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. 1

Team One, Inc.

Emir Khaled Belhaddad

Adam Martin Côté

Anthony Dentinger

Soukaina El-Ghazi

Othman Mounir

Le 5 Décembre 2018

Table des matières

1. Objectifs du projet

Fort de sa présence dans son quartier, le café-bistro Élévation a su s'adapter aux changements de sa clientèle. Bien qu'auparavant des changements de présentation et de menu aient reçus un succès auprès de ses clients, à présent Café-Bistro Élévation, Inc. souhaite ajouter un système informatique original permettant aux clients d'envoyer leur propre liste de chansons depuis une tablette Android. Café-Bistro Élévation, Inc. croit que cela lui permettrait de se démarquer de la nouvelle concurrence, d'autant que sa clientèle est en bonne partie composée d'artistes.

En termes de matériel, le café-bistro a peu d'investissement à effectuer puisque le système sonore est déjà en place et qu'un petit serveur serait nécessaire. Toutefois, Café-Bistro Élévation, Inc. n'ayant pu trouver un système existant convenant à ses besoins, la compagnie a fait appel à un Team One, Inc. afin de développer un système sur-mesure.

Exigences

De manière globale, le système demandé est formé de deux composantes:

1. Une **application Android** pour tablette Galaxy Tab S2. **Les utilisateurs du système interagiront avec cette application qui permettra**, d'une part, aux utilisateurs d'envoyer des chansons à jouer sur le système sonore du café-bistro Élévation, et, d'autre part, à des employés dudit café-bistro de gérer les utilisateurs, et d'arrêter et supprimer des chansons envoyées.
2. Un **serveur** Xilinx Zedboard ayant Arch Linux comme système d'exploitation. Celui-ci recevra les requêtes des tablettes Android et en effectuera le traitement réel. En particulier, **c'est ce serveur qui jouera les chansons** et en transmettra le son via l'interface minijack au système sonore du café-bistro Élévation.

Le système développé avait à répondre à certaines exigences non techniques :

- **Livraison et présentation** au client : le premier livrable devait être remis le 13 Novembre 2018 et avait à contenir le coeur du système, et le livrable final avait à être présenté le 29 Novembre 2018 et contenir l'ensemble du système, des pénalités s'appliquant dans le cas de fonctionnalités manquantes ;
- Des **rapports hebdomadaires d'avancement** du projet quantifiés par une progression de tâches prédécoupées au début du projet.

Les principales exigences techniques de Café-Bistro Élévation, Inc. concernant le système développé pourraient se résumer comme suit :

- **Plateforme matérielle utilisée** pour le serveur (RAM, CPU, entrée/sortie audio, etc.) et pour laquelle le système devra fonctionner ;
- **Langages de programmation** utilisé pour le serveur (C/C++) et l'application Android (Java ou Kotlin) ;
- **Déploiement** du système du serveur à l'aide d'un conteneur ;

- Certaines technologies et librairies utilisées (libmad, alsa/pulseaudio, …) ;
- Interface **REST** à utiliser.

La présente a pour but de permettre au lecteur de comprendre :

1) Certains aspects originaux et particuliers de l'implémentation du système requis ainsi que les choix effectués ;
2) La structure générale du code source ;
3) La démarche utilisée par Team One, Inc. pour répondre aux exigences du projet ainsi que s'assurer un bon niveau de qualité ;
4) Certaines potentielles améliorations techniques et fonctionnalités supplémentaires possibles ; et
5) L'apprentissage des membres de l'équipe.

2. Description du système

Le document d'appel d'offres lancé par Café-Bistro Élévation, Inc. décrivait la possibilité de projets supplémentaires à l'avenir. La présente section décrit donc le système réalisé afin d'introduire à sa conception.

2.1 Le serveur

Technologies utilisées

Le serveur Arch Linux constitue le coeur du système dans la mesure où il effectue les traitements demandés par les tablettes Android et joue les chansons envoyées. Afin d'effectuer le comportement attendu, nous déléguons certaines parties du système à des **librairies et autres technologies spécialisées**.

CMake : Permet d'orchestrer la génération des exécutable et l'édition des liens avec les autres technologies. Appelle également les exécutables de tests unitaires.

Pistache : Permet de développer un serveur REST rapidement sous Linux, mais ne permet pas le chiffrement SSL/TLS pour réaliser du HTTPS.

OpenSSL : Effectue la cryptographie pour chiffrer et déchiffrer les données HTTPS.

libmad : Effectue le décodage des fichiers MP3.

PulseAudio : Cadriciel permettant d'interagir avec le système de son du serveur et de jouer des sons. Nous avons finalement opté pour PulseAudio plutôt qu'Alsa simplement parce que **l'exemple sur lequel nous nous sommes basés utilisait PulseAudio et était déjà fonctionnel**. Nous avions aussi mentionné lors de la réponse à l'appel d'offres que nous utiliserions peut-être ffmpeg, mais le client a au final préféré une solution sur-mesure.

libtag : Effectue le décodage de l'en-tête de divers formats de fichier, dans notre cas les fichiers MP3. Nous avons essayé en parallèle libtag et id3lib, et finalement nous avons réussi à faire fonctionner libtag avant id3lib.

Docker : Crée et gère des conteneurs. Nous avions au départ prévu d'utiliser Docker afin de permettre un déploiement facile et sommes capables de créer des conteneurs fonctionnels pour des architectures AMD64, mais le Xilinx Zedboard utilise une architecture ARMv7 incompatible avec AMD64, et **DockerHub ne semble pas avoir d'image d'Arch Linux disponible pour ARMv7**. Le client a finalement abandonné la solution de déploiement.

SQLite : Permet de gérer une base de données.

Architecture du serveur

Le comportement attendu est réalisé grâce à **quatre processus** et à **deux bases de données**, décrits à la Fig. 2.1 ci-dessous.

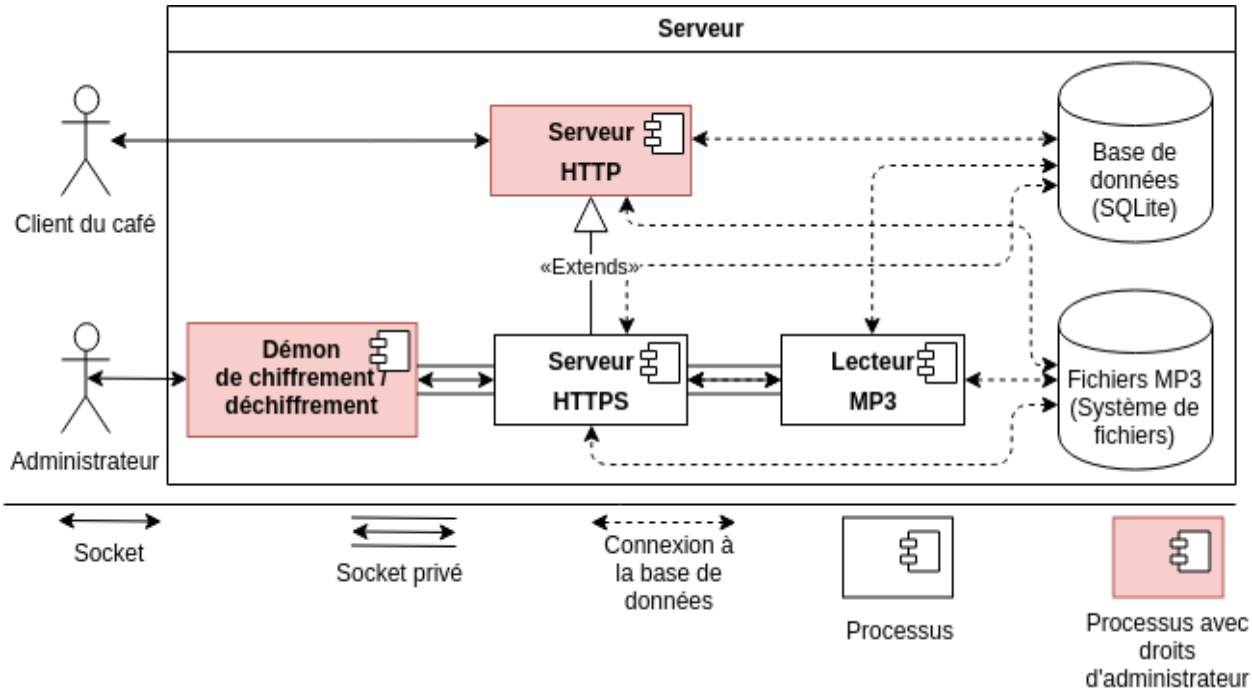

Fig. 2.1 - Aperçu global de la communication au sein du serveur [1]

L'implémentation des processus est répartie dans le dossier `common/` (qui est utilisé par tous les processus) et dans le dossier du même nom que le processus. Ces composantes ont des responsabilités bien définies.

Le **serveur HTTP** répond aux requêtes de l'utilisateur. La classe **`http-server/http/RestApi` agit ici comme chef d'orchestre**, répondant aux requêtes en accédant à la base de données et écrivant les fichiers MP3 au besoin.

Le **serveur HTTPS** répond aux requêtes de l'administrateur. Les actions de l'administrateur étant nombreuses et étant plus à l'aises avec le projet lors de la réalisation du serveur HTTPS, **les comportements sont réalisés dans les classes `https-server/https/descriptions/*`** et agrégées par la classe `https-`

`server/https/SecureRestApi`, qui est une sous-classe de la classe `http-server/http/RestApi` de manière à ce qu'il soit facilement possible de modifier le système pour que toutes les communications soient chiffrées. Le serveur HTTPS possède également un socket (`common/mp3/communication/Mp3EventClientSocket.*`) vers le lecteur MP3 afin de lui envoyer des commandes.

Le **démon SSL** (démon de chiffrement / déchiffrement) est le délégué du serveur HTTPS qui effectue la cryptographie, la technologie Pistache utilisée pour le serveur REST ne permettant pas au serveur HTTPS de le faire directement. Les classes **`ssl-daemon/ssl/*` encapsulent l'interaction avec OpenSSL** pour la cryptographie, et la classe **`ssl-daemon/core/DaemonRunner` gère les connexions SSL individuelles**, ce qui permettrait à plusieurs administrateurs de se connecter en même temps. La communication se fait via des socket Linux, définis sous `common/os/Socket*`.

Le **lecteur MP3** a la responsabilité d'interagir avec le système de son du serveur. C'est donc lui qui décode et **joue les chansons en utilisant la classe `elevation-player/mp3/player/Mp3AutoPlayer`**. Plus précisément, le lecteur MP3 utilise `elevation-player/mp3/Mp3AutoPlayerCallbacks`, qui implémente l'obtention et la suppression des chansons de la base de données. La réception de commandes pour interagir avec le volume et arrêter la chanson courante se fait par l'intermédiaire d'un socket spécialisé (`common/mp3/communication/Mp3EventSocket.*`), et la réalisation de ces commandes est implémenté par `elevation-player/mp3/event/ElevationPlayerMp3EventVisitor`.

L'interaction avec la **base de données SQLite** s'effectue via la classe `common/database/Database`, tandis que l'interaction avec le **système de fichiers** se fait en utilisant `common/filesystem/FileCache`.

Ces commentaires devraient permettre au lecteur pour avoir un bon ordre d'idée du fonctionnement du serveur et de l'utilisation des diverses technologies de manière à pouvoir plus rapidement en lire le code source.

2.2 Tablette

L'application Android est basé sur un système de synchronisation. La principale caractéristique de ce système est qu'il est basé sur une architecture MVCP dérivée des architectures MVC et MVP. Cette architecture est utilisée afin d'accomplir une synchronisation de l'état de la tablette avec celle du serveur. En partant du principe que l'implémentation du serveur est complète et robuste, il nous suffit de nous assurer que l'application ne fait qu'interfacer l'usager et le serveur de façon conviviale.

Pour atteindre cet objectif, nous avons mis au point une interprétation des architectures MVC et MVP pour obtenir le patron Modèle Vue Contrôleur Présentation (MVCP). Voici un schéma des différents modules composants le patron:

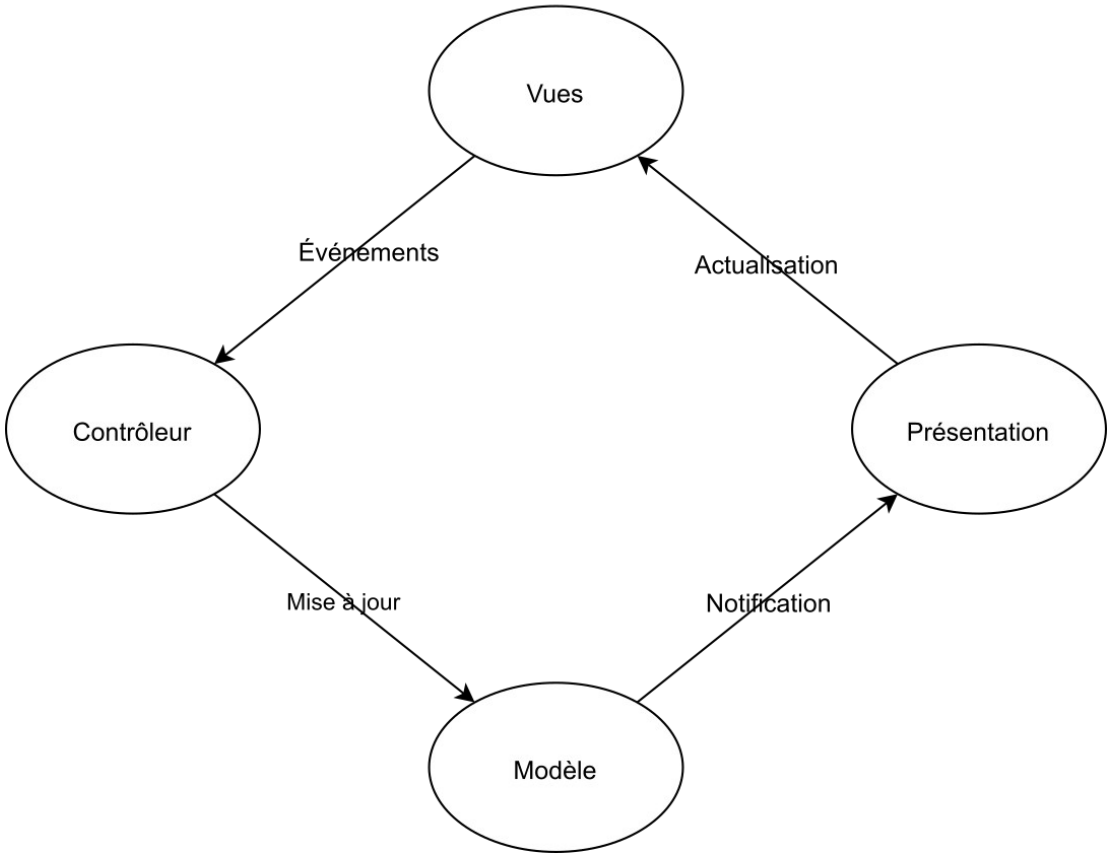

Fig. 2.2 - Schéma du MVCP [1]

Il y a 4 principaux modules constituants MVCP:

- La Vue, qui est chargée de faire l'affichage des données à l'utilisateur. Elle ne contient aucune logique.
- Le Contrôleur, qui est le module gérant tous les flux de données ainsi que l'interfaçage avec les librairies externes (Volley pour les requête réseau, SDK Android) et les événements périodiques.
- Le Modèle, qui contient toutes les données et états de l'application.
- La Présentation, qui se charge de formater les données et mettre à jour la Vue.

Dans notre implémentation de cette architecture, nos classes sont séparées en 4 principaux modules, représentant chacun les 4 modules de MVCP. Les principales classes seront présentés dans le reste de cette section.

Toutes les activités représentant les pages affichées à l'utilisateur héritent de la même classe `AbstractDrawerActivity` définissant tout ce qui est commun aux activités (le menu glissant par exemple). Une particularité d'Android est que les activités sont détruites à chaque fois qu'un changement du format de l'écran se produit (lors de la rotation de la tablette par exemple). Nous n'avons pas eu ce problème car, comme il a été mentionné plus haut, les vues n'ont pas de logique. Cela, couplé au fait que l'on utilise la librairie Koin, une librairie implémentant l'injection de dépendance permettant d'avoir la persistance des classes et des données, nous à permis d'abstraire les particularités gênantes de la SDK d'Android.

Le contrôleur principal de l'application `AppController` est la classe qui reçoit les différents événements (provenant de l'utilisateur ou du réseau) et les met en file d'attente avant de les envoyer aux bons contrôleurs pour qu'ils puissent les gérer. Les requêtes sont mise en file seulement s'il y a de l'information à envoyer. C'est aussi cette classe qui gère l'envoi des événements périodiques comme la requête de récupération de la file du serveur ou celle pour le volume.

Les classes constituant le module Modèle sont très majoritairement des classes de données. La seule classe qui n'en est pas une est la classe principale de ce module, `DataProvider`. Comme son nom l'indique, cette classe gère et met à disposition différent type de données. Elle gère aussi l'envoi de notification de mis à jour des données aux classes s'étant enregistrées.

Le module Présentation n'a pas de classe principale, mais toutes les classes ont un arrangement similaire. Dans l'application, la majorité des données sont présenté sous la forme d'une liste. C'est la raison pour laquelle les classes du module présentation sont des adaptateurs d'Android. Ce sont les adaptateurs qui formate les données et actualise la Vue. Ce sont aussi les classes de présentation qui gère les différents états de la vue (par exemple, lorsque l'on sélectionne une chanson à intervertir dans la file du serveur).

2.3 Fonctionnement général

Pour arriver à faire fonctionner le système, il faut tout d'abord compiler le serveur et configurer la carte FPGA. Tout ce qu'il y a à savoir sur les étapes à exécuter pour compiler et exécuter le code du serveur se trouve dans le `README.md` sous `project/server`, mais nous allons tout revoir plus en détail dans cette section. Nous allons aussi voir comment compiler et exécuter le client.

Serveur

Pour commencer, il faut configurer la carte FPGA afin qu'elle puisse jouer les chansons correctement. Pour ce faire, il faut installer les paquets nécessaires:

```
$ sudo pacman -Sy core/automake core/make core/gcc core/curl
  extra/boost extra/pulseaudio-alsa extra/pulseaudio extra/tcl extra/
  wget extra/git extra/cmake extra/python3
```

Il faut ensuite copier le contenu du dossier `project/server/res/config/etc` dans le dossier `/etc` de la carte (bien sûr avec les droits administrateurs) et exécuter la commande "`pulseaudio -k`" pour arrêter le serveur pulseaudio et qu'il puisse prendre en compte la nouvelle configuration.

L'étape suivante est d'effectuer la compilation. Pour ce faire, il suffit de se rendre dans le dossier `project/server` et d'exécuter la commande "`./elevation -b`". Bien qu'elle soit très simple, cette commande fait plusieurs choses. Premièrement, elle va télécharger les dépendances libmad et SQLite et les place dans le dossier `/lib`, et clone les sous-modules Git. Il va ensuite commencer la compilation des différentes librairies à l'aide de CMake ou de Configure. Une fois la compilation des dépendances terminée, c'est au tour du code du serveur de se faire compiler. Il se peut qu'il y ait

quelques "notes" qui apparaissent, mais elles sont dues à un changement de l'ABI du compilateur et ne nous affecte pas. Attention, la compilation sur la carte FPGA peut prendre une bonne heure.

Après que la compilation soit terminée, il faut créer et initialiser une base de donnée avec "`./elevation -d`". Cette commande ne fait qu'exécuter un script SQL créant la base de données avec le gabarit fourni.

Il est maintenant temps de démarrer le serveur. La commande est simple. Pour démarrer le serveur en mode production, il faut lancer "`./elevation -p`". Il faut prendre note que si l'on vient d'arrêter le serveur et que ce dernier avait une connexion d'ouverte, il se peut qu'il faille attendre jusqu'à une minute avant de le relancer afin que le système d'exploitation supprime les connexions en suspens et libère les ports.

Le lancement du serveur va démarrer en arrière-plan les processus "`elevation-player`", "`http-server`", "`https-server`" et "`ssl-daemon`".

Pour arrêter tous les processus liés à l'exécution du serveur (bref, pour arrêter le serveur), il suffit d'exécuter la commande "`./elevation -o`", qui ne fait qu'exécuter la commande `kill` sur chaque processus du serveur.

Client

Pour lancer le client, la méthode la plus simple est d'ouvrir le projet sur Android Studio et de lancer l'application sur la tablette branchée à l'ordinateur et avec le mode "*USB debugging*" activé.

Figure 2.3.1 - Barre de compilation et d'exécution d'Android Studio

Une autre méthode qui ne nécessite pas d'avoir la tablette mise en mode "*debugging*" est d'empaqueter l'application dans un fichier `.apk` et de l'envoyer sur la tablette pour l'installer.

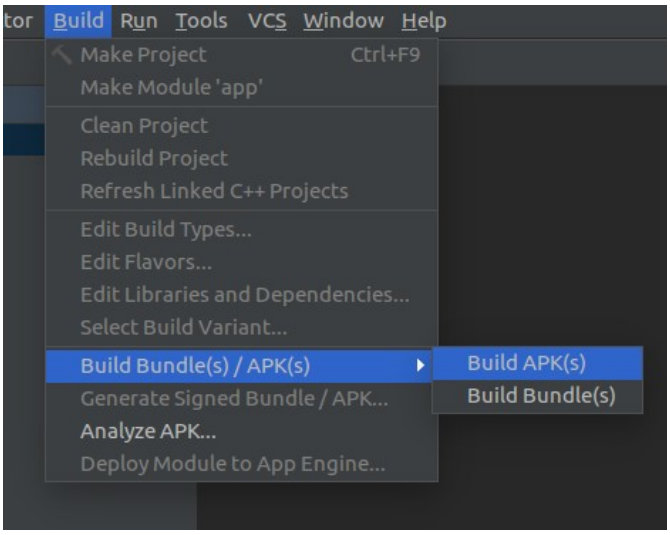

Figure 2.3.2 - Compilation de l'application vers un `.apk` sur Android Studio

3. Résultats des tests de fonctionnement du système complet

Nous avons utilisé plusieurs approches pour s'assurer du bon fonctionnement du système tout au long du projet. Tout d'abord, en implémentant un ensemble de tests unitaires, à l'aide du cadriciel boost, pour tester les fonctionnalités critiques, tels que les accès à la base de données ou encore la lecture de fichier MP3. Ces tests automatisés peuvent être appelés directement par le script de construction du projets (`./elevation --test`), ce qui facilite la validation à chaque itération de développement. De plus, chaque ressource de l'api a été l'objet de tests fonctionnels exhaustifs, à la fois à l'aide de la tablette et avec des requêtes forgées manuellement (`curl`). Nous avons conçu une interface robuste, qui effectue des vérifications à chaque niveau, par exemple pour s'assurer qu'une requête malformée échoue explicitement.

Les tests effectués nous on permit de détecter et de corriger plusieurs anomalies et nous avons maintenant une grande confiance en notre système, qui répond aux spécifications du client avec robustesse et précision. Toutes les fonctionnalités

demandées fonctionnent correctement, tel que démontré lors de la présentation du produit au client. Nous avons également agrémenté l'application d'une icône et d'un fond d'écran sur mesure. Globalement, l'application a été conçue pour être conviviale et intuitive, afin d'assurer une expérience simple et agréable pour l'utilisateur.

4. Déroulement du projet

Le projet est bien déroulé dans l'ensemble, le sens de responsabilité des membres d'équipe était présent. Chaque personne a assumé pleinement ses responsabilités et les tâches étaient accompli à temps. À l'occasion, un membre de l'équipe jugeant qu'il n'avait pas les capacités nécessaires pour une tâche spécifique a dû demander de l'aide afin d'éviter un retard.

La répartition des tâches entre les membres de l'équipe client était bien équilibré. Les tâches étaient simples, atomiques et indépendantes entre elles. Dès qu'une tâche devenait plus complexe et interdépendante, une rencontre avait lieu afin de discuter du découpage de celle-ci. Contrairement à l'équipe client, les tâches du côté serveur étaient lourdes pour quelques membres et il y avait beaucoup de dépendance entre les tâches.

Nous avons suivi de bonnes stratégies pour gérer le projet, conseillées lors de la rencontre de HPR. Nous avons mis en place des réunions hebdomadaires contenant un *Big Brother*, un animateur de gestion et un *scrum master.* Le *Big Brother* devait observer les membres de l'équipe pour guetter tout signe de conflit naissant au sein de l'équipe. L'animateur de gestion s'occupait quant à lui du découpage et de l'interdépendance des tâches. Puis finalement, le *scrum master* s'occupait de l'animation de la réunion pour la rendre plus productive et agréable. Ces stratégies de gestions nous ont permis d'être mieux organisé pendant les réunions hebdomadaires car nous perdions énormément de temps auparavant.

Nous avons adopté la méthodologie GitFlow pour la gestion des branches, qui sépare le code du projet en une version stable et une version de développement comportant les toute dernières fonctionnalités. Chaque branche de fonctionnalité correspondait à un *user story*. L'organisation de notre équipe s'est faite dès le début dans l'esprit de la méthode agile. Afin d'assurer un haut standard de qualité et un code uniforme à travers l'application, nous avons mis en place une stratégie de revue de code, en collaboration avec un autre membres de l'équipe. Cette stratégie améliore la détection de bogues, et l'application des standards, tout en favorisant un apprentissage continu des membres de l'équipe.

Les tests fonctionnels de l'application sont une étape très importante dans le cycle de développement. Cela garantit la qualité de l'application et la satisfaction clientèle. Cependant durant les deux livrables, ces tests ont été effectués tardivement. En conséquence nous avons détecté des bogues à la dernière minute ce qu'a obligé quelques membres de l'équipe à passer des soirées afin d'adopter des correctifs nécessaires.

5. Travaux futurs et recommandations

Une des principales fonctionnalités qu'il serait bénéfique d'implémenter est l'envoi des chansons par flux plutôt que de faire une temporisation des données. Ainsi, pour le moment, nous procédons d'abord à un encodage en base64, puis nous remplissons un tampon en mémoire vive avec les données de la chanson à envoyer. Une fois la chanson en base64 dans le tampon, nous envoyons le tampon entier d'un coup. Ceci a pour effet d'utiliser beaucoup de RAM tant sur l'application Android qui encode que sur le serveur qui décode. Nous étions toutefois limités par la librairie Pistache qui ne supporte pas supporter l'envoi de requêtes sur plusieurs paquets REST. L'ajout de cette fonctionnalité était possible en modifiant légèrement la librairie. Nous n'aurions donc pas eu à changer de librairie (et donc la structure du projet qui avait déjà été mise en place) pour rajouter cette fonctionnalité. Toutefois, à cause de contraintes de temps, nous n'avons pas eu le temps de l'implémenter. L'envoi de chansons par flux aurait eu un impact positif sur les performances du système, que ce soit du côté client ou serveur. Cela aurait permis l'envoi de plus de chansons en même temps, une réduction des délais lors de l'envoi, ainsi que l'élimination du temps de blocage du serveur lors de la réception d'une chanson.

Une deuxième fonctionnalité qui pourrait être implémentée est l'ajout de contrôles supplémentaires, du côté de l'application Android, lors de la lecture d'une chanson. Actuellement l'utilisateur, qu'il soit administrateur ou non, peut seulement ajouter et retirer une chanson. Ainsi, nous aurions aimé pouvoir permettre aux utilisateurs (surtout à l'administrateur) de pouvoir mettre en pause une chanson, la faire repartir, passer à la prochaine ou à la précédente, et faire défiler la chanson grâce à une barre de défilement (*timeline* en anglais).

Une troisième fonctionnalité aurait été l'ajout des images de couverture d'album au côté de la chanson jouée dans l'interface graphique de l'application. Cette fonctionnalité aurait été très simple à implémenter car la récupération des couvertures d'album peut être faite avec la même librairie (TagLib) que nous avons utilisée pour récupérer les informations de la chanson (tout se trouve dans l'en-tête ID3v2). Cependant nous n'avons pas implémenté cette fonctionnalité du fait des contraintes temporelles, du fait d'y avoir pensé trop tard, et par ce que cela ne faisait pas partie des spécifications du client pour l'application.

Une dernière modification que l'on aurait souhaité faire, mais qui n'est pas une fonctionnalité en soi, est le changement du fonctionnement du menu glissant de l'application. Pour l'instant, le menu glissant est instancié plusieurs fois, à chaque fois qu'on a une nouvelle activité. La duplication de code est certes évitée grâce à l'utilisation de l'héritage, mais nous aurions voulu changer ce fonctionnement par l'utilisation de fragment, ce qui est une meilleure pratique. Nous n'avons pas effectué ce changement car nous nous en sommes rendus compte tardivement, et ainsi cela devenait trop risqué car nous aurions pu casser l'application à quelques jours de la remise finale.

6. Apprentissage continu

Emir Khaled Belhaddad :

La lacune majeure que M. Belhaddad a dû surmonter cette session est qu'il a tendance à trop s'imposer dans les prises de décision architecturales. Dans le passé, on lui avait déjà fait remarquer cette tendance et il voulait y remédier dès le début de ce projet. Pour ce faire, il a tenté de ne faire que des propositions en expliquant son raisonnement sans essayer de forcer les autres à le suivre.

Une autre lacune fut qu'il n'avait aucune connaissance en lien avec le langage Kotlin. Il n'en avait même jamais entendu parler avant le début du projet. Il prit donc la première semaine afin de se familiariser avec le langage et il fit un effort d'apprentissage continue tout au long de la session. Cela a eu pour effet qu'il est devenu très familier avec le langage et sa librairie par défaut.

Au début du projet, M. Belhaddad a commis une erreur de communication due à un manque de précision dans son langage, mais cela ne s'est heureusement pas aggravée et a pu être résolu immédiatement. Il a pris conscience des raisons de cette erreur de communication et a pris des mesures afin de préciser son langage lorsqu'il communique avec les membres de son équipe pour ne plus que cela ne se produise.

Adam Martin Côté :

À plusieurs reprises au cours du projet, M. Côté a oublié de remplir les feuilles de temps et de mettre à jour les informations dans les outils de gestion de projet utilisés par l'équipe. Bien que ces pratiques soient des tâches d'assistance ne produisant pas de valeur pour le client, elles permettent de mieux diriger les efforts de l'équipe sur le plan individuel et collectif. Ce projet lui a appris à faire preuve de plus de rigueur au niveau de la gestion de l'avancement de projet.

M. Côté a commis quelques erreurs au cours du projet, en relation avec la définition des tâches et l'estimation du temps requis pour leur exécution. L'importance de bien planifier les différentes tâches et leur interdépendance, ainsi que de définir des objectifs d'une granularité appropriée a été flagrante lorsqu'il a réalisé que plusieurs tâches n'avaient pas été définies au départ du projet et qu'il fallait les compléter rapidement puisque d'autre tâches urgentes en dépendaient.

Pour M. Côté, ce projet a été une occasion de pratiquer les notions de programmation orienté-objet en langage C++, puisque celui-ci avait d'avantage d'expérience en C, qui est purement orienté-donné. Travailler avec des collègues ayant davantage d'expérience en C++ a été formateur et l'a exposé à plusieurs concepts avancés du langage qui lui seront grandement utiles dans le futur.

Anthony Dentinger :

Une des lacunes de M. Dentinger **était son manque d'expertise au niveau de la conception et du développement d'interfaces graphiques et d'applications Android**. Les interfaces graphiques présentent en effet un caractère asynchrone qui est géré par des patrons de conception spécialisés que M. Dentinger n'a jamais réellement expérimentés. Lui confier la conception de l'application Android aurait pu mener à des

erreurs d'architecture nécessitant un travail supplémentaire à régler et une conception peut-être moins judicieuse qu'elle ne l'a été. Bien qu'il ne fût pas contre l'idée de travailler sur le client, son niveau d'expertise se situait au niveau de la partie arrière du système, en particulier du C++ et l'orchestration avec CMake et Bash. Ce problème était **réglé simplement en lui confiant du travail du côté serveur**. M. Dentinger a toutefois effectué trois revues de code de la partie Android, ce qui lui a permis d'avoir une idée générale du fonctionnement des technologies utilisées ainsi que de la conception de l'application Android.

Bien qu'il ait déjà utilisé **l'API Linux** et ait une idée de leur utilisation, l'expérience de M. Dentinger avec ces interfaces étaient très limitées et peut parfois devenir assez complexe, en particulier lorsque l'on introduit des fils d'exécution. La solution à ce problème fut, d'une part, d'utiliser les manuels Linux pour déterminer comment utiliser la librairie Unix, et, d'autre part, **d'encapsuler l'utilisation de l'API Linux dans des classes spécialisées**. La classe *SharedFileMemory* en est un bon exemple, mais l'exemple le plus complet est la famille de classes Socket. Pour ce qui est de l'exécution multi-fils, la séparation de l'implémentation en de multiples classes a également été bénéfique, à l'instar de Mp3Player, ou de la classe Socket.

Soukaina El-Ghazi :

La principale lacune majeure que Soukaina a dû surmonter était le manque de connaissance du langage *Kotlin*. Pour se rattraper et progresser rapidement dans le projet, elle a suivi une formation en ligne pour avoir les notions de base de ce langage.

Une autre lacune était le manque d'expérience avec quelques patrons de conception utilisés dans l'architecture client. Soukaina avait des connaissances théoriques, mais elle n'avait jamais mis en pratique ses connaissances avant le projet. Elle a pu surmonter ce défi grâce à un apprentissage continu et l'expérience en architecture de son coéquipier Emir.

Othman Mounir:

Concernant M. Mounir, la principale lacune qu'il ait eue à surmonter était son manque d'expérience technique. En effet, il est le seul membre de l'équipe n'ayant pas encore d'expérience de programmation en entreprise. Pour y remédier, étant donné qu'il a été assigné au développement du serveur, il a tout d'abord cherché à réaliser le troisième TP lors du projet. Celui-ci consistait en un serveur HTTP avec un gestionnaire d'événements (logger). Il a ainsi participé activement à la réalisation du gestionnaire d'événements une fois que l'architecture du serveur HTTP ait été posée. Cela lui a permis, dans un premier temps, de se rafraîchir la mémoire en pratiquant le C++. Cela lui a aussi permis de commencer à regarder les notions de programmation multifils (multi-threading), qui sont une partie essentielle au bon fonctionnement du serveur HTTP, vu que la logique des tâches devait s'exécuter en parallèle au sein du serveur HTTP. De plus, ceci lui a permis d'en apprendre sur les *move semantics* ainsi que sur les pointeurs intelligents. Enfin, au fur et à mesure que le projet projet avançait, il essayait de travailler dans la même pièce que M. Dentinger pour avoir une référence en

cas de besoin, mais cela permettait aussi de palier au problèmes de conception bien avant qu'elles ne se produisent.

Une deuxième lacune de M. Mounir était son manque de connaissance en termes de normes de codages orienté objet. Pour remédier à cela, nous avons mis en place des normes de codage claires. De plus, au fur et à mesure que le projet avançait, M. Mounir avait plus d'exemples de codes "propre" dans le projet, desquels il pouvait s'inspirer. Cet aspect aurait pu être amelioré si M. Mounir aurait fait plus de recherche sur les bonnes pratiques en C++ au début du projet, ce qui aurait permis de perdre moins de temps lors des phases initiales du projet à cause refactoring .

La troisième lacune qu'il ait dû surmonter était son manque de connaissance en termes d'orchestration et compilation de projet. Ainsi, pour remédier à cela, il a tout d'abord commencé à lire des tutoriels sur le cadriciel CMake. Ne comprenant pas comment le cadriciel fonctionne, il demanda de l'aide à Anthony, qui est notre expert en C++. Anthony a donc mis en place la structure hiérarchique initiale pour la compilation avec CMake. Ainsi M. Mounir avait une source de laquelle il a pu s'inspirer pour faire compiler ses parties du projet. Cet aspect aurait pu être amélioré par M. Mounir s'il avait continué à lire la documentation, malgré la difficulté de cette dernière.

La dernière lacune qu'il ait eu à surmonter était son manque d'expérience dans un environnement de développement Linux. Pour remédier à cela, il a tout d'abord essayé de suivre le tutoriel sur la compilation croisée (cross compiling), préparé par les chargés de laboratoire, pour lui permettre de développer directement dans un environnement Windows sur Visual Studio. Cependant, ne voyant que cette méthode n'était pas très efficace du fait de la structure du serveur sur CMake, il installa sur sa machine une machine virtuelle Ubuntu. Cela lui a permis d'apprendre les bases de Linux, mais aussi des notions de base en scripting *bash.* Cet aspect aurait pu être amélioré s'il avait directement commencé à travailler sur une machine virtuelle Ubuntu plutôt que d'essayer de faire fonctionner la compilation croisée.

7. Conclusion

Nous avons décidé au début du projet d'utiliser une librairie de haut niveau afin de faciliter l'implémentation de l'interface REST. La librairie C++ *Pistache* semblait un outil répondant à nos besoins. Toutefois, la librairie comportait quelques limitations, notamment au niveau de la réception de requête sécurisées (HTTPS), qui n'était pas supporté. La documentation accompagnant cette librairie était également incomplète et n'avait pas été mise à jour suite à plusieurs changements de nommage importants. Nous en somme venu à la conclusion qu'utiliser une librairie de haut niveau qui n'offre pas toutes les fonctionnalités nécessaires est nuisible, puisqu'il faut écrire une grande partie de code et que le résultat final devient disparate et décousu.

La quantité et la qualité des librairies disponibles en C++ semble beaucoup plus variable que pour d'autres langages, comme Python ou Java, ce qui peut s'expliquer par l'usage plus spécialisé et bas niveau du langage. Il est donc très important de faire des recherches lors du choix d'une librairie externe pour s'assurer que celle-ci offre toutes les fonctionnalités nécessaires et d'opter pour une solution sur mesure dans le cas contraire.

En ce qui concerne les tests, nous avons décidé initialement d'utiliser une approche flexible en laissant le choix à chaque développeur d'élaborer des tests seulement lorsqu'il le juge nécessaire. Cette approche semble avoir été la bonne, puisque qu'une couverture complète des tests aurait été très coûteuse en temps et peu utile pour certaines fonctionnalités. Nous en avons conclu que le développement piloté par les tests est un outil précieux mais qui doit être utilisé avec jugement.

8. Références

[1] ‘Team One inc.’, 'Présentation orale - Café-Bistro Élévation', 2018 [entrepôt Git]. Disponible: présentationOrale.pptx.