Rapport de projet mobile

I. Introduction

Ce rapport a pour but de présenter le travail effectué lors du projet de développement mobile du module MI1, consistant à réaliser une application proposant un minuteur HIIT.

Il abordera tout d'abord les étapes du développement, les fonctionnalités de l'application et le point de vue adopté pour atteindre l'objectif fixé. Il présentera ensuite les choix d'implémentation de plusieurs fonctionnalités. Enfin, il fournira des pistes d'améliorations possibles de l'application.

II. Travail réalisé

II.1 Etapes du développement

Le travail réalisé s'est déroulé de la façon suivante.

Le développement de l'application a tout d'abord commencé par l'écriture de son workflow sur papier avec l'ensemble des écrans à afficher et le passage des uns aux autres. Cette étape a été l'occasion de réfléchir aux écrans possibles et a fait surgir l'idée de certaines fonctionnalités, comme la possibilité de proposer des séances prédéfinies ou celle de reprendre la dernière séance effectuée dans l'application. Ce workflow a ensuite rapidement été mis en œuvre dans Android Studio. Une première version de l'application permettant de naviguer entre les écrans prévus mais sans leurs fonctionnalités a ainsi été réalisée.

Contrairement à d'autres étudiants, j'ai poursuivi mes travaux par l'obtention d'une application avec un chronomètre fonctionnel, mais sans base de données. Le but était d'avoir rapidement une application répondant à l'objectif principal, celui d'un minuteur pour entraînement fractionné, et de ne pas avoir à attendre la mise en œuvre d'une base de données pour cela.

La mise au point de cette base est venue ensuite, après la réalisation de la plupart des fonctionnalités du chronomètre. Bien que la précédente version de l'application était déjà satisfaisante et permettait d'effectuer de véritables entraînements fractionnés, l'ajout d'une base de données a permis d'obtenir une application plus aboutie, plus « sérieuse », offrant des séances prédéfinies ou enregistrées, et pour cela plus à même d'être proposée à un utilisateur. Elle a également demandé l'implémentation de concepts et de mécanismes plus compliqués : appels asynchrones à la base (la répartition des traitements à faire entre les méthodes doInBackground et onPostExecute a encore réservé certaines surprises, la première ne pouvant pas modifier les vues de l'application car elle se déroule dans un thread dédié, la seconde ne permettant pas d'appels à la base puisque réalisée dans le thread principal) et listing d'objets avec un recyclerView (ce qui nécessite plusieurs objets interagissant ensemble : un RecyclerView, un Adapter générant des ViewHolder, chacun de ces derniers embarquant la vue d'un élément de la liste à afficher et la donnée associée).

Enfin, le travail s'est focalisé sur la gestion du cycle de vie des activités et l'ajout des effets sonores dans le chronomètre. Concernant le premier point, trois activités nécessitait la mise en œuvre d'une sauvegarde de données pour pouvoir être régénérées au cas où elles seraient mises en pause ou arrêtées. La gestion du cycle de vie a été mise en œuvre pour deux

d'entre elles seulement : l'activité de définition d'une séance et celle faisant l'objet principal de l'application, le chronomètre.

II.2 Workflow et fonctionnalités présentes

L'application développée propose bien évidemment un chronomètre décomptant les différentes étapes d'une séance d'entraînement fractionné, du type décrit dans l'énoncé. L'affichage se fait d'ailleurs à la façon de la figure 1 de celui-ci (mais sans les bordures, celles-ci n'étant pas si simples à obtenir en xml). Des indications visuelles sont fournies : l'activité sportive en cours apparaît en noir et celles en attente sont grisées. Des indications sonores sont également fournies : un type de son est joué au début d'une activité de travail, un autre au début d'une activité de repos. De plus, lors d'une activité de repos, un troisième type de son est joué à la fin et ce de façon répétitive pour annoncer le début de l'activité de travail suivante.

L'application propose également, en début d'utilisation, d'accéder à différents types de séances. Dès la page de menu (le second écran de l'application), quatre choix sont proposés : définir une séance, utiliser des séances prédéfinies, utiliser les séances enregistrées par l'utilisateur lui-même, ou reprendre la dernière séance. Ces choix mettent en œuvre une navigation non triviale.

Si aucune séance n'a été enregistrée, ou si aucune dernière séance n'est disponible, demander ces types de séances conduit vers un écran spécial expliquant à l'utilisateur l'impossibilité d'obtenir ces séances et lui explique comment les avoir plus tard. D'autre part, appeler des séances prédéfinies ou des séances enregistrées (si elles existent) conduit vers un même modèle d'écran listant ces types de séances (listing réalisé à l'aide du RecyclerView mentionné précédemment). Enfin, le choix d'une séance existante (dernière séance, séance prédéfinie ou séance enregistrée), quel que soit l'écran où l'on clique dessus pour l'obtenir, conduit à un nouveau modèle d'écran affichant les caractéristiques de celle-ci. Dans cet écran, on peut même modifier la séance à l'aide d'un clic sur un bouton « modifier » qui change l'interface et permet à l'utilisateur de changer ses différentes valeurs.

Il est à noter que le dernier écran de l'application permet de revenir à l'écran de menu ou de quitter celle-ci.

Le workflow de l'application est disponible dans le fichier lauraca_workflow.jpg fourni avec le dossier. Il s'agit d'une représentation de l'ensemble des activités et la navigation possible entre elles faite sur papier.

II.3 Point de vue adopté pour le développement

Pour répondre à l'objectif fixé, certains concepts des modèles de développement ont été utilisés. Le développement s'est en effet déroulé de façon incrémentale et itérative : les fonctionnalités ont été ajoutées par « blocs » (le chronomètre d'abord, la base de données ensuite) et non toutes à la fois en même temps, le but étant d'avoir à chaque étape une application fonctionnelle. De plus, le développement de chaque fonctionnalité s'est fait de façon itérative : une version minimale mais plus simple à mettre en œuvre était d'abord programmée et une version plus aboutie venait ensuite, si le temps le permettait.

III. Choix d'implémentation

III.1 Classe Seance

Une classe Seance a été écrite pour représenter une séance d'entraînement et embarquer les paramètres associés. Initialement, cette classe contenait une classe Sequence contenant elle-même une classe Cycle. Mais, au fur et à mesure du développement, et afin de simplifier

les traitements, la classe Seance a été réduite à un ensemble d'attributs du type de base int. La classe Seance contient donc des entiers pour chacun des paramètres suivants : le temps de préparation, le nombre de séquences, le nombre de cycles, le temps de travail, le temps de repos et le temps de repos long (les temps sont à fournir en secondes).

Lors de la mise en œuvre de la base de données, la classe Seance a été étendue pour devenir une entité (avec des attributs supplémentaires de type String, correspondant à un titre et une catégorie de séance) grâce aux annotations nécessaires de la bibliothèque Room. Elle a également été rendue « parcelable » pour la gestion du cycle de vie de certaines activités.

Un objet de type Seance est sauvegardé dans une activité et récupéré dans les activités suivantes par l'intermédiaire d'une classe MonApplication héritant d'une classe Application. Celle-ci contient, entre autres, un objet de ce type auquel on affecte l'objet créé dans une activité donnée grâce à l'un de ses setters. On peut ensuite récupérer cet objet dans les activités suivantes grâce au getter correspondant.

III.2 Chronomètre et activité le mettant en œuvre

Une classe Compteur avancée a été proposée pour mettre en œuvre le chronomètre de l'application. Celle-ci dérive d'une autre classe réalisant un mécanisme d'abonnement en association avec une interface.

J'ai réutilisé ce principe pour faire interagir l'activité dans laquelle ce compteur est défini, la classe SeanceActivity, avec celui-ci. SeanceActivity implémente une interface contenant des méthodes d'affichage et de déclenchement du compteur. La classe mère de la classe Compteur, quant à elle, reçoit une instance de la classe SeanceActivity, ce qui permet à un objet Compteur de faire appel aux méthodes de la classe SeanceActivity qu'elle implémente dans l'interface. Par exemple, les différentes durées d'activités sportives, enregistrées dans une liste, sont fournies à un objet compteur et le déclenchement de ces activités se fait par un jeu d'appels de méthodes d'une classe à l'autre. Des méthodes d'affichage d'une instance de la classe SeanceActivity sont également appelées par un objet Compteur lors de son fonctionnement.

Au début du développement, seul l'affichage du décompte du chronomètre a été réalisé. Il a fallu ensuite également indiquer à l'utilisateur où il en est de sa séance sportive. Ceci a nécessité la création d'une classe EtatSeance contenant la séance courante, ainsi que la position (en numéro de séquence et numéro de cycle) à laquelle se trouve actuellement le déroulement des activités sportives. L'affichage de l'interface se fait alors un peu à la manière du paradigme déclaratif : toute l'interface est réaffichée au début d'une activité sportive en fonction de l'état de la séance à ce moment.

III.3 Base de données

La base de données de l'application a été réalisée grâce à la bibliothèque Room et sa mise en œuvre reprend les classes et interface fournies pour le projet.

Cette base n'est constituée que d'une seule table, une table seance correspondant à la classe Seance, telle que définie au paragraphe *III.1*. Les classes et interface utilisées sont au nombre de 4 : la classe Seance, son interface DAO associée SeanceDao dont les méthodes effectuent différentes requêtes à la base, la classe AppDatabase (représentant de la base de données au sein de l'application) et la classe DatabaseClient. Celle-ci met en œuvre le patron de conception singleton et permet d'utiliser la base au sein de l'application. Elle instancie la base de données et permet de l'alimenter par des requêtes initiales lors de sa création.

La classe-entité Seance contient un attribut categorie de type String qui permet d'affecter un type à une séance. C'est grâce à ce type, ou catégorie, que l'on peut proposer à l'utilisateur de choisir différents types de séances dans l'écran de menu et les suivants. La récupération d'une catégorie se fait par une requête contenant une clause WHERE indiquant la catégorie voulue. Il existe trois catégories de séances au sein de l'application : les séances dites prédéfinies (valeur de l'attribut : « preset » - celles-ci sont créées par les requêtes initiales de la classe DatabaseClient et donc disponibles au premier lancement), les séances enregistrées

par l'utilisateur (attribut de valeur « utilisateur ») et la catégorie dite dernière séance (attribut de valeur « last »). Cette dernière catégorie est destinée à ne contenir au plus qu'un seul enregistrement, celui de la dernière séance jouée dans l'application. Pour atteindre ce but, aucun enregistrement de type « last » n'est effectué dans la base lors de sa création. Puis, après le jeu complet d'une séance, lorsque l'application demande à l'utilisateur de poursuivre ou de quitter l'application, le choix de chacune de ces possibilités enregistre en base la séance qui vient de se terminer avec la catégorie « last » : si l'enregistrement existe, il est mis à jour et s'il n'existe pas, il est créé. On s'assure ainsi de n'avoir qu'une seule dernière séance.

L'enregistrement d'une séance en base se fait dans l'activité de définition d'une séance par l'intermédiaire d'un bouton « enregistrer ». L'utilisateur doit avoir rempli toutes les caractéristiques d'une séance (sauf son titre) pour cela.

III.4 Gestion du cycle de vie des activités

Celle-ci a été réalisée pour deux activités, l'activité de définition d'une séance et l'activité principale de l'application, celle de jeu d'une séance. Dans les deux cas, la méthode onSaveInstanceState est redéfinie et sert à créer un Bundle savedInstanceState contenant les données à sauvegarder. Celui-ci est ensuite réutilisé dans la méthode onCreate lors du nouveau lancement de l'activité.

Dans la méthode de définition d'une séance, c'est un objet Seance qui est enregistré dans le Bundle savedInstanceState (cela a nécessité, comme déjà mentionné, de rendre la classe Seance parcelable). Pour l'activité de jeu d'une séance, au contraire, seules trois variables sont sauvegardées, le nombre de variable minimal permettant ensuite de régénérer complètement l'état de la séance et son interface. L'état de la séance est d'abord retrouvé grâce au numéro de tour (dans la liste des activités sportives) sauvegardé, puis c'est l'état du compteur grâce à la sauvegarde de ses attributs « en pause » et updatedTime.

III.5 Effets sonores

Trois sons ont été choisis dans une banque de sons que j'ai obtenue lors d'une formation en design sonore effectuée alors que j'étais acousticien. Ces sons servent à indiquer, pour les deux premiers, le début d'une activité sportive (de type travail ou repos) et, pour le dernier, l'arrivée imminente d'une nouvelle activité de travail à la fin d'une activité de repos.

L'utilisation de ces sons dans l'application se fait grâce à la classe SoundPool et ses méthodes. Un objet de type SoundPool est déclaré dans l'activité de jeu de séance et non dans la classe Compteur qui lui est associée, bien que ce soit un objet de ce type qui déclenche la lecture des sons. Cette répartition a pour but de respecter, pour les sons, le même paradigme MVC mis en œuvre pour les vues.

IV. Améliorations possibles

Plusieurs améliorations peuvent être apportées à l'application développée. La première d'entre elles concerne l'interface.

IV.1 Interface utilisateur

Pour ce projet encore, après celui effectué l'an dernier en DUT, je me suis focalisé en premier lieu sur les aspects fonctionnels de l'application, pensant que ceux-ci primaient sur l'interface. Je voulais cependant améliorer cette dernière par rapport à l'aspect basique de celle que j'avais développée avec mon collègue de binôme l'an dernier. Malheureusement, le temps imparti ne m'a pas permis de le faire : je n'ai pas eu le temps de réfléchir à la façon de la rendre plus attrayante ni de me renseigner sur des widgets permettant de l'améliorer.

Cela permet cependant de se rendre compte de l'importance primordiale de l'interface d'une application. En effet, celle que j'ai développée ne met pas en valeur le travail que j'ai

réalisé et, sans rebuter l'utilisateur, ne lui donne peut-être pas autant envie de l'utiliser que si elle était plaisante. Il existe certes peut-être toujours un fossé entre le travail d'un développeur, l'ensemble des mécanismes qu'il doit mettre en œuvre pour proposer une application ne serait-ce que simple et la façon dont un utilisateur peut appréhender cette dernière : celui-ci ne se rend en effet pas compte de la complexité des traitements sous-jacents. Le développeur peut donc éprouver une certaine déception provenant de la différence entre le travail qu'il a réalisé et ce que l'utilisateur en ressent. Mais ici, l'interface augmente encore plus ce fossé et peut malheureusement laisser penser à l'utilisateur que l'application est à l'image de son interface : pauvre en fonctionnalités, ce qui n'est pas le cas.

IV.2 Chaines de caractères

La plupart des chaines de caractères ont été codées en dur dans l'application. Ce choix a été fait par souci de simplicité et de rapidité, étant donné la difficulté que peut représenter le fait de trouver des noms pour l'ensemble de toutes les chaînes de l'application tout en respectant une certaine convention de nommage.

Il est évident que pour une application de plus grande ampleur, et afin de permettre son internationalisation, il conviendrait de sauvegarder les chaînes dans le fichier strings.xml et de les mentionner dans l'application par le nom qu'il leur est donné dans celui-ci.

IV.3 Classe Compteur et activité SeanceActivity

Il n'est pas certain que l'interaction entre l'activité SeanceActivity et la classe Compteur soit optimale. La compréhension du compteur avancé fourni (notamment les jeux d'appels entre classes) a été pour moi longue et difficile. Je ne pense pas en avoir compris tout de suite la signification dans le cadre du mécanisme d'abonnement reposant sur une interface mentionné dans la classe mère de la classe Compteur (une classe siège d'évènements notifie à une autre leur survenue à laquelle celle-ci réagit). Le fonctionnement que j'en ai tiré pour mes propres classes est probablement trop compliqué et il ne met peut-être pas en œuvre exactement le mécanisme précédent.

Il serait possible de simplifier l'interaction entre ces classes et de réaliser une meilleure répartition des rôles entre elles. Une relecture prochaine du code permettrait de prendre le recul nécessaire pour réexaminer ces questions.

IV.4 Bases de données

Deux améliorations majeures pourraient être effectuées concernant la persistance des données.

Premièrement, afin de ne pas sauvegarder plusieurs fois la même séance, il conviendrait de vérifier que la séance soumise à enregistrement n'est pas déjà présente dans la base. Cette vérification n'a pas été implémentée par manque de temps.

Deuxièmement, il pourrait être intéressant pour l'utilisateur de pouvoir enregistrer les séances qu'il a modifiées. Au stade actuel du développement, en effet, l'utilisateur peut modifier une séance existante en base dont il a demandé l'affichage mais il ne peut pas enregistrer les modifications qu'il a réalisées dessus. Là encore, c'est par manque de temps que cette fonctionnalité n'a pas été mise en œuvre.

IV.5 Effets sonores

Comme précédemment indiqué, un son est joué au début de chaque nouvelle activité sportive (qu'elle soit de repos ou de travail). Mais c'est aussi le cas à la reprise du compteur après sa mise en pause. On pourrait vouloir ne jouer un son que dans le premier cas et pas dans le second. Mais cela suppose une modification de la classe Compteur que je n'ai pas eu le temps de réaliser.

D'autre part, les sons choisis ne sont pas très probants et n'évoquent pas assez des déclenchements d'activités sportives ou un chronomètre. Ils gagneraient à être modifiés en ce sens.