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

Benjamin Heinen

Mohamed-Wassim Guellati

Soukaina Moussaoui

Ibrahima Séga Sangaré

Décembre 2018

Table des matières

1. Objectifs du projet

Le projet *CaféBistro Elevation* a de nombreux objectifs. Il était important que ceux-ci soient respectés afin de satisfaire le client.

Tout d'abord, il fallait que le système développé permette à *Café Bistro* d'attirer une nouvelle clientèle, qu'il ne ciblait pas spécialement auparavant.

De plus, il est important que le système puisse également fidéliser la clientèle déjà existante, afin que le café puisse observer un réel gain de popularité.

Il a donc été décidé de réaliser un système audio, connecté par Internet, qui permet d'envoyer des chansons et ainsi de les partager avec les autres clients du café. Les musiques sont jouées directement sur les haut-parleurs présents chez Café Bistro.

2. Description du système

1.1 Le serveur (Q4.5)

Tout d'abord, le serveur a été développé sur un système *ArchLinux,* en C++.

Afin de parvenir à développer correctement le serveur, nous avons intégré quelques fichiers de configuration. Parmi eux, on en retrouve un qui permet, par exemple, de créer les routes du serveur REST.

De plus, nous avons configuré le serveur de sorte à ce qu'il soit capable de gérer jusqu'à quatre fils d'exécution à la fois. Nous n'avons pas jugé nécessaire d'en mettre plus, car les cartes FPGA contiennent deux cœurs. Elles ne peuvent ainsi utiliser que deux processus physiques, et potentiellement deux autres logiques. Dans la situation où le serveur reçoit plus de quatre requêtes à traiter exactement au même instant, il en met certaines en attente et les traite dès qu'un fil d'exécution est disponible.

Afin d'assurer une excellente consistance des données, en cas de redémarrage de la carte FPGA, nous avons choisi de stocker toutes les données dans des fichiers de métadonnées écrits en JSON.

Nous avons donc utilisé plusieurs technologies sur le serveur, afin de gagner du temps. Nous vous les présentons dans le tableau 1 intitulé *Technologies utilisées dans le serveur*. Néanmoins, nous n'avions pas fait chacun de ces choix dès le départ. Ainsi, lorsque nous avons débuté le projet, nous avions choisi d'utiliser la technologie *mpg123* au lieu de *Libmad* et de *Alsa* afin de décoder puis de lire des fichiers mp3 sur la carte FPGA. Néanmoins, à la suite d'une discussion avec le client, nous nous sommes rendu compte que celui-ci avait finalement plus confiance en *Libmad* et *Alsa*. Nous avons donc fait le choix d'utiliser ces dernières, puisque l'opinion du client nous semblait primordial.

Tableau 1 : Technologies utilisées dans le serveur

Technologies	Raison du choix
Restbed	Multiprocessus et gestion des requêtes sécurisées HTTPS. Librairie flexible qui nous laisse surcharger du code si nous jugeons cela nécessaire.
libmad	Décodage des fichiers *mp3*. La librairie nous offre beaucoup de flexibilité, mais est un peu long à prendre en main en contre-partie.
TagLib	Lecture des entêtes ID3 des fichiers MP3. Très simple à prendre en main
Alsa	Gestion du volume et de la lecture de la musique. Cette librairie est un petit peu compliqué à prendre en main, mais elle nous offre en contrepartie beaucoup de possibilité.
RapidJson	Permet la lecture et l'écriture de fichier JSON. Simple à prendre en main et possède une communauté très active. Cet outil est également très bien documenté, ce qui facilite son apprentissage.

Nous avons également fait le choix de gérer la lecture de la musique dans un micro-service, ce qui n'avait pas été envisagé dans la réponse à l'appel d'offre. Ce choix nous permettait d'assurer une meilleure robustesse à l'application. En effet, si un problème se produit dans le micro-service qui joue la musique, cela ne parviendra pas à provoquer une erreur du serveur. Le serveur continuera à tourner correctement et lancera la musique suivante prévue selon la liste de musiques disponible. Un fil d'exécution est utilisé afin de rechercher continuellement si la liste contient une musique et, le cas échéant, démarre le micro-service.

L'architecture de serveur fut également modifiée par rapport à l'architecture pour laquelle nous avions optée originellement. Celle-ci était au départ était adaptée à un serveur qui ne fonctionnait que sur un seul processus. Néanmoins, comme nous avons fait le choix de rendre l'application parallélisable, permettant ainsi à plusieurs clients d'effectuer des requêtes au même instant, il nous a fallu repenser l'architecture du code afin qu'elle soit capable de gérer différents processus et de synchroniser correctement les données. Par exemple, que se passerait-il si quinze clients différents envoyaient au même instant une musique, en sachant que la liste de musiques ne peut contenir que dix musiques différentes ? Il a donc fallu instaurer de nombre mutex afin d'assurer que les zones critiques du serveur s'exécutent correctement. Il nous a fallu également utiliser ces mêmes mutex afin de gérer la lecture et l'écriture des fichiers de métadonnées. En effet,

si deux processus écrivent en même temps dans un fichier de métadonnées, cela a pour effet de le vider complètement.

Par la suite, nous avons dû nous concentrer sur l'optimisation de la gestion des requêtes REST, c'est-à-dire, comment le serveur doit-il réagir lorsqu'il reçoit une requête REST ? Tout d'abord, au démarrage, le serveur va, à l'aide du fichier de configuration *routes.json*, créer toutes les routes qui permettront au client d'accéder aux différents service web. Par la suite, comme vous pouvez le constater dans la figure 1 'Architecture du serveur', lorsque le serveur recevra une requête, elle sera envoyée à un répartiteur. Il existe trois types de répartiteur : un pour les musiques, un pour les usagers et le dernier pour les statistiques. Chacun des répartiteurs appellera son gestionnaire afin d'effectuer la fonctionnalité voulue. Les gestionnaires auront la charge de gérer les mutex, assurant ainsi la cohérence et la synchronisation des données.

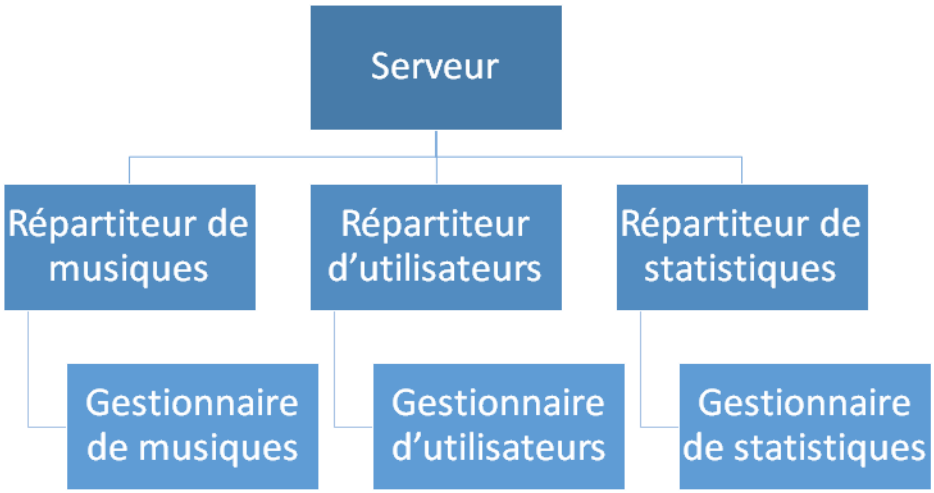

Fig. 1 : *Architecture du serveur*

Enfin, il est très important que le serveur ne puisse pas cesser son exécution en raison d'une erreur. C'est pourquoi il a fallu implémenter un système d'exception rigoureux. Celui-ci est présenté dans la figure 2 intitulé 'gestion des exceptions'. On constate donc que lorsqu'un gestionnaire rencontre une erreur, il en informe son répartiteur. Le répartiteur aura alors pour rôle d'informer le client, en lui envoyant un code d'erreur déterminé selon les spécifications du projet.

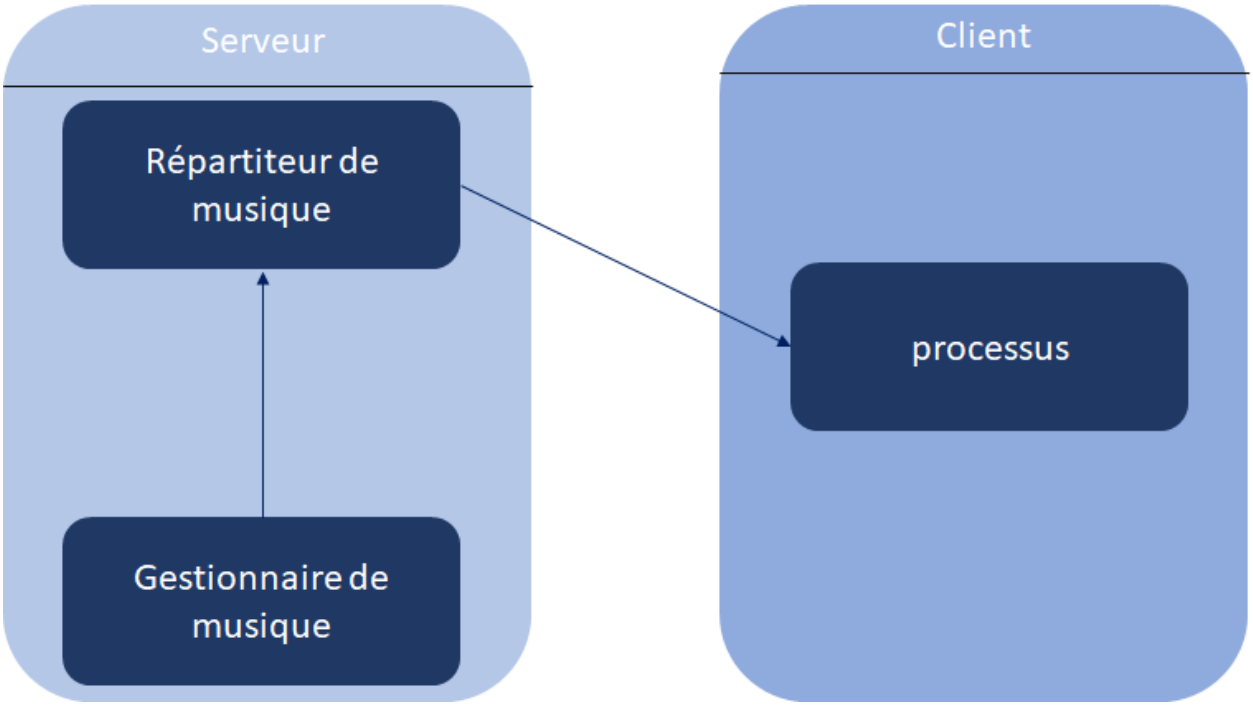

Fig. 2 : *Gestion des erreurs*

1.2 Tablette (Q4.6)

Tout d'abord, il faut savoir que nous avons choisi de répartir notre application sur trois activités. Nous allons donc procéder par étape en expliquant de quels modules est composée chacune de ces activités, comment ils sont agencés, et surtout, quelle est leur utilité.

1. L'identification :

Premièrement, la première activité sur laquelle le client portera son regard, celle de l'identification :

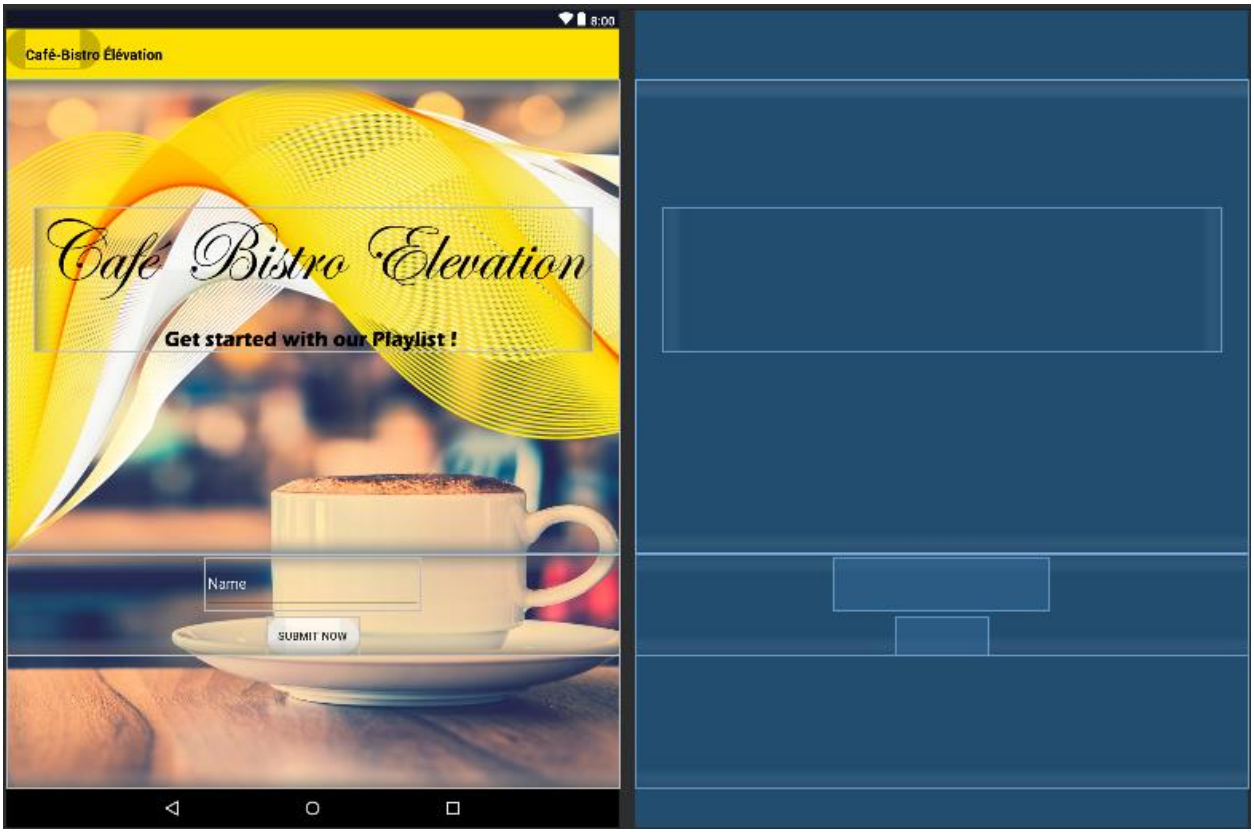

Fig. 3: *Activité d'identification (vue et plan)*

L'interface dédié à l'usager :

Il faut savoir que pour toute l'interface qui suivra, la librairie utilisée est celle d'Android, c'est-à-dire, *Android.support:design*.

Depuis l'image, il est nous possible de voir comment sont agencés les composants que comporte cette activité. En premier lieu, la première chose sur laquelle se fixeront chacun des éléments qui constituent notre vue est un *Linear Layout*. C'est aussi ce plan qui reçoit le fond d'écran qui recouvre l'activité. Par la suite, nous y avons inclus des *ImageViews*. Ce sont le titre de l'application ainsi que l'ondulation en jaune. Pour attirer l'attention du client, nous avons ajouté à ces deux images des animations. Pour ce faire, nous avons créé un répertoire de ressources spécialisé nommé *anim*, dans lequel on retrouve des fichiers au format .xml qui décrivent en détails le comportement de l'animation en question.

Ensuite, nous avons ajouté un champ de texte, le composant chargé de cela se nomme *EditText*. C'est lui qui se verra récupérer l'identifiant du client avant de le transmettre au serveur.

Enfin, un bouton de type *Floating Action Button* dont nous avons redéfini le design à l'aide du fichier button_login_style.xml.

En ce qui concerne la communication avec le serveur que nous avons cité il y a peu, la librairie que nous avons choisi d'utiliser est *Volley*. Pour ce faire, nous avons jugé efficace de réaliser un Singleton que nous avons nommé *CommunicationRest*. C'est ce dernier qui se chargera d'établir une connexion avec le serveur avec un lien url prédéterminé avant de transmettre les données recherchées en utilisant la classe *JsonObjectRequest.* Pour ce faire, à la création, le Singleton *CommunicationRest* reçoit en paramètre l'url de la requête, le type verbe REST de la requête (*DELETE, POST, GET* par exemple), la vue du composant de l'activité qui fait appel au *Singleton* (afin de pouvoir récupérer le contexte par la suite), et enfin un *Component* qui implémentera une interface réalisée par nos soins, l'interface *ComponentsListener*. Implémenter cette interface consiste à posséder une méthode *update().* Cette méthode sera d'une grande utilité pour le Singleton, car en effet, lorsqu'une réponse à la requête de la part du serveur est reçue, c'est la méthode *update()* qui sera appelée afin de traiter les données de la réponse selon nos besoins.

Dans le cas de notre page d'identification, lors de l'appui sur le bouton *Submit now* le composant chargé de traiter la réponse autorisant le client à accéder au contenu de l'application n'est autre que ce bouton-ci. En définitive, notre « Floating Action Button » implémente l'interface *ComponentsListener*, sa méthode *update()* choisit de permettre au client d'accéder au reste de l'application seulement si la réponse du serveur lui est favorable.

L'interface dédiée à l'administrateur :

En effet, l'interface dédiée à l'administrateur figure sur cette même Activité :

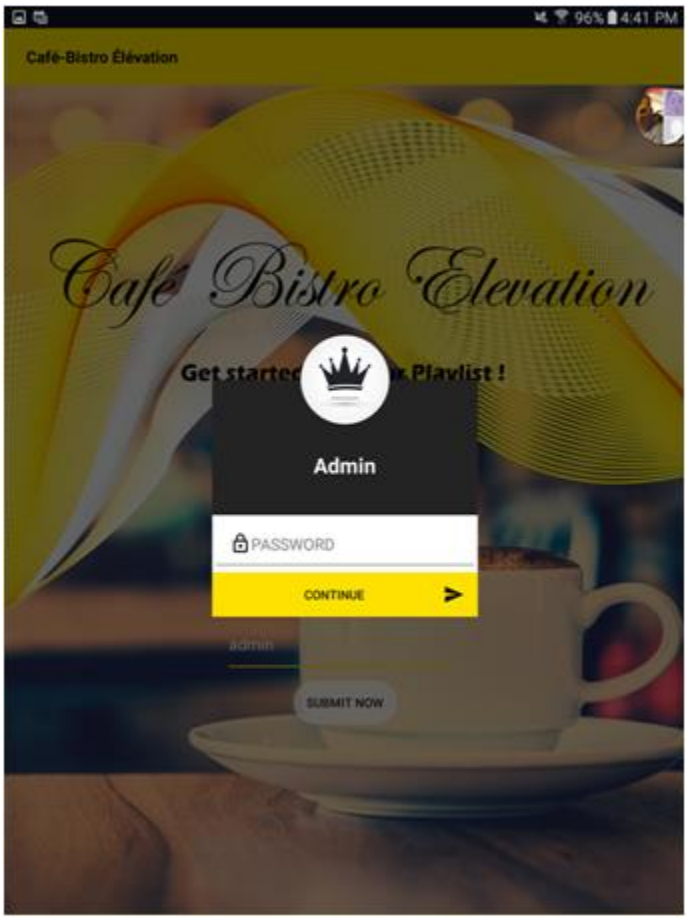

Fig. 4: *Identification de l'administrateur*

La clé de l'interface permettant à l'administrateur de se connecter est un composant nommé *Dialog*. Nous avons ajusté l'apparition et la disparition de ce composant afin qu'il dispose lui aussi d'animations. En effet, ce *Dialog* n'apparaîtra à l'utilisateur seulement s'il choisit de poursuivre dans l'application en ayant rempli le champ identifiant par admin.

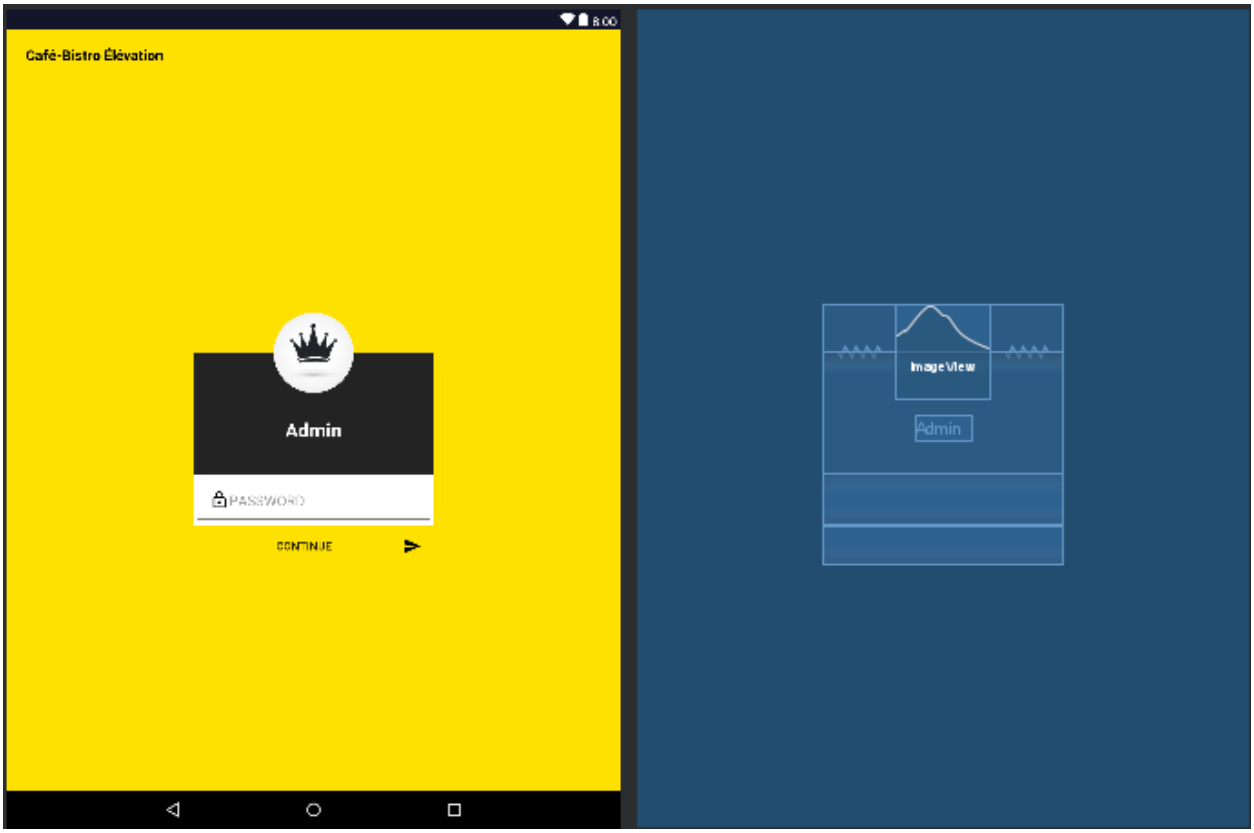

Fig. 5: Composant d'identification administrateur (vue et plan)

En premier lieu, le plan qui recouvre l'intégralité de ce composant est un *RelativeLayout*. Nous avons ajouté à ce dernier une *ImageView*, celle de la couronne qui dépasse volontairement vers le haut. Aussi, nous avons inclus un *TextView* qui fait apparaître le texte *Admin* à l'écran. Enfin, ici aussi, un composant de type *EditText* et un autre de type *Floating Action Button* ont été insérés afin de permettre à l'administrateur d'inscrire son mot de passe avant de l'envoyer au serveur pour vérification.

2. L'activité principale :

Après être parvenu à s'authentifier, l'utilisateur accédera à la vue principale de notre application, une activité sur laquelle nous avons inséré un grand nombre de composants pour offrir la navigation la plus fluide possible.

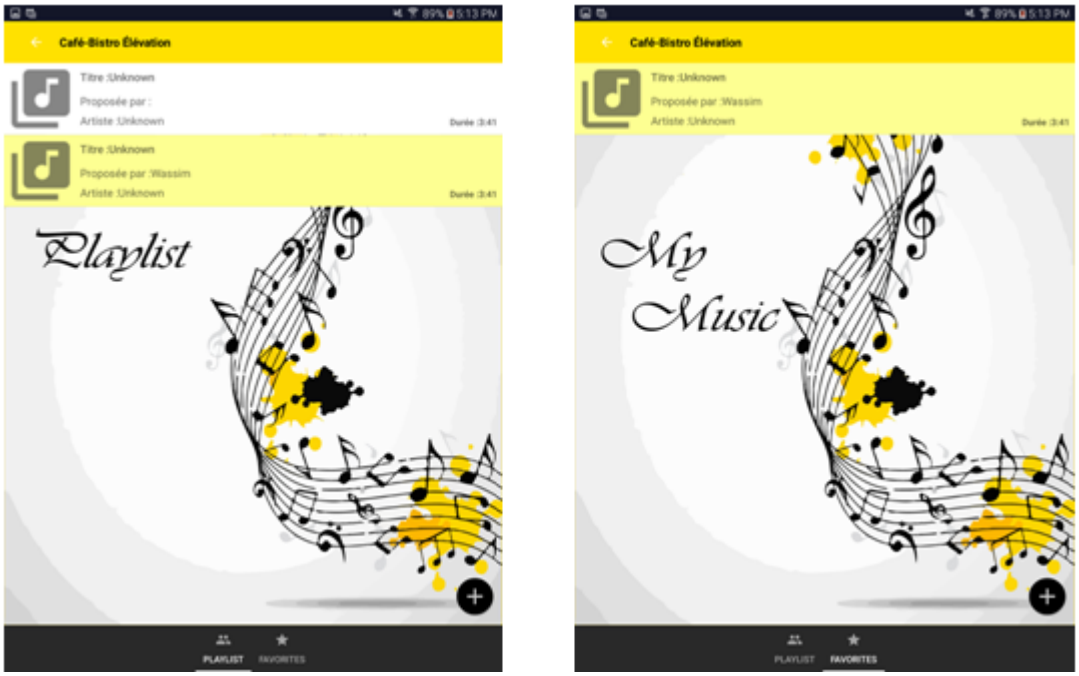

Fig. 6: *Interface de la liste de musique pour un usager normal*

Pour parvenir à offrir cette fluidité dans la navigation, nous avons implémenté un système de navigation par ce qu'on appelle des fragments. Pour ce faire, l'activité fait appel à chacune des classes suivantes :

ViewPager : Cette classe se charge de l'affichage du fragment en permettant à l'utilisateur de passer d'un fragment à un autre d'un simple glissement du doigt dans la direction recherchée.

ViewPagerAdapter : Cette classe désigne le bon fragment à afficher en fonction de la situation.

Fragment : Cette classe s'accapare une section de l'Activité et tout comme l'activité, des composants peuvent s'y fixer afin de produire les vues escomptées. Un fragment peut être créé et supprimé à volonté. Ces fragments ont la particularité de permettre à notre « Activité » de disposer d'une interface multi-volets.

TabLayout : Permet d'accéder directement à la vue d'un fragment recherché, cette classe interagit directement avec le *ViewPager*.

RecyclerView : C'est cette classe qui va représenter notre liste de musiques, elle permet aussi de disposer des musiques d'un simple glissement du doigt.

FloatingActionButton : Ce bouton permet de naviguer vers une autre « Activité », celle chargée de l'envoie des fichiers au format mp3.

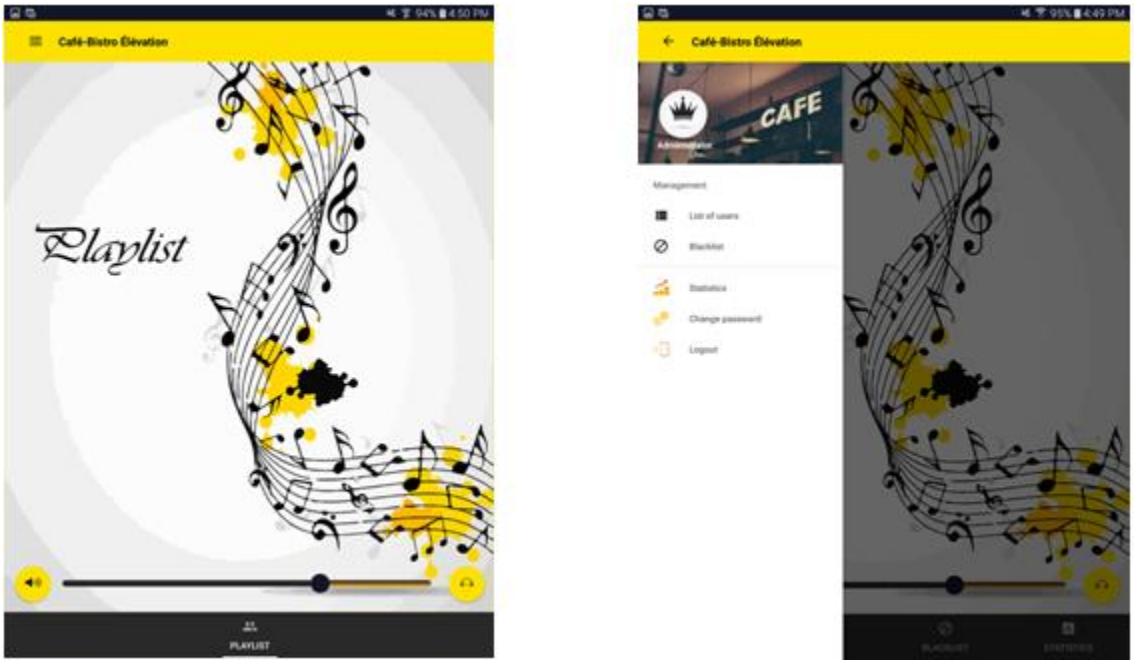

Fig. 7: *Interface de la liste des musiques pour l'administrateur*

Ce qu'on retrouve comme composant dans cette interface de nouveau est:

SeekBar : Permet d'ajuster le volume d'un simple glissement du doigt, en fonction de la progression de la barre, une valeur est récupérée pour ensuite être transmise au serveur.

NavigationView : Ce composant nous a permis d'implémenter un menu disposant de toutes les options nécessaires à l'administrateur, il est directement lié au composant *ViewPager*. En d'autres mots, lorsque l'une des options est pesée du doigt, le fragment correspondant sera créé, l'utilisateur sera redirigé dessus, et le nouvel onglet correspondant sera désormais accessible depuis le composant *TabLayout*. Pour réaliser ce menu, nous avons ajouté une ressource au format .xml au répertoire Menu, c'est ici que nous y avons ajouté l'ensemble des *items* qui forment notre menu. Chaque *item* représente une option mise à la disposition de l'administrateur.

Toggle : Ce composant comporte deux rôles essentiels, il interagit directement avec la *NavigationView*. En effet, il est aussi possible d'ouvrir le menu en pesant sur ce composant que l'on retrouve dans la barre de l'application. De plus, il assure aussi le rôle d'indicateur, en fonction de son icône. Il nous est possible de savoir qu'un menu existe, ainsi que son état, c'est à dire s'il est défilé et donc visible ou bien caché et donc invisible pour l'usager.

3. L'explorateur de fichiers mp3 :

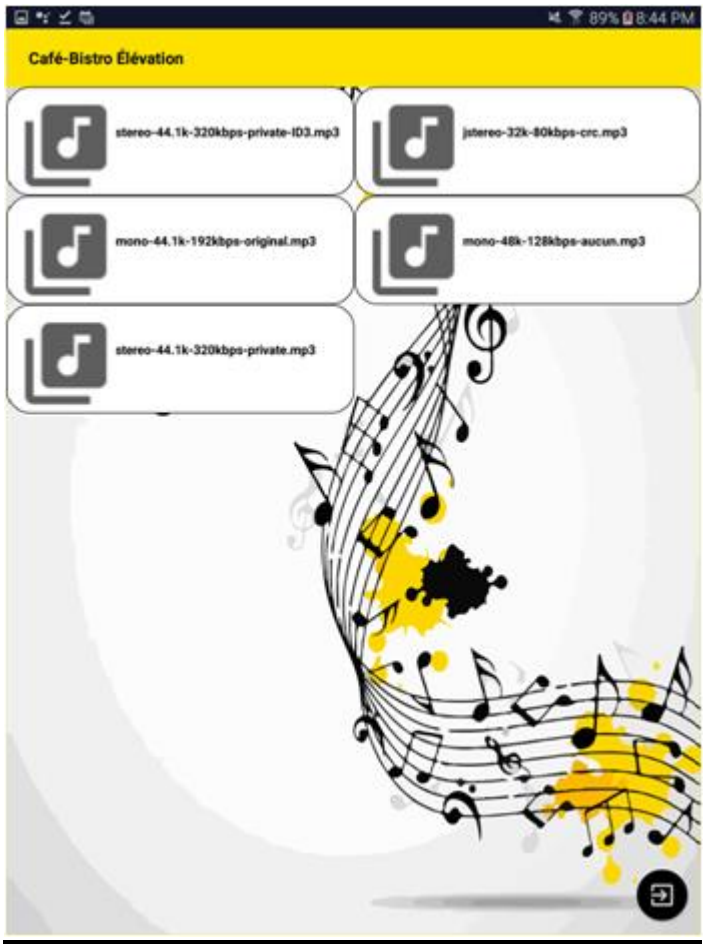

Fig. 8: *Explorateur de fichier mp3*

Cette activité servira affichera à l'utilisateur l'ensemble des fichiers au format mp3 présents sur l'appareil sous la forme d'une liste. Voici les composants que l'activité comporte :

RecyclerView : Ce composant permet d'afficher une liste d'objets, dans notre cas, nous lui transmettons une liste de tous les fichiers au format .mp3 présents sur l'appareil. De cette manière, l'utilisateur peut choisir de les envoyer sur la liste des musiques présentes sur le serveur en pesant du doigt.

FloatingActionButton : Ce composant avec comme icône une porte de sortie offre à l'usager la possibilité de quitter l'activité en cours afin de revenir vers l'activité parente, dans notre cas, c'est à dire vers l'activité principale qui contient la liste de musiques.

1.3 Fonctionnement général (Q5.4)

D'une part le fonctionnement du système sur le serveur se passe sur la carte FPGA. Il faudra posséder une carte SD formatée, comme il a été fait au début du projet, pour accepter le système d'exploitation *ArchLinux*. Ensuite, il faudra se connecter via SSH sur la carte FPGA, en ayant inséré la carte SD dans le port destiné à cet effet au préalable. Le logiciel *Putty* peut être utilisé pour établir cette connexion en indiquant: le type de connexion (SSH) et l'adresse IP utilisée. Une fois dans l'environnement *ArchLinux* visible sur le terminal fourni par *Putty* et à partir du moment où le répertoire du projet se retrouve sur la carte SD, et que l'utilisateur se trouve dans le dossier INF3995-Server, la commande suivante permet de lancer le serveur: *./server*. Par défaut, le serveur s'attend à des requêtes avec l'adresse IP suivante: 132.207.89.35:80 pour les requêtes HTTP et 132.207.89.35:443. Cependant, il est possible de modifier l'adresse IP ainsi que le port réservé pour les requêtes HTTP en suivant le format de commande suivant: *./server* [adresse IP] [port]. Le serveur est alors prêt à recevoir des requêtes de la tablette.

D'autre part, la tablette doit également posséder l'application Android pour faire fonctionner le tout. Lorsque le correcteur se trouvera dans le dossier INF3995-Android, il lui faudra ouvrir le dossier dans Android Studio. Il a le choix de lancer le projet en mode débogage ou sans débogage. Il se peut que certaines erreurs de compilations surgissent pendant le lancement et la phase *Gradle Running*. Cela peut être corrigé en faisant un *rebuild* du projet si nécessaire en cliquant sur le menu *Build* et l'option *Rebuild project*, en haut à gauche, proposé par Android Studio. Lorsque la compilation sera terminée, il sera alors possible d'observer le lancement

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

À présent que le livrable final est terminé, il est important de savoir ce qui fonctionne et ce qui ne fonctionne pas.

Toutes les fonctionnalités présentes dans les spécifications sont fonctionnelles, que ce soit vis-à-vis du client ou du serveur. Ainsi, pour le serveur, on retrouve chaque route qui retourne les résultats escomptés, que ce soit des requêtes encodées (HTTPS) ou non (HTTP). Afin de nous assurer du bon fonctionnement des fonctionnalités, nous avons utilisées des tests qui vérifiaient les routes du système. Les tests vérifient que les erreurs se produisent bien lorsqu'elles doivent se produire ou que le résultat est bien retourné, le cas échéant. Afin de nous assurer du bon fonctionnement durant toutes l'avancée du projet, nous avons utilisé *Travis*. Comme vous pouvez le voir dans la figure 9 intitulée *Validation des fonctionnalités avec Travis et GitHub*, on remarque que le développement d'une fonctionnalité est un succès ou non selon si les tests se passent correctement, avec une coche verte, ou s'ils rencontrent une erreur avec une croix rouge.

Fig. 9 : *Validation des fonctionnalités avec Travis CI et Github*

Néanmoins, au-delà du fonctionnement prévu par les spécifications, nous avons rencontré une lacune au niveau des performances. En effet, nous nous sommes rendus compte d'un bug sur la tablette *Android* : l'usage d'une animation peut créer une fermeture de l'application si l'utilisateur essaie de la provoquer plusieurs fois très rapidement. Il est assez simple de corriger cette erreur. Une solution serait de rajouter un délai avant de pouvoir provoquer une nouvelle animation. Afin de nous assurer que cela ne se produise plus, nous pensons qu'il est important de se concentrer davantage sur les test end-to-end qui, bien qu'ils aient été faits, étaient le meilleur moyen pour détecter ce type d'erreur.

4. Déroulement du projet (Q2.5)

Concernant la gestion du projet, de façon générale, les tâches ont été réalisées telles que prévues selon l'échéancier établi au départ. Tout au long du projet, nous avons fait preuve d'une bonne communication, sans avoir de difficultés à suivre l'avancement des tâches de tous les membres de l'équipe. Grâce aux différents outils d'intégration continue, notamment des *pull requests* sur GitHub, nous étions tous mis à jour sur l'avancement des différentes tâches et nous vérifions et faisions différentes propositions à chaque soumission de code. En outre, étant donné nos différents cheminements académiques, nous avons su faire de bons compromis avec les nombreux travaux des autres cours et nous avons pu soumettre nos livrables dans les temps. D'autre part, du point de vue technique, nous avons fait le choix d'une bonne architecture par rapport à celle pensée dans notre appel d'offre. Effectivement, comme nous l'avons mentionné ci-haut, le choix d'une gestion de musiques à travers un micro-service, d'un serveur multiprocessus et la bonne gestion des requêtes REST ont fait en sorte que le serveur de notre application soit plus robuste et consistant par rapport à celui pensé initialement.

Toutefois, nous avons trouvé certains points à améliorer, notamment du point de vue technique. En premier lieu, même si de nombreux tests ont été effectués, certains auraient pu être améliorés, surtout du côté client. En effet, très peu de fois nous avons remarqué des arrêts soudains dans l'application, dont certains, comme les animations

pour passer d'un fragment à un autre sur l'application mobile, auraient pu être évités à l'aide d'un ajout de délai. En l'occurrence, l'ajout de quelques tests aurait pu nous guider pour la résolution de ces problèmes.

En résumé, de façon générale, le projet s'est très bien déroulé. Certaines modifications techniques ont été apportées afin de livrer un projet plus robuste et consistant. En outre, tous les membres de l'équipe ont eu l'opportunité de travailler sur les deux plateformes : client et serveur, ce qui a fait en sorte que le projet soit encore plus éducatif.

5. Travaux futurs et recommandations (Q3.5)

Tout d'abord, puisqu'un bug graphique a été détecté sur la tablette Android, il est évidemment très important de prendre le temps de le corriger lors de futurs travaux. De plus, certaines interfaces pourraient être améliorées graphiquement. Par exemple, on pourrait un composant graphique qui nous informe qu'une musique est en cours d'envoi. Actuellement, on est informé que la musique est bien envoyée uniquement lorsqu'on la voit apparaître dans la liste des musiques.

De plus, nous pensons qu'il serait intéressant pour *CaféBistro* d'inclure de nouvelles fonctionnalités à celles déjà existantes. Ainsi, il pourrait par exemple proposer un service similaire, mais avec des vidéos. Cela permettrait aux clients de partager une vidéo sur les sorties vidéo du café.

De plus, des sockets pourraient être instaurés afin de gérer les communications entre le client et le serveur. En effet, le client doit actuellement communiquer avec le serveur toutes les cinq secondes afin de voir s'il y a de nouvelles musiques dans la liste de musique. Une communication par socket sera plus intéressante dans cette situation, puisque le serveur pourrait notifier tous les clients qu'il y a une nouvelle musique dans la liste, tout en leur envoyant la liste au même moment. Cela aurait alors pour résultat de diminuer le nombre de communications réseau faites, ainsi que le nombre de processus occupé par le serveur, puisque chacune des requêtes occupent un processus du serveur pendant une certaine durée.

Nous nous sommes également rendu compte qu'il serait intéressant d'étendre l'application à d'autre appareil que les tablettes *Android Samsung S2*. En rendant l'application utilisable par d'autres appareils, mais également par d'autres systèmes d'exploitation tel que iOS, cela augmenterait les possibilités du *CafeBistro* d'attirer de nouveaux clients par la même occasion.

6. Apprentissage continu (Q12)

Benjamin Heinen:

Dès le début du projet, je me suis rendu compte que j'avais des lacunes en C++. Bien que j'utilise ce langage régulièrement dans mon cursus, le développement d'un projet en C++ est bien plus compliqué que l'écriture de quelques fonctions, que ce soit au niveau des configurations, des librairies ou du code en lui-même. Afin de résoudre ce problème, j'ai commencé à lire, dès le début du projet, un ouvrage expliquant les fonctionnalités avancées du C++ suite aux différentes mises à jour. J'aurais pu améliorer mon expertise

du C++ en prenant le temps de développer un peu plus en C++ sur des projets personnels, car la pratique correspond à une grande partie de l'apprentissage d'un langage.

Soukaina Moussaoui:

Avant le projet, je n'avais jamais développé en Android. Pour cela, au tout début, il m'a fallu un peu de temps pour assimiler l'architecture d'une application Android et le lien entre ses différents modules. Avec quelques pratiques à l'aide des tutoriels sur Internet, ces lacunes ont fini par s'estomper. Grâce au critère que notre équipe a maintenu et qui était que chaque membre de l'équipe devait explorer la partie client et serveur, je me suis forcée à sortir de ma zone de confort : le C++. J'ai senti une grande amélioration dans ma façon d'apprendre et le fait de ne pas être spécialisée à un seul côté m'a appris à être plus indépendante pour résoudre les problèmes liés à l'application. Finalement, afin d'améliorer mon expertise, il aurait été préférable de m'entraîner avant que le projet commence, chose qui aurait pu aussi me faire gagner du temps durant le projet.

Mohamed Wassim Guellati:

Au commencement du projet, les premières lacunes auxquelles j'ai dû faire face étaient liées à la maîtrise de la plateforme de développement Android Studio. Par la suite, il s'agissait de parvenir à comprendre le fonctionnement de chacun des composants nécessaires à l'application. Pour y remédier, en plus de l'approche par lecture de la documentation Android, pour être plus efficace, j'ai opté pour le visionnement de vidéos tutoriels et de guides qui ont considérablement accéléré mon apprentissage.

En ce qui concerne la partie liée à notre serveur, la difficulté majeure consistait à la maîtrise des librairies que nous avons utilisées afin de les implémenter comme il se doit. Le problème se solutionnait de lui-même grâce à la pratique continue et répétée de ces dernières.

Ibrahima Séga Sangaré:

En termes de lacunes, il a fallu utiliser les outils mis à disposition pour développer une application Android. Grâce aux ressources comme *OpenClassrooms* et le site officiel d'Android, j'ai pu mieux comprendre comment l'architecture Android était bâtie. J'ai aussi participé aux premières esquisses de l'explorateur de fichiers. Cela m'a permis de travailler sur l'envoi de musiques et utiliser la librairie *Volley* pour les requêtes vers le serveur. Aussi, j'ai participé à la création des composants de l'interface du son pour situer les objets dans l'interface, mettre à jour l'interface selon les réponses du serveur, et faire communiquer des composants ensemble. Utiliser la librairie *restBed* pour traiter les requêtes envoyées m'a permis de mieux comprendre le protocole HTTP comme la récupération et l'envoi de contenu. L'utilisation des librairies *Boost*, *Taglib* et *Alsa* ont été très fructueux. Les aspects à améliorer porteraient sur une plus grande maîtrise du développement sous Android, en y dédiant plus de temps.

7. Conclusion (Q3.6)

Finalement, lorsque nous fait la réponse à l'appel d'offre, nous avions mal évalué certains aspects, que ce soit par rapport au client ou au serveur.

Par rapport à l'architecture du serveur, nous avons développé notre architecture en partant du principe qu'elle ne devait fonctionner qu'avec un seul processus. Néanmoins, nous avons changé d'avis durant le projet à ce propos lorsque nous nous sommes rendu compte que l'on pouvait produire quelque chose de plus performant en utilisant plus de processus. Bien sûr, cela nous a également demandé un important travail de refactorisation, à commencer par repenser complètement l'architecture du serveur.

En revanche, l'architecture que nous avions conçu pour le client était bien pensée, bien qu'incomplète. Elle nous a permis de partir sur de bonnes bases, mais celle-ci auraient pu être plus solide si l'on avait fait un diagramme de classe pour l'application *Android* dans la réponse à l'appel d'offre.

De plus, certaines des technologies que nous avions choisies d'utiliser ont dû être changées, que ce soit après une discussion avec le client ou parce que l'on s'est rendu compte qu'elle ne répondait pas correctement à notre problématique.

Ainsi, à présent que le système est complété, nous pouvons affirmer que notre démarche de conception aurait pu être fortement améliorée. En effet, c'est lors de notre démarche de conception que l'on aurait dû faire attention aux diverses contraintes techniques qui sont apparues par la suite du projet. C'est également à cette étape que nous aurions dû prendre en compte les améliorations possibles de l'architecture, en plus d'y penser au fur et à mesure du déroulement du projet. On se serait alors rendu compte que le serveur devait être capable d'exécuter plusieurs fils d'exécutions en même temps.

Néanmoins, la construction des architectures avait été faite avec beaucoup d'attention, ce qui a permis de gagner du temps lorsque nous avons eu à effectuer des modifications sur celle-ci. En effet, il existait des parties entières de l'architecture qui était réutilisables sans avoir à effectuer le moindre changement.

8. Références

Open Classrooms, 'Créez une application Android'. Tiré de :

https://openclassrooms.com/fr/courses/2023346-creez-des-applications-pour-android/2023968-votre-premiere-application

LibMad, 'MPEG audio decoder library'. Tiré de:

http://m.baert.free.fr/contrib/docs/libmad/doxy/html/index.html

AlsaMixer. 'Gnome AlsaMixer'. Tiré de:

https://doc.ubuntu-fr.org/gnome-alsamixer