherches des utilisateurs. Plusieurs fonctionnalités seront mises à disposition des utilisateurs en plus de celles explicitées précédemment. En effet, le site MuzikFinder possédant une légère dimension social, les utilisateurs pourront se créer un compte, se connecter à leur profil et retrouver l'historique de leurs recherches. Ils auront également la possibilité de se déconnecter et de supprimer leur compte.

4 Partie Client

Pour la partie client, nous avons fait le choix de rédiger chaque page dans une JSP. En effet, les JSP s'interfaçant parfaitement côté serveur avec les servlets, elles nous permettent de manipuler des données (qui peuvent être souvent conséquentes), de formater l'affichage HTML de la page, sans trop alourdir le code.
Notre client est formé de plusieurs composants :

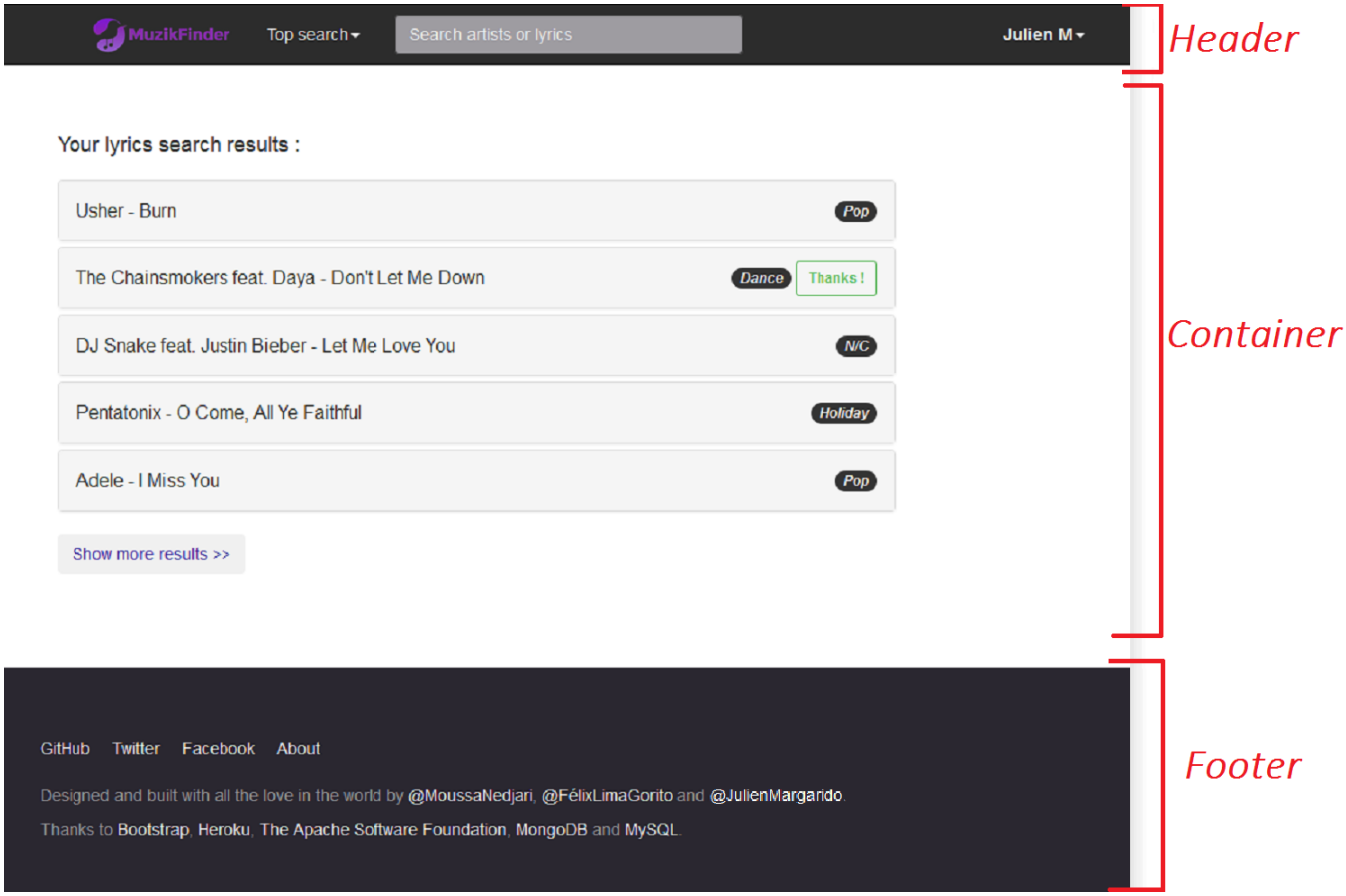
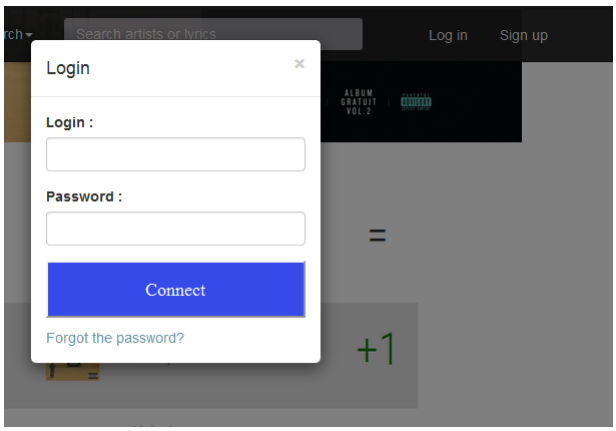


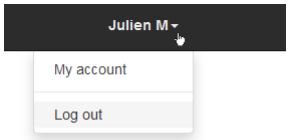
FIGURE 7 – *page de recherche*

Tout le contenu de nos pages est mis dans la partie "Container".

Un "Header" est constamment visible, même si l'on déroule la page. Il permet d'avoir accès aux pages principales de notre application, sans devoir repasser par une page d'accueil, en un clic. Il est composé (dans l'ordre) du logo, d'une liste déroulante d'affichage de "Tops", de la barre de recherche, et enfin d'actions sur des comptes. Ces actions sont de 3 types :

- Soit l'utilisateur est déjà connecté, alors son nom apparaît, accompagné d'une liste déroulante permettant d'accéder à la page de son compte mais aussi de se déconnecter.
- Soit l'utilisateur n'est pas connecté, et donc peut choisir de se connecter ou de s'inscrire.





Enfin, un “Footer” est visible en bas de chaque page, et permet d’obtenir diverses informations sur le site.

Les autres composants sont :

- les 2 “Modals”, ou “fenêtres popups” permettant l’inscription ou la connexion.
- le carousel de la page d’accueil, permettant l’affichage de plusieurs bandeaux d’accueil.

Notre site est composé de 4 pages principales :

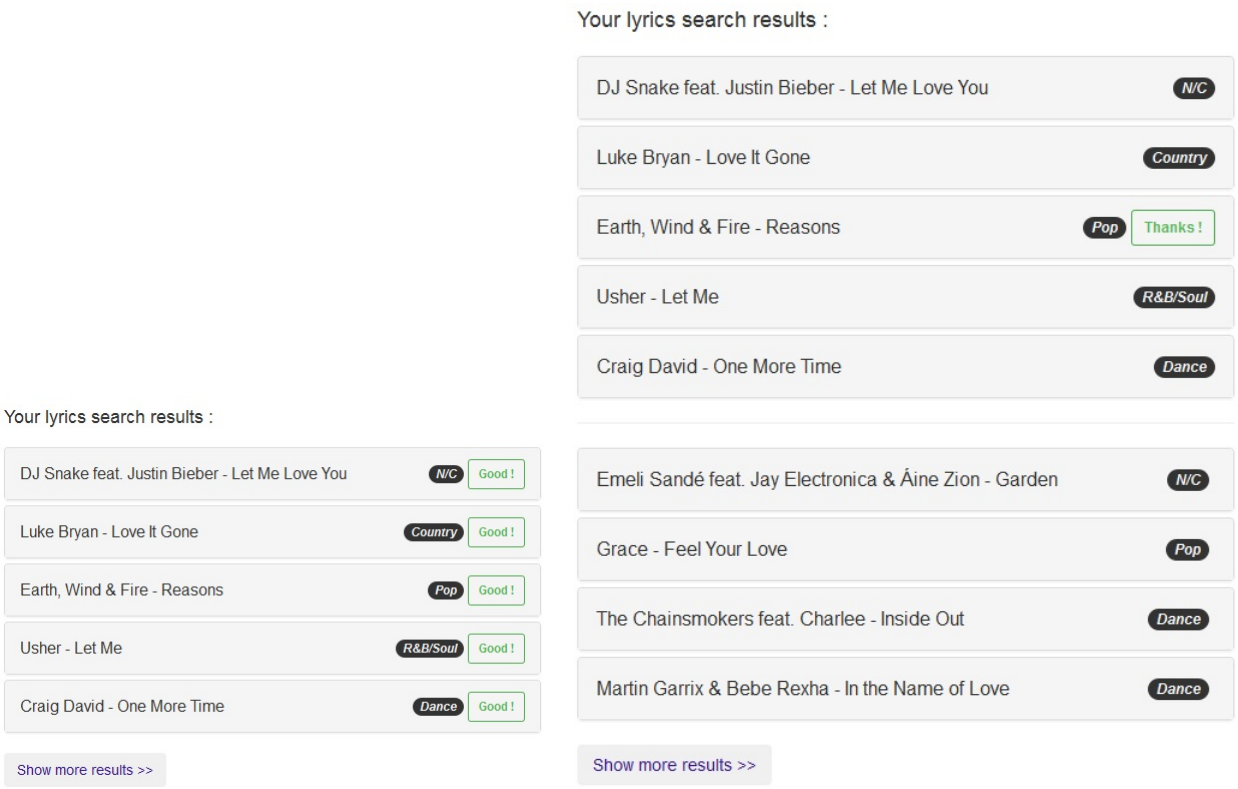
- La page d’accueil “index.jsp” : Elle permet de donner un bonne impression dès l’ouverture du site, et attire l’oeil de l’utilisateur sur les fonctionnalités que l’on peut lui apporter.
- La page de gestion de son compte utilisateur “myAccount.jsp” : Elle permet de modifier ses informations personnelles (mot de passe, mail, date de naissance, . . .), de visualiser l’historique de ses recherches, mais aussi de supprimer son compte utilisateur.
- La page de recherche par artistes ou par musiques : Elle permet d’afficher les résultats de la recherche effectuée dans la barre prévue à cet effet. Cette page propose un premier choix de musiques, triées par pertinence, avec la possibilité d’obtenir des informations sur chaque musique (album, paroles, . . .) et de confirmer par un bouton qu’un résultat est le bon. De plus, l’utilisateur à la possibilité d’obtenir plus de résultats, et ceci en lui proposant des résultats de moins en moins pertinent, mais (de manière générale) très nombreux.
- La page d’affichage des tops recherches : Elle permet de montrer les morceaux les plus recherchés cette semaine, par tranche d’âge, mais aussi pour toutes les tranches d’âges confondus.

4.1 Outils

Dans le but d’obtenir une expérience utilisateur positive (fluidité, adaptabilité, effets,...), et dans le but de nous familiariser avec des outils très utilisés dans le monde du développement web, nous avons fait le choix d’utiliser et intégrer à notre projet diverses bibliothèques et collections d’outils.

4.1.1 JavaScript/JQuery

Tout d’abord, nous avons beaucoup utilisé la bibliothèque Jquery et quelques fonctions écrites en Javascript pour appliquer des effets ou des changements sur certains éléments d’une page. Parmi nos différentes utilisation, nous pouvons prendre en exemple la page des résultats de recherche :



Ici, nous utilisons JQuery pour, lorsqu'un des boutons "Good !" est cliqué, tous les autres (ainsi que les prochains qui devront apparaître) soient sélectionnés et désactivés. Cela permet de n'effectuer qu'un seul vote d'un utilisateur.

4.1.2 Ajax

Pour éviter des rechargements de page (si un champ est mal rempli par l'utilisateur par exemple) et pour améliorer l'expérience utilisateur, nous avons implémenté divers appels ajax dans notre site, au lieu d'une soumission de formulaire. Ceux-ci permettent d'effectuer une requête HTTP paramétrable (méthode, type de retour, paramètres,..) et d'obtenir différents callbacks, permettant de savoir la bonne réception de notre requête, son bon traitement, mais aussi d'obtenir des données fournies par la servlet. Nous avons implémentés 7 différents appels Ajax dans notre site (auto-complétion dans la barre de recherche, récupération des musiques suivantes, modification de son compte, ...), mais nous allons prendre exemple de l'appel effectué lors de l'inscription :

Ici, lorsque l'on détecte que le bouton de soumission de l'inscription est cliqué, nous effectuons une requête POST au serveur (SignUpServlet), en lui passant chaque champs du formulaire. Ceux-ci vont être analysés (conformité, non existence d'un utilisateur au même login, etc...) et un retour sera effectué au client.

```
//checks for the button click event
$("#connectButtonSignup").click(function(e) {
    e.preventDefault();
    var laddaButton = Ladda.create(this);
    laddaButton.start();
    $.ajax({
        type : "POST",
        url : "SignUpServlet",
        data : {
            username : $("#usernameSignup").val(),
            password : $("#passwordSignup").val(),
            mail : $("#mailSignup").val(),
            day : $("#daySignup").val(),
            month : $("#monthSignup").val(),
            year : $("#yearSignup").val()
        },

        //if received a response from the server
        success : function(data, textStatus, jqXHR) {
            laddaButton.stop();
            if (data.success) {
                location.reload();
            } else { //display error message
                $("#strongErrorMessageSignup").text(data.message);
                $("#errorMessageSignup").show();
            }
        },
        error : function(jqXHR, textStatus, errorThrown) {
            alert("No response from the server");
            laddaButton.stop();
        },
        always : function() {
            laddaButton.stop();
        }
    });
```

FIGURE 8 – Appel Ajax à la servlet SignUp

Celui-ci, suivant si le serveur lui indique le succès ou non de la soumission :

- soit il affichera un message d'erreur (fournis par le serveur) à l'utilisateur
- soit il rechargera la page, ce qui aura pour effet d'afficher à l'utilisateur un header en mode "utilisateur connecté".

Ces appels se font de manière rapides (suivant la qualité de connexion réseau) et invisibles. De plus, l'ajout de boutons de la bibliothèque "Ladda" (voir http ://lab.hakim.se/ladda/) permet à l'utilisateur d'indiquer qu'une action est en cours.

4.1.3 Bootstrap

Enfin, nous avons intégré le célèbre framework CSS Bootstrap 3. Celui-ci nous apporte une palette de composants que nous avons intégré à notre site (navbar, buttons, carousel, modals, thumbnail,...), tous possédant des attributs CSS et des fonctionnalités en Javascript, permettant une fluidité et une ergonomie très agréable. De plus, avec l'aide de ce framework, nous avons pu faire en sorte que tout notre site soit "Responsive", et ainsi qu'il puisse s'adapter à différentes plateformes sur lesquels nous l'avons testé (PC, smartphones, tablettes et même TV).

4.2 Gestion des cookies

Au départ, pour savoir si un utilisateur étais déjà authentifié ou non, nous avions mis en place un système d'enregistrement et de récupération de l'identifiant de l'utilisateur dans la session, côté serveur.

Cependant, après réflexion et quelques recherches, nous en avions déduis que ceci était mauvais car :

- X millions d'utilisateurs simultanés reviendrait à stocker X millions d'identifiant côté serveur.
- En cas de répartition de l'application sur plusieurs serveurs, l'identifiant serait soit perdu, soit à dupliquer.

La solution simple que nous avons trouvée est d'insérer cet identifiant dans les cookies du client. Ainsi, cette donnée n'est pas stocker sur notre serveur, et est accessible et modifiable à chaque requête. Nous avons fait le choix de maintenir en vie 2 heures ce cookie, mais repousser ce délai à chaque action de l'utilisateur. De ce fait, au bout de 2 heures d'inactivité, l'utilisateur devra obligatoirement se reconnecter s'il désire continuer d'utiliser les fonctionnalités de notre site.

5 Extensions possibles

Dans cette partie, nous parlerons des fonctionnalités que l'on aurait pu ajouter, avec plus de temps. Tout d'abord, la première extension que serait l'ajout d'un lecteur permettant à l'utilisateur d'entendre les différentes musiques trouvées, et pas uniquement de voir les paroles des musiques.

Une autre fonctionnalité serait d'ajouter des statistiques non plus uniquement sur l'âge des utilisateurs, mais également sur le type des musiques (pop, rock, hip hop, ...), ce qui permettrait d'avoir des statistiques plus affinées : connaître le genre de musique recherché par les moins de 18 ans, le genre de musique le plus recherché sur le site, ...

On aurait également pu améliorer l'algorithme de recherche, en ajoutant un critère de recherche propre au traitement, qui se baserait également sur le classement des recherches du moment. En effet, plus une musique est populaire, plus la probabilité qu'elle soit recherchée est grande. Un autre moyen d'améliorer la recherche serait d'ajouter des critères de recherche à l'utilisateur, tels que le nom d'un artiste participant à la musique (featuring), sélectionner le style de la musique, ...

Actuellement notre site est totalement en anglais, mais on pourrait externaliser tous les textes dans des fichier properties, pour que le site s'adapte à l'utilisateur. Si l'utilisateur utilise un navigateur en français, on chargerait alors le fichier properties français pour mettre le site en français.

6 Points positifs et négatifs

Nous allons maintenant citer, le plus objectivement possible, les différents points forts et points faibles de notre application.

6.1 Point positifs

- Peu couplé aux différentes bases de données et API externes : l'architecture mise en place sur ce projet permet que si un problème survient avec une base ou une API, en quelques heures de développement (simplement en recodant une ou plusieurs classes de service) le système pourra s'adapter à la nouveauté. Par exemple, il suffira de changer une ligne dans le constructeur de la classe API et de coder une seule classe SpotifyService (et d'éventuels classes utilitaires) pour passer d'un système utilisant MusixMatch à Spotify.

— Code documenté (commentaires, javadoc), noms parlants correspondant au contexte (classes, méthodes, attributs, fichiers,...).
— Facilité pour ajouter de nouveaux pays : Actuellement, le daemon d'ajout des musiques dans notre base passe sur 7 Pays différents ("us", "gb", "au", "fr", "ca", "be"), tous francophones ou anglophones. Ceci est dû au fait que nos dictionnaires d'exclusion des mots soient en anglais et en français pour le moment. Mais il suffirait d'ajouter un nouveau dictionnaire pour ajouter une nouvelle langue, et donc pouvoir intégrer de nouveaux pays, et donc de nouvelles musiques.
— Toutes les préférences et choix numériques de conception se trouvent dans une classe MuzikFinderPreferences.
— La recherche d'une musique sur notre site est plus précise que la recherche par paroles de MusixMatch, l'API que nous utilisons (qui dispose d'un système de recherche similaire), malgré que nous disposons des mêmes données qu'eux.

6.2 Point négatifs

— Nous ne pouvons obtenir la totalité des paroles, dû à une contrainte de MusixMatch qui nous fournit uniquement la moitié des paroles dans la version gratuite. Solutions : obtenir la version payante à 25.000 dollars, ou trouver une autre API gratuite et similaire.
— Entièrement dépendant de l'hébergeur Heroku pour le moment. Si celui-ci à un problème ou décide de bannir notre compte, alors l'application deviendrait inaccessible. Solution : Héberger l'application sur nos propres serveurs.

7 Conclusion

Pour conclure, nous pouvons dire que ce projet nous a beaucoup apporté dans le domaine du développement web. Nous avons rencontrés plusieurs difficultés tout au long de ce projet. Tout d'abord, trouver une idée pouvant répondre à un besoin précis fut assez compliquée. Ensuite, après avoir identifié la direction que prendrait notre projet, il nous a fallu mettre en place les différents moyens qui nous permettraient de mener à bien le projet, en en commençant par chercher, tester et valider une API gratuite nous fournissant les données nécessaires, ainsi qu'un hébergement simple, permettant de déployer et de tester rapidement notre application.

Nous tenons à remercier nos enseignants, Monsieur Romain Demangeon, Monsieur Gregoire Henry et Monsieur Benjamin Canou, pour les connaissances qu'ils nous ont apportés et pour leur aide durant l'élaboration du projet.