LOG8430 : Architecture logicielle et conception avancée

TP2

Présenté à : Manel Abdellatif

Soumis par

Benjamin Cotton, 1734287
Louis-Charles Hamelin, 1742949
Raphael Christian-Roy, 1743344

Section 01

Dimanche 3 décembre 2017

Table des matières

README

Pour l'explication complète du TP2 et de son fonctionnement, voir le README sur notre Github ici: https://github.com/raph63/log8430/tree/master/TP2.

Logical View

Le logiciel étant en HTML et Javascript, le concept d'une "classe" n'est pas très explicite. Par contre, on réussi à ressortir des classes lorsque l'on analyse bien le code complet. On retrouve ces classes ainsi que leurs composants dans le diagramme de classes à la figure 1.

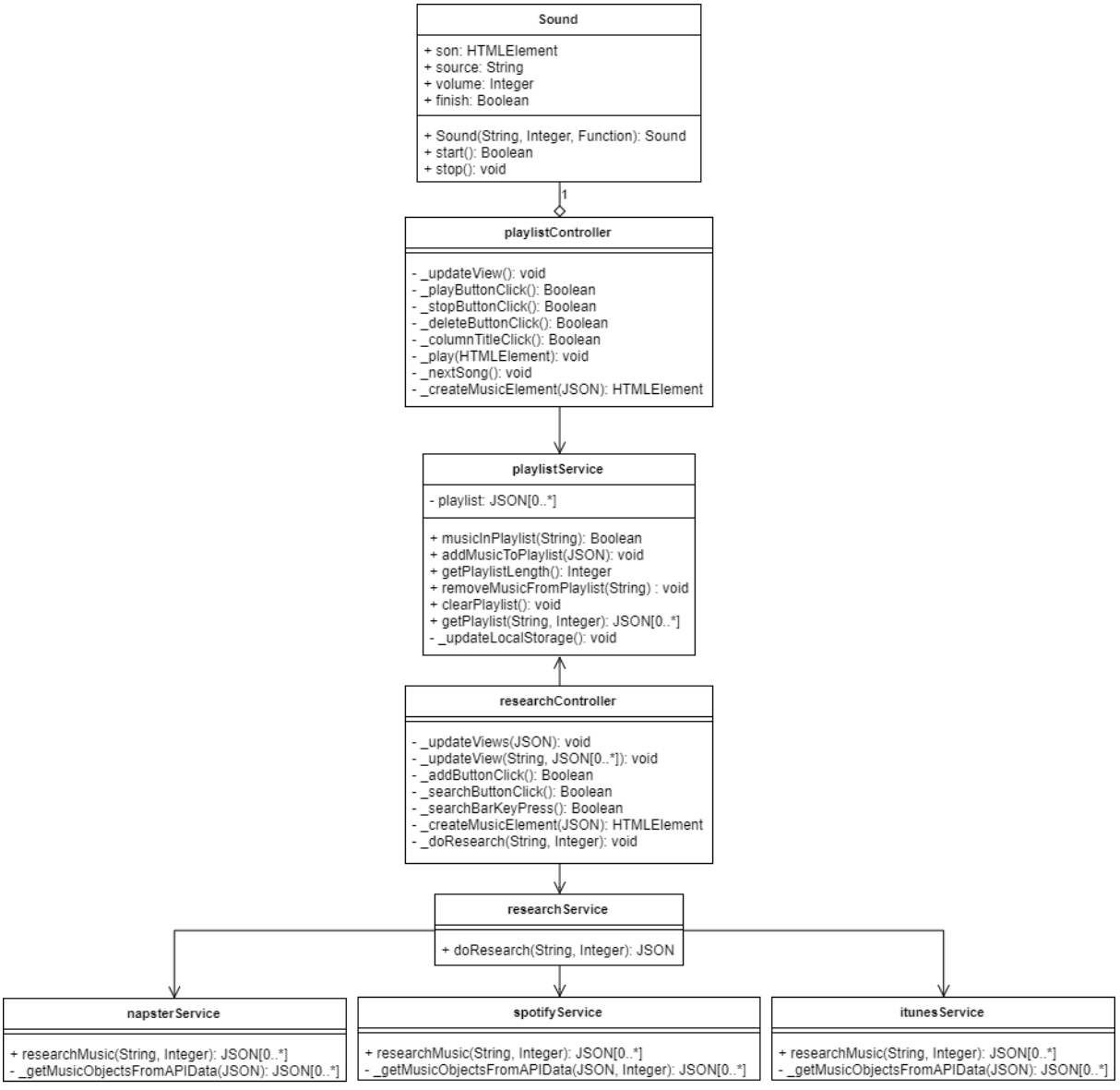


Figure 1 : Diagramme de classes

Development View

Le logiciel ne possède pas plusieurs composants. La figure 2 ci-dessous illustre le model MVC bien présent dans notre application ainsi que ses principaux composants.

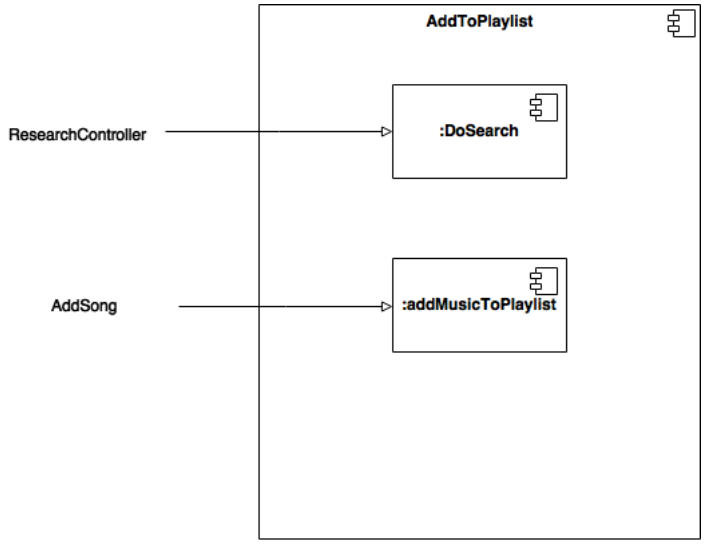


Figure 2 : Diagramme de composants

La figure 3 ci-dessous illustre le paquetage de la création de liste de lecture.

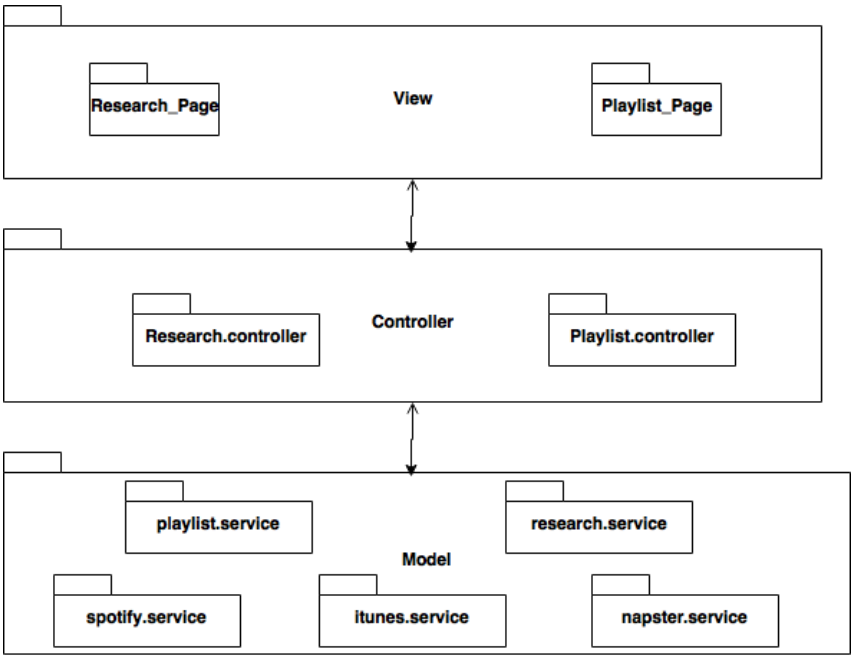
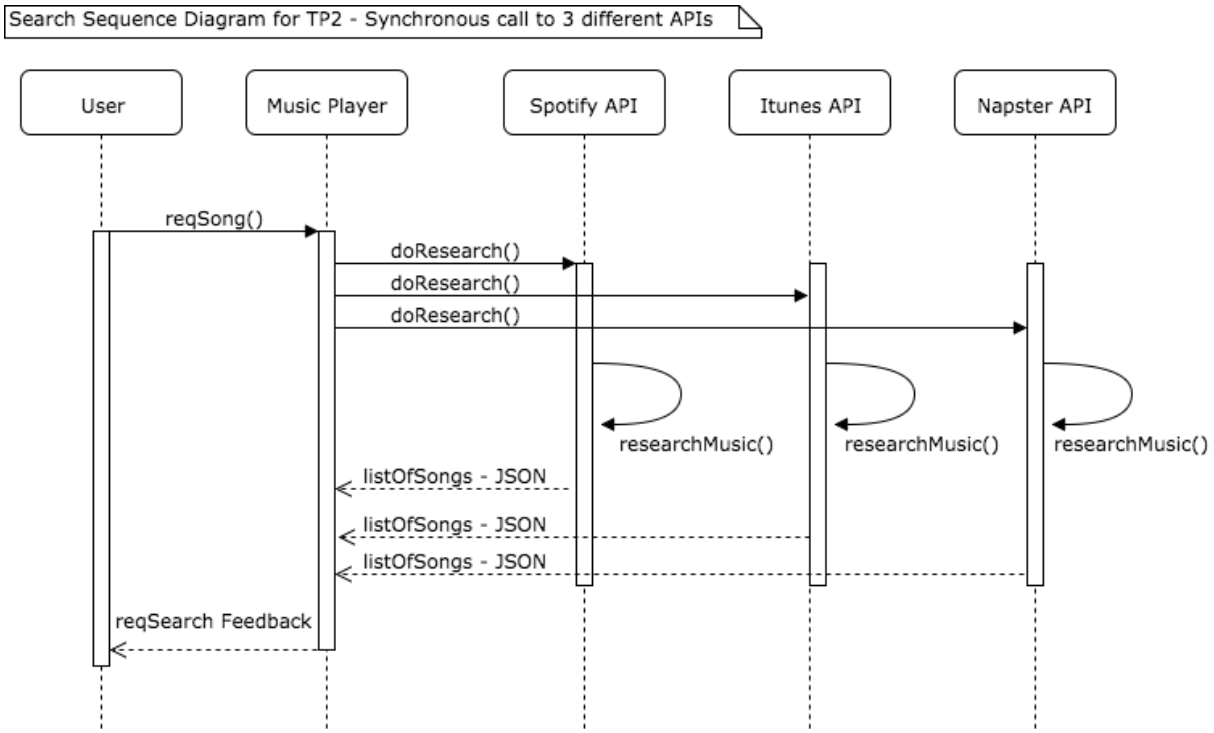


Figure 3 : Diagramme de paquetage

Process View

Pour le diagramme de process view, nous avons priorisé le diagramme de séquence pour le module recherche de notre web app. En partant de la requête de l'usager pour une chanson, le music player, fera 3 requêtes différentes à chaque web API de chaque service de musique en ligne. (Spotify, Itunes et Napster) Ces listes de chansons JSON sont par la suite retourner en feedback à l'app qui permettra ensuite à l'usager de sélectionner la chanson voulu (selon le bon service de musique en ligne). Il pourra donc l'ajouter à la playlist et faire play.



Physical View

Comme notre application est encore en version HTML et Javascript, il n'y a pas vraiment d'artéfacts ou de composants et il n'y a pas de "jar". C'est pourquoi le diagramme de déploiement de la figure 5 ne contient que les dossier et les fichiers inclus dans ces dossier avec les liens entre eux.

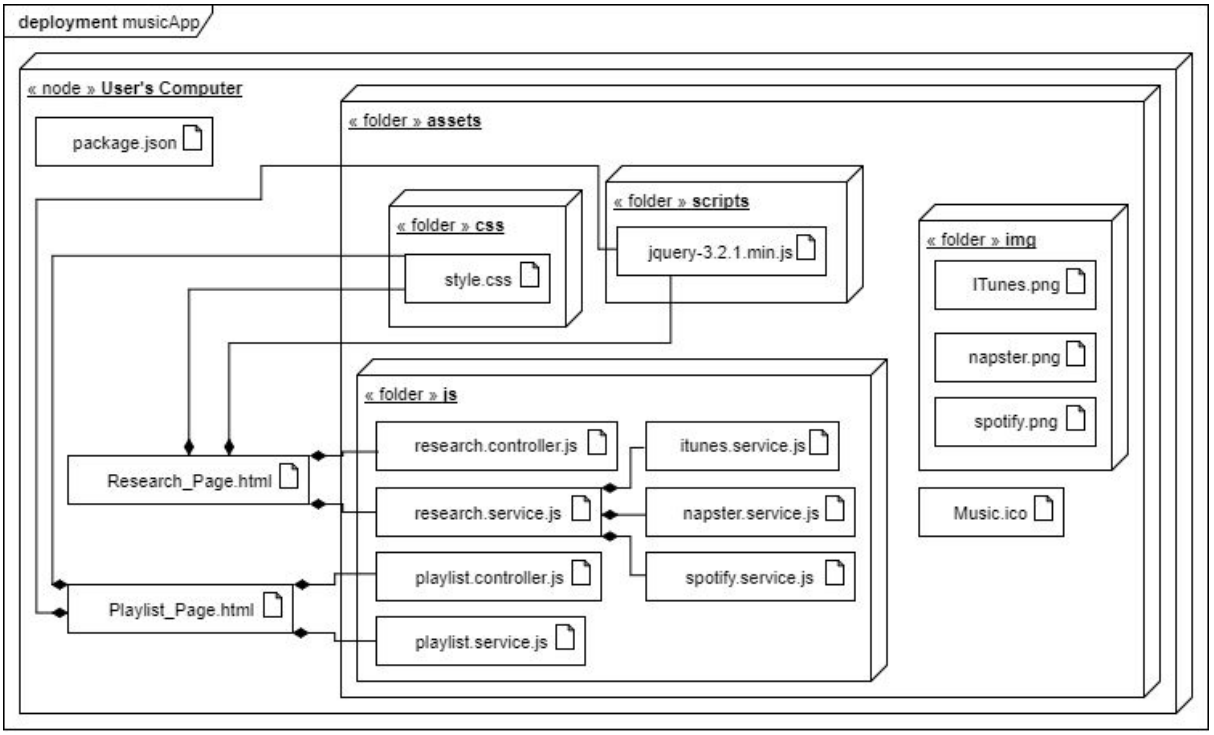


Figure 5 : Diagramme de déploiement

Scénarios

Dans le diagramme de cas d'utilisation ci-dessous à la figure 6, on peut voir tous les cas d'utilisation primaire de notre application.

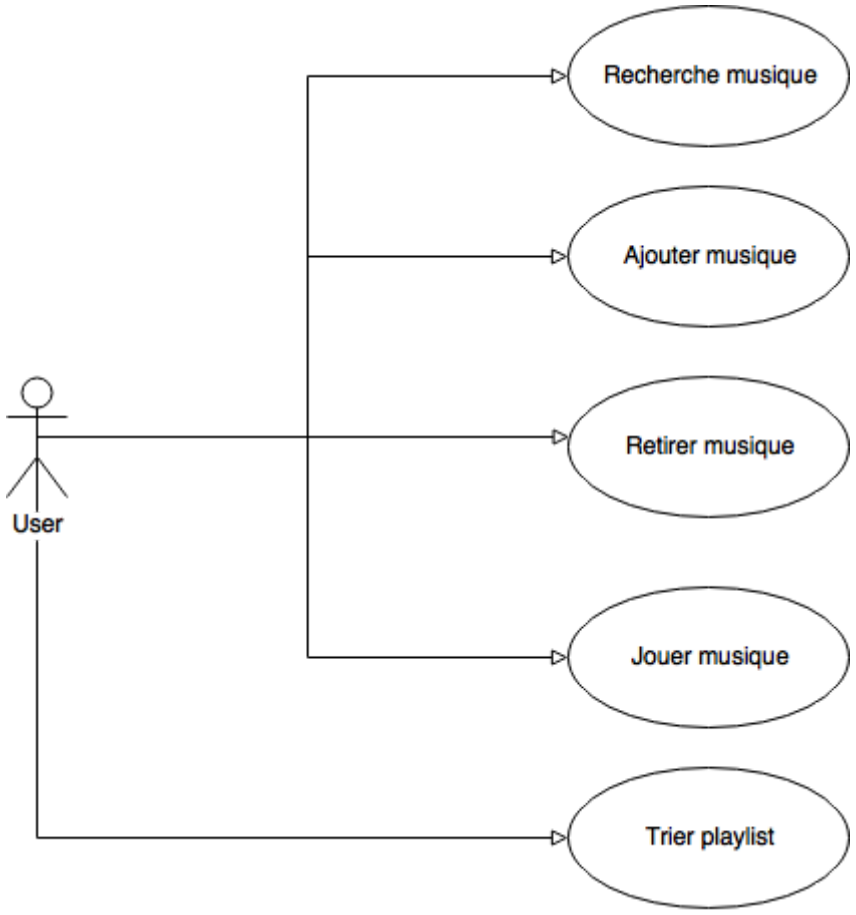
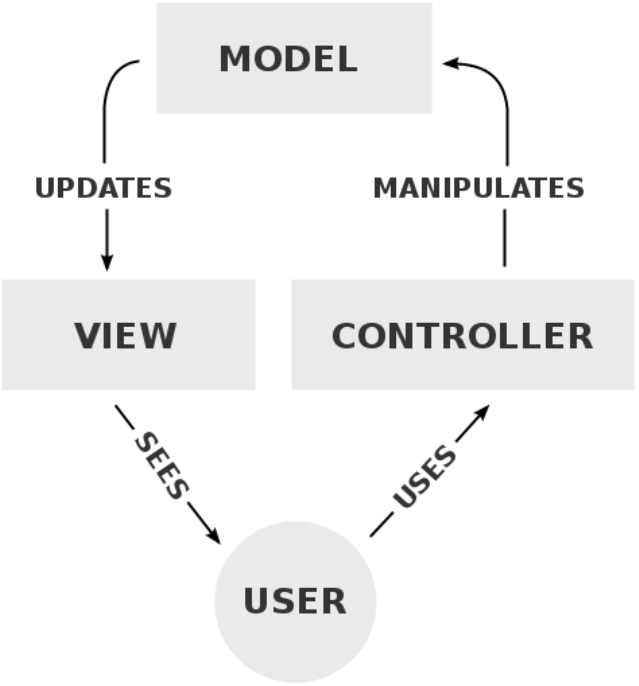


Figure 6 : Diagramme de cas d'utilisation

Patrons de conception

Dans ce laboratoire, nous étions libre de choisir l'architecture et la technologie pour réaliser notre gestion de liste de lecture avec trois API différentes. Ainsi, nous avons opté pour une technologie web en utilisant le patron de conception model-view-controller (MVC).



Model-view-controller (MVC) est un modèle d'architecture logicielle permettant d'implémenter des interfaces utilisateur sur des ordinateurs. Il divise une application donnée en trois parties interconnectées. Ceci est fait pour séparer les représentations internes de l'information des façons dont l'information est présentée à l'utilisateur et acceptée par celui-ci. Le modèle de conception MVC dissocie ces composants majeurs permettant une réutilisation du code efficace et un développement parallèle. Ainsi, dans notre cas, nos vues consistent en nos pages HTML de recherche de musique et d'affichage de liste de lectures, nous avons un controller pour effectuer la recherche et les modèles sont les services qui se connectent au API napster, itunes et spotify.

Choix architecturaux et technologiques

Dans ce laboratoire, nous étions libre de choisir l'architecture et la technologie pour réaliser notre gestion de liste de lecture avec trois API différentes. Ainsi, nous avons opté pour une technologie web en utilisant le patron de conception model-view-controller (MVC). En effet, notre choix d'utiliser une technologie web était simplement un question d'intérêt et de simplification avec le patron de conception MVC. De plus, en faisant ce choix technologique, nous nous simplifions la tâche pour le TP suivant qui utilisait des services. Nous avons donc pu réutiliser certaines composantes de notre code du TP2 pour concevoir le TP3.

Tests

Pour les tests, nous avons utilisé la librairie QUnit pour faire des tests automatisés sur les services en javascript et nous avons fait des tests manuels pour tester le contrôleur et le UI.

Pour les tests QUnit, il faut tout d'abord faire "npm install" dans le dossier "TP2" pour pouvoir les rouler. Ensuite, il suffit d'ouvrir le fichier "tests.html" dans le dossier "tests". Tous les tests sont alors exécutés et les problèmes trouvés sont ressortis.

Pour les tests manuels du UI et des contrôleurs, voir le tableau 1 suivant :

Nom du test	**Étapes**	**Résultat attendu**	**Résultat (P pour Passed, F pour Fail)**
1. Ouvrir le site pour la première fois	Ouvrir "Research_Page.html"	"No music found with this API." est supposé être inscrit sous chaque API.	**P**
2. Rechercher "success"	Faire le test précédent Dans la barre de recherche, inscrire "success"	10 musiques ayant un lien avec "success" sont listés sous chaque API avec un "+" sur le côté gauche de chaques éléments des listes.	**P**
3. Ajouter de la musique à la liste de lecture	Faire le test précédent Appuyer sur le "+" d'une musique pour chacun des APIs (avec différents noms de titre et d'artiste)	Le "+" des 3 musiques est maintenant un crochet et il n'y a plus de main lorsque l'on passe sur le crochet. NOTE : Si on rafraichit la page et refait la même recherche, les musiques ajoutés ont encore un crochet.	**P**
4. Vérifier la musique dans la liste de lecture	Faire le test précédent Ouvrir la page "Playlist" à l'aide de l'onglet sur la gauche	La page "Playlist" s'affiche avec les 3 musiques précédemment ajoutés en ordre alphabétique de titres. De Plus, le symbole des APIs correspondants sont assignés à chaques musiques.	**P**

5. Trier avec ordre alphabétique inversé	Faire le test précédent Cliquer sur le titre de colonne “Name”	La page “Playlist” s’affiche avec les 3 musiques précédemment ajoutés en ordre alphabétique inversée de titres.	**P**
6. Trier avec autres colonnes	Faire le test 5 avec la colonne “API”, “Artist”, “Time”; deux fois	La première fois que l’on clique sur la colonne, les musiques se classent en ordre alphabétique. La deuxième fois, en ordre alphabétique inversée.	**P**
7. Trier avec colonne invalide	Faire le test 6 avec la colonne contenant les boutons de lectures et avec la colonne contenant les “X”.	L’ordre ne change pas.	**P**
8. Jouer liste lecture	Faire test 4 Appuyer sur le bouton de lecture de la première musique et laisser les 3 musiques jouer.	Les 3 musiques jouent une après l’autre et, après la troisième, il n’y a plus de son.	**P**
9. Jouer liste lecture et réordonner	Faire test 8, mais réordonner la liste pendant qu’une musique joue.	La prochaine musique jouée sera la prochaine musique dans le nouvel ordre de la liste de lecture.	**P**
10. Arrêter musique	Faire test 4 Appuyer sur le bouton de lecture d’une musique Appuyer sur le bouton d’arrêt de cette même musique.	La musique s’arrête.	**P**
11. Jouer autre musique	Faire test 4 Appuyer sur le bouton de lecture d’une musique. Appuyer sur le bouton de lecture d’une autre musique.	La musique qui jouait s’arrête et la nouvelle commence.	**P**
12. Jouer autre musique et arrêter musique	Faire test 10 Appuyer sur le bouton d’arrêt de la musique	La musique s’arrête, sans erreur.	**P**
13. Plusieurs arrêt et lecture complète	Faire test 11 Faire test 8	Vérifier que les 2 tests passent lorsque réalisés un après l’autre.	**P**

14. Supprimer une musique	Faire test 8, mais en appuyant sur le bouton “X” de la 2e musique lorsque la première joue.	La musique 2 disparait de la liste de lecture, la 3e joue suite à première et tout s’arrête sans erreur après les 2 musiques.	**P**
15. Supprimer musique en cours	Faire test 4 Appuyer sur le bouton de lecture d’une musique Appuyer sur le bouton “X” de cette même musique	La musique s’arrête, sans erreur, et on peut jouer les musiques restantes sans problème.	**P**
16. Supprimer toute la liste	Faire test 4 Appuyer sur le bouton “X” de toutes les musiques.	“There is no music in the playlist” est affiché au lieu de la table de liste de lecture,	**P**

Tableau 1 : Tests manuels

Analyse de la qualité du projet

Les métriques utilisées pour analyser la qualité du code de notre web app tp2 sont majoritairement les métriques ressorties par SIG (Software Improvement Group) pour analyser la maintenabilité d'un système. La maintenabilité est une des caractéristiques de la qualité d'un produit logiciel les plus importantes, car sans maintenabilité, améliorer le produit devient une tâche des plus difficile.

Métrique	**Outil pour obtenir la métrique**	**Valeurs de référence**
Fonctions de petite taille	Outil en ligne pour web app	Selon SIG : ● Au plus 6.9% avec plus de 60 lignes ● Au plus 22.3% avec plus de 30 lignes ● Au plus 43.7% avec plus de 15 lignes ● Au moins 56.3% avec au plus 15 lignes
Complexité cyclomatique faible	Outil en ligne pour web app	Selon SIG : ● Au plus 1.5% avec un McCabe au-dessus de 25 ● Au plus 10% avec un McCabe au-dessus de 10 ● Au plus 25.2% avec un McCabe au-dessus de 5 ● Au moins 74.8% avec un McCabe de 5 et moins
Petit nombre de paramètres envoyés aux fonctions	Outil en ligne pour web app	Selon SIG : ● Au plus 0.7% avec plus de 7 paramètres ● Au plus 2.7% avec plus de 4 paramètres ● Au plus 13.8% avec plus de 2 paramètres ● Au moins 86.2% avec 2 paramètres ou moins
Minimum de code non utilisé	Outil en ligne pour web app	0% serait l'idéal

Avoir assez de commentaires	Outil en ligne pour web app	Ratio de nombre de lignes de commentaire sur le nombre de lignes de code de plus de 5%

Tableau 1 : Métriques utilisées pour l'analyse de la qualité du projet ainsi que les valeurs de références de ces métriques

Premièrement, il est important d'avoir des fonctions de petite taille, car elles sont plus faciles à comprendre, à réutiliser et à tester. Pour notre tp2, 0% des méthodes contiennent plus de 60 lignes, 0.8% en contiennent plus de 30, 36% plus de 15 et 64% en ont un maximum 15. Selon SIG, notre music player tp2 serait considéré comme un système d'au moins 4 étoiles pour la grosseur des méthodes, ce qui est un très bon score.

Deuxièmement, il faut avoir une complexité cyclomatique la plus faible possible pour aider à tester et à garder les fonctions le plus simples possible. Considérant que notre app était une app web, il est plutôt dur d'évaluer concrètement combien serait la complexité cyclomatique de toutes les fonctions. Or, considérant la précisions et la complexité des fonctions évaluées, nous sommes capable de dire que selon SIG, notre music player tp2 se classe comme un système d'au moins 4 étoiles pour la complexité cyclomatique, ce qui est très bien.

Troisièmement, il est bien de limiter le nombre de paramètres passés aux fonctions pour améliorer la compréhension et la réutilisabilité de celles-ci. Pour JabRef, 0% des fonctions ont plus de 7 paramètres, 0% en ont plus de 4, 40% en ont plus de 2 et 60% en ont 2 ou moins. Selon SIG, ceci classerait ce projet tp2 à au minimum 4 étoiles pour le nombre de paramètres par fonctions, ce qui est excellent.

Quatrièmement, malgré que ce ne fasse pas parti des critères importants de SIG, une autre partie importante de la maintenabilité est de ne pas garder du code qui n'est plus utilisé. Ainsi, on réduit le nombre de codes et cela permet de pouvoir mieux analyser le tout par la suite. Notre tp2 contient un total de 0 instances de variables non utilisées, 0 variables locales non utilisées ainsi que 0 méthodes non

utilisées. L'avantage avec le JavaScript et avec un projet de la grosseur de celui-ci, c'est qu'une variable inutilisé est vite identifiable et donc on les a toutes enlevées.

Cinquièmement et dernièrement, une des choses les plus importantes pour aider à comprendre du code, surtout pour un projet open source où n'importe qui peut venir aider, est de commenter le code. Ceci permet à un nouveau développeur sur le projet de comprendre rapidement la fonctionnalité et la raison d'être d'une fonction ou d'une variable, sans avoir à regarder une dizaine de fichiers. Notre tp2 respect aussi ce critère, il contient des en têtes de fonctions bien définies pour chaque fonctions et nous avons commenté toutes les lignes nécessaire à la compréhension de la logique du code.

Finalement, en web un des critère super essentiel, la modularité des fichiers et des modules de l'app. Comme le démontre les diagramme au début de ce rapport, nous avons effectué une très bonne séparation de nos modules afin de créer une app le plus modulaire possible en vu de possible futures expansions logiciel. Utilisant le patron MVC, modularisé toutes nos modules était bien évidemment plus facile.

Critique de la solution proposée

Pour une simple application de bureau qui serait utilisé par quelqu'un ayant un compte payant avec Spotify, Napster et Itunes, cette application est assez pratique. En effet, elle permet d'avoir l'aperçu des musiques disponibles sur chacune de ces 3 applications de musique. Ainsi, l'utilisateur pourrait savoir sur quelle application il pourra écouter la musique qu'il désire, sans avoir à faire une recherche individuel sur ces 3 applications. Si l'utilisateur ne possède pas de comptes avec ces dernières, une page web permettant de faire une liste de lecture d'aperçus ne lui sera pas vraiment utile.

En supposant que l'on pouvait avoir accès aux musiques entières lors de la recherche de musique sur les APIs choisis, notre application serait très intéressante! En effet, elle permettrait d'avoir les musiques qui sont disponibles sur seulement une application au même endroit. Il y aurait donc une plus grande base de donnée disponible pour notre application que pour les 3 autres utilisés individuellement.

En analysant plus la façon de faire de notre solution, il y a plusieurs choses qui pourraient être faites pour améliorer le produit. Premièrement, on pourrait mettre nos HTML sur le web plutôt que d'utiliser le fichier localement. Ainsi, n'importe quel utilisateur pourrait l'utiliser son son ordinateur sans avoir besoin de télécharger quoique ce soit.

Deuxièmement, certaines parties, comme la recherche de musiques avec les 3 APIs, pourraient être encapsulées dans des services pour rendre le programme plus réutilisable et modifiable facilement. En plus de cela, cela permettrait de ne plus avoir à utiliser l'extension de Google Chrome "Allow-Control-Access-Origin: *" qui n'est vraiment pas une extension sécuritaire. L'utilisateur n'aurait donc pas à installer une extension non-sécuritaire en plus de ne pas avoir à penser à activer l'extension lors de l'utilisation de l'application et à la désactiver lors de l'arrêt.

Troisièmement, les 2 HTMLs ont une tous les deux la même colonne sur la gauche pour le choix de la page à afficher, ce qui fait une duplication de code pour rien. On pourrait remédier à cela avec une technologie comme "Pug" qui permet de réutiliser une même

disposition de page avec une simple modification dynamique de certaines sections en fonction de l'adresse URL entrée.

Quatrièmement, il aurait pu être intéressant d'avoir la possibilité de faire plusieurs listes de lectures. De plus, ces listes auraient pu être partageable avec les autres utilisateurs, n'obligeant ainsi personne à faire la même recherche plusieurs fois pour chacun de ces appareils. En effet, notre application garde en mémoire la liste de lecture pour chaque appareil indépendamment. Un utilisateur avec plusieurs ordinateurs pourrait bénéficier du fait de pouvoir sauvegarder des listes de lectures en ligne.

Cinquièmement, une page de connection aux différents API aurait pu permettre, pour certains API, d'avoir accès aux musiques complètes plutôt que d'avoir accès aux aperçu seulement. Un utilisateur ayant un compte payant avec une des 3 applications utilisés aurait donc un meilleur service qu'un autre ne payant pour aucune application. Par contre, ceci rajouterais beaucoup de gestion de comptes utilisateurs et prendrait beaucoup de temps de d'argent pour tester le tout avec plusieurs compte utilisateurs pour plusieurs APIs.