Rapport Projet Industriel

Arthur FLAMBEAU
Yohann VAUBOURG

Réalisation d'une application de streaming musical afin de tester différents algorithmes de recommandation

M2 MIAGE SID

2016 - 2017

Encadrants :
Geoffray BONNIN
Amaury L'HUILLIER
Sylvain CASTAGNOS
Equipe KIWI

Entreprise d'accueil : Loria

Table des matières

Remerciements

Nous tenons à remercier tout particulièrement Mr Sylvain CASTAGNOS, pour nous avoir donné l'opportunité d'effectuer notre projet industriel au sein du Loria.

Nous remercions aussi Mr Geoffray BONNIN et Mr Amaury L'HUILLIER, pour leur accompagnement tout au long de ce projet industriel, pour leurs conseils et leur écoute, ainsi que leur chaleureux accueil.

Enfin, nous remercions Mr Khalid Benali, responsable du Master 2 SID, sans qui cette opportunité de stage ne serait pas possible.

Introduction

Notre projet industriel se place dans le cadre de notre formation de Master deuxième année SID (Systèmes d'Informations Distribués). Il a pour but de nous mettre en situation de travail en entreprise un jour par semaine, et ce, pendant 12 semaines. Au total nous avons donc 12 jours pour réaliser le projet qui nous est confié pour ce projet industriel. Il s'agit du développement d'une application web de streaming musical, qui a pour but de permettre aux chercheurs de tester différents algorithmes de recommandation de musique. Ce projet est intéressant car il nous permet de mettre en application les différents éléments vus en cours, tant sur la conception que sur la réalisation. La principale difficulté réside dans la durée. En effet bien que ce projet industriel soit d'une durée de 12 semaines, seulement un jour par semaine nous est alloué pour y consacrer du temps, ce qui implique une gestion optimisée du temps de travail. Au terme du temps qui nous a été imparti, nous avons finalement pu livrer une application fonctionnelle, permettant l'intégration de différents algorithmes et une gestion complète des actions utilisateurs en base de données, ainsi qu'un site web permettant l'écoute de musique via deux modes de navigations différents, et le choix des recommandations faites par les algorithmes. Cette application vous sera présentée tout au long de ce rapport.

Le LORIA, laboratoire de recherche en informatique, est constitué de nombreuses équipes travaillant dans différents domaines de l'informatique. L'équipe KIWI est une des équipes de recherche et plusieurs de ses membres travaillent dans le domaine de la recommandation musicale. L'équipe a donc imaginé un certain nombre de modèles d'apprentissage pour modéliser automatiquement les préférences des utilisateurs dans le but de fournir des recommandations de musiques, d'artistes ou de playlists.

Afin de mettre en œuvre et de tester certains de ces modèles, l'équipe avait besoin d'un support, une plateforme d'écoute de musique en ligne, permettant de récolter des données utilisateurs et de tester les différents algorithmes de recommandation.

Une première version de plateforme musicale a été réalisée en 2014/2015 par deux groupes d'étudiants lors d'un projet industriel, puis améliorée par Myriam DELARUELLE lors de son stage de fin d'année. Il a de nouveau été complété dans une troisième version par un nouveau groupe d'étudiants lors de leur projet industriel en 2015/2016.

Notre rôle, pour ce projet industriel, est de mettre en place une plateforme qui soit le plus rapidement opérationnelle, les versions précédentes n'ayant pas pu être exploitées pour diverses raisons techniques. L'objectif final est de permettre la récolte de données par les chercheurs afin d'éprouver et de mettre en œuvre les différents algorithmes mis en place par l'équipe KIWI.

Contexte

L'application à développer doit permettre de collecter des informations assez précises concernant les habitudes des utilisateurs. Pour ce faire, il faut des données complètes concernant les musiques écoutées. Les versions précédentes se sont servies de différentes API Web :

- Première version : Groove Shark
- Deuxième version : Napster / Rhapsody
- Troisième version : YouTube

Ces versions ne sont actuellement plus fonctionnelles dû à l'évolution des API Webs, aucune ne peut donc être exploitée en l'état. Ceci est dû principalement à des problèmes techniques : la complexité des applications pour les rendre aussi génériques et modulables que possible, combinée à la rapide évolution des technologies et des API exploitées ont rendu les choses impossibles.

Les objectifs du projet ont donc été redéfinis, et suite à une réunion avec Mr CASTAGNOS, Mr BONNIN et Mr L'HUILLIER, il a été convenu que nous devrions mettre en place le plus rapidement possible un code fonctionnel permettant aux utilisateurs d'écouter une musique. Par la suite l'application serait amenée à évoluer mais il s'agissait de l'objectif principal.

Afin d'explorer les différentes possibilités qui s'offraient à nous pour réaliser une application de lecture d'une musique dans les plus bref délais, il nous était nécessaire d'étudier l'application précédemment développée par les étudiants. Le frontend de la v3 était à revoir complétement, et le backend était quant à lui quasiment fonctionnel (remplacement d'une des API Web utilisée), mais incomplet (pas de possibilité de mettre en place des évaluations, seulement d'utiliser l'interface de lecture de musique).

Face à ce constat, il était possible soit de reprendre la v3 (compléter le backend et refaire le frontend), soit d'abandonner l'existant. D'un commun accord avec nos encadrants, et afin de s'assurer que le résultat serait exploitable par l'équipe KIWI, la seconde solution a été choisie.

Choix des technologies

Etant donné que lors de la phase de réflexion sur l'existant nous avons eu à manipuler Angular, nous avons dans un premier temps décidé de rester dans la continuité et d'utiliser la nouvelle version de ce framework. Angular 2 n'est autre que l'évolution du framework JavaScript développé par Google. Cette nouvelle version voit le jour dans l'optique d'améliorer les quatre défauts principaux de la version précédente. Il s'agit d'un tout nouveau framework qui est très différent du précédent. L'apparition du TypeScript, qui est le langage de développement recommandé, permet une auto complétion plus poussée, une syntaxe plus stricte et un typage statique. Il permet de plus de rendre le code plus lisible et maintenable.

La documentation disponible à l'heure actuelle sur Angular2 n'était pas assez fournie et ralentissait notre progression. De plus, nous avons reçu sur la fin d'année des cours sur Angular1, et nous avons pris la décision de repasser sur cette technologie avec l'accord de nos encadrants, afin de nous assurer de pouvoir rendre le travail qui nous était demandé.

Comme convenu avec l'équipe de recherche se chargeant de l'implémentation des algorithmes, le back-end serait en Java. Etant donné que notre front-end requiert déjà divers appels à des API REST et que les algorithmes des chercheurs sont implémentés en Java, nous avons choisi de procéder de même avec notre back-end et pour ce faire nous utilisons spring-boot, qui est un framework Java que nous avons eu l'occasion d'étudier en cours.

Afin de gérer la persistance de nos données et de coupler le tout à notre spring-boot, nous avons fait le choix d'hibernate, étant de même un outil déjà étudié.

Git est un système de contrôle de version décentralisé, réalisé dans l'optique de gérer aussi bien des petits que des gros projets avec rapidité et efficacité. Il nous permet de mieux gérer notre code et de donner aux encadrants la possibilité de voir l'évolution du code.

Conception

Présentation de l'application

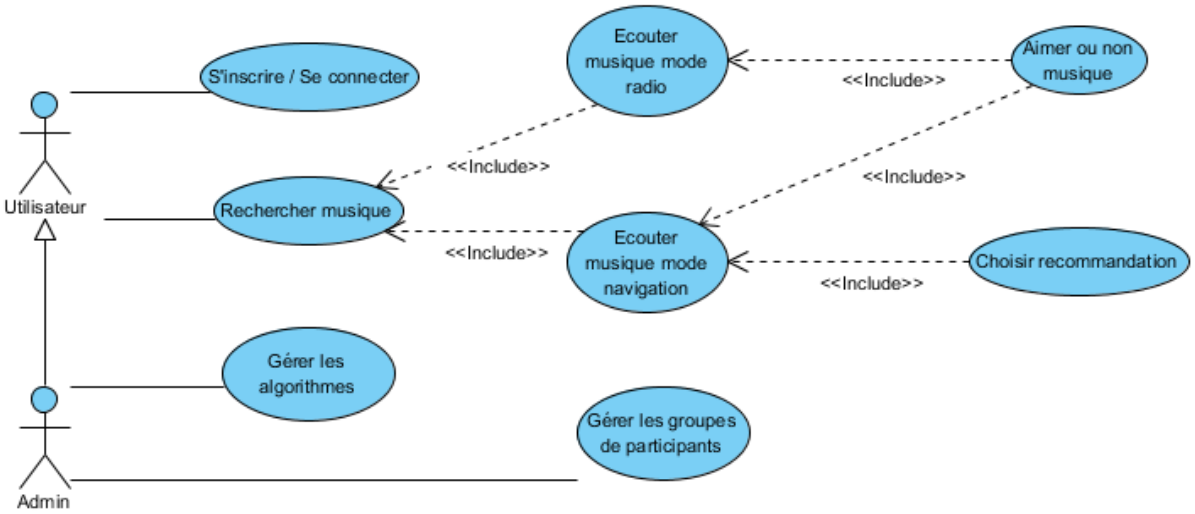

Figure 1 : Diagramme de cas d'utilisation

Le diagramme des cas d'utilisation en Figure 1 présente toutes les fonctionnalités que notre application doit proposer à un utilisateur. Dans un premier temps, un utilisateur quelconque doit pouvoir s'inscrire et se connecter à l'application.

Ensuite, l'utilisateur doit pouvoir rechercher une musique et choisir quel mode de lecture il veut utiliser (mode radio ou mode navigation).Ces modes seront expliqués plus en détail par la suite. De plus, pour chaque musique qu'il écoute l'utilisateur doit pouvoir dire s'il aime la musique ou non peu importe le mode qu'il a choisi. Lorsqu'il utilise le mode navigation l'utilisateur doit pouvoir sélectionner une musique recommandée par les algorithmes.

Enfin, un utilisateur de l'application peut être un chercheur, ce dernier doit pouvoir se servir de l'application comme un utilisateur normal mais aussi gérer les algorithmes que l'application utilise, c'est à dire ajouter, modifier et choisir les algorithmes actifs.
Outre ces fonctionnalités, l'application doit aussi permettre la collecte d'informations utilisables par les algorithmes. C'est pourquoi, toutes les actions d'un utilisateur sont sauvegardées (entre autres, choix d'une recommandation, aimer ou non une musique, passer à la musique suivante).

Modélisation de la base de données

Voici une version simplifiée de notre diagramme de classe représentant la première version, que nous avons fait valider par nos encadrants, de la base de données de notre application. Ce schéma nous a servi de base pour le développement de notre API, il a donc évolué tout au long du projet pour mieux répondre aux besoins.

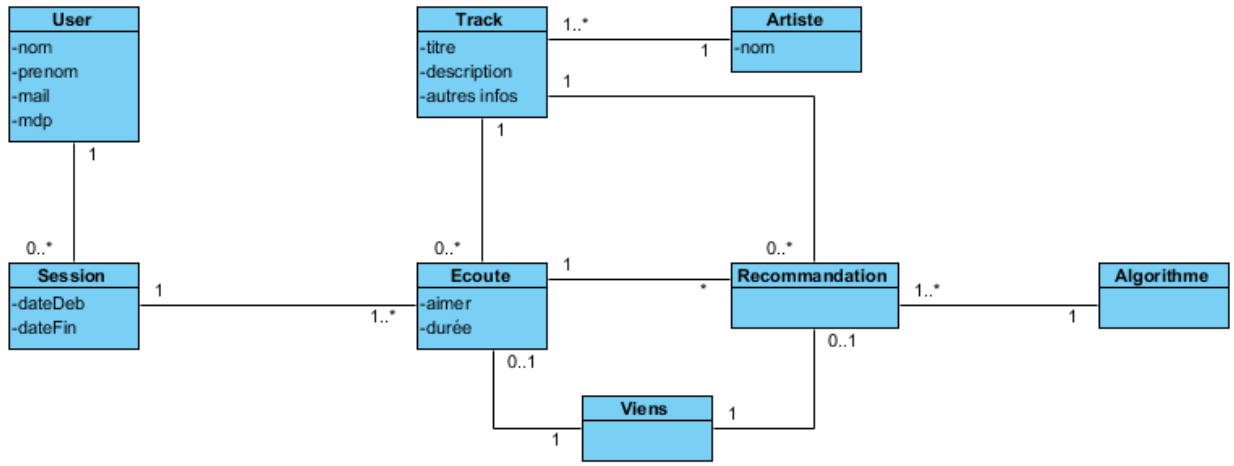

Figure 2 : Première version du diagramme de classe

Comme le montre le diagramme en figure 2, notre base de données se découpe en trois parties distinctes. La première concerne les données musicales provenant de l'extérieur et qui ne changent jamais ou très peu. En effet, l'application a besoin de données sur les artistes et la musique venant d'API externes, comme le genre, l'acoustique, la « dansabilité » (...). La deuxième permet d'identifier l'utilisateur et de définir ses habitudes. La dernière sert à stocker les comportements des algorithmes des chercheurs. L'utilité de l'ensemble des classes de ces parties va maintenant être détaillée.

Les tables track et artiste sont donc remplies grâce aux différentes API utilisées (LastFM, Spotify et YouTube) afin de limiter les appels à ces dernières. Nous stockons toutes les informations nécessaires aux algorithmes mais aussi l'identifiant YouTube de la musique permettant de la jouer.

Concernant la gestion des utilisateurs, nous avons une table permettant de stocker toutes les informations propres à l'utilisateur, mais aussi une table session permettant de stocker toutes les musiques qu'il a écouté lors d'un passage sur le site. De plus, comme une session est datée, on peut connaître quel style de musique l'utilisateur écoute à quel moment de la journée.

La table écoute, elle, permet de savoir si l'utilisateur a aimé une musique et combien de temps il l'a écoutée. On pourrait se dire que ces informations sont redondantes, car si un utilisateur écoute peu de temps une musique c'est qu'il ne l'aime pas, néanmoins, il peut très bien aimer la musique mais ne pas avoir envie de l'écouter à ce moment-là.

Pour les recommandations, nous avons besoin de trois tables. La table algorithme qui permet d'identifier de quel algorithme provient la recommandation effectuée. La table recommandation qui permet de stocker l'ensemble des recommandations faites pour une écoute. Ces deux premières tables sont nécessaires aux chercheurs du Loria car ils peuvent analyser le comportement de leurs algorithmes en fonction de la musique jouée (style, « dansabilité »…). Enfin, la table "vient" qui permet de savoir si la musique jouée est issue d'une recommandation ou d'une nouvelle recherche de l'utilisateur. Cette table permet de voir si un algorithme est choisi et permet aux chercheurs de faire des statistiques pour savoir lequel est le plus efficace par exemple.

Comme indiqué au début de la partie, le diagramme de classe a évolué tout au long du projet. Nous allons donc maintenant vous présenter le schéma final de la base de données (figure 3) et vous expliquer les changements apportés.

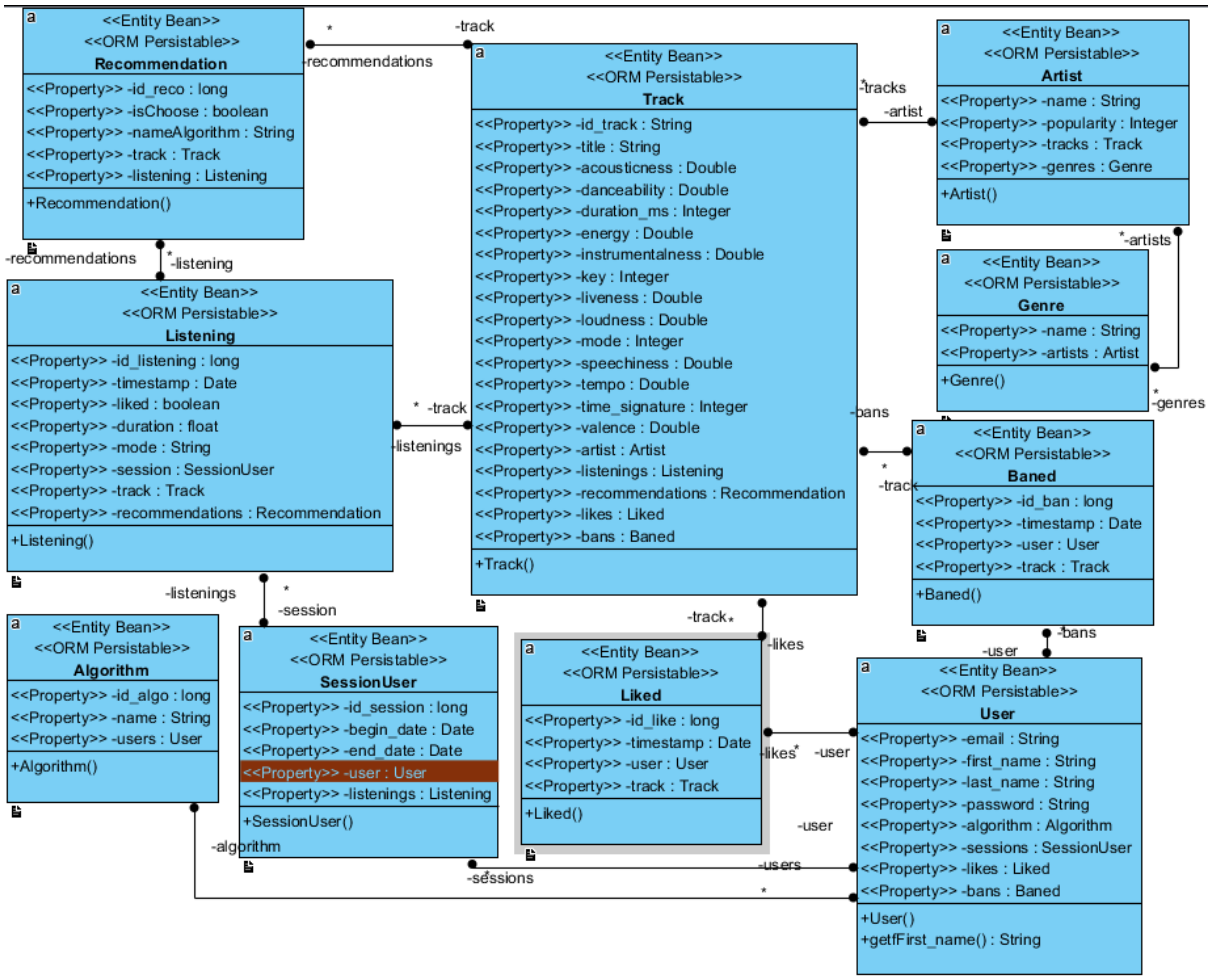

Figure 3 : Diagramme de classe final

Le premier, le plus bénin, est que toutes les classes et les attributs portent des noms anglophones. C'est pourquoi, par exemple, la table "ecoute" devient la table "listening".

Ensuite, nous avons ajouté une table genre reliée à la table artist dans le but de définir une liste de genres pour chaque artiste. Ceci a été nécessaire afin de pouvoir regrouper l'ensemble des artistes en fonction de leurs genres. Cet ajout a été apporté parce qu'il apporte des informations supplémentaires dans la base de données permettant aux algorithmes de sélectionner des recommandations pertinentes de ce point de vue. Par exemple si un utilisateur écoute une musique d'un artiste dont le genre est axé sur le rock, l'algorithme peut décider de proposer des recommandations basées sur ce critère.

De plus, nous avons également créé 2 nouvelles tables nommées "liked" et "baned". Ces tables permettent de déterminer si l'utilisateur a aimé (table "liked") ou non (table "baned") une musique et quand. De la même manière, ces tables ajoutent de l'information utilisable afin de permettre aux algorithmes de se baser sur plusieurs critères pour le calcul des recommandations.

Enfin, nous avons également décidé de créer un lien entre la table "user" et "algorithm", afin d'attribuer un algorithme à un utilisateur lors de son inscription. Ceci a été nécessaire pour la gestion du mode radio de l'application. Dans le but de tester l'efficacité des algorithmes, il faut pouvoir tester différents algorithmes sur une multitude d'utilisateurs tout en sachant quel algorithme est utilisé pour quel utilisateur.

Nous pouvons remarquer que la table "track" est relativement chargée. Toutes ses informations sont récupérées principalement de l'api de Spotify et permettent de décrire précisément une musique en fonction de différents critères comme le tempo, la « dansabilité », l'acoustique, etc. Toutes ces informations sont des données utiles pour le calcul des recommandations.

Maquettage des IHM

Afin de donner à l'ensemble des personnes travaillant sur le projet une idée plus précise de l'interface finale de l'application, nous avons décidé de réaliser une maquette. Ainsi il a été décidé dans les premières semaines, qu'une fois connecté, l'utilisateur aurait accès à une page de recherche, avec un choix pour accéder soit au mode radio soit au mode navigation.

Choix mode radio

Il s'agit d'un simple lecteur qui permet la lecture automatique des musiques proposées dans les recommandations. L'utilisateur a tout de même la possibilité d'aimer ou non une musique, ou de passer à la suivante. La figure 4 présente un aperçu de l'interface finale prévue dans la maquette pour le mode radio.

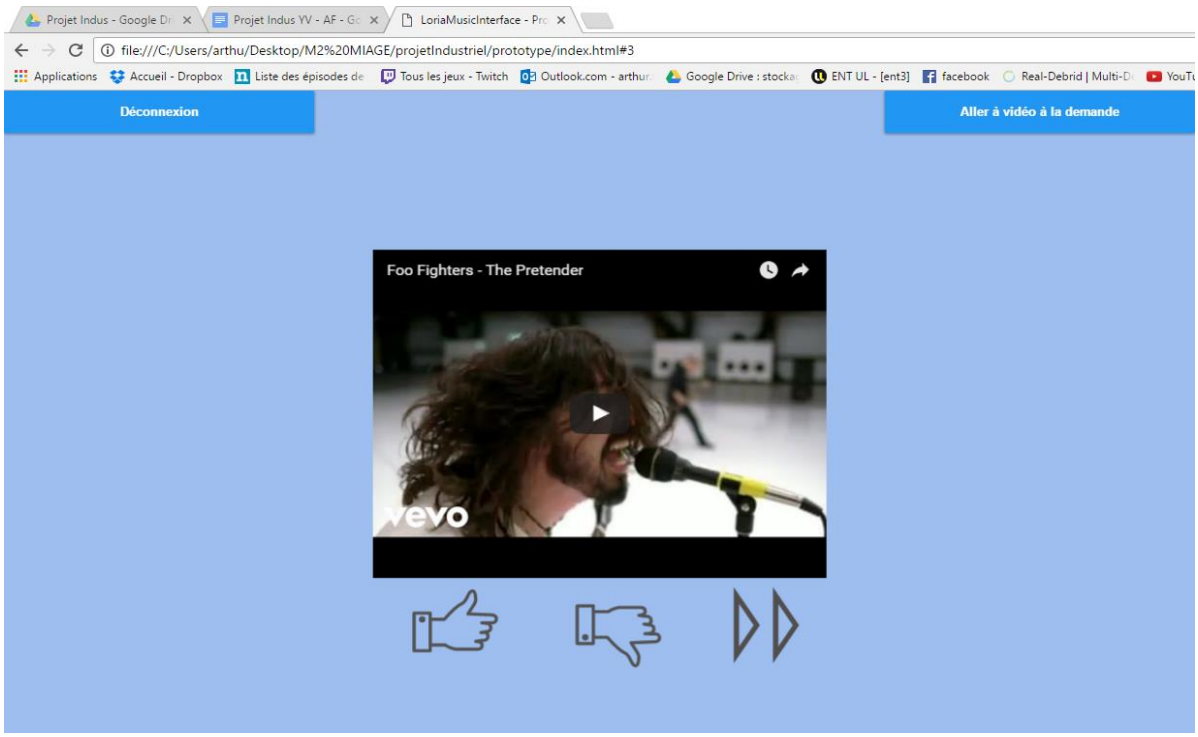

Figure 4 Maquette du mode Radio

Choix du mode navigation

Dans ce dernier, en plus du lecteur, une liste de propositions de musiques est disponible pour l'utilisateur. Ainsi il peut sélectionner la musique qui lui convient le mieux parmi les différentes propositions. De même que pour le mode radio, à la fin de la musique en cours, la prochaine est chargée automatiquement, cette fois-ci, en fonction de l'algorithme de la dernière recommandation choisie par l'utilisateur. Comme on peut le constater dans la figure 5, on y retrouve aussi les boutons j'aime, je n'aime pas, suivant, présents dans le mode radio.

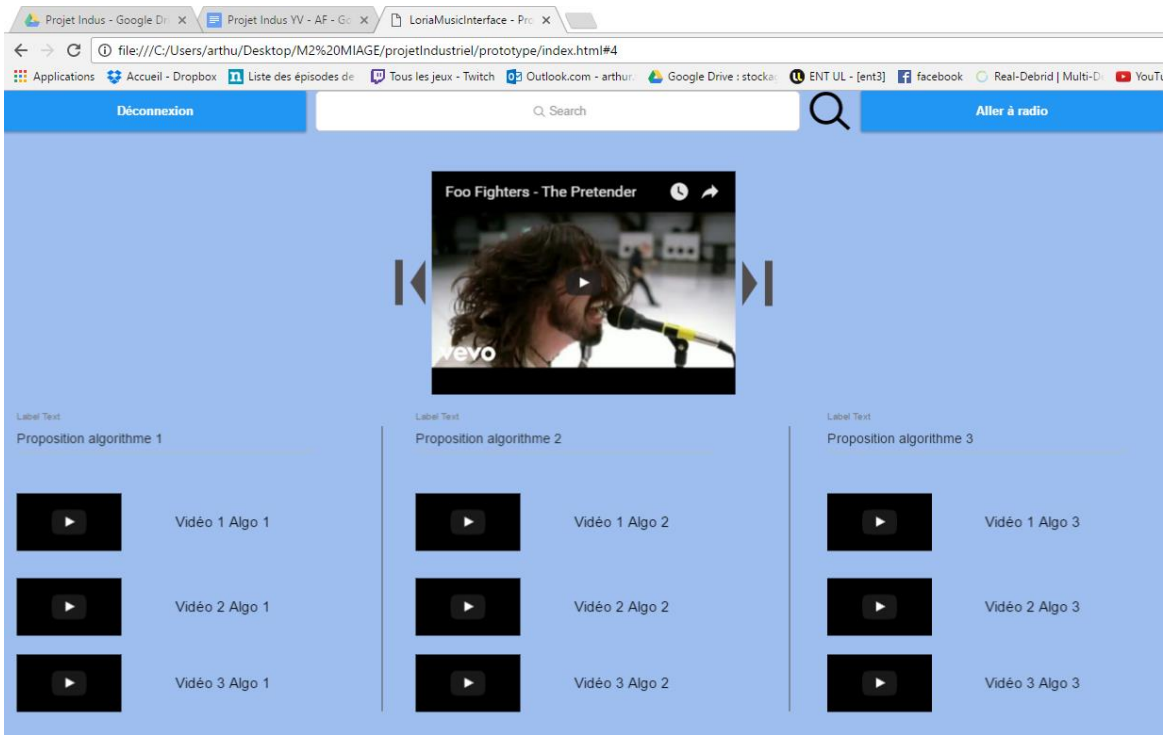

Figure 5 : Maquette du mode Navigation

Ce maquettage nous a permis dans un premier temps de faire valider rapidement par nos encadrants l'interface envisagée pour notre application. Et dans un second temps, de modifier l'interface graphique d'une manière simple en fonction des différents besoins qui pourraient apparaître.

Développement

Choix de Conception (API)

Pour la conception de notre API REST, nous avons réfléchi à une arborescence permettant de répartir au mieux les différentes classes. Pour ce faire, nous avons créé 5 packages rassemblant les classes par rôle.

C'est pourquoi, nous avons:

- un package "entity" (où sont placées toutes les classes servant de « mapping » avec la base de données),
- un package "dao" (où toutes les classes permettant l'accès et la manipulation des données sont stockées),
- un package "boundary" (où sont regroupées toutes les classes servant de contrôleurs REST),
- un package "api" (où les classes permettant les appels aux apis de Youtube et de Spotify sont définies)
- un package "algorithm" (où les classes décrivant la gestion des algorithmes sont placées).

Le programme principal (pour démarrer l'application) est quant à lui placé dans une classe située à la racine du projet.

Ensuite, nous avons fait en sorte que la partie client soit le plus détaché possible de la partie serveur et, ainsi, que le client n'utilise qu'une partie infime des données. En d'autres termes, la partie client ne connaît rien de la logique métier et du modèle de la base de données. Par exemple, le client ne fait jamais d'appel à l'api avec des identifiants de "session" ou de "listening", ce qui signifie que le serveur se charge lui-même de rechercher la session et l'écoute concernées par la requête faites par le client

Choix de conception (Application Angular)

Comme expliqué précédemment dans le choix des technologies, nous avons dans un premier temps essayé de développer notre application sur Angular2. Mais le manque de documentation en comparaison d'Angular 1 ralentissait notre progression. De plus, nous nous sommes aperçu que sur la fin de l'année, l'un des cours qui nous serait donné porterait sur Angular1, et nous permettrait ainsi de mieux comprendre l'architecture d'un projet Angular et de s'assurer de pouvoir produire une application fonctionnelle.

Architecture/Arborescence

Après plusieurs recherches afin de s'assurer de posséder de bonnes bases sur Angular1, nous avons commencé le développement de l'application côté client. La première des tâches réalisées a été la partie de connexion et d'inscription. En effet c'est sur cette partie que nous avons trouvé le plus de documentation, et cela nous a permis de bien assimiler le mode de fonctionnement d'Angular.

Ainsi notre projet est structuré de telle façon que chaque page de notre interface graphique se situe dans un dossier spécifique, contenant sa page html et son contrôleur JavaScript. A la racine du répertoire, se trouve le fichier app.js, qui permet de définir le routeProvider, qui s'occupe de gérer la redirection entre les différentes pages de notre application et le chargement des contrôleurs associés. Nous pouvons donc retrouver les répertoires suivants :

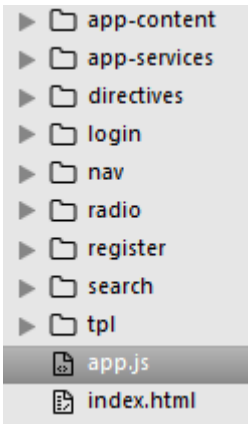

Figure 6 : Arborescence du projet

Pour ce qui est de l'inscription et de la connexion il paraît clair de voir apparaître les dossiers register et login. Mais une fois que l'utilisateur s'est inscrit ou connecté, il arrive alors sur la page de recherche.

Recherche

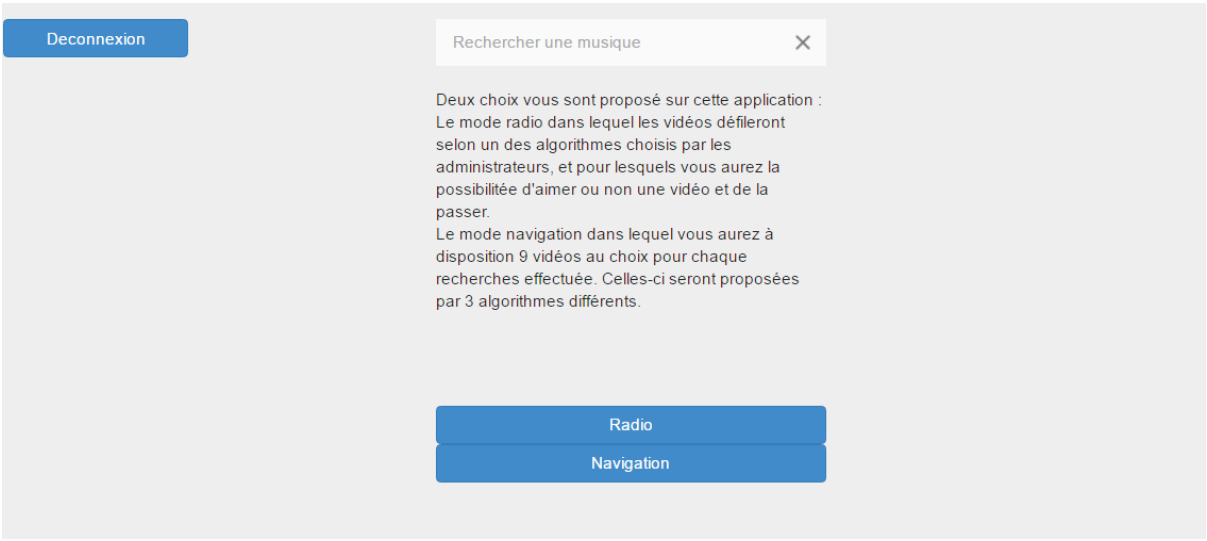

Figure 7 : Capture d'écran page Recherche

La figure 7 représente la version finale de la page recherche, incluant toutes les fonctionnalités définies lors du maquettage de l'application. Ainsi on retrouve bien le bouton permettant la déconnexion, le champ de recherche (en auto-complétion), et les deux boutons permettant d'accéder aux deux modes de lecture. On y retrouve aussi un texte explicatif, permettant d'expliquer à l'utilisateur les choix qui s'offrent à lui.

Radio

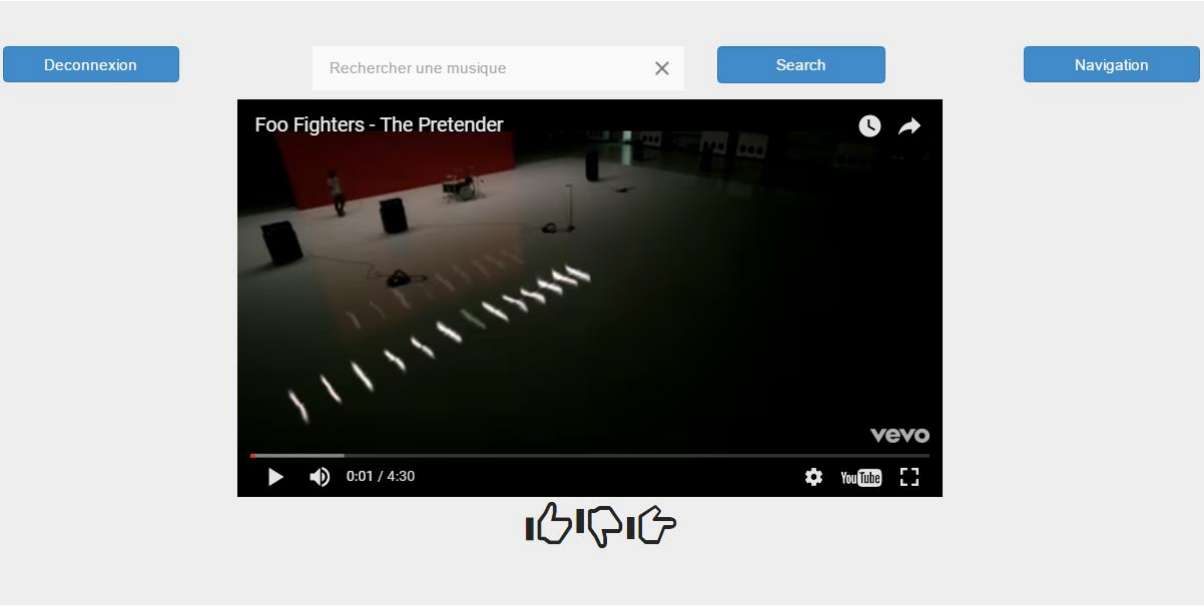

Figure 8 : Capture d'écran page Radio

Voici en figure 8, l'interface du mode radio. Comme convenu, ce mode restera basique, permettant simplement à l'utilisateur d'aimer ou non sa musique, ou de la passer. Dans les deux premier cas, les informations sont envoyées à l'API et sauvegardées en base de données, afin de permettre à l'algorithme de prendre en compte toutes les actions effectuées par l'utilisateur.

Dans le cas où l'utilisateur choisit de passer une musique, la vidéo de l'algorithme associé à l'utilisateur sera chargée. Cet algorithme est défini lors de l'inscription de l'utilisateur de manière aléatoire. Encore une fois, l'action de passer à la musique suivante est sauvegardée en base de données, permettant ainsi de constater le temps d'écoute lors de la musique.

Navigation

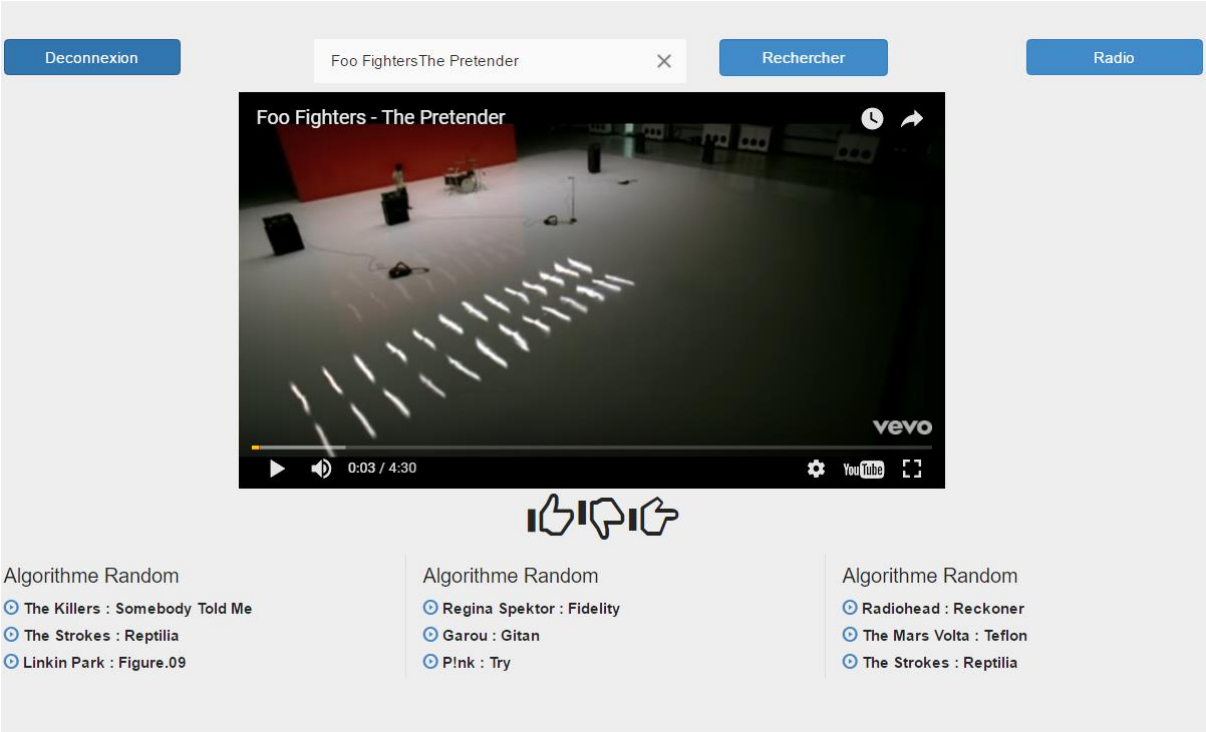

Figure 9 : Capture d'écran page Navigation

A l'inverse, si c'est le mode de navigation qui est choisit, nous retrouverons l'interface présente en figure 9. On y retrouve donc tous les éléments du mode radio, avec en plus en bas de page, les vidéos recommandées par les trois algorithmes actifs.

Les actions supplémentaires sauvegardées en base de données sont les choix de recommandations de musique. En effet lorsque l'utilisateur choisira l'une des musiques proposées, cette dernière ainsi que l'algorithme l'ayant proposée seront sauvegardés.

De plus, lors du passage à la musique suivante (de manière automatique à la fin d'une musique, ou grâce au bouton associé), le dernier algorithme sélectionné est gardé en mémoire afin de charger la musique en fonction du dernier choix effectué par l'utilisateur.

A tout moment sur les deux modes, l'utilisateur peut effectuer une recherche afin de changer de musique si celles proposées par les algorithmes ne lui conviennent pas. Il peut aussi décider de passer d'un mode à l'autre, ou de se déconnecter à n'importe quel moment. Lors du passage d'un mode à l'autre, la musique en cours d'écoute est conservée.

Utilisation d'API Web

Pour mener à bien le développement du projet, nous avons dû utiliser 3 apis externes différentes. La première, LastFm, est utilisée pour l'auto-complétion afin de corriger les erreurs de l'utilisateur. Par exemple, lors d'une recherche pour "Adle Helo", l'api va corriger automatiquement les erreurs et nous renvoyer un résultat pour "Adele Hello". La deuxième api est celle de Spotify, celle-ci nous permet de récupérer des informations détaillées sur les artistes et les musiques qui sont essentielles pour les algorithmes de calcul des recommandations. Enfin, nous utilisons l'api de Youtube pour pouvoir récupérer un identifiant d'une vidéo youtube correspondant à la recherche afin de permettre à l'application de lire les musiques issues de l'API web Youtube.

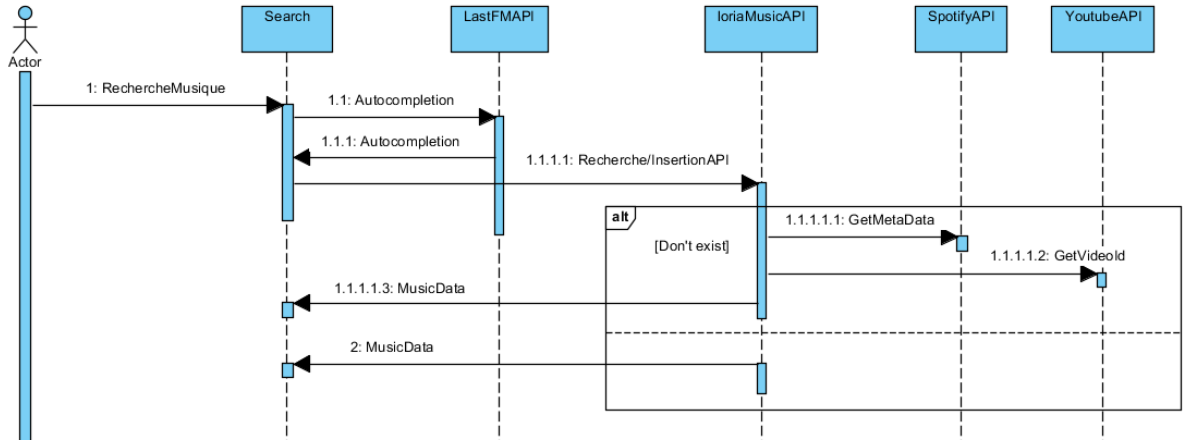

Figure 10 : Diagramme de séquence Appels API

Ce diagramme de séquence permet d'illustrer les différents appels à ces apis externes. Dans un premier temps, la partie client fait appel à l'api de LastFM lorsque l'utilisateur fait une recherche de musique. Grâce à cela, nous avons mis en place un système d'auto-complétion à partir de cet appel. Ceci signifie, qu'à chaque caractère ajouté ou supprimé par l'utilisateur un appel api est fait pour mettre à jour la liste de musique proposée.

Lorsque l'utilisateur lance une recherche (clic sur le bouton), le client adresse une requête à notre api en envoyant comme information le nom de l'artiste et le titre de la musique. Quand le serveur reçoit ces informations, il vérifie leur présence en base de données. Si les données ne sont pas contenues dans la base de données, le serveur fait appel à l'api de Spotify puis de Youtube pour récupérer les informations et les insérer dans la base de données (ceci est détaillé dans la partie suivante). Une fois ce traitement effectué, il envoie la musique au client pour qu'il puisse afficher la musique et la lancer.

Gestion des demandes de musiques

Le besoin principal de l'application est de fournir des musiques à l'utilisateur mais aussi d'avoir des données conséquentes sur ces dernières. Pour ce faire lorsque le client demande une musique au serveur ce dernier doit vérifier un certain nombre de choses avant de pouvoir fournir la musique.

La première vérification concerne l'artiste, le serveur contrôle si l'artiste existe ou non. Si ce dernier n'existe pas dans la base de données le serveur fait appel à l'api de Spotify, grâce à la classe "spotify" dans le package "apicall", afin de récupérer des données le concernant, comme sa popularité et les genres de musiques qu'il produit. La seconde concerne la musique, comme précédemment, le serveur contrôle si la musique existe ou non. Si la musique ne se trouve pas dans la base de données, le serveur récupère des informations dans l'api de Spotify, de la même manière que pour l'artiste. De plus, il fait appel à l'api de Youtube via la classe "youtube" du package "apicall" pour récupérer l'identifiant Youtube permettant au client de récupérer la vidéo correspondante.

Une telle requête peut être très gourmande en termes de temps d'exécution lorsque ni l'artiste, ni la musique n'existent dans la base de données. En effet, grâce à une multitude de tests, nous avons estimé le temps de réponse du serveur entre 2 et 3 secondes lorsque l'on doit ajouter l'artiste et la musique dans la base, entre 1 et 2 secondes lorsque seule la musique n'existe pas et moins de 0.5 secondes quand toutes les données sont présentes. Pour résoudre en partie ce problème, nous avons créé une classe "importData" qui, au démarrage du serveur, va lire un fichier contenant plus de 150 000 musiques afin de les ajouter à notre base de données. Ceci signifie que le lancement du serveur sera long, mais une fois lancé la base de données sera déjà remplie de beaucoup de musiques et permettra d'avoir des temps de réponses respectables dans la majorité des demandes d'un utilisateur, rendant l'application plus agréable d'utilisation pour ce dernier.

Gestion des Algorithmes

Afin de permettre aux chercheurs du Loria de tester leurs algorithmes, nous avons dû mettre en place un module de gestion des algorithmes. Pour ce faire, nous avons utilisé le patron de conception “abstract factory”. Nous avons donc créé une classe abstraite définissant l’ensemble des méthodes que ses classes filles doivent implémenter, celles-ci sont “computeRecommendation” et “getNameAlgorithm”. La première fonction doit fournir l’ensemble des recommandations pour un utilisateur et une musique donnés, la seconde doit renvoyer le nom de l’algorithme.

De plus, au niveau de la base de données, nous stockons les algorithmes actifs. En effet, pour le mode navigation, le client doit proposer 9 recommandations à l’utilisateur issus de 3 algorithmes c’est-à-dire 3 recommandations par algorithme. Dans la table “algorithm”, nous avons les algorithmes actifs définis par un nom.

Enfin, dans la factory nous avons un “switch”, qui contient autant de cas que de classes filles. Ce dernier teste la valeur d’une chaîne de caractères correspondant au nom de l'algorithme donné dans la base de données pour créer le bon objet.

Ainsi, lorsqu’une demande de recommandations est émise par le client, le serveur récupère les 3 algorithmes actifs dans la base de données puis pour chacun de ses algorithmes, nous faisons appel à la factory pour créer les objets adéquats et exécuter la méthode générant les recommandations.

Procédure d'installation

Nos choix de technologies nous permettent d'avoir une procédure d'installation extrêmement simple. Pour ce faire il suffit de récupérer les sources sur le dépôt git (https://github.com/Yodaii/LoriaMusicRestAPI), et de faire la procédure d'importation de projet classique sur un IDE tel que NetBeans ou Eclipse.

Avant d'exécuter le programme, il faut s'assurer que la base de données a été créée et bien paramétrée. Nous avons choisi d'utiliser MariaDB, afin de l'installer, il suffit de se rendre sur le site internet et de faire l'installation de base. Lors de cette dernière il est demandé de choisir le mot de passe pour l'utilisateur "root". Une fois installé, il est possible d'avoir accès à HeidiSQL un SGBD assez basique mais qui suffit. Il ne reste plus qu'à se connecter avec les informations par défaut et créer la base "loriamusic". Il est possible de configurer le serveur directement dans le fichier properties de l'application.

Afin d'assurer la sécurité de l'accès à notre API, nous avons spécifié un CORS (Cross-Origin Ressource Sharing) sur notre serveur. Pour ce faire il suffit de rajouter une annotation comme ceci : « CrossOrigin(origins = "http://client") » sur chaque méthode appelée par notre client. Il faudra bien entendu remplacer client par l'adresse d'hébergement de l'application (par exemple « localhost :8081 » dans notre cas).

Difficultés Rencontrées (API)

Concernant le développement de la partie serveur, nous avons rencontré différentes difficultés. La première était la technologie en elle-même car nous avons eu peu de cours sur cette dernière (module de 9h), c'était donc la première fois que nous développions une api REST et qui plus est sur un projet assez complexe. En effet, certaines fonctionnalités de Spring tel que la configuration de l'api ou certaines annotations n'ont pas été simples à assimiler.

La deuxième difficulté fut de modéliser une base de données cohérente. En effet, nous devions concevoir une base permettant de stocker toutes les informations nécessaires pour, dans un premier temps, recueillir toutes les actions d'un utilisateur et, dans un second, faciliter l'accès et la manipulation des données par les chercheurs et leurs algorithmes.

Retro Planning

- Les deux premières séances ont été utilisées afin de définir les limites de l'utilisation des différentes API Web (la limite dans le nombre d'appel à chacune d'entre elles), ainsi que leur utilisation.
- La troisième semaine a été consacrée à l'étude de l'existant, qui a donc été abandonné au profit d'une nouvelle application.
- Lors de la quatrième semaine, nous avons réalisé les maquettes et le MCD de notre application, avec validation de la part de nos tuteurs.
- Cinquième et sixième semaine, découverte d'Angular 2 et mise en place des bases de notre API.
- Lors de la septième semaine, notre progression s'est retrouvée entravée par le manque de documentation d'Angular 2, nous passons donc sur Angular 1.
- De la huitième semaine à la fin, progression dans le développement de l'application et point d'avancement en fonction de la disponibilité de nos tuteurs.

Conclusion

Afin de conclure ce rapport, nous pouvons dire que ce projet industriel fut une expérience très enrichissante sur le point de vue technologique. En effet ce projet regroupe plusieurs éléments de cours que nous avons abordés lors de cette dernière année et nous a permis de les mettre en application tout au long de ces douze semaines.

De plus nous avons trouvé le sujet très intéressant, car il s'agit d'un outil que nous utilisons tous les jours et de travailler dessus nous a permis de comprendre les tenants et les aboutissants des applications similaires que nous sommes amenés à utiliser au quotidien.

Enfin, les choix de conception nous appartenaient, ils ont permis de développer notre analyse du besoin et d'y répondre à l'aide de technologie et de solutions adaptés.