haute école
neuchâtel berne jura

Zoukfiesta

Rapport Développement Mobile

Antoines Lestrade, Mathias Marty, Alexis Portmann

28.01.2021

Table des matières

1 Introduction

Nous avons imaginé une application qui permet de gérer les musiques jouées à une soirée par tous les participants.

L'host de la soirée héberge une « zouk » en sélectionnant un dossier de son téléphone contenant des fichiers de musique. Les autres participants peuvent alors rejoindre sa « zouk » et proposer des musiques du dossier de l'host qui s'ajoutent ensuite dans la liste de lecture. Ils peuvent aussi voter pour sortir une musique de la liste de lecture ou passer à la musique suivante.

L'host a les mêmes droits que les participants avec comme privilège supplémentaire de mettre la musique en pause, de changer le temps de la musique en cours et de virer des participants.

La détection des « zouks » en cours, l'échange d'information sur les musiques disponibles, la playlist, la musique en cours de lecture, son temps et les votes se fait à l'aide de l'api « Google Nearby Connections ».

2 Interface utilisateur

2.1 Interface commun client/serveur

2.1.1 Menu d'accueil

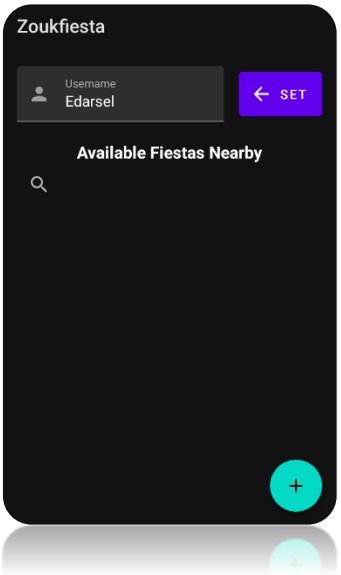

Figure 1 : Menu d'accueil

Au lancement de l'application, « Google Nearby » lance une découverte des différents hôtes possibles. Il charge ensuite toutes les « zouks » disponibles dans une liste. En cliquant sur une « zouk » l'application lance le mécanisme de connexion, propre à notre protocole, et passe au salon de « zouk ».

L'utilisateur a un pseudonyme attribué par défaut. S'il le change, il est enregistré dans un fichier JSON et est chargé au prochain lancement de l'application.

Le bouton « + » en bas à droite de l'écran ouvre le menu de création d'une « zouk ».

2.1.2 Salon de « zouk »

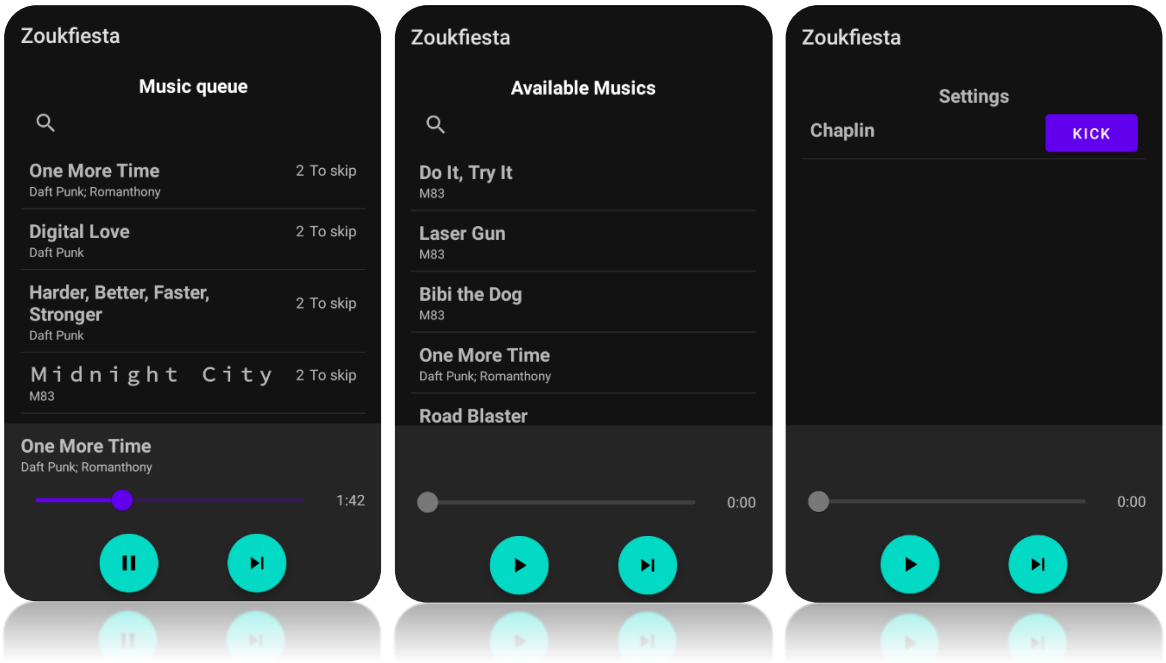

Figure 2: Playlist en cours, musiques possibles et men de kick

Le salon de « zouk » possède un « ViewPager » qui permet de défiler entre différents fragments ainsi qu'un lecteur de musique qui affiche les informations de la musique en cours de lecture. Ces informations sont :

- Le titre de la musique.
- Le temps écoulé.
- Un slider qui représente la progression de la musique.

Les clients possèdent deux fragments dans son « ViewPager » :

- Le premier contient la file d'attente en cours de « Zouk » sur laquelle le client peut cliquer sur une musique de la liste, ce qui demande au serveur de lui ajout un vote de skip.
- Son deuxième fragment affiche une liste avec les musiques disponibles qui peuvent être ajoutées à la file d'attente. Un click sur une musique envoie une requête au serveur pour l'ajouter à la liste de lecture gérée par un singleton : « MusicStore ». Le serveur va ensuite distribuer la nouvelle file d'attente aux clients.

Quant à lui, le serveur possède un troisième fragment avec la liste des clients de la « zouk ». Il peut les expulser en cliquant dessus, ce qui termine leur connexion.

2.2 Interface serveur

2.2.1 Menu de création d'une « zouk »

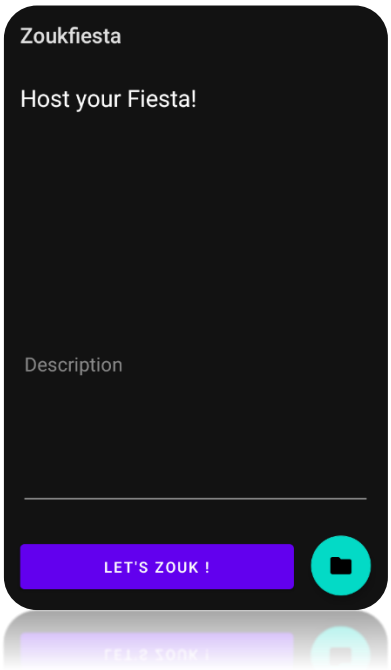

Figure 3 : Menu création de Fiesta

Lorsque l'on crée une « zouk », il faut entrer sa description et cliquer sur un bouton pour choisir un dossier contenant de la musique.

Ce dernier ouvre un « Document Tree » d'Android qui permet de se déplacer dans l'arborescence du système de fichier et de choisir un dossier. Le chemin du dossier choisi est renvoyé à l'application qui charge dans une liste tous les chemins des fichiers audios que le dossier contient.Ils sont alors enregistrés dans le singleton « MusicStore ». Cette opération peut être répétée plusieurs fois pour charger plusieurs dossiers.

Après cela, le bouton « Let's zouk ! » lance une « zouk » et broadcast ses informations à l'aide de « Google Nearby Connexion », de sorte qu'un client puisse s'y connecter.

2.2.2 Lecture d'une musique et pause

Lorsqu'une musique est ajoutée à la file d'attente, les clients sont notifiés de sa mise à jour. Le serveur lance alors la première musique de la file d'attente en appelant la méthode play du singleton « MusicPlayer », qui encapsule un « MediaPlayer » d'Android, en lui fournissant son uri récupéré depuis « MusicStore ».

Lorsque le serveur appuie sur le bouton pause/play, plusieurs événements se produisent :

- La méthode pause/play du « Music Player » est appelée.
- L'avancée du slider est arrêtée/reprise dans son fragment.
- Un message est envoyé à tous les clients, qui arrêtent/reprennent aussi l'avancée leur slider.

2.2.3 Changer la position du player

Le slider contenu dans le player est non-sélectionnable pour les clients ainsi que pour le serveur si sa liste de lecture est vide.

Lorsque la position du slider est changée par le serveur, une fonction de callback est appelée et met à jour le temps courant de la musique dans le « MusicPlayer ». Un message est alors broadcasté à tous les clients qui le mettent à jour sur leur interface.

2.2.4 Passer une musique

Il y a 2 cas de figure pour lesquels une musique est passée :

- La musique précédente a fini de jouer
- Son nombre de « voteskip » dépasse la moitié du nombre de clients (arrondie vers le haut).

Lorsqu'une musique est passée, l'application appelle le « MusicStore » pour la supprimer de la liste de lecture. Elle tente ensuite de récupérer la nouvelle musique en tête de liste du « MusicStore » pour la lancer et broadcast l'information aux clients. Si c'était la dernière musique, le slider du serveur est désactivé.

3 Réseau

3.1 Google Nearby Connections

« Google Nearby Connections » et une API destinée au réseau peer-to-peer. Celle-ci permet aux applications qui l'utilisent de découvrir, se connecter et échanger des données facilement avec d'autres appareils en temps réels. Le choix du périphérique utilisé pour la connexion est totalement encapsulé par l'API, elle peut utiliser :

- Le Bluetooth
- Le Wifi
- BLE (Bluetooth à basse consommation)

La connexion est totalement encryptée.

3.2 Protocole

Le protocole définit plusieurs commandes. Elles sont soit envoyées par le client pour le serveur, soit par le serveur pour le client. Ci-dessous un tableau récapitulant les commandes envoyées par le client :

Nom	Description	Paramètres
Gestion de la musique		
skip	*Enlever la musique en cours ou une musique de la playlist*	*-Nom de la musique à passer*
add	*Ajouter une musique à la playlist*	*-Nom de la musique à ajouter*

Ci-dessous les commandes envoyées par le serveur pour le client.

Nom	Description	Paramètres
Gestion de la musique		
playlist	*Renvoie la playlist, les informations concernant la musique en cours et tous les votes en cours*	*-Tableau associatif qui associe chaque titre dans la playlist à un nombre de votes, trié dans l'ordre de la playlist. La première musique est celle en cours.* *-Timestamp de la musique en cours (s)*

		-Durée de la musique en cours (s)
available	*Renvoie la liste des titres de musiques possibles*	*-Liste des musiques*
pause	*Demande au client de stopper la musique*	*-Un boolean représentant si oui ou non la musique est en cours de lecture*
Administration		
kick	*Envoie à un client qu'il faut kick*	

3.3 Structure

Pour simplifier l'implémentation des différentes commandes, deux interfaces ont été codées : « INearbyServer » et « INearbyClient ».

Elles contiennent les différentes méthodes qui permettront l'envoi des commandes appropriées, et les différents attributs qui stockent les méthodes appelées (callback) lors de la réception des commandes qui leur sont destinées.

Par exemple pour la commande « skip » du client :

- La méthode « sendSkip » avec comme paramètres une chaine de caractère sera déclarée dans « INearbyClient ».
- L'attribut « onSkip », qui est une fonction qui prend en paramètre une chaine de caractère sera déclaré dans « INearbyServer ».

Les différentes interfaces sont ensuite implémentées dans les classes « NearbyServer » et « NearbyClient ».

Les différentes méthodes relatives à « Google Nearby Connections » sont déclarées dans l'interface « NearbyListener ».

Ci-dessous un diagramme de classe de la partie réseau du projet :

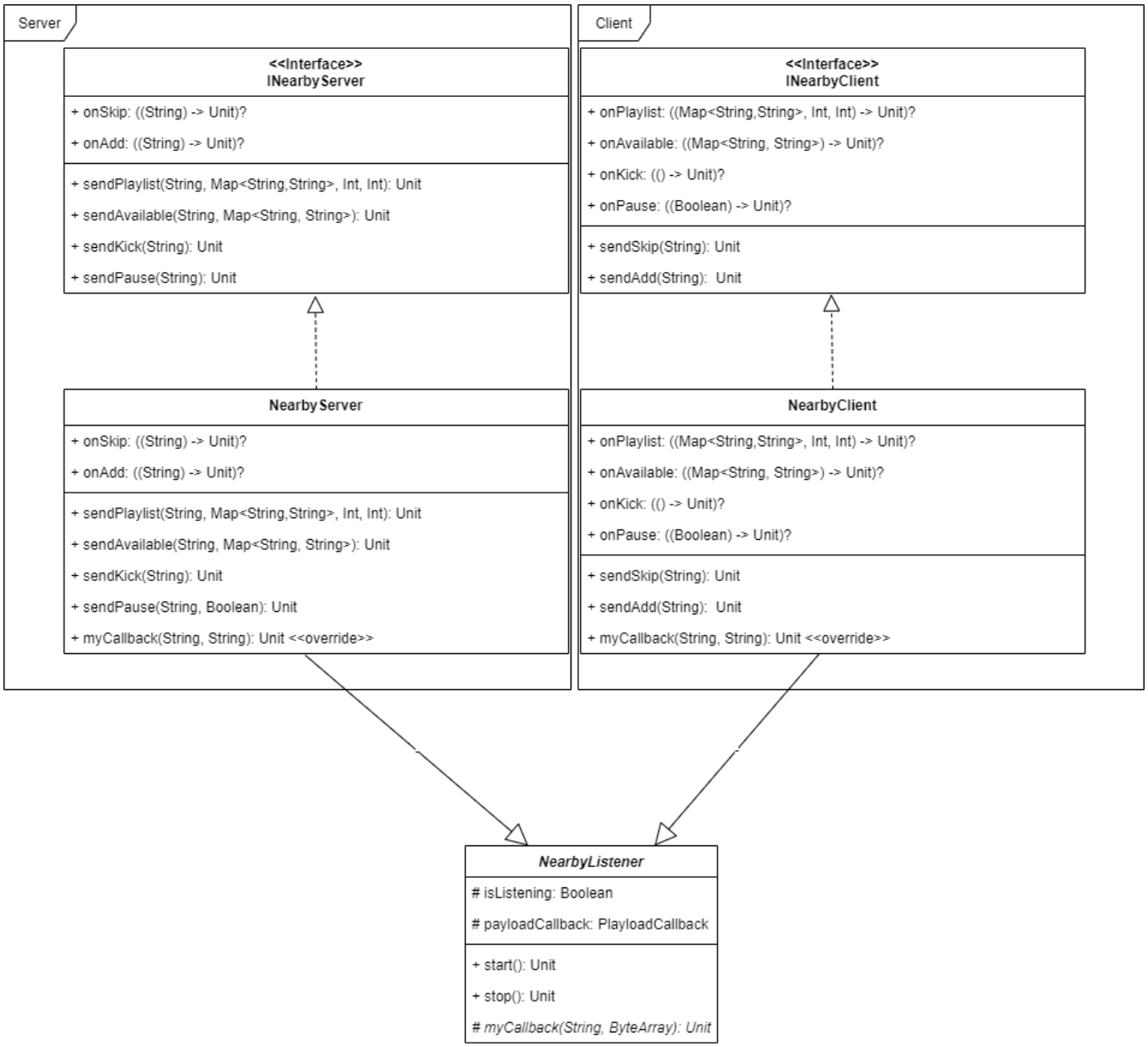

Figure 4 : Diagramme de classe de la partie réseau

3.4 Implémentation

L'implémentation est faite dans les classes « NearbyServer » et « NearbyClient ». Les données (le nom de la commande et ses arguments), sont sérialisées en JSON pour être ensuite converties en tableau de byte. Le tableau de byte peut être envoyé tel quel dans le réseau.

Deux classes qui permettent l'encapsulation des données ont été créées pour les structures plus compliquées : « DataPlaylist » et « DataAvailable ». Elles sont créées dans le fichier « Tools.kt » et sont annotées par « @Serializable », ce qui permet de les sérialiser en JSON.

Lors de la réception d'un message, le JSON est désérialisé. Plusieurs formats sont attendus, la désérialisation est donc faite dans des « try and catch » pour récupérer le message dans le bon format.

Après avoir parsé correctement les données, le programme appel la fonction de callback, appelé : « onXXX », et lui passe paramètres attendus.

Cette structure assez abstraite du réseau a permis de bien séparer l'implémentation de la logique pour qu'elle n'interfère pas avec le réseau. De cette façon, nous avons pu séparer le travail d'une manière efficace[1].

4 Problèmes et solutions

4.1 L'affaire des Schedule

Problème : Notre première implémentation du lecteur de musique utilisait une fonction pour passer à la musique suivante. Celle-ci était passée à un schedule paramétré avec le temps de musique restant. Cela alourdissait le code de façon considérable. En effet, lors d'une mise en pause ou d'un changement de temps forcé par l'host, le schedule devait être annulé puis relancé à la reprise de la musique. Par conséquent nous devions aussi recalculer à chaque fois le temps restant à partir du temps total et du temps courant de la musique.

Solution : Utiliser la fonction de callback du « MediaPlayer » qui est appelée lorsqu'une musique est terminée.

4.2 L'emboitage des fragments

Problème : Nous avons tenté d'emboiter de multiples fragments les uns dans les autres. Cela n'aboutissait à rien, car l'appel des fonctions dans cette hiérarchie compliquait trop le code.

Solution : Ne pas emboiter des fragments avec une profondeur N>2.

4.3 L'embuscade des linear layouts

Problème : Tels de jeunes androdins inexpérimentés, nous avons commencé à décorer notre interface d'une tapisserie de linear layouts entremêlés. Après de nombreuses larmes versées, au fond de l'abime du désespoir, une lueur se fit voir : LE COURS D'ANDROID DE NOTRE BIEN AIMEE AISHA RIZZOTI !

Solution : De beaux et élégants constraints layouts, source de toute interface qui se respecte.

5 Planning

Le planning a été relativement bien respecté. Le peer-programming a cependant été beaucoup utilisé lorsqu'un membre de l'équipe avait du temps libre. Le peer-programming n'est pas indiqué dans le planning, c'est pourquoi il n'est pas 100% fidèle à la réalité.

Le planning se trouve au point 8.1 ci-dessous.

6 Conclusion

Zoukfiesta est une application qui implémente un concept inédit ! Incorporant un design tout aussi ergonomique que dynamique, elle s'adresse à un large publique public, sans aucune discrimination d'âge, de sexe, de fatigabilité ou de handicap.

L'application répond aux critères élaborés dans le cahier des charges. Le planning a su être respecté.

[1] Bien qu'il ait fallu modifier le protocole réseau vers la fin du projet, car des fonctionnalités étaient manquantes

7 Remerciements

Nous tenons tout d'abord à remercier celle sans qui aucune ligne de code de ce projet n'aurait été possible : Madame Aisha Rizzoti.

Nous remercions très chaleureusement Mademoiselle Marie-Arielle Béguin, qui malgré sa charge de travail étouffante, a consacré un temps non négligeable a la création de notre logo.

Nous remercions finalement Nicola Cosi qui nous a encouragés tout au long de ce projet, quand bien même nous étions la risée de nos camarades.

8 Annexes

8.1 Planning