Département de génie informatique et de génie logiciel

INF3995

Projet de conception d'un système informatique

Rapport final de projet
Conception d'un système audio pour café Internet

Équipe No. 3

Constantin Bouis
Christella Cidolit
Léandre Guertin
Anis Redjdal
Issifath Sanni

Avril 2019

Table des matières

1 Objectifs du projet

Dans le cadre de ce projet, nous avions pour but de concevoir un système audio participatif pour le café-bistro Élévation. Notre système rend le café plus attrayant en permettant aux clients utilisant la connexion internet de soumettre à partir de leurs appareils mobiles Android des chansons à un serveur conçu sous Linux pour fonctionner sur une carte FPGA. Nous offrons également la possibilité aux employés du café de gérer les musiques, utilisateurs et le son à partir d'un mode superviseur sur des tablettes Android. Enfin, le propriétaire de l'établissement a accès à une application sur PC lui permettant de suivre les activités du serveur et administrer les comptes de ses employés.

Pour rendre possibles ces fonctionnalités, les produits conçus devaient répondre à différentes exigences. En ce qui concerne le serveur, son rôle est de traiter les différentes requêtes qu'il reçoit des applications via l'interface REST définie, puis de diffuser la musique décodée à partir de fichiers MP3 qu'il achemine vers la sortie audio standard de la carte FPGA. Par la suite l'application Android vient en deux modes avec une interface simple et instinctive. Le premier mode est pour les clients, il doit leur permettre de soumettre au serveur les chansons présentes localement sur leur appareil (pour un maximum de cinq en même temps soumises au serveur) et voir la file de musiques en attente d'être jouées sur le serveur. Le second est le mode superviseur permettant de changer l'ordre ou supprimer des chansons de la liste sur le serveur, ainsi que de bloquer des utilisateurs ou de changer le volume sonore. Pour finir l'application PC de l'administrateur lui permet de suivre les activités principales du serveur, quelques statistiques et d'ajouter ou supprimer les comptes superviseurs de la base de données.

Ainsi, nos objectifs étaient de satisfaire le propriétaire du café-bistro et les clients avec un produit simple d'utilisation afin d'assurer le gain de popularité du concept. Nous avons donc consacré du temps pour améliorer l'aspect esthétique des interfaces et l'expérience utilisateur. Notamment, nous nous sommes assurés que notre application soit utilisable autant sur tablette que sur téléphone afin que le café puisse choisir de fournir des tablettes ou que les utilisateurs puissent utiliser leurs propres appareils. De plus, tout au long de la conception nous avons gardé à l'esprit les aspects de qualité et performances du système, que ce soit au niveau de la fluidité et robustesse des interfaces, des interactions entre les composantes ou encore en ce qui concerne la qualité du son.

2 Description du système

2.1 Le serveur (Q4.5)

Le serveur a été codé en C++ comme il nous l'a été demandé dans l'appel d'offres. Néanmoins nous avons eu à opérer de nombreux choix en ce qui concerne les librairies qui sont chargées de recevoir, traiter et répondre aux requêtes des applications sur la tablette Android et sur PC. Les librairies que nous avons choisi d'utiliser sont Restbed, RapidJson, Libmad, Alsa, Taglib et Zlib. La libraire Restbed est celle qui est utilisée pour créer un service web et gérer la création des routes correspondant aux différentes requêtes Http et Https disponibles. Pour manipuler des objets de type Json, nous avons choisi la librairie RapidJson. Les autres librairies ont servi à décoder et à lire les fichiers mp3 sur la carte fpga.

En ce qui concerne l'architecture du serveur, les points essentiels sont abordés dans la suite, mais un diagramme de classes est disponible en annexe de ce rapport pour plus de détails.
Le dossier du serveur disponible sur le répertoire Git contient trois sous dossiers notamment *database*, *docs* et *src*. Le dossier database qui contient entre autres les fichiers dans lesquels sont stockées les informations de connexion des superviseurs et de l'administrateur du système, le dossier contenant les fichiers mp3 des chansons qui sont présentes dans la file de musiques. Le dossier *docs* comprend quant à lui les fichiers nécessaires à lancer automatiquement le serveur au démarrage de la carte, le fichier de configuration d'Alsa que nous avons utilisé et qui devrait être placé dans le dossier */etc* de la carte. En ce qui concerne le dernier dossier, il contient tous les fichiers d'entêtes et sources du code.

Notre serveur peut être globalement divisé en 4 modules. Le premier est celui qui contient le code permettant d'utiliser les fonctions de la librairie Restbed pour gérer les requêtes. Les principaux fichiers sont Requests_Handler qui instancie et définit les paramètres tels que le port et l'adresse IP du service, les classes Requests_Client, Requests_Supervisor et Requests_Admin qui implémentent respectivement les méthodes de gestion des requêtes GET, POST et DELETE du mode usager normal, des superviseurs et de l'application sur PC. Une autre classe de ce module qu'il est intéressant de mentionner est la classe Requests_Tools qui comprend les instances statiques des gestionnaires du son, de l'authentification et des statistiques et logs.
Le deuxième module est celui de l'authentification qui permet de gérer la génération des identifiants des clients, la connexion, la déconnexion et le changement de mot de passe des superviseurs ainsi que ceux de l'administrateur. Il permet également, toujours en communiquant avec notre système de fichiers stocké dans le dossier database, de créer et de supprimer des superviseurs. Tout cela est fait grâce à une classe FileManager disponible dans le dossier Utils qui, grâce au mot clé template, permet de manipuler tous les fichiers de la database peu importe leur structure. Ensuite, le module qui s'occupe du son est composé d'une classe SoundManager qui

instancie un objet Player afin de décoder et de jouer une musique sur la carte fpga, ainsi qu'un objet VolumeHandler qui permet de gérer le son comme les requêtes de l'interface REST le demandent. La classe SongsQueueHandler de cette partie du système permet de gérer la file de musiques du côté du serveur et une chanson est représentée par un objet Song qui est caractérisé par l'utilisateur soumissionnaire et les informations obtenues du décodage de l'entête du fichier mp3. Pour simplifier, nous enregistrons les fichiers mp3 dans la base de données (le répertoire mentionné plus haut) sous le numéro qu'ils occupent en fonction du moment de leur soumission tout au long de la journée. À chaque démarrage du serveur, le dossier des chansons est vidé et le compte recommence à 0, ce qui diffère du numéro de la file d'attente qui va uniquement de 0 à 9.

Pour finir, les statistiques et le journal d'activités du serveur envoyés à l'application PC sont générés grâce au code contenu respectivement dans les fichiers StatisticsHandler et LogsHandler. Les méthodes de ces deux classes sont appelées dans le traitement des requêtes lorsque nécessaire. Les statistiques et les logs sont envoyés à chaque requête GET correspondante reçue en provenance de l'application PC et les logs sont écrits par lot de 5 dans un fichier.
Par ailleurs, l'utilisation de la librairie Restbed a facilité la gestion d'erreur des requêtes, car pour chaque requête reçue de l'une des applications, une session est ouverte et à sa fermeture, on renvoie le bon code d'erreur.

D'autres choix d'implémentation peuvent être soulignés. Par exemple, tout au long du projet nous avons développé sur nos ordinateurs alors nous avons trouvé intéressant de garder la possibilité de compiler le code aussi bien sur nos ordinateurs que sur la carte. C'est pourquoi nous avons laissé la possibilité de lancer le serveur sur un PC en utilisant la commande *make runl* et un *make runc* dans le cas de la carte. À noter que si l'exécutable est déjà compilé on peut aussi bien effectuer la commande *./main -l* pour le PC et *-c* pour le carte. De plus, pour les requêtes Https, les clés générées sont d'une longueur de 4096 bits pour la carte pour une meilleure sécurité. En outre, nous avons bloqué les requêtes Http pour le mode superviseur et l'administrateur. Enfin, nous avons trouvé intéressant de conserver les affichages dans la console du serveur, car tout au long du projet ils nous ont permis de mieux avancer et de déboguer. Ces affichages sont également disponibles dans le journal d'activité *journalctl* de SystemD puisque notre serveur est intégré comme un service et se lance au démarrage de la carte. Les commandes et fichiers utiles pour la compréhension de cette utilisation de SystemD se trouvent dans le dossier *docs/boot_with_systemd*.

Un dernier point qu'il nous semble intéressant de mentionner concernant notre code est que pour éviter d'avoir des soucis de cohérence entre les données et limiter le couplage des classes, nous avons utilisé de nombreux objets et méthodes statiques. En effet, nous avons divers modules qui permettent de maintenir à jour les informations du système ou d'effectuer des services dont de nombreuses classes ont besoin. Le fait qu'ils soient statiques fait en sort que l'on peut créer de nouvelles instances par moule et toujours avoir les mêmes informations sans avoir besoin de

faire des liens inappropriés entre les classes. Nous avions également la contrainte d'objets et d'attributs statiques à cause de certaines librairies telles que Restbed qui contenaient des méthodes devant être statiques et donc ne pouvaient opérer sur des objets non statiques.

2.2 Tablette (Q4.6)

En ce qui concerne notre application Android, nous nous sommes servis du langage java. En effet nous avions le choix entre java et kotlin, mais n'étant pas à l'aise avec kotlin, java nous semblait un choix plus judicieux puisque nous connaissions tous ce dernier. Pour ce qui est de l'architecture de notre application, le plan Modèle Vue Présentateur (MVP) nous a semblé être une évidence tant ce modèle a déjà fait ses preuves auparavant, que ce soit en termes de performance ou de maintenabilité. Afin de mieux comprendre comment nous avons appliqué le modèle MVP, voici un tableau qui décrit son fonctionnement :

Module	Nom du module	Rôle
Modèle	Models	Décrit comment chaque objet est modelé dans notre système, notamment les objets qui se transmettent sur le réseau.
Vue	View	La vue contient toutes les activités de l'application, ainsi que les fragments. Soit tout ce qui est directement lié à la vue de l'application.
Présentateur	Presenter	Nos présentateurs sont principalement utilisés pour faire les requêtes liées à chacune des activités.
Service	Service	Un service est un module autonome qui à un seul objectif précis et peut être facilement utilisé partout dans le code.
Adaptateur	Adapter	Un adaptateur permet de lier des éléments de la vue avec des données, c'est-à-dire décrire comment une vue complexe doit s'approprier des données.
Ecouteur	Listener	Un écouteur permet de réagir aux différents événements qui peuvent se produire sur la vue.
Fragment	Fragment	Un fragment est un morceau de vue complexe et réutilisable.

Tableau 1: Modules de l'application Android

Une version plus complexe et détaillée de notre architecture est disponible en annexe 1.

Nos modèles utilisent la librairie Gson pour être facilement convertis d'un objet à Json ou de Json à un objet. Cette librairie permet de faciliter, d'uniformiser et surtout d'encapsuler toutes les conversions Json.

Afin de centraliser toutes nos requêtes, nous avons mis en place un "RequestManager", ce dernier est appelé par tous les présentateurs qui ont besoin de faire des requêtes. Cela favorise grandement la maintenabilité du code.

D'ailleurs, pour ce qui est des requêtes, nous utilisons la librairie Volley. C'est une librairie efficace et avec de la bonne documentation, de plus les requêtes sont asynchrones et suivent le modèle REST.

Afin d'obtenir les paroles des chansons, nous n'avons pas pu utiliser une API classique qui se contente de faire une requête et de recevoir les paroles, de fait, l'API musixmatch est payante et nous ne pouvions pas inclure cela dans notre budget. Ainsi, nous avons utilisé le site *"genius.com"*. Nous faisons donc une recherche sur ce site, récupérons la page entière retournée par la recherche puis sélectionnons les paroles depuis cette page. Si genius retourne une erreur, nous faisons une recherche google et récupérons de la même manière les paroles depuis cette recherche. Cette manière non conventionnelle de récupérer les paroles des chansons est loin d'être infaillible, car *genius* comme *google* peuvent avoir des difficultés à trouver la chanson en question. Toutefois cette méthode est celle qui nous a permis de couvrir le plus de chansons parmi celles que nous avons considérées.

Pour ce qui est de la recherche de musiques dans la mémoire de l'appareil, nous avons pris l'initiative de rechercher sur la mémoire locale de l'appareil ainsi que sur la mémoire externe de ce même appareil. Ainsi, si un client possède des chansons sur sa carte SD plutôt que la mémoire locale de son appareil, il pourra les envoyer au serveur. Ce cas est plutôt commun, alors bien qu'il puisse légèrement ralentir la recherche de musique dans la mémoire de l'appareil, il est cependant nécessaire dans notre contexte.

Un point principal de notre design est le fragment de 'trackList' qui implémente une 'recyclerView' avec le 'drag and drop' ainsi que le 'swipe to delete'. Ce fragment est utilisé du côté client comme du côté superviseur. Cette implémentation permet de rendre l'expérience usager plus fluide et plus intuitive. Cela participe également au design de l'application. De plus, nous avons arrondi tous les boutons, ce qui donne un bien meilleur rendu visuellement parlant. La dernière particularité est la présence d'un petit bouton d'aide qui permet à l'usager de prendre en main l'application facilement.

2.3 Application sur PC

Du côté de l'application PC, nous avons utilisé le langage de programmation Java. L'utilisation de ce langage nous a permis de réutiliser quelques concepts que nous avons abordés du côté Android. Nous avons utilisé la librairie graphique Swing afin de construire l'interface utilisateur. Nous avons opté pour le modèle de conception MVC, soit Modèle-Vue-Contrôleur, afin de bien séparer les différentes couches de notre application.

La librairie Swing a seulement été utilisée du côté de la vue. Cette librairie nous permettait de créer des fenêtres, de boutons, des champs de texte, etc. Ainsi, nous avons ajouté des auditeurs sur les boutons et certains champs pour connaître l'état de l'usager et ses décisions prises dans l'application.

Du côté des modèles, nous avons utilisé la librairie Gson. Celle-ci nous permettait de faire une conversion des modèles Java en format Json. En utilisant le paramètre "@SerializedName("champ")", il est possible d'indiquer le nom du champ dans le format Json en français, tout en gardant une bonne consistance du code en anglais.

En ce qui concerne l'architecture du code de notre application, nous avons séparé chaque groupe de classe dans des modules différents. Le tableau 2 nous indique l'organisation du code, soit ce que nous avons dans chacun de ses modules.

Module	Nom du module	Rôle
Modèle	model	Ce module contient toutes les classes qui permettent de faire la traduction d'une classe Java en une chaîne de caractère au format Json.
Vue	view	Celui-ci contient tout ce qui interagit avec l'interface utilisateur. Il possède un contrôleur qui gère les requêtes, mais ne fait pas de grande logique dans le code. Il est aussi utilisé pour afficher le contenu des modèles, soit pour ceux des journaux et des statistiques du serveur.
Contrôleur	controller	Il permet de faire l'intermédiaire entre le service de requêtes fait au serveur et la vue. Il ne renvoie à la vue que les informations importantes et filtre les exceptions des codes de retour Https.
Service	service	Dans ce module, nous avons des classes qui possèdent des méthodes statiques. Elles sont utilisées partout dans l'application afin de fournir des

		fonctions utilitaires.
Composante	component	Ce module contient des classes qui étendent des classes fournies par Swing afin d'ajouter une fonctionnalité qui permettra d'uniformiser l'application.
Exception	exception	Module contenant toutes les exceptions utilisées dans l'application.
Interface	interfaces	Module contenant une interface utilisée pour la vue.
Ressource	resources	Dans ce module, nous avons regroupé toutes les ressources de l'application, soient les images et les textes.

Tableau 2 : Organisation du code dans leur module respectif

Pour ce qui est de notre architecture logicielle, nous avons essayé de découpler les composantes le plus que possible. Comme nous pouvons le voir dans à l'annexe 2, les vues (View) contiennent un contrôleur (Controller), qui eux, modifient les modèles pour qu'ils soient affichés sur la vue.

Dans nos services particuliers, nous avons un RequestManager qui fait la gestion de toutes les requêtes de l'application. En effet, elle gère tous les GET et POST du modèle REST. Tout comme du côté Android, nous pouvons transformer les objets en format Json, ce qui fait en sorte que ces deux fonctions peuvent prendre en paramètre n'importe quel objet. Ainsi, Gson remplit le Json avec les annotations "@SerializedName("champ")". Au retour d'une requête POST, comme nous insérons dans la méthode un attribut qui indique la classe de l'objet de retour, nous pouvons générer un objet du type de la classe attendue avec Gson à partir du format Json.

Le deuxième service particulier est le LabelUtils. Nous avons ajouté un fichier de ressource qui contient tous les textes que nous avons dans l'application. De cette manière, nous pouvons facilement retrouver un texte, car ils se situent tous au même endroit. De plus, si nous avions à traduire notre application, nous pouvons facilement juste ajouter un nouveau fichier texte et changer la langue que nous avons choisie.

Dans tous nos contrôleurs, nous attrapons beaucoup d'exception venant du RequestManager. Ceci vient du fait que nous gérons toutes les erreurs Http pour chaque requête, mais celles-ci ne sont pas toujours renvoyées à la vue. En effet, la vue n'a pas besoin de voir toutes ces exceptions, mais il arrive que nous renvoyions une exception différente à la vue afin d'améliorer la lisibilité du code.

Pour ce qui est de la vue principale, nous avons sorti les panneaux des statistiques et les journaux du serveur. Ainsi, nous pouvons réduire légèrement le nombre de lignes pour la vue dans la page d'accueil, qui est déjà monstrueux. Dans la liste des journaux, le modèle de Log implémente la méthode toString, ce qui permet de faire l'affichage de la liste sans avoir à directement indiquer dans la vue comment un Log doit être affiché.

2.4 Fonctionnement général (Q5.4)

- Installation et lancement du serveur :

Pour parvenir à configurer notre serveur et à le faire fonctionner sur Archlinux, plusieurs des étapes suivantes sont nécessaires. Tout d'abord, il faut installer les librairies dont les dépendances sont requises dans le Makefile pour lancer le serveur à savoir restbed, rapidJson, libmad, alsa-plugins, alsa-utils, taglib et zlib. Pour cela, la commande *pacman -Sy* du gestionnaire de paquets de la distribution Archlinux peut être utilisée.
Ensuite, pour compiler le tout, se placer le dossier src. Le fichier main.cpp contient l'adresse IP du serveur. Il faut donc remplacer la constante IP_ADDR_CARD avec celle de votre serveur. La prochaine étape est d'ouvrir un terminal en root dans le dossier *serveur* et de faire un *make runc -j4* pour compiler puis lancer un serveur distant ou *make runl -j4* pour le lancer en local.
Par défaut, le contrôleur du volume utilisé est celui de la carte de son de la fpga. Ainsi, pour lancer le serveur en local sur une machine, il faut mettre la valeur de la variable booléenne PLAYING_ON_CARD qui se trouve dans le fichier serveur/src/soundManager/VolumeHandler.h à *false*. Une fois que le serveur sera lancé, les applications sur PC et sur Android peuvent être lancées et les interactions entre les applications peuvent être visualisées dans la console du serveur (à condition que celles-ci aient l'adresse IP du serveur).

- Fonctionnement de l'application Android :

Le code de notre application Android peut être visualisé et compilé avec le logiciel Android Studio qui nous permet d'exporter notre projet en Java vers un appareil Android branché en USB lors qu'un Build est lancé. Notre application s'installe donc dans une apk avec notre logo et le nom Yellemon. Si on veut suivre le déroulement du lancement de l'application et les étapes de fonctionnement de celle-ci (quelle activité est ouverte par exemple), l'onglet "RUN" nous permet de voir en temps réel ces informations si l'appareil est toujours connecté au pc qui a "build" et "run" le code.

Pour ce qui est de la connexion au serveur, il faut changer l'URL de l'adresse IP dans la classe RequestManager.java en remplaçant par l'URL du serveur auquel on veut se connecter. De plus, on doit nécessairement autoriser l'accès aux musiques locales sur l'appareil Android sur lequel l'application est installée lorsqu'on est connecté en tant que Client afin de pouvoir téléverser une musique de l'appareil sur la file de musique du café-bistro.

En ce qui concerne l'interface de l'application Android, notre application est divisée en deux parties : client et superviseur dans la première activité et l'utilisateur doit choisir entre ces deux profils afin d'obtenir l'interface qui lui est associée. Pour le client c'est relativement simple, l'utilisateur a accès à la file de musique et à ses musiques locales et voit un bouton assez intuitif pour téléverser ses musiques sur le serveur et sur la file. Du côté superviseur, on trouve trois onglets qui permettent respectivement d'apporter des modifications à la liste de musique (changer l'ordre, supprimer une musique de la file…), à la liste noire (bloquer ou débloquer un utilisateur) et aux réglages de l'application (ajuster volume, changer mot de passe…).

- Fonctionnement du PC :

Tout d'abord, il faut ouvrir le projet avec Eclipse et s'assurer que le projet est configuré comme étant un projet Maven. De cette façon, toutes les dépendances seront automatiquement téléchargées. Afin de réussir à établir une connexion avec le serveur, il faut modifier la constante dans le fichier RequestManager.java qui indique l'adresse du serveur. Cette constante est dans le module "service" et porte le nom de SERVER_URL. Ainsi, il faut tout simplement aller dans le fichier MainView.java et compiler le projet avec *Run as Java Application*. Afin d'éviter tout problème de compatibilité, il faut s'assurer d'avoir la version 8 de Java JDK ou si ce n'est simplement que pour rouler le Jar, il ne faut que la version 8 du JRE. La page d'authentification de l'administrateur devra alors s'afficher et si le serveur est en ligne, il sera possible de se connecter pour faire l'utilisation complète de l'application.

Lors de l'ouverture de l'application, une page d'authentification de l'administrateur apparaît. Après avoir entré son identifiant et son mot de passe (nom d'utilisateur : admin, mot de passe : mario), l'administrateur se retrouvera sur une page d'accueil contenant toute l'information nécessaire à son utilisation, soit les journaux et les statistiques du serveur. Toutes les autres fonctionnalités sont indiquées par les boutons au coin supérieur droit, où une nouvelle fenêtre apparaîtra selon le bouton pesé.

3 Résultats des tests de fonctionnement du système complet (Q2.4)

En ce qui concerne les résultats de nos tests de fonctionnement de l'application Android avec le serveur et l'application PC réunis, le système au complet fonctionne globalement bien et la présentation de l'application s'est déroulée sans aucun "bug" ou arrêt du système ce qui prouve la robustesse de notre produit final.

Néanmoins, nous avons nous-mêmes constaté certaines limitations, telles que le fait que certains tags de fichiers MP3 ne contiennent pas toutes les informations concernant le titre et l'artiste sur certaines musiques, ce qui entraîne un manque d'information sur la file de musique du client et du superviseur. Cette défaillance peut également survenir du fait que l'entête du fichier MP3 ne possède pas les informations dans un format que notre librairie (Taglib) peut gérer, le serveur est donc amené à envoyer dans la file une musique avec des titres et artistes "inconnus".

De plus, nous avons aussi remarqué qu'une erreur de type *Broken pipe* survient très rarement sur certaines musiques bien précises du côté du serveur. Une telle erreur se produit lorsqu'on essaye d'écrire dans un tube qui a déjà été fermé. Cette erreur causée par certaines rares musiques reste inconnue à nos yeux.

En ce qui concerne les performances, on a calculé un temps d'envoi de musique au serveur moyen de 5.76 secondes pour des musiques de longueur comprise entre 1.5 et 7 minutes ce que nous considérons comme acceptable pour le client.

Du côté de l'application Android, lorsqu'on est connecté en tant que superviseur pour effectuer des réglages et que l'administrateur de l'application PC supprime le compte superviseur avec lequel on est connecté, la session n'est pas fermée avant que le superviseur ne se déconnecte. On pourrait interpréter cela comme un problème, car le superviseur n'est pas censé pouvoir apporter des modifications alors qu'il n'est techniquement plus dans la base de données. Cependant, nous le comptons comme étant un choix d'implémentation de notre part. Nous avons décidé qu'une fois qu'un superviseur est connecté, il gardait sa session ouverte puisque l'administrateur du café connaît ses employés et peut facilement refuser l'accès à son superviseur ou lui enlever la tablette.

De plus, pour les paroles des musiques du côté client, nous utilisons deux API qui sont Génius ainsi que Google. Néanmoins, même à avec ces deux API qui constituent une réponse à notre demande de paroles assez solide, il existe des musiques qui n'ont pas la totalité des paroles affichées donc on pense qu'une autre API payante ou ajouter d'autres sites à notre fonctionnement pourrait être une

meilleure solution pour la fonctionnalité “Paroles de chansons” que l’on propose dans notre application.

Du côté de l’application PC, toutes les fonctionnalités sont présentes, mais il serait tout de même possible d’améliorer la qualité du produit rendu. En effet, il se peut que café-bistro veuille faire en sorte que l’administrateur se déconnecte directement lorsqu’il n’est plus capable de communiquer avec le serveur. Nous n’avons pas implémenté cette fonctionnalité puisque ce n’était pas demandé et qu’il était intéressant de toujours voir le journal du serveur même après que le serveur soit fermé, ou qu’il y ait eu une erreur.

Finalement, lors de nos tests de robustesse des interfaces et capacité de gestion des requêtes nombreuses et concurrentes du serveur nous avons réussi à corriger la grande majorité des bugs que nous avons découverts, et nous n’en avons pas trouvé davantage lors de la démonstration du livrable 2.

4 Déroulement du projet (Q2.5)

Tout au long du projet, l’équipe a pu compter sur ses membres pour faire preuve d’une bonne organisation, de rigueur dans le travail, mais également de motivation. C’est ce qui nous a permis d’avoir les fonctionnalités présentes et opérationnelles pour chacune des démonstrations. De plus, nous avons su entretenir une bonne communication ce qui a fait que nous avons pu faire avancer les applications en parallèle, mais de manière cohérente. En effet, dès qu’une fonctionnalité était disponible du côté d’une des applications, on s'assurait de rendre le traitement de la requête disponible du côté du serveur assez rapidement pour faciliter les tests et le fonctionnement, et vice-versa. D’une manière générale, nous avons essayé d’écrire du code de qualité du premier coup pour limiter les heures à passer sur les revues de code et le réusinage de code. Ainsi nous avons dû faire une seule revue de code tous ensemble, car l’équipe faisait suffisamment confiance à ses membres pour la conception et l’implémentation des fonctionnalités.

Par ailleurs, nous avons essayé de faire valider nos conceptions au fur et à mesure du projet en discutant avec les chargés de laboratoire et le professeur. Ce qui nous a permis de nous rendre compte au premier livrable que notre interface de l’application Android ne répondait pas totalement aux requis et de la revoir tous ensemble avant la seconde remise. En outre, nous avons très tôt compris l’utilité des suivis hebdomadaires et nous les avons faits consciencieusement, car ils nous permettaient de faire un bilan et une planification à chaque semaine. Malheureusement, l’élaboration des fichiers de suivi n’était pas suffisante et nous

avons dû faire une rencontre de replanification générale pendant la semaine de relâche.

Dans notre gestion d'équipe, nous avions attribué certains rôles qui nous semblaient importants au départ, mais nous nous sommes adaptés avec les réels besoins de l'équipe et les différentes situations. Par exemple, nous avions prévu faire des scrums tous les jeudis matin, mais nous n'avons pas toujours réussi à respecter cela. Aussi, nous avons dû nous adapter et nous n'avons pas toujours été en mesure de suivre à la lettre les dates des sprints que nous nous étions fixées par rapport aux dépendances entre les tâches et celles qui nous prenaient plus de temps que prévu. En effet, il était difficile au début du projet de cerner entièrement les composantes qui auraient un impact sur d'autres. Ce qui a retardé le déploiement de certaines fonctionnalités et inversement certaines fonctionnalités pour lesquelles nous avions alloué beaucoup de temps au départ se faisaient relativement vite. C'est ainsi que par rapport à notre planification initiale certaines tâches nous ont pris plus de temps, notamment décodage et lecture de fichier audio. Nous avons eu du mal à faire jouer le son à la bonne fréquence et cela nous a amené à devoir changer les fichiers de configurations d'Alsa en plus de faire un réajustement de la fréquence du son lorsque celle-ci est à 44100 KH. Malgré cela, nous avancions relativement bien dans notre planification et nous avons pu livrer tout ce que nous devions livrer à nos dates critiques.

Autre chose que nous aurions aimé améliorer est le fait que nous n'avons pas fait de tests inclus dans notre remise de projet alors que nous avions prévu les faire et nous en connaissions la pertinence. Mais à chaque fois, la priorité était ailleurs. Il fallait faire soit du développement, de l'intégration ou encore du débogage. Il est à noter que nous avons toutefois réalisé des fichiers de tests pour nous même, sur certaines parties du code demandant plus de logique (particulièrement côté serveur), mais qui n'étaient pas assez complets et que nous avons décidé de ne pas inclure dans la remise finale. Nos tests ont donc été répétés manuellement de nombreuses fois, mais pas automatisés.

Enfin, les personnes auxquelles nous avions attribué les tâches dans la planification initiale ne sont pas souvent celles qui ont travaillé dessus en fin de compte. Ceci a créé une confusion initialement avec la notion de responsable de la tâche sur Gantt et nous avons dû clarifier cela. En dépit de toutes ces difficultés, nous avons su mener le projet à bien, car nous apprenions rapidement de nos erreurs et nous avions une bonne capacité de réadaptation.

5 Travaux futurs et recommandations (Q3.5)

Le premier point qui nous a interpellé dès le départ est l'aspect sécuritaire du système qui n'a pas été au centre de notre développement. Bien que ce n'était pas un requis dans le travail qui nous a été demandé, nous pensons que l'on pourrait améliorer ce point par exemple en ayant un réel certificat au lieu de celui que l'on utilise que nous avons dû signer nous-mêmes pour l'utilisation d'Https par les applications qui acceptent alors tout type de certificat ce qui se révèle être une faille de sécurité. De plus, le mode client ne bénéficie pas de la version sécurisée d'Http, ce qui pourrait s'avérer plus intéressant. Particulièrement par rapport au fait que nous ayons découvert que la toute dernière mise à jour d'Android bloquait parfois les requêtes des applications qui n'étaient pas effectuées en Https.
Dans le même esprit, afin de sécuriser et simplifier les requêtes et transmissions d'informations entre les composantes, nous pensons que l'utilisation de socket et de sessions avec Http2 apporterait des possibilités intéressantes. Nous serions alors en mesure d'utiliser des token pour établir une connexion directe entre le serveur et les applications afin de savoir plus simplement qui est en train d'utiliser et sur quel appareil. Ceci ferait en sorte qu'au lieu de recevoir de manière asynchrone de nombreuses requêtes des applications, le serveur serait en mesure de leur transmettre directement les informations importantes en cas de modifications de l'état du système.

Autre chose que nous aimerions ajouter pour améliorer l'expérience utilisateur est la possibilité de voir les paroles de davantage de chansons. Comme mentionné précédemment, nous utilisons deux API, ce qui fait que nous ne sommes parfois pas en mesure d'obtenir toutes les paroles de toutes les chansons. Pour améliorer cela, nous pourrions utiliser plus d'API ou encore payer un abonnement à l'une d'entre elles qui permettrait d'accéder aux paroles d'un plus grand pourcentage de musiques.
Par ailleurs, un obstacle au gain de popularité du produit est le fait qu'il ne soit disponible que sur Android. Le rendre accessible sur IOS ferait que l'application serait plus accessible et utilisable par un plus grand nombre de clients. Nous avons nous aussi été confrontés à cette limitation puisque la majorité des membres de notre équipe ne possédait pas d'Android pour effectuer des tests, étant donné que les performances avec les simulateurs d'Android Studio ne reflétaient pas toujours la réalité.

De plus, nous considérons que nous pourrions compléter notre système avec des actions supplémentaires que pourraient effectuer les superviseurs pour s'assurer de la convivialité et de l'atmosphère du café. Notamment, il pourrait y avoir des options leur permettant de faire pause à la musique, passer à la suivante, même retirer celle en cours en cas de chanson inappropriée, etc. De même du côté utilisateur, l'ajout d'un bouton de recherche dans ses musiques locales pourraient être un élément intéressant. Et enfin, suivre la lecture de la musique en temps réel serait une

fonctionnalité pratique, qui serait probablement plus possible avec l'utilisation de sessions comme mentionné précédemment.

Ensuite, du côté de l'application PC, il serait intéressant d'ajouter une enveloppe sur le projet afin de pouvoir le déployer sur n'importe quelle machine. Une machine n'ayant pas un JRE ne sera pas capable de rouler notre exécutable JAR, mais le sera avec cette enveloppe d'installation des librairies requises. De plus, il nous permettrait de changer le logo par défaut de l'exécutable lors de l'installation, ce qui ajouterait de l'esthétisme à notre application.

Pour finir, le dernier point que nous pensons qu'il serait intéressant d'ajouter à notre projet serait la capacité du serveur de redémarrer et retrouver son état d'avant en cas de cessation subite de son fonctionnement, notamment en effectuant des fichiers contenant des logs plus exhaustifs et sauvegardant de temps en temps des sauvegardes de l'état. Ceci serait également mieux implémenté avec la notion de sessions, car les applications pourraient être tenues informées de la déconnexion du serveur et déconnecter automatiquement les utilisateurs par exemple.

6 Apprentissage continu (Q12)

Léandre :

1. Ma principale lacune se situe parfois sur l'appréciation du code. Lorsque je n'aime pas une partie du code ou qu'il ne fonctionne pas et que je n'ai pas le goût de corriger un bogue qui serait plus rapide de juste recommencer, je supprime le code tout bêtement et je le recommence pour le mettre à ma façon.
2. Je devrais parfois plus coopérer et essayer de garder ce que l'autre personne a fait. Il faudrait aussi prendre rendez-vous avec la personne pour lui indiquer ce qui ne va pas.
3. En indiquant ce qui ne fonctionne pas avec le code, on peut alors faire d'une pierre deux coups. Premièrement, il se peut que le bogue soit corrigé. Deuxièmement, je serai plus disposé à lire le code dans cette condition que lorsque je me retrouve à le lire seul et sans comprendre nécessairement ce que la personne voulait faire.

Constantin :

1. Je pense que je travaille trop tout seul chez moi et peut-être pas assez durant les périodes de laboratoire. Puisque je considère ces périodes comme des temps durant lesquels tout le monde doit se mettre à jour avec les avancées des autres membres. Ce type de fonctionnement que j'adopte aussi parce que je peux être fatigué ou en pleine digestion peut parfois introduire un manque

de communication puisque tout le monde n'emploie pas ce mode de travail. J'ai aussi bien plus de facilité à me concentrer et à faire des tâches complexes lorsque je suis seul chez moi, c'est ce qui m'incite à plus travailler chez moi, voire à travailler de nuit et à être fatigué lors des rencontres en laboratoire.
2. Afin de quand même produire durant les laboratoires, je me suis souvent penché vers du code qui nécessite moins de réflexion afin de ne pas rester bloqué. J'ai aussi utilisé ces périodes pour assister des membres du groupe qui avaient des problèmes particuliers.
3. Pour résoudre cette lacune, il faut que je me force à ne pas travailler de nuit et essayer de dormir plus afin de me forcer à me concentrer durant les périodes de laboratoire. Par ailleurs, je ne devrais pas manger un grand plat juste avant de commencer une session de laboratoire.

Anis :

1. Une de mes lacunes est sans doute que je vais trop vite sans suffisamment penser à l'architecture et à la robustesse lorsque je code. Mon seul but était de faire fonctionner l'application tout en soignant le design front-end, il m'arrive souvent de laisser le back-end sans revenir dessus en termes d'optimisation, réutilisation de code ou autre. De plus, un des problèmes que j'ai rencontré durant cette session est une vérification peu fréquente de notre canal de communication "Slack". En effet, j'ai rarement eu à utiliser cet outil et communication et je demandais sans cesse à mes partenaires quelles étaient les prochaines directives alors que celles-ci se trouvaient sur "Slack".
2. Pour combler la lacune de coder trop vite sans penser au back-end et à la robustesse j'ai eu plus d'une fois à prendre exemple sur Léandre pour améliorer la robustesse de mon code et ma façon de penser pour y parvenir cela m'a été d'une grande aide. Pour ce qui est de "Slack", j'ai installé la version mobile de l'application de communication afin de recevoir les notifications de chaque nouveau message des canaux que je suivais. De plus, je me suis efforcé de ne plus utiliser de réseaux sociaux classiques pour communiquer pendant que mon étude du projet.
3. Une méthode qui aurait pu me permettre d'améliorer ma lacune principale est de trouver différentes manières de coder une fonctionnalité en back-end pour chaque fonctionnalité et d'ensuite choisir laquelle des méthodes trouvées (que j'aurais au préalable fait fonctionner) assureraient une meilleure robustesse du code.

Christella :

1. Au cours de ce projet, je pense que j'ai eu trop tendance à vouloir trop faire seule. Souvent quand il y avait une tâche annexe au projet je me proposais simplement pour la faire alors que les autres membres de mon équipe étaient également présents. Une autre chose que j'ai remarquée est le fait que si je

rencontre un bug au cours de mon développement, j'aurai beaucoup de mal à le laisser de côté si je suis bloquée et passer à autre chose pour y revenir plus tard, ce qui peut parfois faire perdre du temps.
2. Afin de limiter cette charge de travail que je prenais, j'ai essayé de parler davantage avec mon équipe pour savoir si quelqu'un pouvait aider à réaliser telle ou telle tâche si je n'avais pas le temps de le faire. De même, communiquer avec mes partenaires m'a permis de me rendre compte de si je passais trop de temps sur quelque chose alors que d'autres soucis étaient plus prioritaires. Je me suis également moi-même surveillée pour me donner des limites de temps pour réussir à arrêter quelque chose et le laisser dans un état non problématique, même s'il n'est pas complètement fini.
3. Dans l'optique d'améliorer encore plus la première lacune que j'ai citée, il faudrait que je délègue davantage et m'exprime sur ce que je veux ou peux faire. Une meilleure organisation me permettrait sûrement de pouvoir prévoir en amont ce qu'il y a à faire, le temps que ça prendrait afin d'aviser et voir qui serait à même de le faire. Enfin pour le deuxième point que j'ai soulevé de manière générale il faut que je me fixe des priorités et des limites de temps pour savoir quand arrêter une tâche, et savoir aller demander de l'aide avant de perdre trop de temps.

Issifath :

1. Je ne m'exprime pas assez, j'ai des attentes envers les membres de mon équipe et j'attends d'eux qu'ils les devinent alors que ce n'est pas toujours évident. De plus, je ne prends pas de risques, car j'aurais aimé travailler sur l'application Android, mais j'avais peur de ralentir le projet.
2. Pour ne pas me sentir en dehors de l'application Android, je me suis énormément investie dans les tests d'intégration de l'ensemble et je posais des questions sur comment les fonctionnalités avaient été implémentées du côté de l'application client.
3. J'aurais pu améliorer cela en demandant aux autres leurs attentes par rapport à moi, car cela aurait instauré le dialogue et m'aurait aidé à dire les miennes.

7 Conclusion (Q3.6)

Au début du projet, en réponse à l'appel d'offres, nous avons eu à faire une planification des tâches et nous avons réparti les fonctionnalités sur quatre sprints. Nous nous sommes très vite rendu compte que c'était très approximatif et qu'il faudrait faire des ajustements au fur et à mesure. Nous avons eu dès le départ une bonne idée de ce que l'ensemble du projet représentait en matière de temps de conception et d'investissement. Même si à certains moments, nous avons sous-estimé le développement de certaines parties du système, le temps supplémentaire n'est pas aberrant.

Ce que nous retenons c'est qu'il est important, lorsqu'on travaille avec du matériel, comme dans le cas de la carte fpga, de faire tous les tests là-dessus pour éviter d'avoir des surprises de dernière minute. En ce qui concerne les applications, nous avons un peu négligé l'esthétisme de l'interface graphique au début de la conception, car nous voulions surtout que cela fonctionne. Et nous nous sommes rendu compte un peu tard en faisant utiliser notre application par d'autres personnes que nos interfaces n'étaient pas toujours très instinctives. En outre, nous aurions aimé avoir des tests automatisés afin de s'assurer de la disponibilité et fonctionnalité du système après les intégrations et de mettre moins de temps et d'efforts dans les tests de fonctionnement en boite noire.

Néanmoins, nous avons globalement eu une bonne approche de développement et nous avons su prendre du recul et porter un regard critique sur nos savoirs et nos savoir-faire. C'est cela qui nous a permis de surmonter les difficultés rencontrées et d'avoir des applications qui interagissent bien ensemble.

8 Références

Références utilisées autres que celles utilisées par le professeur durant le cours. Un site web utilisé pour la programmation (cadre de travail, outils, librairies) devrait être listé.
…

[1] "Download Android Studio and SDK tools", *Android Developers*, 2019. [En ligne]. Disponible: https://developer.android.com/studio. [Accédée le : 14 avril 2019].

[2] "Volley overview | Android Developers", *Android Developers*, 2019. [En ligne]. Disponible: https://developer.android.com/training/volley. [Accédée le : 14 avril 2019].

[3] "google/gson", *GitHub*, 2019. [En ligne]. Disponible: https://github.com/google/gson. [Accédée le : 14 avril 2019].

[4] E. Foundation, "Eclipse Downloads | The Eclipse Foundation", *Eclipse.org*, 2019. [En ligne]. Disponible: https://www.eclipse.org/downloads/. [Accédée le : 14 avril 2019].

[5] "javax.swing (Java Platform SE 8)", *Docs.oracle.com*, 2019. [En ligne]. Disponible: https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html. [Accédée le : 14 avril 2019].

[6] "MiG Layout Java Layout Manager for Swing and SWT", Miglayout.com, 2019. [En ligne]. Disponible: http://www.miglayout.com/. [Accédée le : 14 avril 2019].

[7] "Create a List with RecyclerView | Android Developers", *Android Developers*, 2019. [En ligne]. Disponible: https://developer.android.com/guide/topics/ui/layout/recyclerview. [Accédée le : 14 avril 2019].

[8] "Corvusoft/restbed", *GitHub*, 2019. [En ligne]. Disponible: https://github.com/Corvusoft/restbed. [Accédée le : 14 avril 2019].

[9] "RapidJSON: Main Page", *Rapidjson.org*, 2019. [En ligne]. Disponible: http://rapidjson.org/index.html. [Accédée le : 14 avril 2019].

[10] A. Wenger, "fasterthanlime/libmad", *GitHub*, 2019. [En ligne]. Disponible: https://github.com/fasterthanlime/libmad/blob/master/minimad.c. [Accédée le : 14 avril 2019].

[11] "taglib/taglib", *GitHub*, 2019. [En ligne]. Disponible: https://github.com/taglib/taglib. [Accédée le : 14 avril 2019].

[12] "zlib Home Site", *Zlib.net*, 2019. [En ligne]. Disponible: https://www.zlib.net/. [Accédée le : 14 avril 2019].

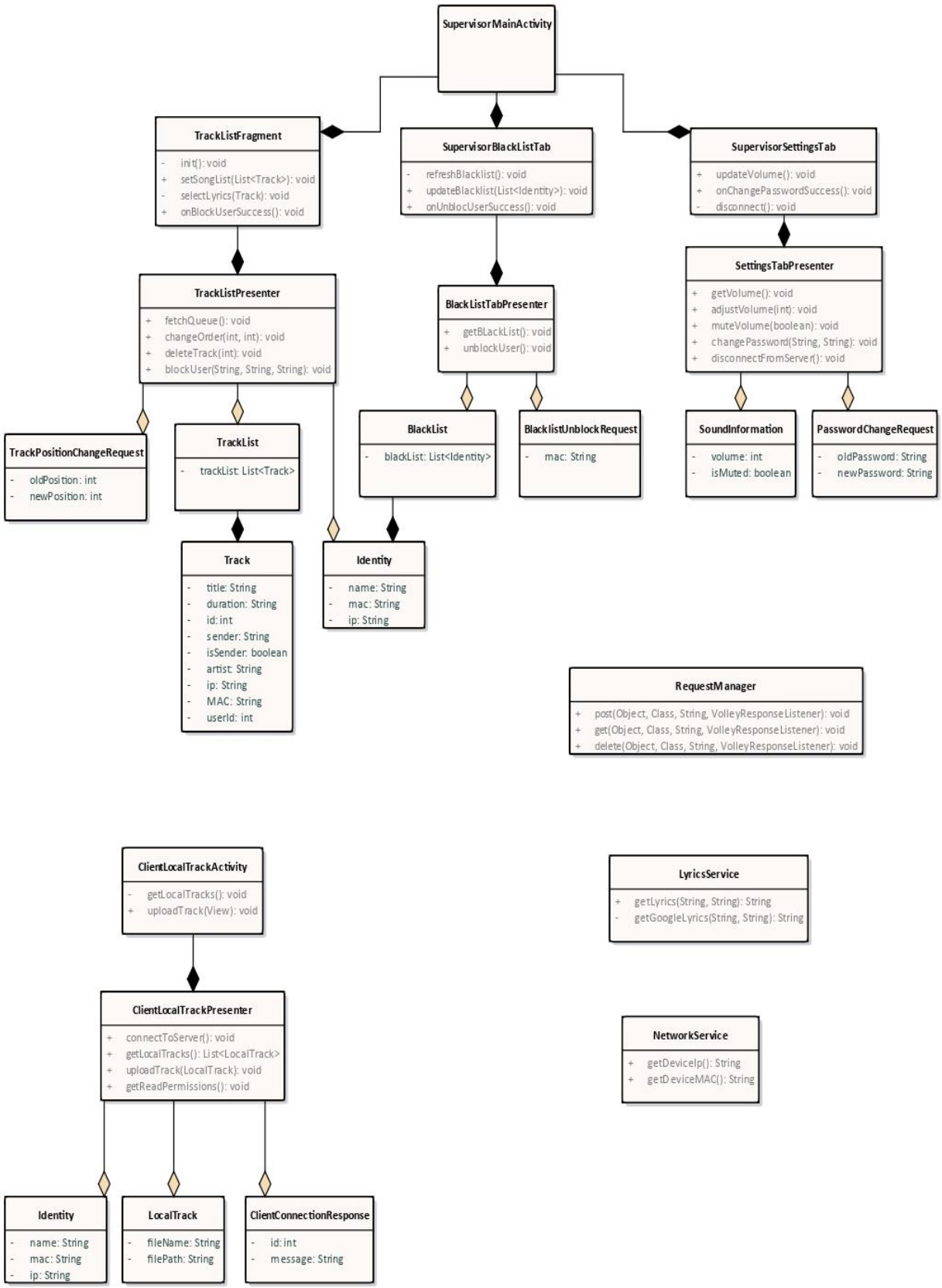
[13] "Where work happens", *Slack*, 2019. [En ligne]. Disponible: https://slack.com/. [Accédée le : 14 avril 2019].

[14] "Stack Overflow - Where Developers Learn, Share, & Build Careers", *Stack Overflow*, 2019. [En ligne]. Disponible: https://stackoverflow.com/. [Accédée le : 14 avril 2019].

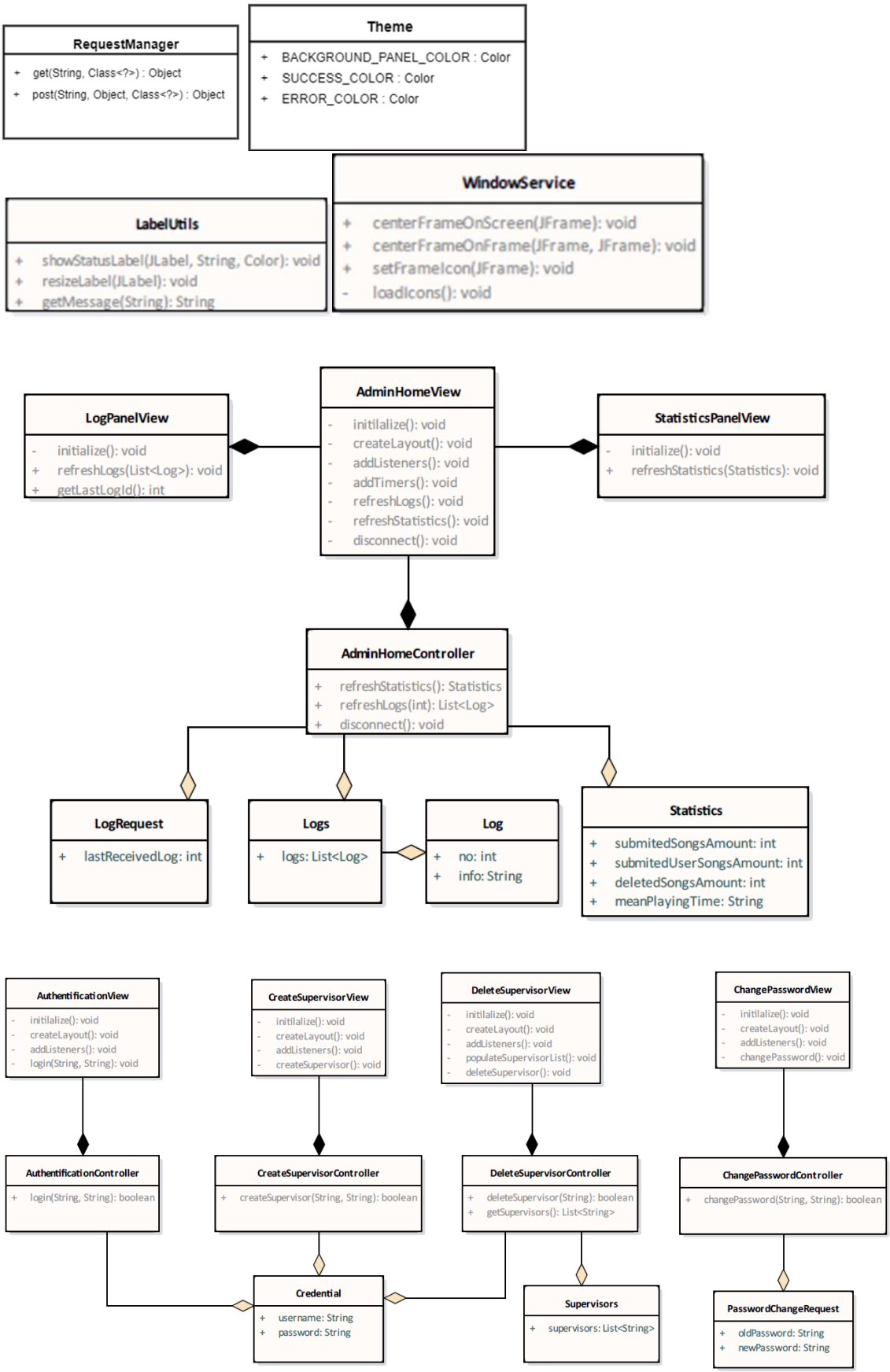
[15] "Medium – a place to read and write big ideas and important stories", *Medium*, 2019. [En ligne]. Disponible: https://medium.com/. [Accédée le : 14 avril 2019].

[16] “Cross-platform Java executable wrapper”, Launch4j, 2019. [En ligne]. Disponible: http://launch4j.sourceforge.net/. [Accédée le : 14 avril 2019].

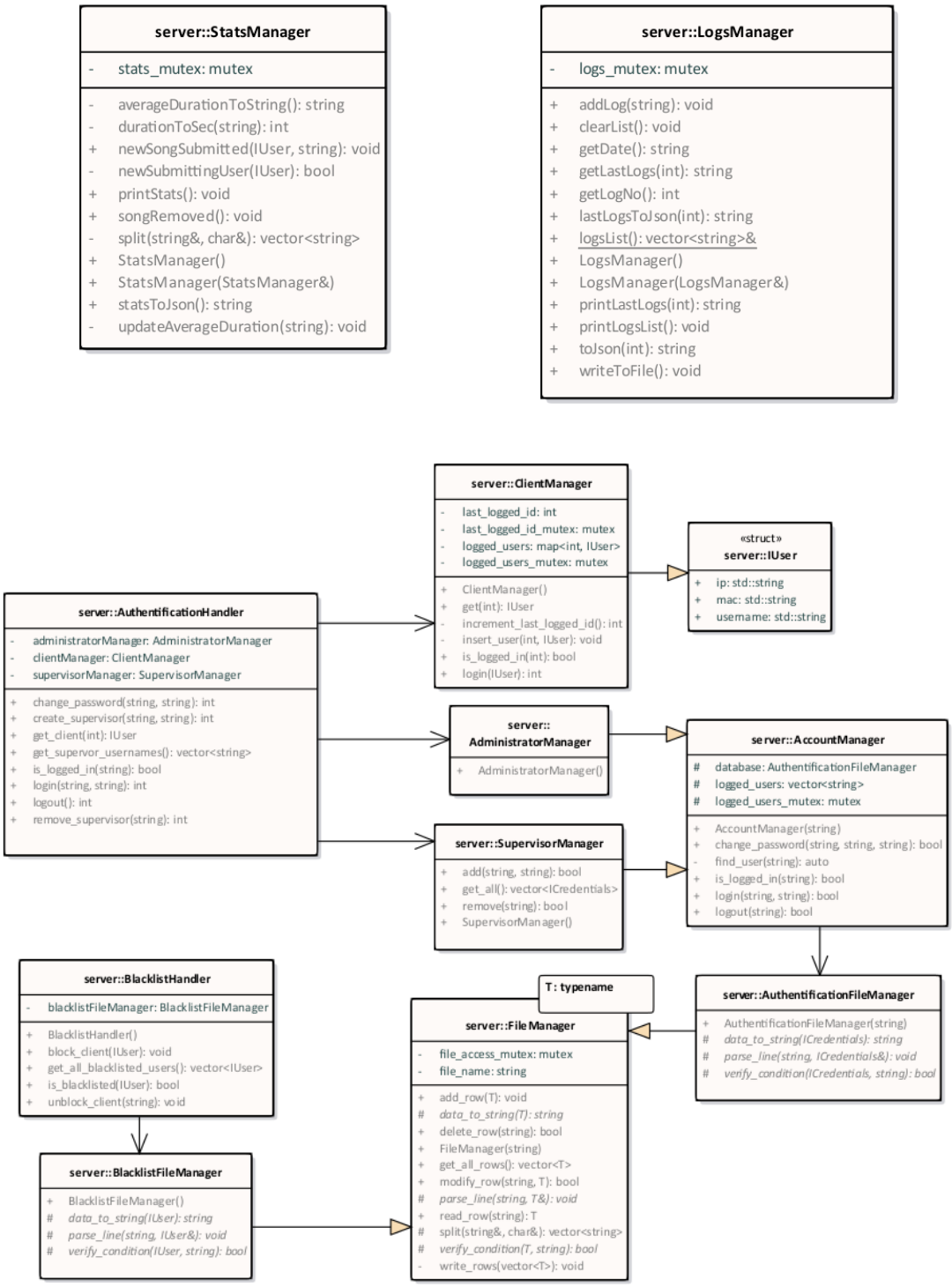
9 Annexe



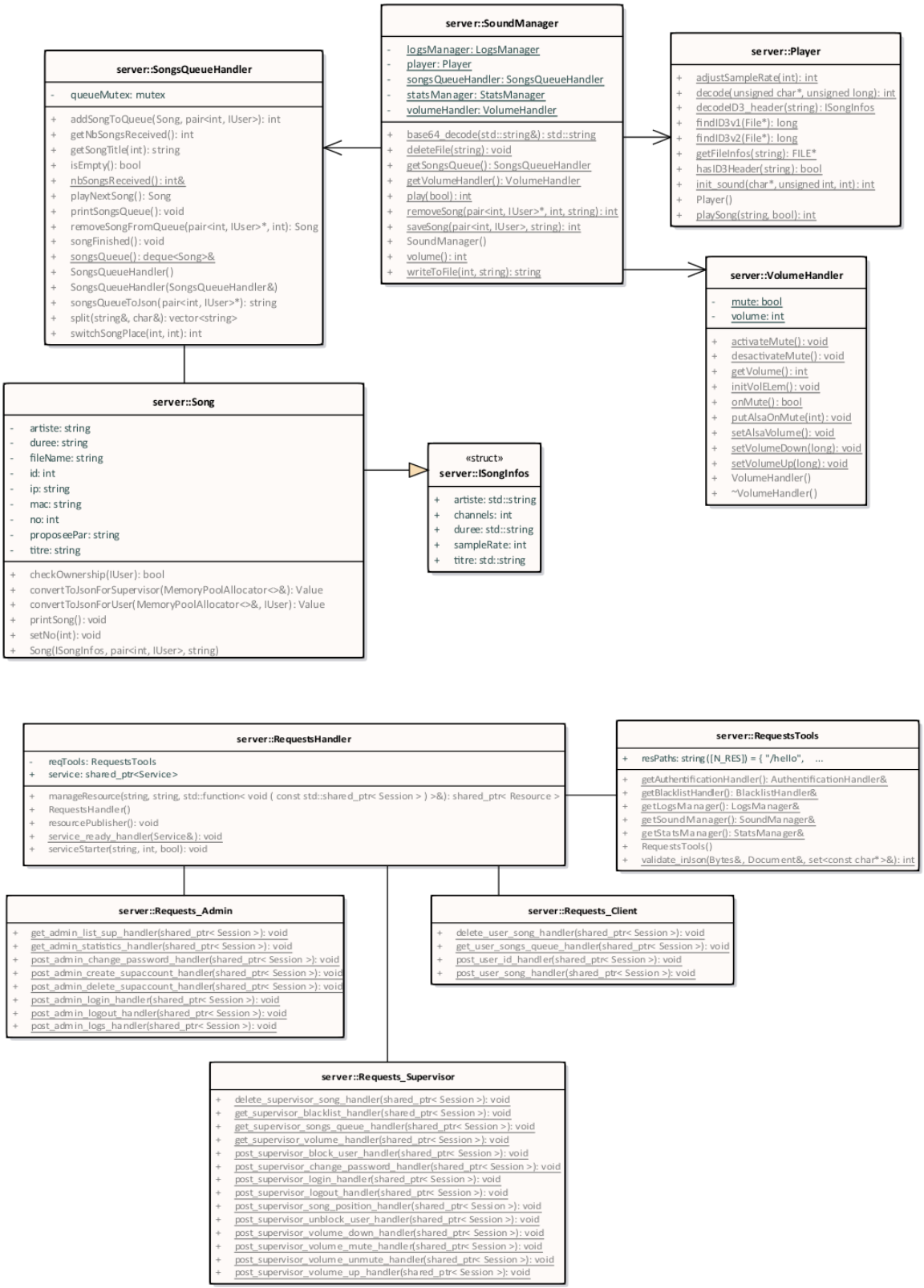
[1] Diagramme de classe de la partie Android



[2] Diagramme de classe de l'application PC



[3] Diagramme de classes du serveur (Modules : Logs/Statistiques et Authentification)



[4] Diagramme de classes du serveur (Modules : Son et Requêtes)