

Android Cycling Trainer



Lechaire Thomas – FIN2
ETML - Lausanne
Chef de projet :
M. Patrick Chenaux
Durée Projet : environ 102 heures

Table des matières

1	Spécifications	3
1.1	Objectifs et portée du projet.....	3
1.2	Fonctionnalités requises (du point de vue de l'utilisateur)	3
2	Planification	3
2.1	Planning initial.....	3
3	Analyse	4
3.1	Faisabilité.....	4
3.1.1	Les compétences à acquérir :.....	4
3.1.2	Matériel/Logiciels.....	4
3.1.3	Recherches d'informations/documentation	4
3.1.4	Gestion du temps	4
3.1.5	Choix des logiciels	5
3.2	Document d'analyse et conception.....	7
3.2.1	Maquette graphique	7
3.2.2	Développement	10
3.2.3	Hiérarchie des activités/vues	12
3.2.4	Base de Données.....	12
3.3	Conception des tests	14
3.4	Planning détaillé.....	15
4	Réalisation.....	15
4.1	Installation et préparation de l'environnement de travail	15
4.2	Implémentation de l'application	17
4.3	Implémentation de la base de données	43
4.4	Le dossier Ressources - res.....	44
4.5	Modifications	46
5	Tests	46
5.1	Dossier des tests.....	46
6	Conclusion	47
6.1	Bilan des fonctionnalités demandées	47
6.2	Bilan de la planification.....	48
6.3	Bilan personnel	49
6.4	Améliorations possibles	50
6.5	Remerciements.....	50
7	Annexes	51
7.1	Bibliographie.....	51
7.2	Webographie	51
7.3	Version d'Android.....	Error! Bookmark not defined.
7.4	ANNEXE 1 - LEXIQUE.....	51
7.5	ANNEXE 2 - TABLEAU DES VERSIONS ANDROID	51
7.6	ANNEXE 3 – JOURNAL DE TRAVAIL	51
7.7	ANNEXE 4 - PLANIFICATION INITIALE	51
7.8	ANNEXE 5 - PLANIFICATION DÉTAILLÉE	51
7.9	ANNEXE 6 – CODE SOURCE	51

1 Spécifications

1.1 Objectifs et portée du projet

- Analyser attentivement le cahier des charges.
- Proposer une interface graphique simple et intuitive.
- Réaliser l'application sous Android.
- Tester l'application.

1.2 Fonctionnalités requises (du point de vue de l'utilisateur)

L'utilisateur doit pouvoir :

- Créer de façon simple une séance d'entraînement par exemple :
- Entraînement route « force » : 6 sprints 8''/52'' + 6 sprints 12''/48'' + 5 x 1' + 3 x 3' + 1 x 5' + 3 x 3' + 6 sprints 10''/50''.
- Entraînement VTT « côtes » : 2 tours de 3 côtes de 5' une doublée (7 côtes) : 4^e et 5^e en 15/15 et 6^e et 7^e en 20/20.
- Entraînement route « rythme » : 30' échauffement avec 10 sprints de 6 à 8'', 10' 20/20 côte 160-165 bpm, 15' vitesse 130 bpm, 3' 30/30 côte à bloc 180 bpm, 15' vitesse 130bpm, 20' contre-la-montre 165 bpm souplesse, 15' vitesse 130 bpm, 20' contre-la-montre 165 bpm braquet + gros, 20' retour au calme.
- Afficher un résumé de la séance d'entraînement, cela permet d'avoir un aperçu global de l'entraînement.
- Démarrer la séquence d'entraînement, chaque partie de la séquence doit être affichée de façon claire, au moyen de couleurs et contrôles graphiques adéquats, barres de défilement, boutons, clignotements etc.
- Mettre en pause la séquence d'entraînement et la redémarrer au moment souhaité.
- Sauvegarder une séance d'entraînement créée.
- Charger une séance d'entraînement précédemment sauvegardée.
- Insérer sa fréquence cardiaque de repos et maximale, elle pourra être utilisée pour travailler ou afficher une certaine zone cardiaque.
- Calculer un indice de récupération sur le vélo pour une date précise :
- $F_{cmax} - FC$ 1'30'' après effort, cette valeur augmente lorsque la forme physique augmente, afficher l'historique des valeurs enregistrées.

2 Planification

2.1 Planning initial

- Dates de réalisation : du lundi 11.05.2015 au lundi 08.06.2015
- Horaire de travail :
 - Lundi** 08h00-12h15/13h10-15h40 Pentecôte le 25 mai
 - Mardi** 08h00-11h25/13h10-16h35
 - Mercredi** - 13h10-16h35
 - Jeudi** 08h00-11h25/12h20-15h40 Ascension le 14 mai
 - Vendredi** 08h00-11h25/12h20-15h40 Pont de l'Asc. le 15 mai
- Nombres d'heures : 102

Le lien vers le fichier de planification initiale est disponible en annexe

3 Analyse

3.1 Faisabilité

Dans le cadre de ce projet, une analyse est nécessaire afin de pouvoir anticiper les problèmes qui pourraient survenir durant la réalisation ou la programmation. Il faut vérifier certains points comme :

- Les compétences à acquérir ou approfondir.
- Le matériel nécessaire ou à exploiter.
- Les recherches d'informations particulières.
- La gestion du temps de travail.
- La sélection des logiciels.
- Les difficultés potentielles et les solutions envisagées.
- Les besoins de l'utilisateur.

3.1.1 Les compétences à acquérir :

- Compétences en java et POO (Programmation Orientée Objets).
- Compétences en programmation Android.
- Compétences de base en XML.
- Compétences en SQL ou Base de données.
- Connaissances de l'environnement Android Studio.

3.1.2 Matériel/Logiciels

- Un Ordinateur PC.
- Microsoft Office pour la rédaction du rapport.
- Android Studio pour le développement.
- Virtual Box pour l'émulation des téléphones.
- Gimp ou Inkscape pour le traitement d'image.
- Navigateur internet pour la recherche.

3.1.3 Recherches d'informations/documentation

- Recherches sur les bases de données Android.
- Documentation sur l'environnement Android Studio.
- Recherches sur les interfaces Android.
- Documentation de Google pour le développement Android.
- Recherche d'exemples d'applications sur le vélo.
- Recherche d'information sur le domaine du cyclisme.

3.1.4 Gestion du temps

La gestion du temps se fait grâce à la planification Initiale et détaillée ainsi qu'au journal de bord, qui servent de fil rouge. La gestion du temps peut facilement devenir problématique, il est donc important de respecter au maximum les délais fixés dans la planification (initiale et détaillée) afin de mener le projet à son terme. Des zones « tampons » vont être mises en place pour anticiper l'apparition d'éventuels problèmes.

3.1.5 Choix des logiciels

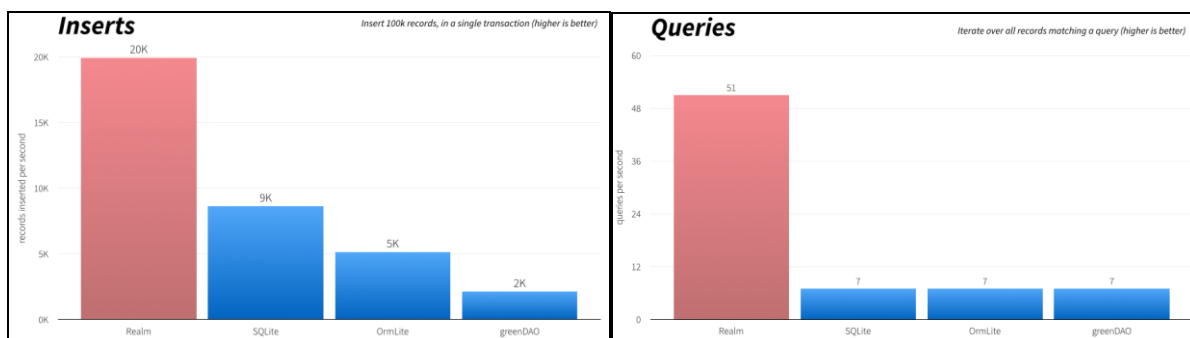
Concernant les logiciels de développement, deux grands logiciels peuvent être envisagés. Eclipse, projet d'Eclipse Foundation, qui possède un plugin (ADT) s'intégrant facilement et qui permet de développer sous Android, ainsi que Android Studio qui est développé par Google et qui base son noyau sur IntelliJ IDEA (autre logiciel de développement).

Note: If you have been using Eclipse with ADT, be aware that **Android Studio** is now the official IDE for Android, so you should migrate to Android Studio to receive all the latest IDE updates. For help moving projects, see [Migrating to Android Studio](#).

3.1 Note de Google concernant l'arrêt du développement d'éclipse pour Android

ADT pour Eclipse n'étant plus mis à jour et Google recommandant son produit pour obtenir les dernières mises à jour, Android Studio a donc été choisi pour développer le projet.

Concernant la partie base de données, SQLite est la solution que de nombreux développeurs utilisent pour Android. Cependant, les recherches et les forums parlent d'un grand nombre de développeurs ayant des problèmes. Principalement des problèmes liés à la vitesse d'exécution des requêtes et à la difficulté d'implémentation des bases de données avec SQLite. Comme solution, beaucoup de réponses mettaient en avant « Realm » qui est une librairie (cf. [Lexique](#) – point 1) permettant la création de base de données sur mobile. Pourquoi choisir Realm plutôt que SQL Lite ou un ORM (cf. [Lexique](#) – point 2) comme greenDAO (<http://greendao-orm.com/>) ? Premièrement, car Realm est facile à implémenter, il utilise son propre mécanisme de persistance (cf. [Lexique](#) – point 3) et, est capable de sauvegarder n'importe quel objet qui étend la classe RealmObject. Deuxièmement, car il est multiplateformes. Un fichier *.realm, contenant une base de données, peut facilement être importée dans un autre projet, que ce soit un projet Android ou IOS. Enfin, car il est d'après le benchmark trouvé, plus rapide qu'un ORM ou qu'une base de données SQLite. Realm est capable d'insérer 2000 enregistrements à la seconde contre 900 pour SQLite et 500 pour OrmLite et exécuter 51 requêtes seconde contre 7 pour ses concurrents. Il faut bien sûr prendre du recul sur ces résultats, car ils ont été effectués par les créateurs de Realm et non par un tiers.



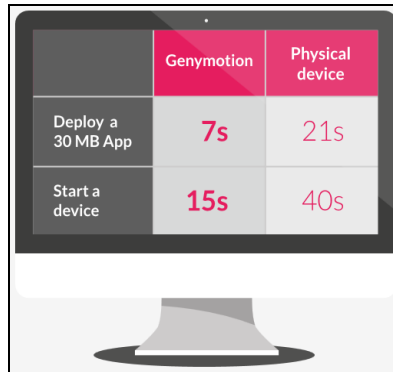
Benchmark tiré de : <http://realm.io/news/realm-for-android/#realm-for-android>

3.2 Requetes avec Realm (51 par sec)

3.3 Insertions avec Realm (2k par sec)

Pour tester l'application, un Samsung S3 version 4.3 a été utilisé ainsi que GenyMotion pour émuler d'autres terminaux. Android permet l'émulation de téléphones, mais la mise en place est longue et complexe pour un résultat et une rapidité très moyenne. GenyMotion est une solution rapide et facile à mettre en place puisqu'un plugin est directement téléchargeable depuis Android Studio. Il s'intègre dans la barre des tâches et propose une liste de plus de 80 téléphones dans 7 versions Android différentes. L'installation est simple car se sont

des Machines Virtuelles (cf. [Lexique](#) – point 4) gérées grâce par le logiciel VirtualBox. Le téléchargement de la machine virtuelle suffit à avoir un émulateur prêt à être lancé. Un gain de temps énorme pour développer et tester son application sur un maximum d'appareils différents et ainsi augmenter la compatibilité. GenyMotion offre aussi la possibilité de prendre des captures d'écran, de simuler des positions GPS, d'installer des applications par glisser-déposer ou encore de simuler les capteurs comme l'accéléromètre.



	Genymotion	Physical device
Deploy a 30 MB App	7s	21s
Start a device	15s	40s

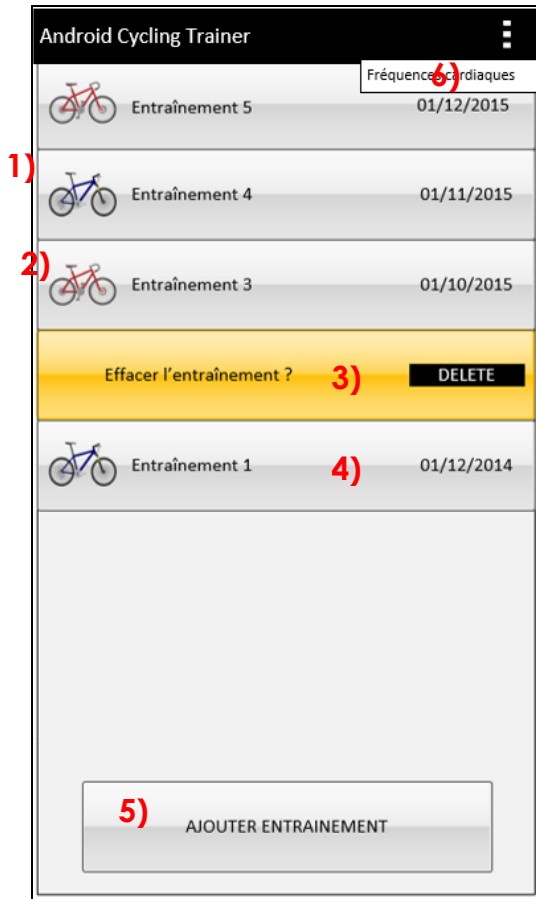
3.4 Comparaison entre GenyMotion et un Téléphone

Concernant le reste des logiciels utilisés, ils sont les suivants : Microsoft Office 2016 en preview pour la gestion de la partie administrative et la rédaction du rapport de projet. L'école fournit des ordinateurs avec Office, mais sans les droits administrateurs. Installer Android Studio dans une Machine Virtuelle et émuler encore à l'intérieur des téléphones aurait ralenti considérablement l'ordinateur. Gimp et Paint.Net (deux logiciels de traitement d'images) pour tout ce qui est la partie design et retouche (principalement pour les interfaces), Chrome pour la navigation et eduncanet, qui est la plateforme de l'ETML pour la correspondance. Tous ces outils sont des outils gratuits et ils ont été choisis pour cela.

Pour sauvegarder le projet le logiciel SourceTree et le site web <https://bitbucket.org>, qui sont des outils de gestion de versions, ont été utilisés. Ce choix est motivé par mes précédentes expériences lors de projets et ma connaissance de ces outils. Le site web bitbucket.org permet de créer des « repository » ou répertoire de sauvegarde. Le logiciel SourceTree clone le dossier distant dans un dossier de l'ordinateur. Les modifications effectuées dans le dossier physique sont envoyées chaque soir, de manière manuelle, sur le dossier de sauvegarde bitbucket.

3.2 Document d'analyse et conception

3.2.1 Maquette graphique



3.5 Activité entraînements

Dans cette activité (cf. [Lexique](#) – point 5), chaque entraînement affiche le nom choisi par l'utilisateur, une image vélo différente selon le type d'entraînement et la date du jour de création de l'entraînement.

Deux types d'entraînements sont disponibles:

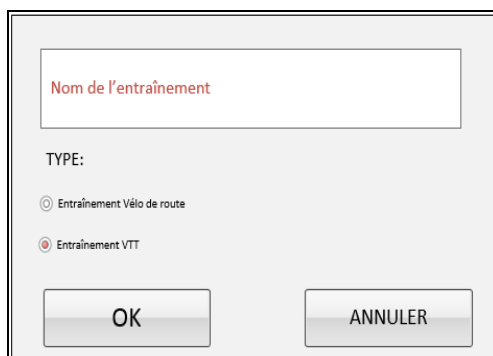
- VTT (en Bleu) (1)
- Vélo de route (en rouge) (2)

(6) Le menu paramètre permet d'ajouter les fréquences cardiaques de repos et maximale. (Voir image 3.7)

(3) Un geste « Swipe » (cf. [Lexique](#) – point 6) vers la gauche laissera apparaître un bouton permettant d'effacer l'entraînement.

(4) Chaque entraînement (ligne) est cliquable. Le clic ouvre une nouvelle activité et laisse apparaître les séquences contenues dans l'entraînement. (Voir Image 3.8 Eléments entraînement)

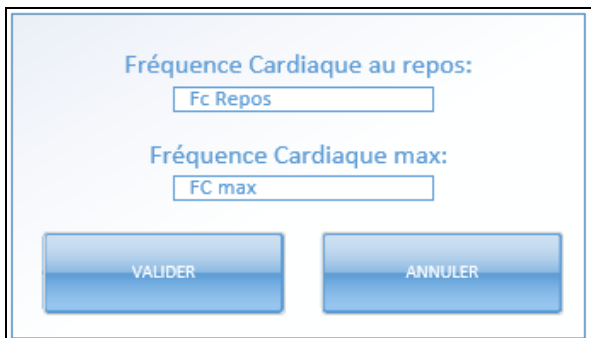
(5) Le Bouton « Ajouter Entraînement » ouvre une boîte de dialogue (cf. [Lexique](#)- point 7) permettant l'ajout d'un entraînement. (Voir : Image 3.6 Boîte de dialogue.)



3.6 Boîte de dialogue

La boîte de dialogue contient :

- Une zone de texte pour ajouter le nom de l'entraînement
- 2 boutons pour définir le type d'entraînement (VTT ou Route.)
- 2 boutons pour valider ou annuler l'ajout.



3.7 Insertion des fréquences cardiaques

Au clic sur le bouton paramètres de l'activité entraînement (6) une boîte de dialogue s'ouvre et permet à l'utilisateur d'introduire dans la base de données sa fréquence cardiaque de repos et sa fréquence cardiaque maximale (qui sera utilisée pour calculer un indice de récupération).



3.8 Séquences d'entraînements

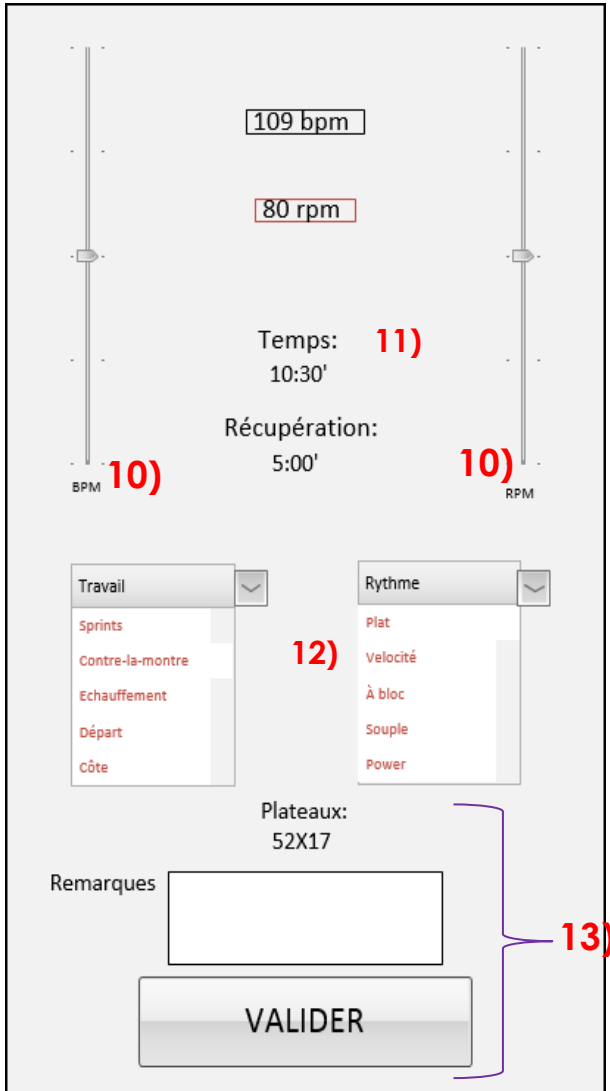
(7) Un bouton pour ajouter une séquence à l'entraînement. L'action sur ce bouton ouvre l'activité « Détails ». (Voir : Image 3.9 Détails activité)

Chaque ligne représente une séquence de l'entraînement, comme des sprints, de la récupération ou un contre-la-montre. La ligne indique à l'utilisateur le temps de la séquence, les [Bpm](#) et les [Rpm](#) ou encore les plateaux. Pour les termes techniques voir Annexes : [Lexique](#) – point 8,9 et 10.

(8) Comme pour l'activité « entraînement ». Un geste (Swipe) vers la gauche laissera apparaître un bouton pour effacer la séquence.

Chaque élément (ligne) est aussi cliquable permettant de modifier les valeurs de la séquence choisie.

(9) Le bouton « démarrer » lance l'activité Timer qui démarre l'entraînement de vélo avec le chronomètre. (Voir : image 3.10 Timer.)

**3.9 Détails activité**

L'activité détails est accessible en créant une séquence ou en cliquant sur une ligne de l'activité précédente permettant ainsi la modification. (Voir Image 3.8)

Dans cette activité Il est possible d'ajouter plusieurs informations.


Les deux barres verticales permettent de choisir les Bpm et les Rpm (10) et mettent à jour automatiquement les valeurs dans les deux TextView (cf. [Lexique](#) – point 11) prévues à cet effet.

(11) Le temps et la récupération sont validés par des TimePicker (cf. [Lexique](#) – point 12), présent dans Android. Le choix d'afficher le TimePicker directement ou dans une boîte de dialogue, est encore à faire.

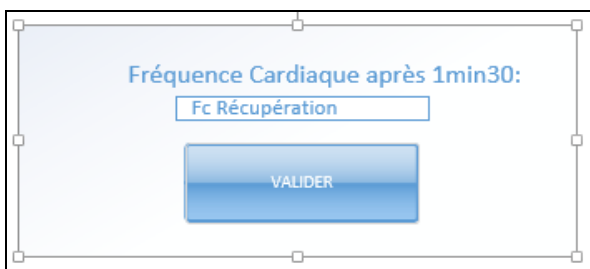
(12) Deux listes déroulantes ou Spinners (cf. [Lexique](#) – point 13) seront implémentées. Le premier, pour choisir le type de travail à effectuer (sprint, contre-la-montre, côtes) et le second pour choisir le rythme de la séquence (à bloc, Vélocité etc...)

Les derniers éléments (13) de l'activité sont :

Un élément pour choisir les plateaux avant et arrière, une boîte de texte pour ajouter des remarques personnelles et un bouton « valider » pour enregistrer la séquence et l'ajouter à l'entraînement.



3.10 Activité Timer



3.11 Boîte de dialogue « Récupération »

Toutes les informations affichées (14) proviennent de la séquence en cours et ont été entrés par l'utilisateur dans l'activité « Détails ». L'affichage permet à l'utilisateur de savoir la durée du travail et le rythme. Le type de travail, le rythme, les Bpm et Rpm, les plateaux, des remarques et même la séquence suivante seront affichés à l'écran lorsque le compte à rebours arrive à zéro.

L'activité « Timer » permet de lancer deux ProgressBar (cd. Lexique – point 14) affichant le temps de la séquence (17) et le temps de l'entraînement (18).

Le point 15 représente le compte-à-rebours de la séquence. Il a été défini par l'utilisateur dans l'activité « Détails » (Image 3.9).

Une série de contrôles (16) permettent d'arrêter, de mettre sur pause ou de stopper l'entraînement.

En fin d'entraînement lorsque le temps est terminé, une boîte de dialogue permettra à l'utilisateur de rentrer sa fréquence cardiaque après 1m30 de récupération. Grâce à cette fréquence et la fréquence maximale entrée dans les paramètres, un indice de récupération (pour l'entraînement effectué) est calculée et enregistrée.

3.2.2 Développement

Les recherches et la documentation sur internet ont permis de trouver des solutions rapides pour la base de données, la gestion du Swipe lors de l'effacement ainsi que certaines librairies gratuites.

Méthodologie :

La première étape de la phase de développement, consistera à découper l'ensemble du travail en plusieurs petites tâches. La méthode la plus efficace est de développer

Auteur : Léchaire Thomas

Version : 1.0

Création : 11.05.2015

Rapport-TPI_Lechaireth.doc

l'application par activité. Création d'une activité, création de son interface, puis création des méthodes et des fonctionnalités requises pour cette activité et ainsi de suite. Ce choix est motivé par le fait qu'il y a une hiérarchie (Voir Image 3.12) entre les activités et qu'il n'est pas possible de naviguer dans une application comme sur un site internet.

Fonctionnalités à développer :

L'image 3.5, représente la première activité de l'application. L'utilisateur doit pouvoir ajouter un entraînement grâce au bouton « Ajouter Entraînement », lors de l'ajout d'un entraînement un nom et le type de l'entraînement sont demandés au moyen d'une boîte de dialogue permettant de remplir ces informations. Il doit pouvoir supprimer l'entraînement lors du swipe ou voir les séquences en cliquant sur la ligne de l'entraînement. Chaque nouvel ajout entrainera automatiquement la sauvegarde de celui-ci dans la base de données.

L'activité de l'image 3.8 (activité séquences d'entraînement) représente les séquences d'un entraînement. La structure par liste est utilisée pour afficher les éléments comme celle présente dans l'activité précédente (image 3.5). Le swipe pour effacer une séquence est présent lui aussi, cependant pour ajouter une séquence il faut utiliser le bouton « + » situé en haut à droite dans l'ActionBar (cf. [Lexique](#) – point 15) (Voir point 7 Image 3.8). Le clic sur ce bouton lance l'activité « Détails » permettant d'ajouter une séquence à notre entraînement. Le bouton « démarrer l'entraînement » situé en bas de l'interface ouvre l'activité Timer. La modification d'une séquence enregistrée se fait par clic sur la ligne que l'utilisateur souhaite modifier.

L'activité « Détails » présentée à l'image 3.9 est l'interface qui s'ouvre pour ajouter une séquence à l'entraînement. Il est possible d'ouvrir cette activité si l'utilisateur désire modifier une des séquences de son entraînement. Ajouter une séquence ouvre l'activité « Détails » avec des valeurs vides. Modifier une séquence ouvre la même activité avec les valeurs de la séquence que l'on souhaite modifier. Il faudra, lors de la validation, vérifier qu'il s'agit bien de la modification d'une séquence déjà existante afin de ne pas rajouter une nouvelle séquence. Pour cela, deux méthodes différentes pourraient être nécessaires.

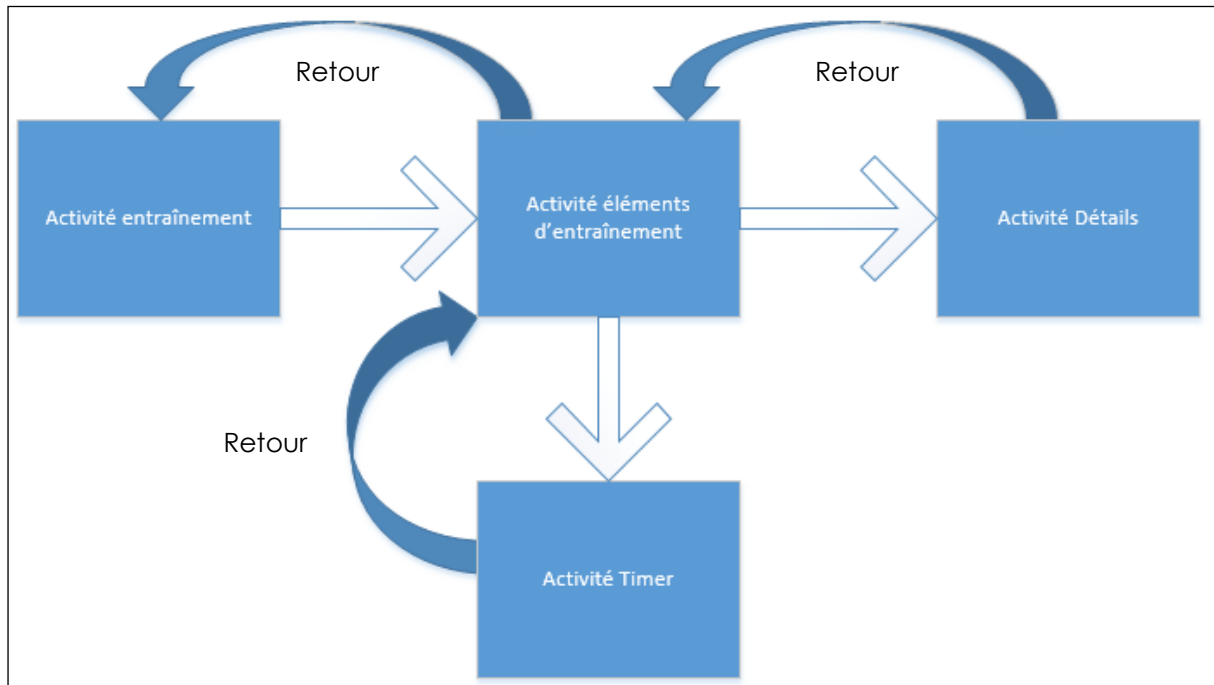
La dernière activité, est l'activité « Timer », présentée par l'image 3.10. C'est l'activité centrale de l'application, car c'est cette activité qui permet à l'utilisateur (cycliste) d'effectuer son entraînement. Quand l'utilisateur lance son entraînement, l'activité « Timer » prend alors le temps total de l'entraînement, ainsi que le temps et les informations issues de la première séquence de l'entraînement. Elle affiche alors les remarques, plateaux, Bpm, Rpm et toutes les autres informations présentes dans la séquence et démarre les timers du temps total de l'entraînement et du temps de la séquence. Une fois la séquence terminée, les données et le temps de la séquence suivante sont affichés, mettant à jour les informations déjà présentes dans l'activité. Le timer de l'entraînement continue de tourner normalement, mais le timer de la nouvelle séquence se met à jour avec le temps correspondant et redémarre à zéro. La partie de gestion des timers est quelque chose de compliquer qu'il faudra gérer correctement afin de ne pas laisser des timers tourner alors que l'application n'est plus active.

Les paramètres de l'application permettront d'ouvrir une boîte de dialogue afin d'enregistrer deux valeurs dans la base de données, la fréquence cardiaque de repos et la fréquence cardiaque maximale.

À la fin de l'entraînement, lorsque le chronomètre se termine, une boîte de dialogue s'ouvre et permet à l'utilisateur d'entrer sa fréquence de récupération (Fréquence obtenue 1min30 après la fin de l'entraînement). Un indice est calculé à partir de cette fréquence et de la fréquence maximale entrée dans les paramètres. L'utilisateur devra utiliser sa montre lui

donnant sa fréquence cardiaque, car elle ne sera pas récupérée automatique par l'application.

3.2.3 Hiérarchie des activités/vues

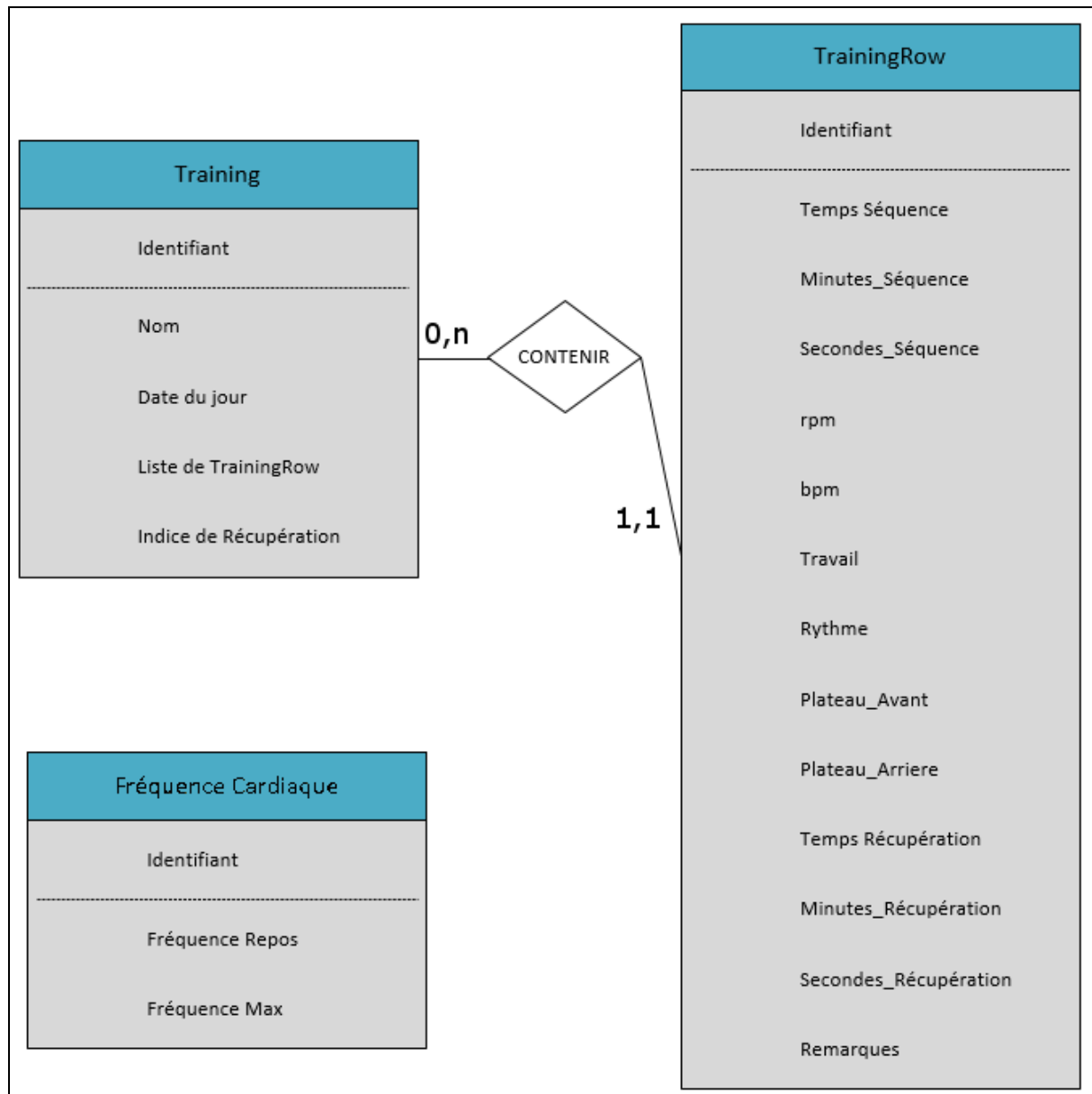


3.12 Hiérarchie des activités

La navigation ne s'effectuant pas comme sur un site internet, l'image ci-dessus représente les activités qui peuvent être appelées ou déclenchées par l'activité en cours. Le schéma permet de voir qu'il est, par exemple, impossible pour l'activité « Entraînement » de déclencher (ouvrir) l'activité « Timer » ou « Détails ». Chaque activité a une fonction retour, qui, sur Android est gérée par le bouton « retour » présent sur chaque téléphone. L'activité « Timer » aura un retour vers l'activité « Séquences d'entraînement » qui mettra fin à l'entraînement en cours entraînant la perte de la progression déjà effectuée par l'utilisateur.

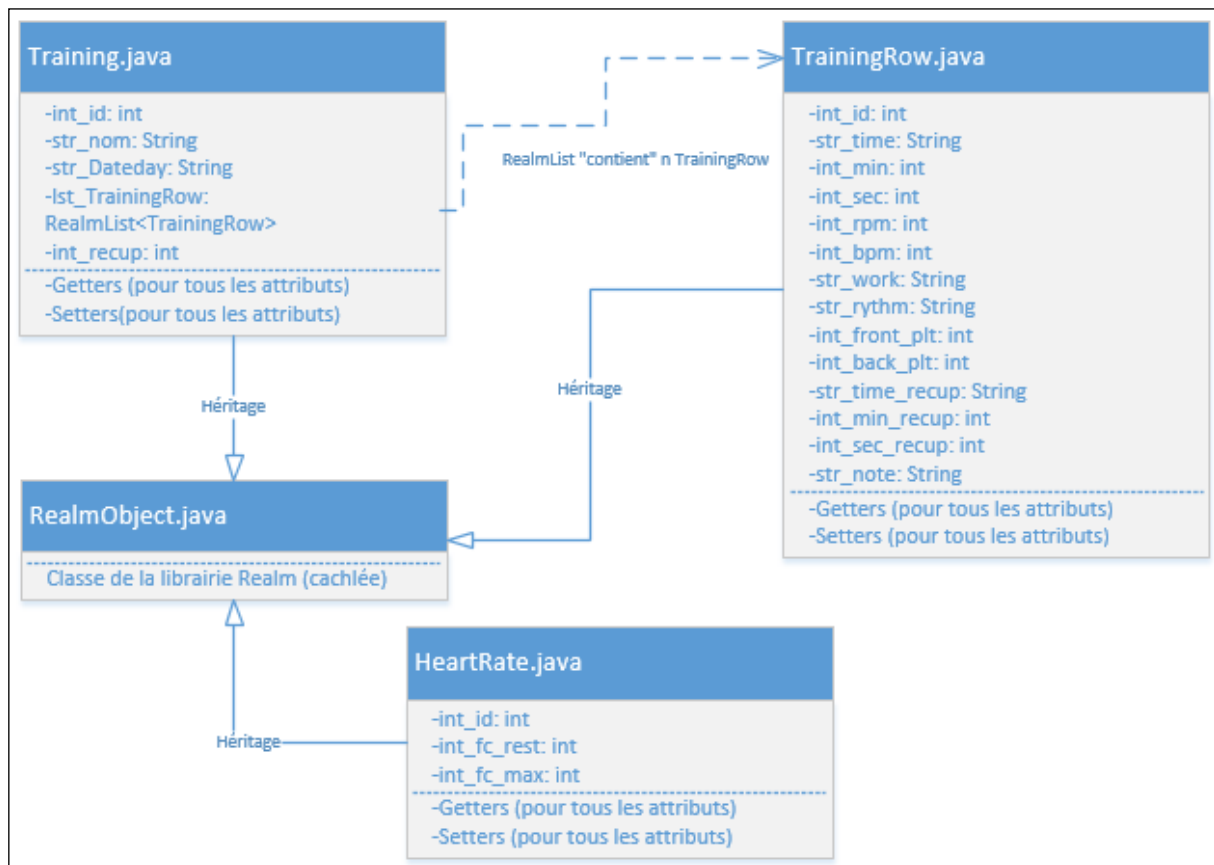
3.2.4 Base de Données

Realm sera utilisé pour la base de données. Cette librairie crée, à partir d'une classe objet java, une table dans la base de données. Cette table n'étant pas modifiable, seul le MLD est présenté ici : Une seule base de données avec 3 tables. Une table pour les entraînements (Training) une table pour les séquences (TrainingRow) et une table pour les fréquences Cardiaques. La table fréquence cardiaque n'a pas de lien avec les deux autres. La table Training a un lien avec la table TrainingRow, car Training peut contenir 0 ou plusieurs TrainingRow. TrainingRow appartient à un et un seul entraînement.



3.13 MCD de la base de données

La base de données étant gérée grâce aux classes objet java, un model UML de Design de classes permettra de compenser l'absence de MLD, MPD.



3.14 Design de classes UML

Le design de classes UML, remplace le MLD, MPD. Trois classes java héritent de la super classe RealmObject provenant de la librairie Realm. Realm s'occupe de convertir les classes qui héritent de RealmObject.java. Cet héritage permet de dire à Realm quelles classes il faut sauvegarder dans la base de données.

Training.java gère les entraînements. TrainingRow.java gère les séquences dans un entraînement. Pour représenter un lien 0 à plusieurs (0 à n) avec Realm, il suffit d'instancier un tableau/liste (RealmList), dans une des classes, contenant les objets de la seconde.

Dans l'UML, la classe Training.java (entraînements) possède une RealmList (lst_TrainingRow) d'objets TrainingRow (séquences), au même titre qu'un entraînement possède 0 ou plusieurs séquences.

3.3 Conception des tests

La partie des tests est complexe et beaucoup de fonctionnalités/méthodes devront être testées sur le moment, car la hiérarchie des vues sous-entend que si une fonctionnalité n'est pas implémentée correctement, celles dépendantes de la première ne pourront pas non plus fonctionner. Le temps imparti en fin de planification pour les tests finaux devra être utilisé correctement, mais il est réservé pour les tests finaux à savoir le comportement général de l'application.

Aucune méthodologie spécifique n'est prévue. Chaque bouton a une action spécifique, chaque partie de l'application doit se comporter comme décrit dans la partie analyse

graphique et fonctionnalité. Si l'une des fonctionnalités ne s'exécute pas correctement lors d'une vérification (compilation de l'application pour vérifier le comportement de l'application), cela pourra être assimilé à un bug qu'il faudra résoudre. De par le fait que ce genre de contrôle fait partie du développement de l'application, elle s'intègre dans la planification sous l'élément « programmation ». En effet, si une fonctionnalité de clic, d'ouverture de nouvelle activité ne fonctionne pas, il faut résoudre le problème avant d'avancer, voilà pourquoi il n'y a pas de temps réservé aux tests avant la dernière semaine du projet. Cependant, selon le cahier des charges, la validation de l'interface et la validation de l'application se feront à travers l'utilisation de l'application par le chef de projet qui jouera le rôle de bêta testeur.

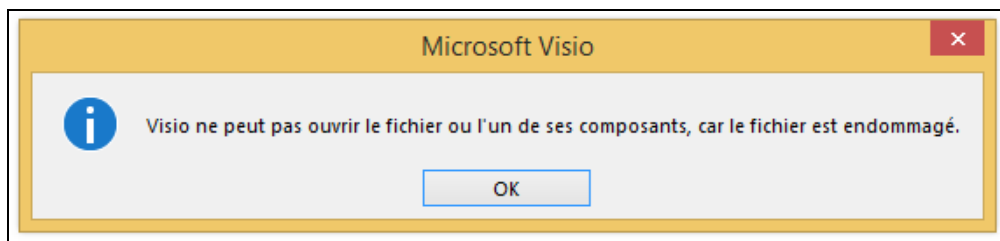
3.4 Planning détaillé

Selon les indications du chef de projet, le lien vers la planification détaillée se trouve en annexe.

4 Réalisation

4.1 Installation et préparation de l'environnement de travail

Office Visio étant absent de l'environnement de travail, il a été nécessaire de trouver une Machine Virtuelle ayant la suite Office installée. La création des schémas s'est faite à l'intérieur de l'environnement virtuel. Un problème est survenu lors de la reprise du travail, le lendemain, car le fichier visio a été endommagé.

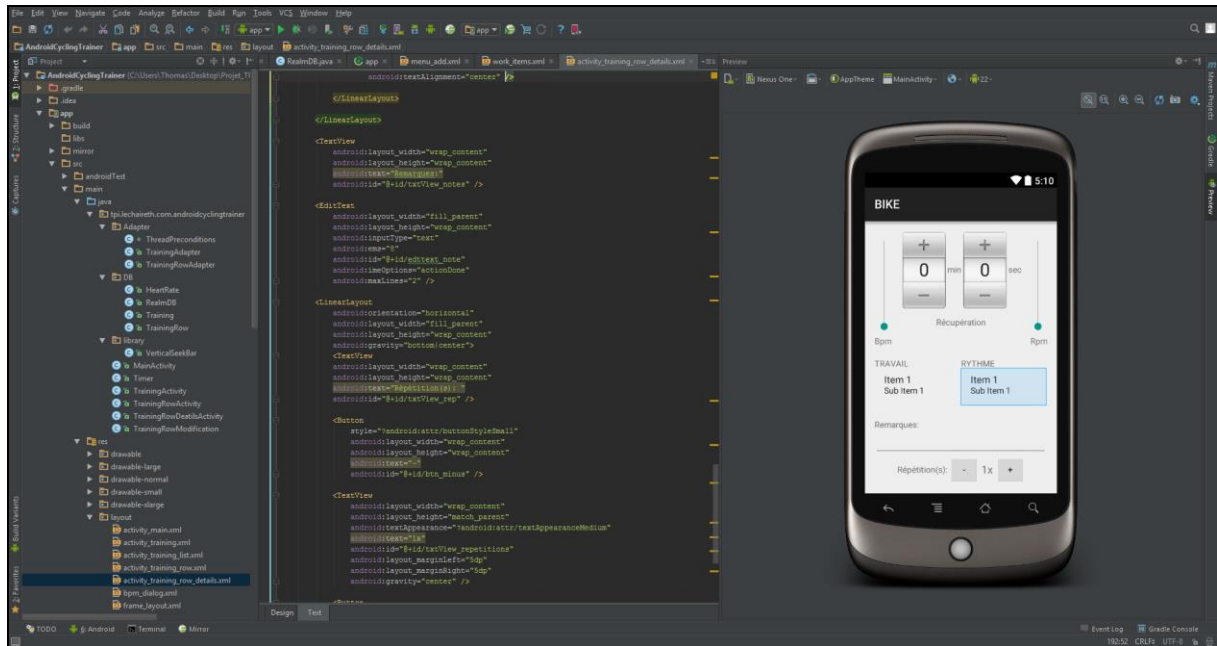


4.1 Erreur du fichier Visio

Une partie du travail a dû être refaite et environ dix quarts d'heures ont été utilisés pour recréer les schémas et interfaces.

Concernant l'installation de Android Studio (version 24.2), le téléchargement de L'IDE (cf. [Lexique](#) - point 16) & du SDK (cf. [Lexique](#) – point 17) s'est fait depuis le site de Google. (<https://developer.android.com/sdk/index.html>)

L'environnement de développement (IDE) Android Studio est un logiciel complet. Beaucoup de fonctionnalités sont présentes et utiles au bon développement d'applications. L'image ci-dessous montre une vue générale de l'interface graphique d'Android Studio. Chaque fichier est accessible via l'arborescence dans la colonne de gauche. L'IDE selon le type de fichier est capable de simuler une représentation graphique, comme il est possible de voir dans la colonne de droite, tout en affichant le code dans la colonne du milieu. C'est un environnement très intuitif et l'auto-complétion des méthodes et des variables est très agréable et rapidement prise en main.



4.2 Interface de l'environnement de développement Android Studio

La création d'un nouveau projet sous Android Studio se fait intuitivement et l'utilisateur est guidé dans les différentes étapes de création. Pour l'installation des librairies utiles dans l'application, il suffit d'ajouter une ligne de texte dans le fichier build.gradle (cf. [Lexique](#) – point 18) de l'application une fois le projet créé. Exemple :

```
//android Realm Library
compile 'io.realm:realm-android:0.80.1'
```

La ligne ci-dessus permet d'ajouter Realm dans notre projet/application.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.0.0'
    //android Realm Library
    compile 'io.realm:realm-android:0.80.1'
    /* android library for swipe gesture */
    compile 'com.android.support:recyclerview-v7:21.0.0'
    compile 'com.android.support:support-v4:22.+'
    compile 'com.daimajia.swipelayout:library:1.2.0@aar'
    /* android Yoyo Library */
    compile 'com.nineoldandroids:library:2.4.0'
    compile 'com.daimajia.easing:library:1.0.1@aar'
    compile 'com.daimajia.androidanimations:library:1.1.3@aar'
    //android Better Picker Library
    compile ("com.doomonafireball.betterpickers:library:1.6.0") {
        exclude group: 'com.android.support', module: 'support-v4'
    }
}
```

4.3 Installation des librairies

Le fichier build.gradle contient aussi les versions d'Android sur lesquelles le programme pourra fonctionner. Chaque version ou « API Level » correspond à une version de la plateforme. Par exemple, la plus petite version de l'Api est la 1 et la plus grande la 22. La version 1 est la

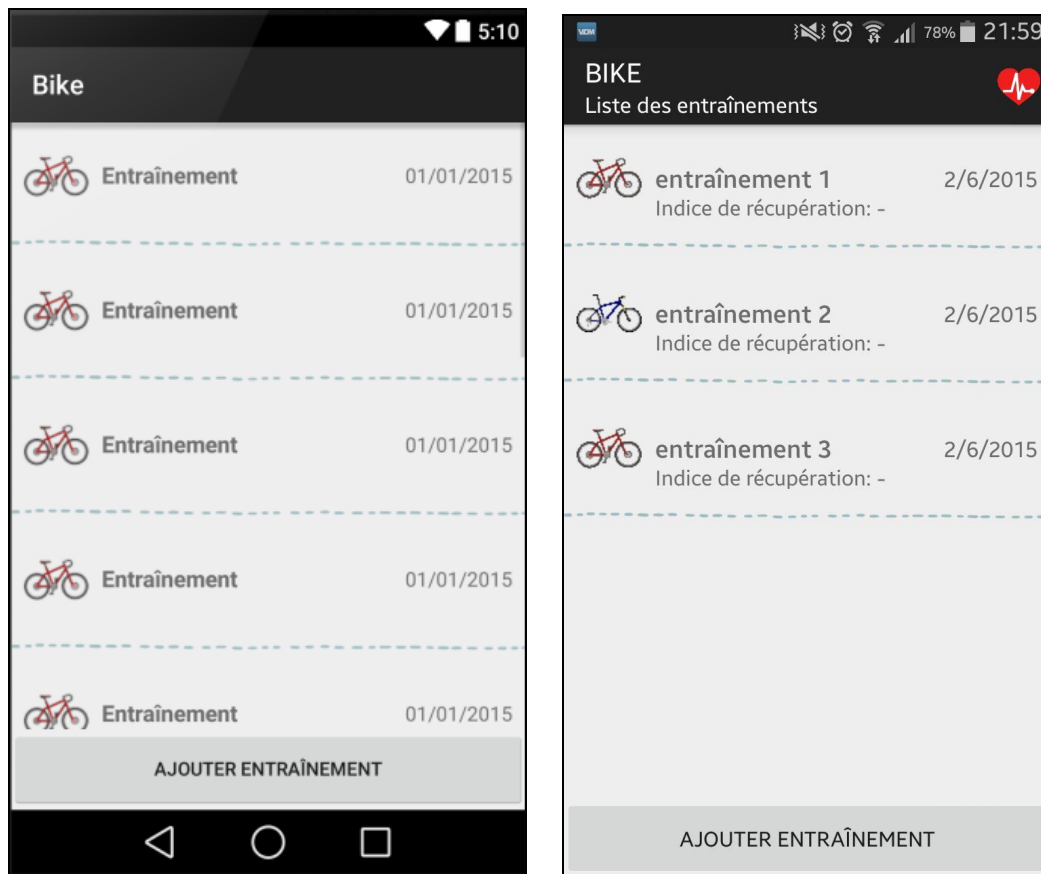
version nommée « BASE » et la version 22 se nomme « LOLLIPOP ». Une table de mise en relation entre le niveau de l'Api et la version d'Android est disponible en annexe. Cette application a pour minimum une version 11 et une version cible à 22. Ce choix s'est fait car certaines librairies requièrent une version minimum. Dans notre cas une des librairies utilisées n'aurait pas fonctionné sur une version 8, le choix s'est imposé à nous. Cela ne pose que peu de problèmes, car l'application reste compatible avec plus de 95% des smartphones utilisant Android comme système d'exploitation.

4.2 Implémentation de l'application

La création de l'application commence par la création des interfaces graphiques. La première interface créée, est liée grâce au nom du fichier XML à la classe `Training_activity.java` par la ligne de code ci-dessous.

```
setContentView(R.layout.activity_training_list);
```

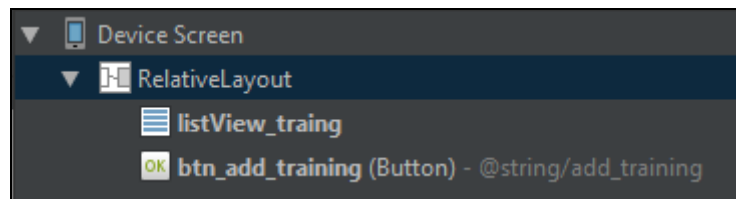
Le fichier `activity_training_list.xml` est le layout de la première activité de l'application. Il ressemble dans sa première implémentation à l'image de gauche ci-dessous.



4.4 Interface `Training_activity.java` avant et après modifications.

Les éléments qu'il contient sont :

Un layout de type « `RelativeLayout` » (cf. [Lexique](#) – point 19) contenant une liste du type « `ListView` » (cf. [Lexique](#) – point 20) pour afficher les entraînements au fur et à mesure, ainsi qu'un Bouton qui va lancer la boîte de dialogue (`btn_add_training`) permettant l'ajout d'un entraînement et de son type.



4.5 Hiérarchie des éléments du fichier activity_training_list.xml

La ListView (lstView_training) contient, elle, un autre layout (Voir Image : 4.6) qui représente une ligne de la liste. C'est la ListView qui se charge de prendre le layout pour la ligne et de le multiplier par le nombre d'éléments contenus dans la liste par le biais d'un Adapter (cf. [Lexique](#) – point 21). Toutes ces opérations sont réalisées dans un « Adapter ». Un adapter est une classe java qui va servir au programme pour traiter toutes les lignes d'une listView et permettre d'avoir un affichage et des interactions identiques sur chaque ligne.

L'application a donc besoin de deux Adapter, un pour la liste des entraînements et un pour la liste des séquences. Le layout pour une seule ligne d'entraînement. Comporte un CustomLayout nommé swipe. Ce type de layout est issu de la librairie AndroidSwipeLayout qui permet de faire apparaître un second layout par un geste de swipe vers la gauche. Ce dernier est utilisé pour effacer l'entraînement si l'utilisateur le désire.

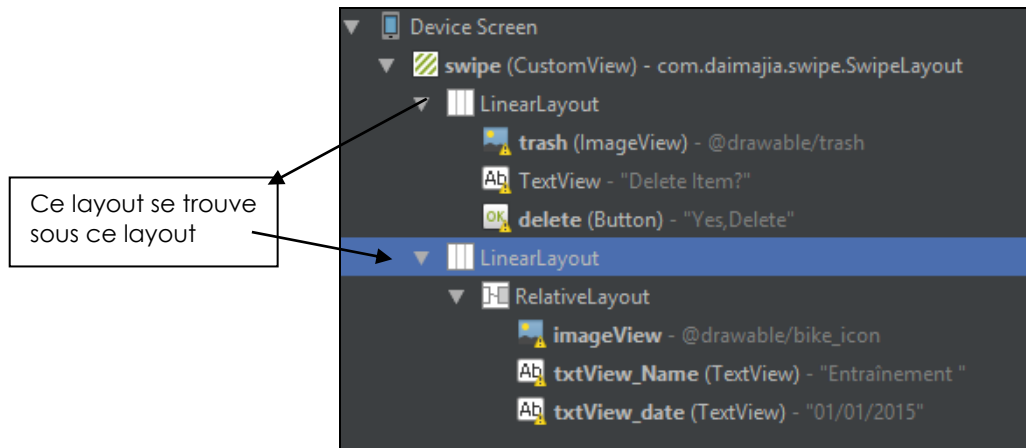


4.6 Layout d'un élément de la ListView entraînement



4.7 Layout qui apparaît lors du swipe

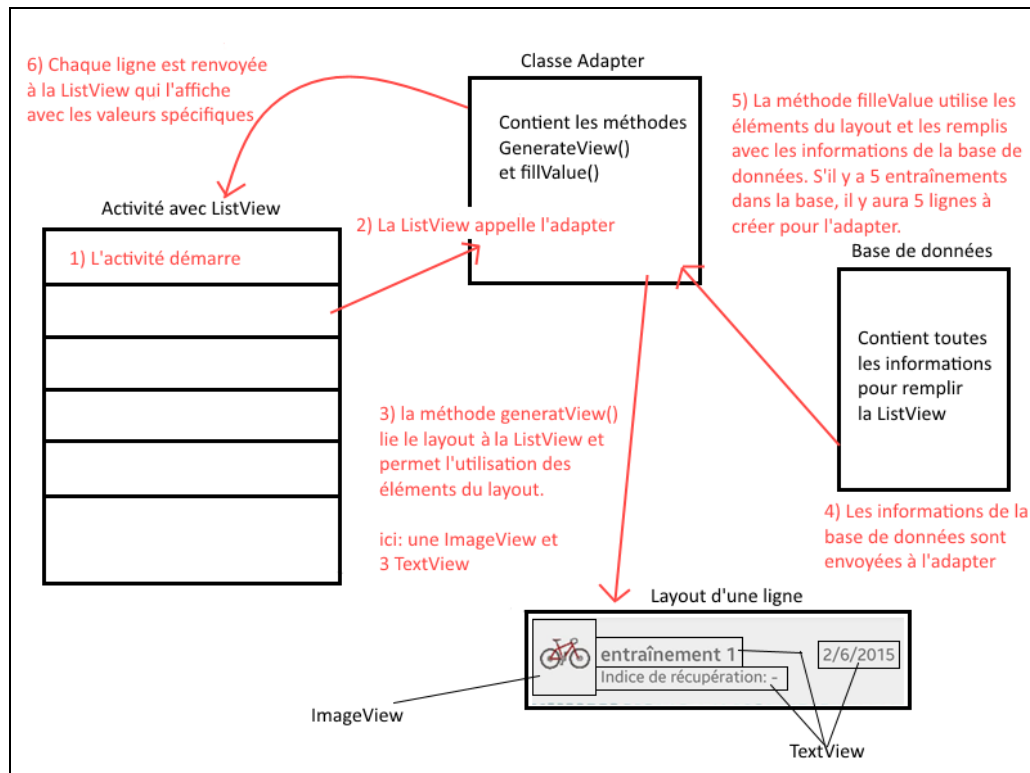
Comme vu précédemment, le layout pour la ligne est spécial car il provient d'une librairie extérieure à Android. L'élément (CustomView) « swipe » contient ici un premier LinearLayout avec 3 éléments. « trash » qui est l'icône de la poubelle. Une TextView pour la partie Texte, qui demande si l'on veut vraiment supprimer l'entraînement et un « Button » pour lancer l'action et effacer l'entraînement. Cette partie n'est visible, que lors du swipe.



4.8 Hiérarchie des éléments du fichier training_list_item.xml

Le layout qui recouvre ce premier layout est lui aussi un « LinearLayout » qui contient une « ImageView » qui permet d'afficher soit le vélo Bleu pour le vtt ou le vélo rouge pour le vélo de route. Deux « TextView » permettent l'affichage du nom de l'entraînement et de la date. Le tout est contenu dans un « RelativeLayout » afin que tous les éléments s'affichent les uns par rapport aux autres. Le « RelativeLayout » a été rajouté ici car, lorsqu'on utilise seulement le « LinearLayout » les dates ne s'alignent pas à droite de l'écran, mais se collent au nom de l'entraînement, ce qui donne un effet peu esthétique.

Pour gérer l'affichage de la ListView, il faut sous Android, un Adapter. Le premier Adapter est implémenté dans la classe java TrainingAdapter car comme son nom l'indique il est lié à la ListView des entraînements. Il faut savoir que des types d'Adapters de base sont déjà définis utilisables dans de cas où la ligne ne contient que du texte. Dans les cas où la ligne contient des images ou des éléments plus spécifiques, on parle de CustomAdapter, car ce sont des Adapters qui sont liés à un layout non standard.



4.9 Schéma du fonctionnement d'un Adapter

Pour l'application l'adapter devra, en plus, répondre aux contraintes d'implémentations imposées par l'utilisation de la librairie AndroidSwipeLayout. Ces contraintes ne sont que minimales comparées au temps que la librairie permet d'économiser pour l'implémentation de cette fonctionnalité. L'utilisation de la CustomView swipe est une des contraintes imposées lors de l'utilisation.

La seconde contrainte se situe dans l'Adapter. La méthode `generateView()` qui permet de lier le layout de la ligne à l'adapter. Les éléments contenus dans le layout peuvent alors être utilisés. Exemple : accéder au bouton effacer et définir son action ou insérer le titre dans la TextView. C'est dans cette méthode que la gestion du swipe est gérée et, point important, cette méthode retourne une vue.

```
public View generateView(int position, ViewGroup viewGroup)
```

Toute la gestion des clics et des listeners (cf. [Lexique](#) – point 22) a été mise, lors de la première implémentation, dans cette classe. Un problème avec la récupération de l'identifiant est apparu. L'implémentation des listeners était correcte, mais la variable « position » présent dans les paramètres de la méthode, prenait la valeur zéro au lieu de prendre la valeur de la ligne sélectionnée. C'est sur le forum de la librairie que la solution a été trouvée. La position était égale à zéro car les listeners n'étaient pas dans la bonne méthode. Déplacer les listeners dans la seconde méthode `fillValues()` a suffi à résoudre l'erreur.

La seconde méthode `fillValues()` comme son nom l'indique permet de remplir les valeurs des éléments du layout. Par exemple : mettre le nom de l'entraînement dans la TextView prévue à cet effet (`txtView_Name`). Il est important de noter que cette méthode a pour paramètres un `int` représentant la position de la ligne et une `View` (cf. [Lexique](#) – point 23) qui provient du retour de la méthode `generateView()`.

```
public void fillValues(final int position, View convertView)
```

Le paramètre « position » représente le numéro de la ligne et « convertView » le layout avec les éléments de ce dernier. Dans cette méthode il faut ajouter un listener qui se déclenchera lors du clic sur l'entraînement. Cette méthode récupère simplement l'id de l'entraînement, dans la base de données, grâce à la position de la ligne. Le clic lance l'activité suivante (Voir image 4.20) et affiche les séquences présentes dans la RealmList de l'entraînement. Un second listener est implémenté pour gérer le clic sur le bouton « EFFACER », il permet de lancer la méthode de la classe RealmDB qui va récupérer l'entraînement dans la base de données grâce à la position et à l'id de ce dernier.

Concernant la suppression d'un entraînement, la méthode choisie est celle de la récupération complète de ce dernier grâce à son id, puis de lancer sa suppression. Cela est dû au fait qu'il faut supprimer l'entraînement dans la base de données, mais aussi dans la ListView affichant les entraînements. Il faut supprimer l'entraînement de la ListView avant de le supprimer dans la base de données, car l'inverse cause une erreur de récupération de l'entraînement. En effet, l'entraînement n'existant plus dans la base de données il est impossible de le supprimer de la ListView, car la ligne (de la ListView) n'est plus liée à la référence en base de données de l'entraînement, cette dernière n'existant plus. La méthode pour la suppression de l'entraînement provient de la classe RealmDb.java qui contient toutes les méthodes d'interaction avec la base de données Realm.

```
//When we click on the delete button
viewHolder.btnDelete.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        RealmDB realmDB = new RealmDB(mContext);

        //get the training we want to delete
        Training t = realmDB.getATrainingWithID(trainingList.get(position).getInt_id());

        //delete the training from the list first
        trainingList.remove(t);

        //from the DB after
        realmDB.removeTraining(t);

        //hide the swipe Layout and display infos and training Name
        swipeLayout.close();

        //we notify that the Data list has changed
        notifyDataSetChanged();
    }
});
```

4.10 Méthode de suppression des entraînements

Lors de l'implémentation de cette première listView, un problème avec le swipe s'est déclenché. Une mauvaise reconnaissance du geste et un ralentissement lors de l'affichage sont apparus. En supprimant la ligne de code ,

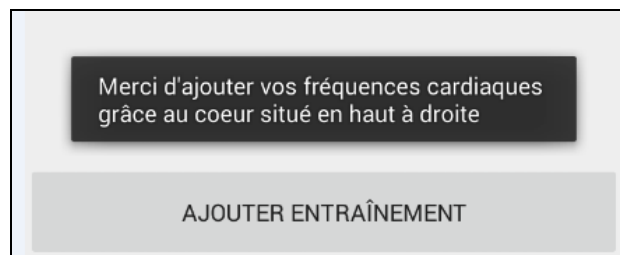
```
SwipeLayout.setShowMode(SwipeLayout.ShowMode.LayDown);
```

correspondant au type d'animation s'exécutant lors du swipe, le problème a disparu. Cette animation n'est pas reconnue dans toutes les versions d'Android et pose des problèmes dans certains cas.

L'affichage des entraînements est la première interface lorsque l'utilisateur ouvre l'application. Le clic sur le bouton (1) (Image 4.11) lance la méthode `setOnClickListener` liée à ce dernier. Ce listener se déclenche lorsqu'il intercepte un clic sur le bouton ajouter entraînement. Il se charge de contrôler que les fréquences cardiaques de l'utilisateur ont été enregistrées en allant vérifier cela grâce à la méthode `getHeartRate()` de la classe `RealmDB`.

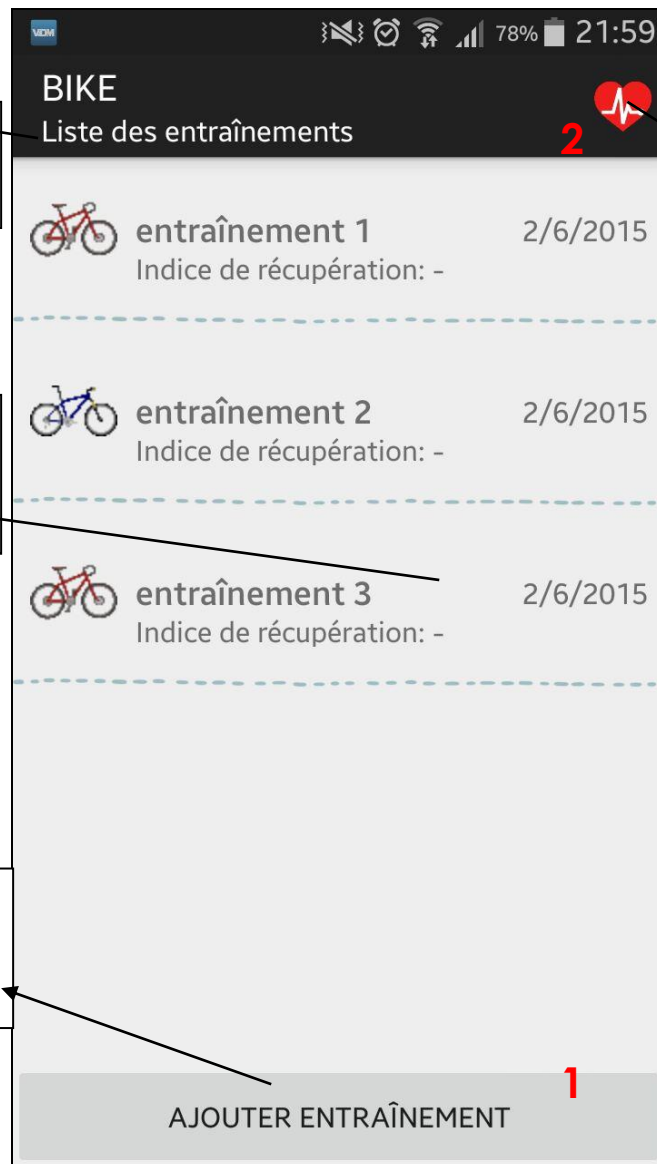
Dans le cas où les fréquences cardiaques n'ont pas été renseignées, un « Toast » (cf. [Lexique](#) – point 24) informe l'utilisateur qu'il doit remplir ses fréquences cardiaques grâce au bouton se situant en haut à droite de l'ActionBar.

Les fréquences cardiaques sont vides. Le Toast apparaît



4.11 Toast indiquant que les fréquences cardiaques doivent être remplies

La dernière implémentation de l'interface utilisateur donne un bon aperçu des fonctionnalités et interactions possibles avec l'application.



4.12 Interface finale de l'affichage des entraînements

Pour l'ajout des fréquences cardiaques, une icône présente dans l'ActionBar permet d'ouvrir une boîte de dialogue. Pour ajouter ce bouton, deux méthodes, un fichier pour le menu et un drawable (cf. Voir [Lexique](#) – point 25) sont nécessaires.

Le fichier pour le menu et le fichier menu_fc.xml présent dans le dossier ressources « menu ». (voir chapitre 4.4 – Ressources)

```

<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:compat="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/action_add_fc" compat:showAsAction="always"
        android:icon="@drawable/fc_button" android:title="Fréquences Cardiaques" />
</menu>

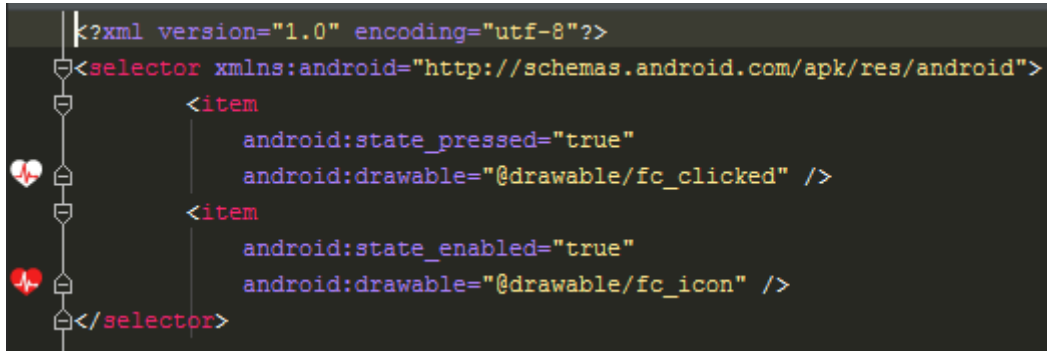
```

4.13 fichier XML du menu fréquence cardiaque

La ligne 4 affiche le lien vers le drawable utilisé pour l'icône du bouton qui permet de créer le lien avec l'icône (point 2 de l'image 4.9).

```
android:icon="@drawable/fc_button"
```

Le fichier est lui aussi un fichier XML qui contient deux images, une pour l'état standard et une image lors du clic. Ainsi les deux états sont différenciés.



```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:state_pressed="true"
        android:drawable="@drawable/fc_clicked" />
    <item
        android:state_enabled="true"
        android:drawable="@drawable/fc_icon" />
</selector>
```

4.14 fichier XML de l'icône (Point 2 Image 4.12)

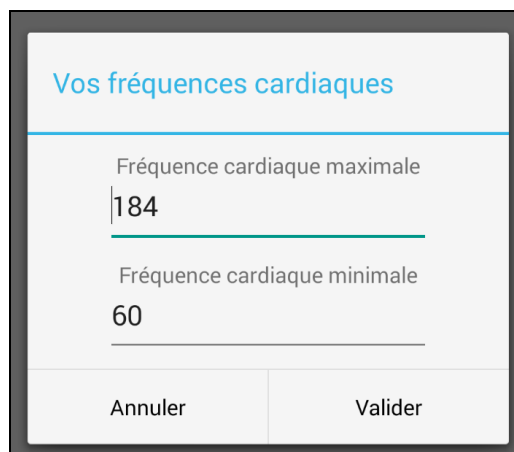
L'affichage de l'icône se fait par le biais de deux méthodes. La première méthode :

```
public boolean onCreateOptionsMenu(Menu menu)
```

Est la méthode qui permet de lier le fichier XML du menu (Image 4.13) au menu de l'application. (Image 4.12 point 2). La seconde méthode est la méthode qui permet de définir les actions pour chaque élément du menu. Exemple : que faire lors du clic.

```
public boolean onOptionsItemSelected(MenuItem item)
```

On remarque que cette méthode prend en paramètre un MenuItem qui représente l'item dans le menu. On peut ainsi implémenter une méthode qui va effectuer ce que l'on souhaite lors du clic sur le bouton des fréquences cardiaques. Le clic aura pour effet d'ouvrir la boîte de dialogue ci-dessous.



4.15 Boîte de dialogue pour l'insertion des fréquences cardiaques

C'est ici que l'utilisateur doit rentrer ses fréquences cardiaques. La boîte de dialogue est liée à une classe privée CheckValueHeartListener

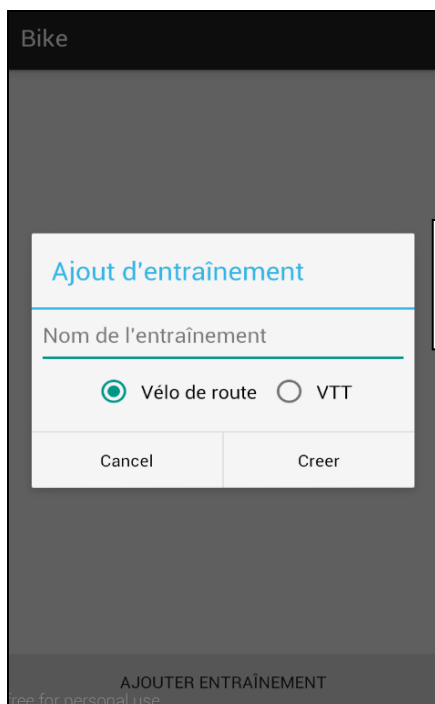
```
class CheckValueHeartRateListener implements View.OnClickListener
```

Cette classe étend la classe View.OnClickListener qui permet de contrôler les deux champs d'édition (EditText) (cf. [Lexique](#) – point 26) au moment du clic sur le bouton « Valider ». Le contrôle teste que les valeurs ne soient pas du texte, soient des valeurs entre 30 et 220 et qu'elles ne correspondent pas à rien. (Voir modification du 03.06.2015)

Une fois le contrôle et la validation effectuées, une méthode de la classe RealmDB se charge de sauvegarder les fréquences dans la base de données.

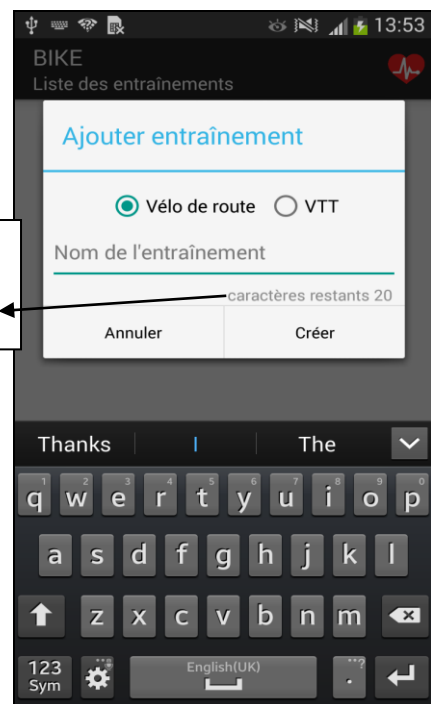
```
realmDB.saveHeartRate(int max, int min);
```

Les valeurs sont enregistrées dans la base de données et il est maintenant possible pour l'utilisateur d'ajouter un entraînement.



4.16 Boîte de dialogue pour l'ajout d'entraînement

TextWatcher pour le contrôle du nombre de caractères



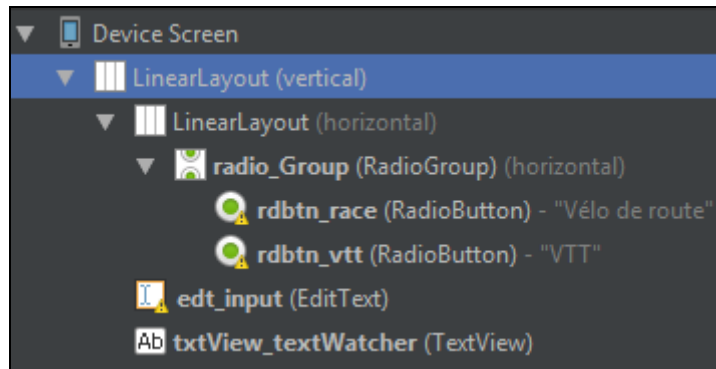
4.17 Version finale

La version finale indique à l'utilisateur le nombre de caractères restant pour le nom de l'entraînement et ouvre le clavier directement en simulant la sélection de la ligne d'édition. Cliquer dans l'EditText permet de faire apparaître le clavier. L'ajout de la ligne de code ci-dessous rend l'utilisation de l'application par l'utilisateur plus agréable, car elle ouvre le clavier automatiquement dès l'apparition de la boîte de dialogue.

```
edt_input.requestFocus();
```

L'EditText de cette boîte de dialogue est liée à un TextWatcher (cf. [Lexique](#) – point 27) qui permet de contrôler le nombre de caractères entrés. La boîte de dialogue est, elle, liée à une classe privée checkValueTrainingListener qui s'occupe de contrôler que le nom de

l'entraînement n'est pas déjà utilisé, qu'il n'est pas vide, ni trop grand par rapport au nombre de caractères maximums admis.



4.18 Hiérarchie des éléments de la boîte de dialogue entraînement

La boîte de dialogue est créée en utilisant le layout du fichier training_dialog.xml. (Voir Image 4.18). Les éléments sont alors accessibles depuis le code et permettent d'implémenter le TextWatcher et les méthodes de contrôle qui sont liées à une classe privée comme pour la boîte de dialogue des fréquences cardiaques. Ces deux classes privées se trouvent en bas de la classe TrainingActivity.java présente dans les annexes.

```
class CheckValueTrainingListener implements View.OnClickListener
```

Cette classe privée contrôle que l'entraînement n'est pas déjà existant en utilisant une méthode de la classe RealmDB.

```
public boolean isUnique(String name)
```

isUnique prend en paramètre le nom de l'entraînement, cherche si ce nom existe déjà dans la base de données et retourne un booléen faux si le nom n'est pas déjà utilisé. Le contrôle se poursuit en vérifiant que l'entraînement n'a pas un nom vide ou plus de 20 caractères. Les 20 caractères permettent de limiter la taille du nom de l'entraînement afin qu'il ne déborde pas sur les autres éléments du layout.

Les deux radios boutons doivent être dans un seul radio groupe afin que lorsqu'un des éléments est sélectionné, cela désélectionne automatiquement l'autre. Le choix de l'un des deux éléments (VTT ou Vélo de route) va être sauvegardé dans la base de données et utilisé pour afficher l'image du vélo bleu ou rouge. L'entraînement est alors créé en récupérant la date du jour, le nom de l'entraînement et le type. C'est après les contrôles des éléments que la classe RealmDB, qui s'occupe des interactions avec la base de données, va créer l'entraînement et l'ajouter à la base. (Pour mieux comprendre le fonctionnement de Realm et de la classe RealmDB voir le chapitre 4.3 – Implémentation de la base de données).

```
public void createTraining(String name, boolean isVtt)
```

La méthode createTraining qui a pour paramètres le nom de l'entraînement et son type (représenté par le boolean isVtt). Va ajouter le nouvel objet dans la base de données. Un entraînement est un objet Training issu de la classe du même nom (Training.java). Encore une fois le chapitre 4.3 explique le fonctionnement des classes de la base de données et des interactions entre l'application et les éléments sauvegardés.

Le nombre du mois a posé un problème lors de la sauvegarde de la date. Chaque mois s'affichait dans la listView avec un mois de retard. Après recherche sur internet, le problème était dû au fait qu'Android initialise le mois de janvier à zéro et non à un. Pour résoudre le problème la variable concernant le mois a été initialisée à 1+le nombre du mois en cours.

Le nouvel entraînement créé est alors sélectionnable dans la listView et ouvre l'activité TrainingRowActivity. Le layout de cette activité est similaire à celui de la classe Training_activity.java. Une liste, contenant toutes les séquences d'un entraînement. Le fonctionnement est identique à l'interface affichant les entraînements et permettant aussi l'effacement grâce au swipe.

Le layout du fichier XML activity_training_row contient un « RelativeLayout » qui lui contient 3 éléments qui sont, une « ListView », un « LinearLayout » (cf. Lexique – point 28) et un « Button » pour démarrer l'entraînement. Ce bouton lance l'activité « Timer ». Le « LinearLayout » permet d'afficher une barre de résumé affichant, 3 « TextView » qui affichent respectivement, le nombre de séquences dans l'entraînement, la moyenne des Bpm et le temps total de l'entraînement. Cette partie du layout n'était pas présente dans la maquette graphique du départ. Cet ajout a été motivé pour améliorer l'expérience utilisateur et pour répondre au cahier des charges qui demandait un bref résumé de l'entraînement. En effet, il est pratique pour un utilisateur souhaitant s'entraîner de connaître quelques informations sur l'entraînement qu'il désire effectuer avant de le démarrer. Pour le nombre de séquences, cela correspond simplement à la taille de la liste. L'implémentation des deux autres méthodes, celle qui permet de calculer la moyenne de Bpm et celle pour le temps total, se fait dans la classe RealmDB. La première méthode pour calculer la moyenne des Bpm.

```
public int getAverageBpmOfTraining(int _id)
```

Cette méthode récupère l'entraînement grâce à une requête et au paramètre passé à la fonction. Ci-dessous la requête Realm pour récupérer l'entraînement.

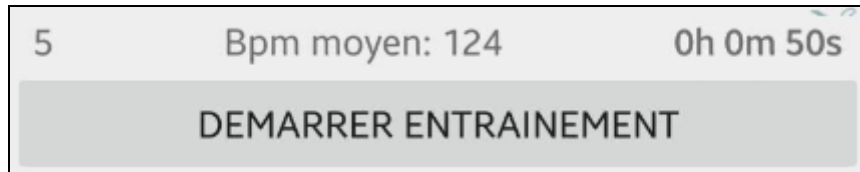
```
realm.where(Training.class).equalTo("int_id", _id).findFirst();
```

Le nombre de séquences est alors connu grâce à la taille de la liste. Il suffit de faire une boucle qui additionne toutes les valeurs des Bpm contenues dans chacune des séquences et de diviser le résultat par la taille de la liste (correspondant aussi au nombre total des séquences).

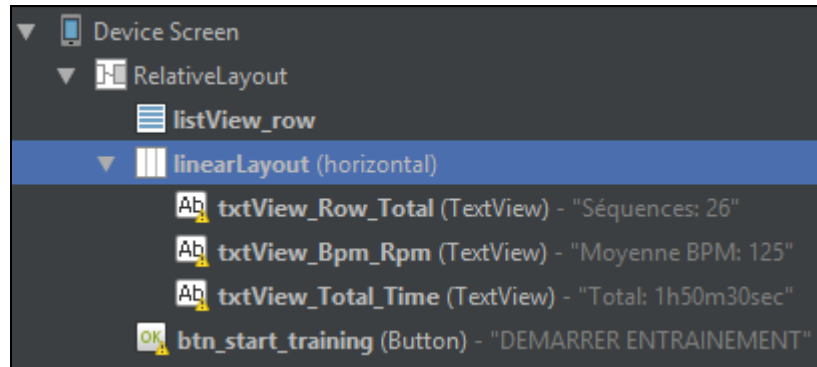
La seconde méthode comme la première récupère et additionne toutes les minutes et toutes les secondes de chaque séquence dans deux variables. Puis divise la somme des minutes par soixante pour trouver les heures, effectue un modulo soixante pour trouver les minutes et divise ces minutes par soixante encore pour trouver les secondes. Ci-dessous les calculs, dans la méthode, pour trouver le temps total.

```
//total min/ 60 and we get hours
int hour = totalMin / 60;
//totalMin % 60 and we get minutes
int min = totalMin % 60;
//totalSec / 60 and we get the minutes
min += totalSec / 60;
//totalSec % 50 and we get the seconds
int sec = totalSec%60;
```

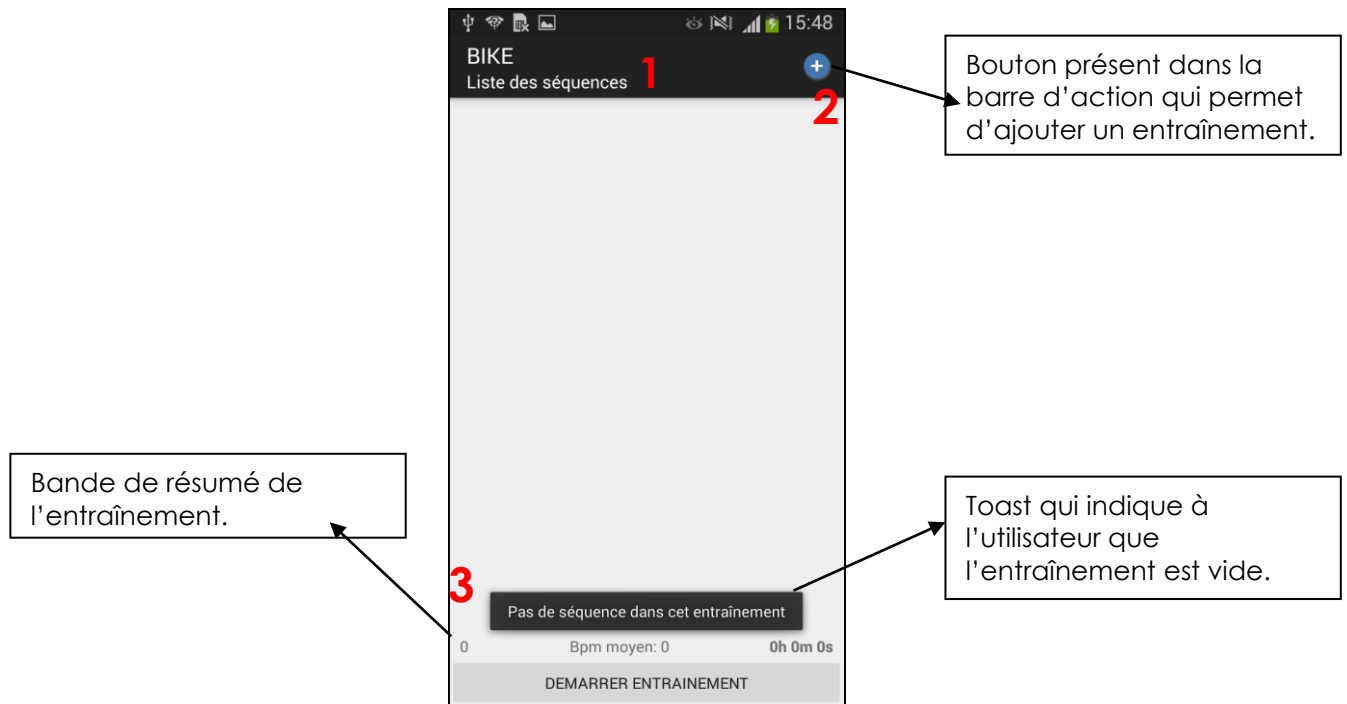
Image 4.19 est l'affichage du résultat des méthodes dans la barre de résumé de l'entraînement.



4.19 Bande de résumé d'entraînement



4.20 Hiérarchie des éléments fichier activity_training_row.xml

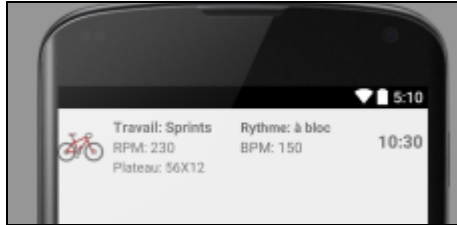


4.21 Représentation graphique du fichier activity_training_row.xml

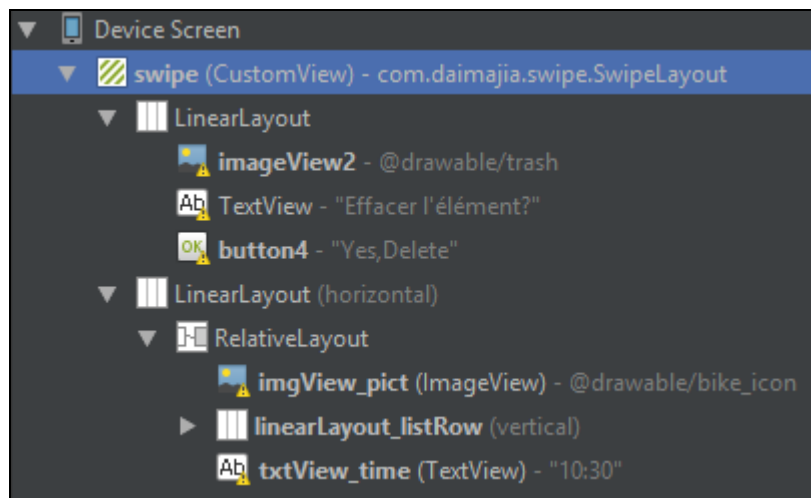
Lorsque l'utilisateur arrive pour la première fois sur la liste des séquences (1) un toast s'affiche (Point 3 de l'image 4.21) indiquant qu'il n'y a aucune séquence d'enregistrée dans l'entraînement. Si l'utilisateur tente de démarrer l'entraînement alors qu'il n'y a pas de séquence, un toast s'affiche lui demandant d'ajouter au moins une séquence à l'aide du bouton en haut à droite. (Image 4.21 point 2).

L'image 4.22, représente une ligne de la « ListView ». Comme pour la « ListView » de l'activité Training, une « CustomView » de la librairie AndroidSwipeLayout. Le premier « LinearLayout » est celui de l'élément du dessous qui ne sera visible qu'après avoir swipé. Le second « LinearLayout » est celui de l'affichage des données avec un « RelativeLayout » à l'intérieur

pour faciliter l'affichage des éléments les uns par rapport aux autres. Une « ImageView » pour l'image et un sous layout de type « LinearLayout », (LinearLayout_listRow) permettent l'affichage des valeurs propres à la séquence, comme le type de « Travail », le « rythme », les BPMs, les RPMs, et/ou les plateaux.



4.22 Représentation graphique du fichier training_row_list_item.xml



4.23 Hiérarchie du fichier training_row_list_item.xml

L'activité a besoin comme vu précédemment de deux boutons. Un pour démarrer l'entraînement (btn_start_training) (cf. Image 4.20) et un autre bouton pour ajouter une séquence d'entraînement (Image 4.21 point 2).

Comme dans la première activité de l'application, ce second bouton n'est pas matérialisé dans les éléments du fichier activity_training_row.xml, car il s'affiche sur l'ActionBar. Pour implémenter ce bouton il faut à nouveau implémenter les deux méthodes vues précédemment.

```
public boolean onCreateOptionsMenu(Menu menu)
```

```
public boolean onOptionsItemSelected(MenuItem item)
```

Le fichier XML pour le bouton qui utilise deux images correspondant aux deux états du bouton. Comme pour le bouton pour ajouter les fréquences cardiaques.

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:state_pressed="true"
        android:state_enabled="true"
        android:drawable="@drawable/rond_pressed" />
    <item
        android:state_enabled="true"
        android:drawable="@drawable/rond_over" />
</selector>
```

4.24 Fichier XML pour le bouton

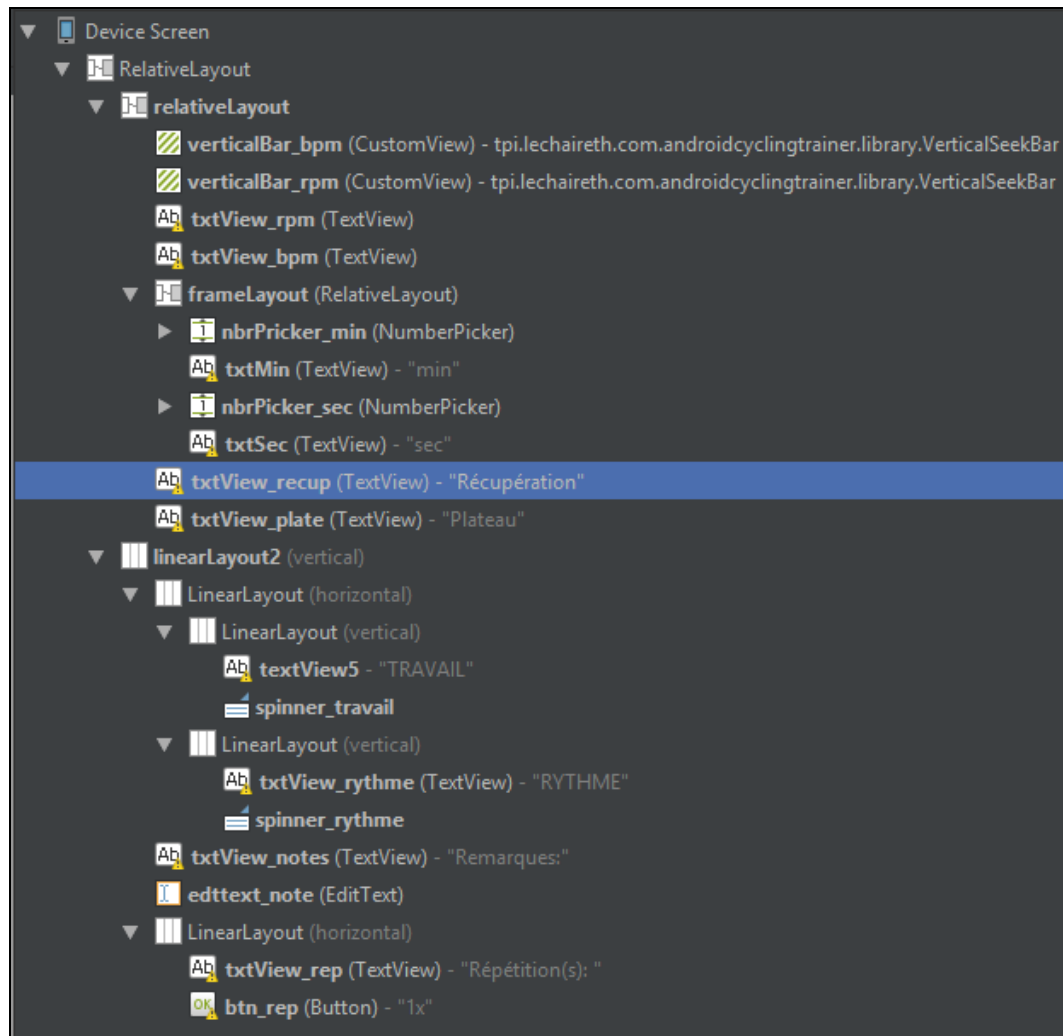
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:compat="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/action_add" android:title="Ajouter"
        android:icon="@drawable/add_button" compat:showAsAction="always" />
</menu>
```

4.25 Menu du bouton ajouter

Lors du test, le bouton ne voulait pas apparaître. L'ajout de l'attribut dans le fichier xml du menu a résolu le problème d'affichage.

```
compat:showAsAction="always"
```

C'est lors de l'ajout d'une séquence que le layout de l'activité TrainingRowDetailsActivity.java est utilisé. Il comprend tous les éléments pour enregistrer une séquence. Plusieurs Layouts sont imbriqués les uns dans les autres afin de garantir la structure sur n'importe quel téléphone Android. Tous les éléments de l'image 4.31 sont présents dans la hiérarchie.



4.26 Hiérarchie du fichier activity_training_row_details.xml

Le layout contient, deux « VerticalBar » de la librairie Vertical-SeekBar-Android. On remarque dans l'image précédente que les deux éléments verticalBar_bpm et verticalBar_rpm sont des CustomView provenant bien de la librairie. Ces deux VerticalBar héritent d'une classe abstraite et nécessitent l'implémentation de 3 méthodes qui permettent d'effectuer des changements lorsque l'on commence à toucher la barre, lorsque l'on arrête ou lors de chaque changement de valeur de la barre. Cette dernière méthode est la seule nécessaire pour l'application. L'important est de ne pas oublier de définir la valeur maximale que la barre peut atteindre grâce à la fonction setMax(int).

```
//max bpm value
verticalBar_bpm.setMax(max_bpm-min_bpm);

/*****
 * Method for the seek bar change listener for bpm bar
 *****/
verticalBar_bpm.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        //maximum = 200 and min = 30;
        int_progress_bpm = progress + min_bpm;
        //set progression to textView
        txtView_bpm.setText(int_progress_bpm+"");
    }

    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {
    }

    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {
    }
});
```

4.27 Méthodes abstraites à implémenter pour les VerticalBar

Deux TextView pour afficher les valeurs des Bpm et de Rpm choisies grâce aux « VerticalBar ». Deux « NumberPicker » permettant de sélectionner les minutes et les secondes de la séquence. Deux « TextView » cliquables pour ajouter le temps de récupération et le choix des plateaux du vélo. Lors du clic sur un de ces deux éléments (Image 4.31 point **1** & **2**) une AlertDialog ou boîte de dialogue s'ouvre. La création d'une boîte de dialogue passe par la création de deux objets. Un premier objet AlertDialog et un second AlertDialog.Builder. Ce second objet permet de renseigner plusieurs informations qui sont utiles lors de l'affichage.


```
//Build an Alert Dialog
AlertDialog.Builder alrt_builder = new AlertDialog.Builder(TrainingRowModification.this);
//set the view of the Dialog
alrt_builder.setView(v1);
alrt_builder.setTitle(getResources().getString(R.string.choice_braquets)); //title
alrt_builder.setPositiveButton("Valider", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) { //OK Button
        int int_front_value, int_back_value;
        //numberPickers
        int_front_value = nbrPicker_front.getValue();
        int_back_value = nbrPicker_back.getValue();
        txtView_gear.setText("Plateaux:"+int_front_value + "X" + int_back_value);
    }
});
alrt_builder.setNegativeButton("Annuler", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) { //Cancel Button
        //do nothing
        dialog.dismiss();
    }
});

//create the alert dialog
AlertDialog = alrt_builder.create();
//now show the alert dialog to user
AlertDialog.show();
```

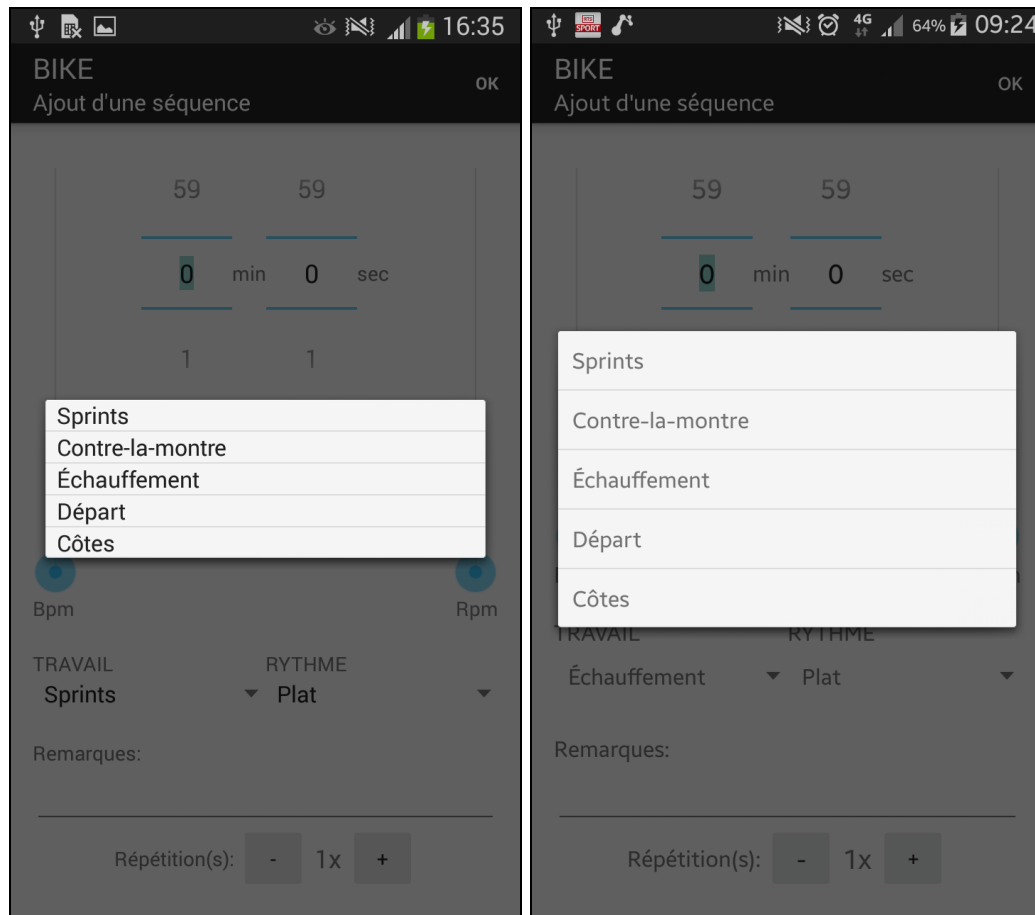
4.28 Exemple de création d'une Boîte de dialogue

L'objet `AlertDialog.Builder` présenté ci-dessus offre plusieurs méthodes, ici comme l'`AlertDialog` devra avoir deux boutons « Valider » et « Annuler », il faut implémenter les méthodes `setPositiveButton()` et `setNegativeButton()` qui se déclencheront lors du choix de l'utilisateur. La méthode `setTitle()` permet d'ajouter un titre à notre boîte de dialogue et la méthode `setView()` permet de définir la vue qu'aura notre boîte de dialogue. Une fois les paramètres de l'`AlertDialog.Builder` renseignés, il est possible de créer la boîte de dialogue, avec la méthode `create()`, et de l'insérer dans un objet `AlertDialog`.

Les vues travail et le rythme ouvrent des « Spinners » ou plus communément appelés menus déroulants. Ils permettent une sélection parmi plusieurs types de travail ou rythme prédéfinis. Ces menus déroulants sont liés à des fichiers XML regroupant la liste des éléments sélectionnables. Les deux images ci-dessous montrent le fichier XML avec les items du Spinner « Travail » et l'affichage graphique de ce dernier. Voir modification 04.06.2015.

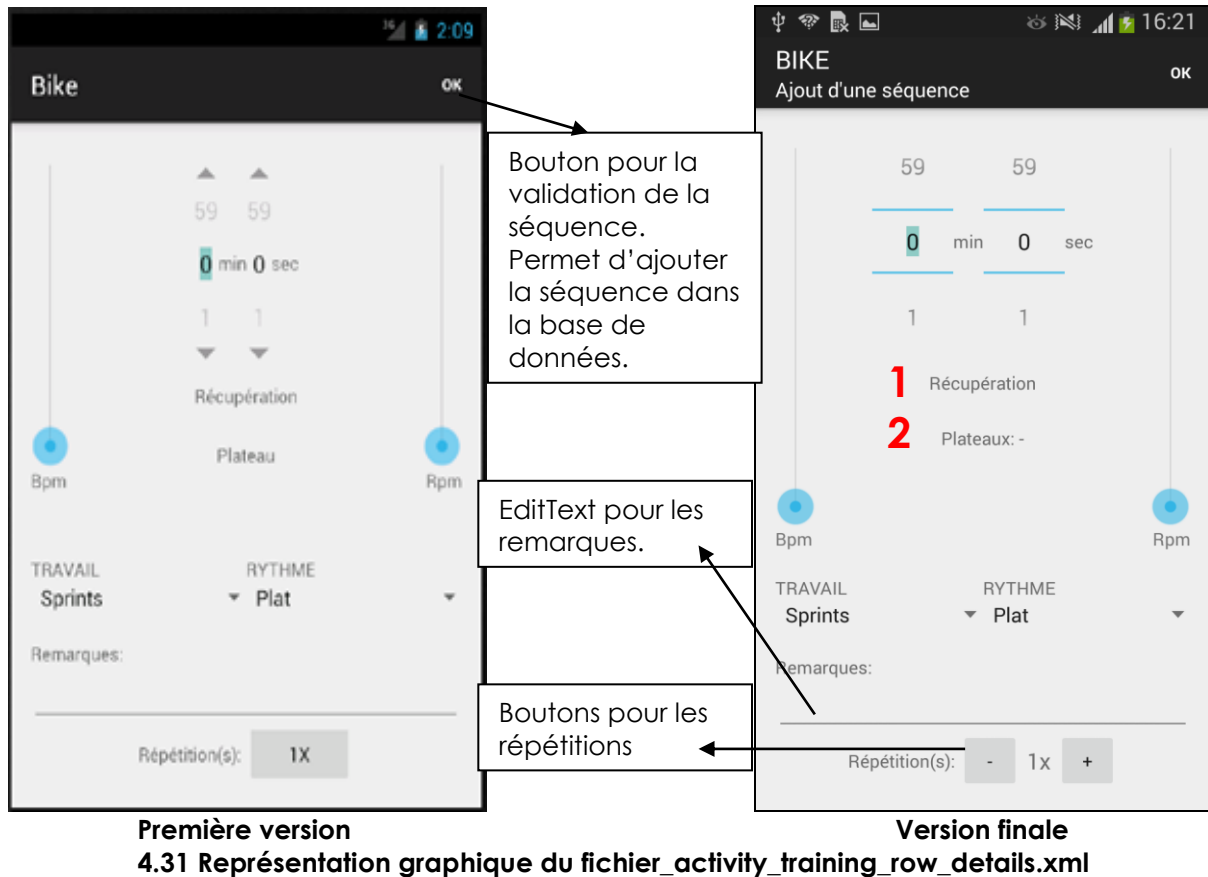
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="work">
        <item>Sprints</item>
        <item>Contre-la-montre</item>
        <item>Échauffement</item>
        <item>Départ</item>
        <item>Côtes</item>
    </string-array>
</resources>
```

4.29 Éléments du menu déroulant "Travail"



4.30 Affichage des items du Spinner avant modification et après

Le bas de l'écran laisse la place à l'utilisateur d'ajouter une remarque concernant la séquence. Cet EditText n'a pas de contrôle spécifique et affiche le texte sur une seule et unique ligne afin de ne pas modifier la disposition de l'interface graphique. Grâce à deux boutons un « + » et un « - » l'utilisateur est capable de choisir le nombre de fois qu'il souhaite répéter une séquence. L'image suivante montre l'interface dans sa version initiale et dans sa version finale.



Les quelques modifications entre les deux interfaces par rapport à la maquette graphique ont été requises par le mandataire. (cf. modification du 26.05.2015).

Le bouton pour valider la séquence se trouve dans l'ActionBar et n'apparaît pas dans la hiérarchie des éléments du layout. Comme pour les deux activités précédentes un fichier de menu est nécessaire, ainsi que l'implémentation des deux méthodes vues précédemment. (onCreateOptionsMenu et onOptionsItemSelected)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:compat="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/action_validate" android:title="OK" compat:showAsAction="always"/>
</menu>
```

4.32 Fichier XML du menu de l'activité TrainingRowDetailsActivity

Ce menu n'a pas d'icône comme les autres interfaces, mais simplement un texte « OK » plus explicite. Lors du clic sur ce bouton, une méthode de contrôle est appelée. Elle vérifie que trois éléments sont bien renseignés, les Bpm, les Rpm et la durée de la séquence. Dans le cas contraire un toast invite l'utilisateur à compléter les données manquantes. La méthode de validation retourne un booléen qui doit être égal à « vrai » pour continuer. Si le booléen est « vrai » alors l'ajout dans la base de données commence et une boucle for contenant le nombre de répétitions choisies par l'utilisateur est implémentée.

```
//this loop check how many times the user wants to repeat a row
for (int i = 0; i < nbr_repeat; i++){
    //add a row to the training we have selected
    realmDB.addAtrainingRowToTraining(int_id, realmDB.createRow(
        int_min_work, //min
        int_sec_work, //sec
        int_rpm, //rpm
        int_bpm, //bpm
        int_min_rest, //min rest
        int_sec_rest, //sec rest
        str_gear, //gear
        str_work, //work
        str_rythm, //rythm
        str_note)); //notes

    //here we check if bln_recup is true
    if(bln_recup){
        //if it's true we add a new training row for rest
        //add a row to the training we have selected
        realmDB.addAtrainingRowToTraining(int_id, realmDB.createRow(
            int_recupMin_value, //change min value by rest min value
            int_recupSec_value, //change sec value by rest sec value
            0, //no bpm
            0, //no rpm
            0, //no min rest
            0, //no sec rest
            str_gear, //gear
            "Récupération: -", //name of the row
            "", //rythm
            str_note)); //notes
    } //if(bln_recup)
} //for
```

4.33 boucle for pour l'ajout de séquence

Nbr_repeat représente le nombre de fois que le cycliste souhaite faire la même séquence. L'appel à la fonction de la classe RealmDB permet l'ajout de la séquence (Row) à un entraînement (Training).

```
realmDB.addAtrainingRowToTraining(int_id, TrainingRow trow)
```

l'id en paramètres est l'identifiant de l'entraînement et trow représente la séquence. Dans l'Image 4.33, la TrainingRow est créée directement grâce à l'appel d'une seconde méthode dans les paramètres de la méthode précédente. La méthode

```
realmDB.createRow
```

reçoit en paramètre tous les éléments nécessaires à la création d'une séquence. À savoir les minutes et les secondes de l'entraînement, les Rpm, les Bpm, les minutes et les secondes de récupération, les plateaux de vitesse, le type de travail, le rythme et les éventuelles remarques. La méthode createRow de la classe RealmDB s'occupe de créer l'objet TrainingRow (objet qui représente une séquence d'entraînement). La boucle for va alors vérifier si le temps de récupération a été modifié ou non. Dans le cas où le booléen bln_recup est « vrai » cela veut dire que l'utilisateur souhaite entre chaque séquence se

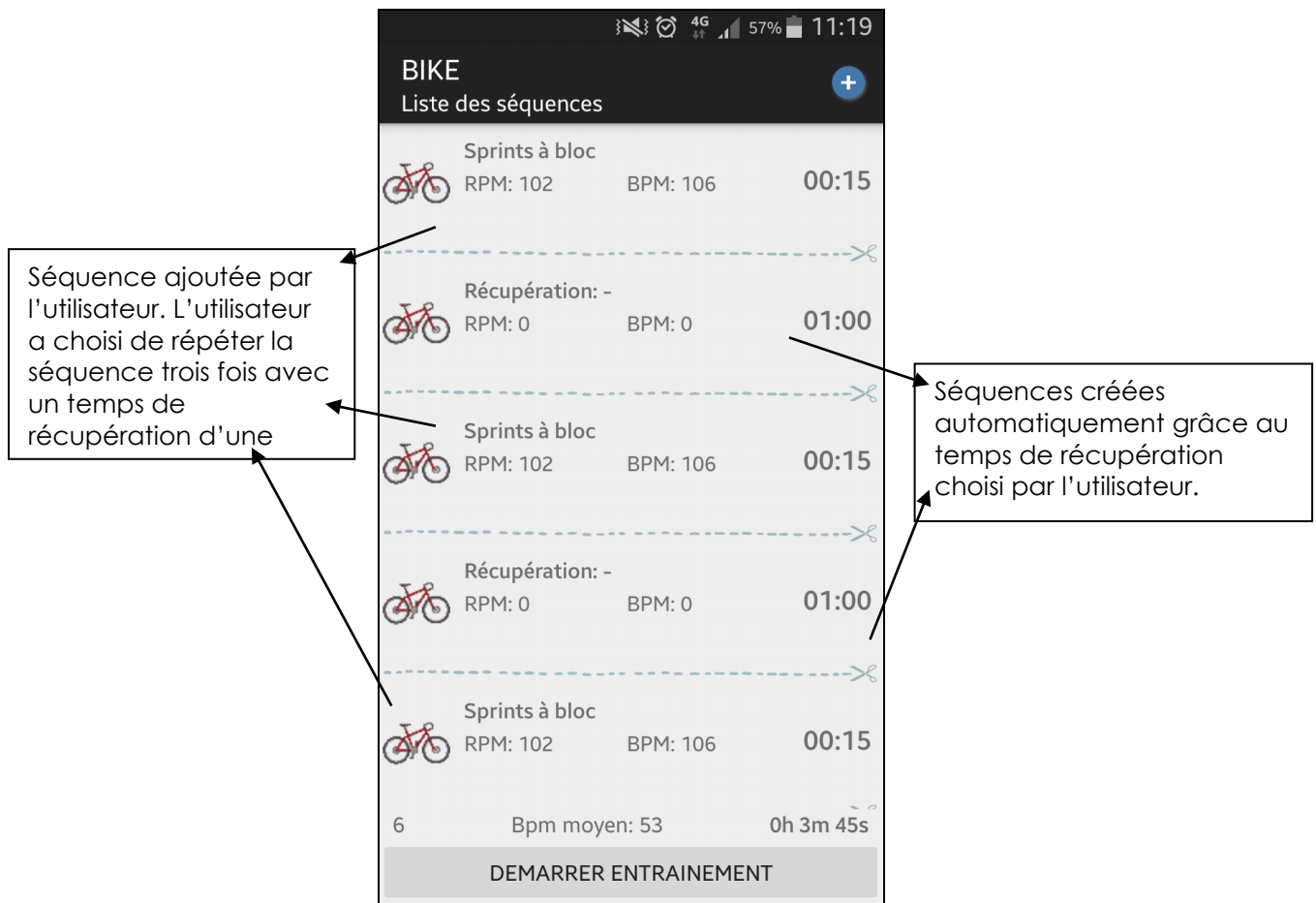
Auteur : Léchaire Thomas

Version : 1.0

Création : 11.05.2015

Rapport-TPI_Lechaireth.doc

reposer. À ce moment, une séquence de récupération avec les minutes et les secondes choisies va s'ajouter entre chaque séquence d'effort.



4.34 Exemple d'affichage après passage dans la boucle for

Cela permet à l'utilisateur d'avoir un affichage clair des temps de récupération dans le résumé de l'entraînement. Cette seconde liste est identique à la première ListView implémentée pour l'affichage des entraînements, elle utilise aussi un Adapter présent dans la classe TrainingRowAdapter.java, qui joue le rôle de lien entre les informations de la base de données et le layout de l'image 4.22. Dans cet Adapter le geste de swipe permet de faire apparaître le layout ci-dessous.



4.35 Après le swipe sur une séquence de la ListView

Comme pour la première ListView ici lors du clic sur le bouton effacer. C'est d'abord toute la séquence qui est récupérée avant de l'effacer de la liste puis de la base de données. C'est la méthode `getARowWithID()` qui permet de faire cela.

```
realmDB.getARowWithID(trainingRowList.get(position).getId());
```

Elle retourne une séquence grâce à son Id. Cette méthode n'a pas tout de suite fonctionné, car lors de la création d'une séquence l'id n'était pas correctement incrémenté et certaines séquences avaient alors le même identifiant. Pour résoudre cette erreur une requête permettant de trouver le plus grand id a été créée. Cet id a ensuite été incrémenté avant d'être enregistré dans une variable et utilisé pour l'identifiant de la nouvelle séquence.

```
int nextID = (int) (realm.where(TrainingRow.class).maximumInt("id") + 1);
```

Le problème ainsi résolu a permis d'effacer l'élément de manière correcte par rapport à sa position dans la ListView.

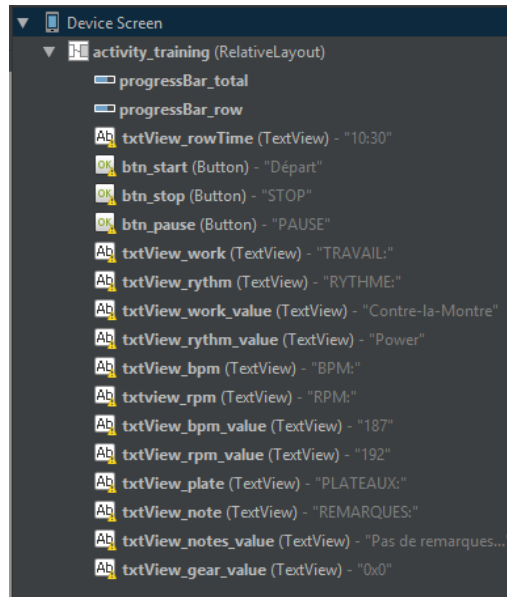
Le clic sur une ligne permet la modification d'une séquence. Pour éviter un code trop complexe dans une seule classe et une perte de temps, une seconde classe utilisant le même layout que sur l'image 4.31 a été créée. La nouvelle classe TrainingRowModification.java permet de gérer les modifications sur une séquence déjà enregistrée dans la base de données. En premier lieu, il est nécessaire de récupérer la séquence en question et pour cela une méthode de la classe RealmDB est utilisée.

```
//get the current row.  
tRow = realmdb.getArowWithID(int_rowId);
```

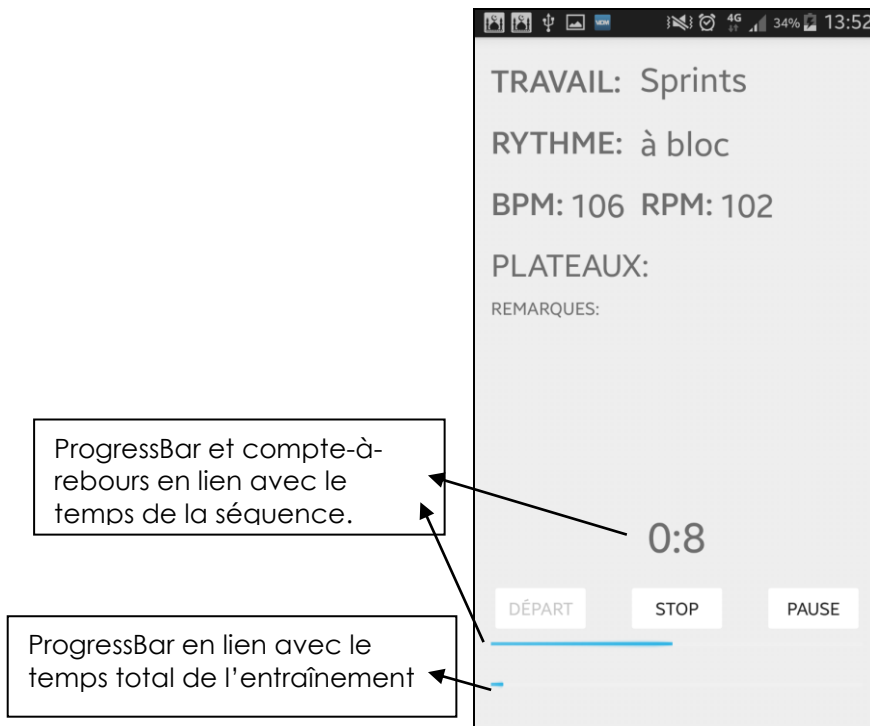
Cette méthode retourne la séquence sélectionnée et permet de récupérer les valeurs et les afficher à l'écran. Les valeurs de la séquence sont donc affichées et il est possible de les modifier. Pour gérer cela, il faut désactiver certains boutons, comme les boutons pour répéter la séquence ou celui pour ajouter du temps de récupération. Cela empêche l'ajout, dans l'entraînement, de nouvelles séquences entre deux et évite d'avoir des incohérences. En effet, on pourrait imaginer que l'utilisateur rajoute un temps de récupération entre deux autres séquences de récupération se retrouvant alors avec plusieurs séquences de récupération les unes derrière les autres.

Pour éviter ce genre d'incohérences et garder un certain ordre dans l'entraînement, il faut désactiver l'ajout de temps de récupération et les boutons permettant de répéter la séquence. Il est possible par contre de modifier la durée, les Bpm, les Rpm, ajouter des plateaux, des remarques ou encore changer le type de travail et/ou de rythme. Cette nouvelle classe est utile, car la méthode de validation des éléments est différente de la méthode présente dans la classe gérant la création d'une nouvelle séquence. Ici il faut reprendre l'id de la séquence et modifier les valeurs qui ont été changées.

Les séquences maintenant ajoutées ou modifiées, il est possible de démarrer l'entraînement en cliquant sur le bouton « DEMMARRER ENTRAINEMENT ». Cela ouvre l'activité suivante (Timer).



4.36 Hiérarchie du fichier activity_training.xml



4.37 Représentation graphique du fichier activity_training.xml

Dans cette interface, plusieurs TextView affichent les éléments contenus dans la séquence en cours. L'utilisation d'un « RelativeLayout » a permis de faire une mise en page sommaire, le mandataire souhaitant principalement que l'application fonctionne avant d'améliorer le design/graphisme. La partie importante de cette activité est la partie avec le compte à rebours et les 2 ProgressBar. Ces dernières, ont pour fonction d'afficher la progression du temps de la séquence (1^{ère} ProgressBar) et la progression du temps total de l'entraînement (2^{ème} ProgressBar). Les 3 Boutons, Départ, Stop et Pause permettent la gestion du compte à rebours.

Lors de la création de cette classe (Timer), beaucoup de problèmes ont été rencontrés. Les premiers essais d'implémentation ont été fait grâce à une AsyncTask tournant en tâche de fond. Cependant, des actions, comme mettre en pause, stopper ou reprendre le compte à rebours, étaient impossibles. Les boutons déclenchant les commandes de pause ou de reprise sont instanciés dans le thread (cf. [Lexique](#) – point 29) principal et l'interaction, entre un thread en arrière-plan et le thread principal, est déconseillée et difficile à mettre en place. De plus, la classe AsyncTask est généralement utilisée pour effectuer des tâches, continues et sans interruption, en arrière-plan.

Sans compliquer la tâche du Timer, des problèmes d'affichage des informations contenues dans les séquences sont apparus, cela car les objets Realm ne sont accessibles que dans le thread où ils ont été créés. Pour résoudre ce point, il a fallu sauvegarder toute la séquence d'entraînement, issue de la base de données, dans des variables globales (cf. [Lexique](#) – point 30). Neuf variables ont été créées et prennent les valeurs contenues dans la séquence en cours.

```
//create Training Row
int_min = lst_trainingRow.get(int_i).getInt_min();
int_sec = lst_trainingRow.get(int_i).getInt_sec();
int_rpm = lst_trainingRow.get(int_i).getInt_rpm();
int_bpm = lst_trainingRow.get(int_i).getInt_bpm();
str_gear = lst_trainingRow.get(int_i).getStr_gear();
str_time = lst_trainingRow.get(int_i).getStr_time();
str_rythm = lst_trainingRow.get(int_i).getStr_rythm();
str_work = lst_trainingRow.get(int_i).getStr_work();
str_note = lst_trainingRow.get(int_i).getStr_note();
```

4.38 Variables globales pour sauver les valeurs de la séquence

Elles sont mise à jour à nouveau en début de fonction par rapport à la valeur de « int_i » afin de s'actualiser lors du changement de séquence. La variable « int_i » est incrémentée lorsque le temps de la séquence précédente se termine.

Pour résoudre l'impossibilité de stopper et reprendre le compte à rebours, le Timer a été implémenté avec la classe CountDownTimer qui est une classe Android. Elle possède deux méthodes abstraites qui sont, `ontick(int millisInFuture)` et `onfinish()`. La première sert à faire un décompte à partir de la variable passée en paramètre (variable `millisInFuture`) et la seconde est appelée lorsque le décompte de `onTick` se termine. Lors de la première implémentation, le timer a fonctionné. Cependant, la mise en pause du compte à rebours s'est avérée compliquée. La première problématique à résoudre était la sauvegarde de la progression, du temps et de l'avancement des séquences dans l'entraînement, avant le déclenchement du bouton pause. Pour comprendre, il faut savoir qu'un objet de la classe CountDownTimer ne se met pas en pause. Il s'arrête et se recrée.

Lors de chaque pause il faut stopper le premier Timer, sauvegarder les données de la progression et démarrer un second. Un timer que l'on vient d'arrêter ne peut pas être recréé directement derrière. Théoriquement un Timer annulé devrait pouvoir être recréé, mais après plusieurs périodes de recherches, il semble que la classe CountDownTimer, implémentée par Android, ne gère pas correctement l'arrêt des Timers. Lors de l'appel de la méthode `cancel()` qui permet de stopper le Timer, le premier objet ne se stoppe pas.

C'est donc après lancement du nouveau Timer (reprise de l'entraînement) que les erreurs sont apparues. La TextView gérant le compte-à-rebours affichait tour à tour le temps du premier, puis du second Timer, bien que la méthode `cancel()` ait été appelée sur le premier.

Une des solutions proposée pour éviter ce problème est de modifier la classe `CountDownTimer` selon l'exemple trouvé ici :

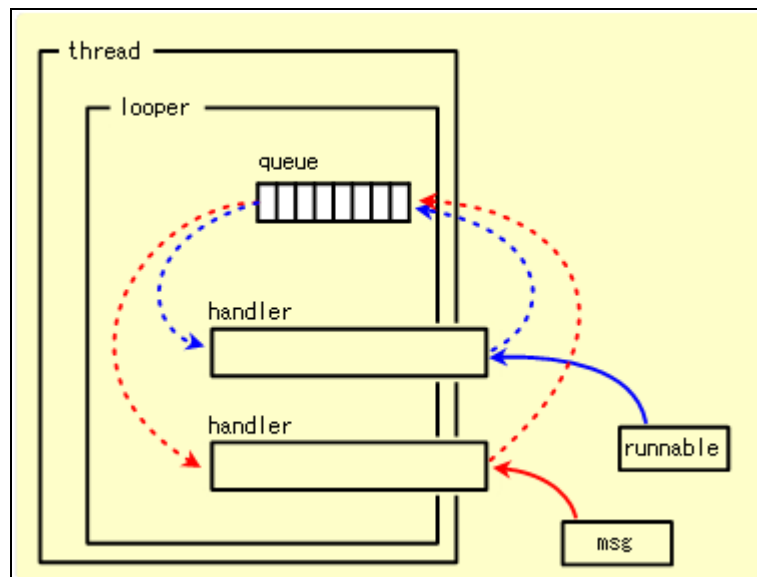
<https://code.google.com/p/android/issues/detail?id=58668>.

La modification effectuée n'a rien changé au problème d'arrêt du premier timer qui, malgré l'appel de la méthode pour le stopper, continuait à tourner. Il a fallu trouver une solution rapide à mettre en place pour ne pas perdre trop de temps à la réalisation de cette tâche qui en a demandé plus que prévu.

Après quelques recherches et la lecture de forums où les gens conseillaient d'éviter les classes `Chronomètre` et `CountDownTimer`, présentes dans Android, une solution a été trouvée. Le Timer est géré grâce à une classe privée `TimerRunnable` qui implémente la classe `Runnable`.

```
private class TimerRunnable implements Runnable
```

Ici, un « Handler » (cf. Voir [Lexique](#) – point 31) permet d'envoyer les informations contenues dans un « Runnable » (cf. Voir [Lexique](#) – point 32). Le Runnable est le conteneur d'information que l'on souhaite envoyer au thread. C'est le Handler qui l'envoie à la queue d'exécution.



4.39 Représentation graphique du fonctionnement d'un Handler

Dans la classe `TimerRunnable` gérant le timer, il y a deux méthodes abstraites. La première méthode `run()` est la méthode contenant le code à exécuter. L'astuce pour implémenter un Timer avec un « Runnable » est de rappeler le Handler à la fin de la méthode `run()` pour relancer le même Runnable cette fois en ajoutant un délai d'une seconde ou dans ce cas de 100ms.

```
//delayed post handler every 100 mililSecond  
handler.postDelayed(this, 100);
```

Ainsi on démarre une fois le Runnable lors de du clic sur le bouton « Départ », puis on utilise le Runnable pour rappeler le Handler qui va relancer le même Runnable. Une boucle infinie se crée, relançant le Runnable sans cesse. La seconde méthode `killRunnable()` est importante, car c'est elle qui permet de dire au Runnable de s'arrêter ou non.

```
public void killRunnable(){  
    //when timer is paused or stopped isCanceled is set to true.  
    isCanceled = true;  
}
```

Cette fonction est utilisée lors de la pause du Timer ou lors de son arrêt. Lors de la pause/arrêt de l'entraînement le booléen `isCanceled` passe à « vrai », puis dans la méthode `run()` on vérifie si le booléen est « vrai » ou que le temps total n'est pas égal à la progression du temps de l'entraînement (ce qui voudrait dire que l'entraînement est terminé). Dans le cas où l'on souhaite mettre en pause le Timer, le Runnable passe à « null » et, c'est le bouton pour la « Reprise » qui permettra d'appeler à nouveau le Handler pour poster le Runnable dans la liste des instructions à exécuter. Si l'entraînement est terminé, le Runnable passe aussi à « null », mais cette fois une boîte de dialogue apparaît.

Ainsi, un seul Runnable est utilisé et il est utilisable sans avoir besoin de le recréer. De plus grâce à l'utilisation de variables globales il n'est plus nécessaire d'enregistrer les variables lors de la pause, car tout est maintenu à jour automatiquement et la reprise du compte à rebours ou l'avancement des `progressBar` ne pose aucun problème.

Le timer fonctionne, mais une erreur de calcul est apparue lorsque le temps est supérieur à cinquante-neuf secondes. Le calcul du temps en millisecondes était mal instancié. $(\text{Min} * 60) + \text{Sec} * 1000$ au lieu de $((\text{Min} * 60) + \text{sec}) * 1000$. Cette correction a résolu cette erreur. Toute la classe est bien instanciée et disponible en annexe pour plus d'informations.

La dernière méthode implémentée, lorsque l'entraînement est terminé, permet le calcul de l'indice de récupération et sa sauvegarde. Comme dit précédemment, lorsque l'entraînement se termine une boîte de dialogue s'ouvre et invite l'utilisateur à patienter une minute et demie, puis à entrer sa fréquence cardiaque.



4.40 Boîte de dialogue en fin d'entraînement

La validation de cette boîte de dialogue lance une méthode de la classe RealmDB.

```
public void setRestIndice(int id, int fc_after_1min30)
```

Cette méthode prend la valeur inscrite par l'utilisateur et la soustrait à la fréquence cardiaque maximale entrée au début de l'application. Calculant un indice de récupération, lié à l'entraînement, qui sera enregistré dans la base de données. La variable en paramètre « id » permet de récupérer l'entraînement et « fc_after_1min30 » permet de calculer l'indice de récupération.

```
//calculate the rest indice  
int rest indice = fc max - fc after 1min30;
```

4.3 Implémentation de la base de données

Pour implémenter une base de données en accord avec la librairie Realm, la création de 3 classes d'objets et d'une classe de méthodes d'interaction avec la base a été nécessaire.

La Classe Training.java

Cette classe permet la sauvegarde des entraînements. (cf. Voir Annexe). Elle est implémentée selon le modèle de classe présent sur l'image 3.14. Une modification a été cependant apportée suite à un problème rencontré (cf. Modification du 28.05.2015). En effet, il faut sauvegarder une information supplémentaire concernant l'entraînement, afin de savoir s'il s'agit d'un entraînement de vélo de route ou de VTT.

La Classe TrainingRow.java

Classe représentant les séquences d'entraînements contenues dans un entraînement (cf. Voir Annexe). Ces séquences sont contenues dans une RealmList<TrainingRow> qui est un élément de la classe Training.java. C'est de cette manière que les séquences des entraînements sont reliées aux entraînements. Une séquence appartient à un et un seul entraînement.

La Classe HeartRate.java

Classe représentant la fréquence maximale et minimum de l'utilisateur.

La Classe RealmDB.java

Classe contenant toutes les méthodes d'interaction avec la base de données. Une partie des méthodes de cette classe a été présentée dans le chapitre 4.2 Implémentation de l'application.

Les classes Training, TrainingRow et HeartRate héritent de la classe RealmObject. La librairie Realm vient alors utiliser les méthodes (getters et setters) des classes en question, permettant ainsi de sauvegarder, de restaurer ou encore modifier les objets de ces classes. La base de données n'est pas visible, car c'est la librairie Realm qui s'occupe de la gestion des éléments et qui sauvegarde le tout dans un fichier *.realm. Realm propose cependant des méthodes pour faciliter les requêtes faites à la base de données.

La classe RealmDB contient de nombreuses méthodes d'écriture ou de lecture de la base de données. Durant l'implémentation de ces méthodes il faut faire attention car chaque méthode doit récupérer une instance de Realm avant de pouvoir interagir avec la base de données.

```
//get a realm Instance  
realm = Realm.getInstance(context);  
//start transaction  
realm.beginTransaction();  
//commit transaction  
realm.commitTransaction();
```

4.41 Exemple de transaction realm

Pour chaque interaction avec la base Realm, il faut entourer les instructions que l'on souhaite envoyer à la base de données entre deux lignes de codes. `Realm.beginTransaction()` et `Realm.CommitTransaction()` ou `Realm.CancelTransaction()` (si l'on ne souhaite pas sauvegarder les instructions).

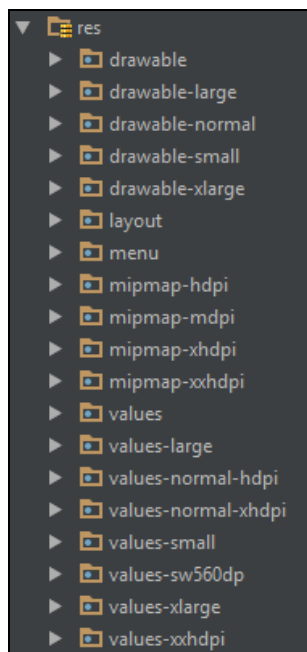
Durant les premières phases d'essais de sauvegarde des entraînements, un problème avec ces méthodes est survenu. L'entraînement était sauvegardé correctement et sans erreur. Cependant, lorsqu'on souhaitait récupérer les entraînements sauvegardés l'application avait tendance à s'arrêter. En effet, une transaction était parfois démarrée avec la méthode `beginTransaction()` mais jamais refermée ou annulée. Dès lors, quand qu'une nouvelle transaction tentait de démarrer, une erreur survenait interdisant de démarrer la seconde transaction avant que la première ne soit validée ou annulée. La vérification dans le code que toutes les transactions sont bien fermées avant le lancement de la nouvelle transaction a permis d'éviter cette erreur.

Concernant la suppression des éléments Realm met à disposition une méthode, `removeFromRealm()` qui permet d'effacer l'élément que l'on souhaite. Il faut bien prendre en compte que, comme un entraînement contient des séquences, s'il est supprimé, les séquences le seront aussi.

La modification des éléments s'effectue par une simple réécriture des valeurs de la base de données. Pour réécrire par-dessus une valeur de la base de données, il faut utiliser les setters qui ont été mis en place dans les classes objets. Par exemple : `setTrainingName(« Nouveau nom »)` remplacera le nom de l'entraînement par « Nouveau nom » dans la base de données.

L'utilisation de Realm est très intuitive, car il s'agit, pour résumer, d'objets que l'on peut sauvegarder. Pour plus de détails les classes d'objets et la classe `RealmDB` sont disponibles en annexe.

4.4 Le dossier Ressources - res



4.42 Dossier des ressources

Le dossier ressources est le dossier qui contient toutes les ressources du projet. Les différents types de ressources sont :

- Les images.
- Les layouts.
- Les menus.
- L'icône de l'application.
- Les chaînes de caractères.
- Les dimensions.
- Les fichiers de style.
- Les boutons personnels.
- Les couleurs.
- Etc...

Dans notre projet nous utilisons

Les Drawables : Les drawables peuvent être des images ou des fichiers xml avec des liens vers des images. L'image du vélo (rouge ou bleu) est un drawable. Mais le fichier XML de l'icône des fréquences cardiaques est aussi un drawable. Dans l'image 4.42, cinq dossiers de drawables sont présents. En effet, les téléphones qui utilisent Android comme système d'exploitation sont nombreux et il faut fournir des images adaptées aux tailles des écrans. Le dossier drawable-large sera utilisé pour les écrans de grande taille, tandis que le dossier drawable-small pour ceux de taille petite.

Les layouts : Les layouts sont des fichiers XML qui représentent les interfaces graphiques des activités. Comme pour les drawables il est possible d'avoir un dossier de layout pour chaque taille d'écran, ainsi que pour chaque orientation (paysage ou portrait). Il est donc possible de créer un dossier layout-large et de changer la disposition du layout pour s'adapter aux tablettes, par exemple. Un seul dossier est suffisant pour l'application, car premièrement l'application cible les téléphones uniquement et deuxièmement, l'utilisation de RelativeLayout permet aux éléments graphiques de mieux s'adapter selon la taille de l'écran.

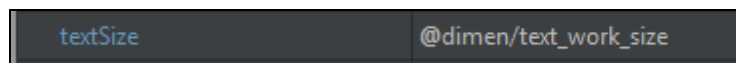
Les Dimensions : Les dimensions sont contenues dans des fichiers XML eux-mêmes contenus dans des dossiers « values ». Un dossier « values » comme un dossier layout ou drawable peut utiliser une indication de taille. Il faut alors définir la dimension dans tous les fichiers de chaque dossier pour que l'application puisse récupérer la valeur. Par exemple :

Fichier dimens.xml du dossier values-normal-hdpi

```
<dimen name="text_work_size">22sp</dimen>
```

Fichier dimens.xml du dossier values-large

```
<dimen name="text_work_size">25sp</dimen>
```



4.43 Utilisation de la dimension dans le layout

Les chaînes de caractères : Les chaînes de caractères sont toutes définies dans un fichier String. Pour les chaînes de caractères il est possible de définir la langue. Cela permet de traduire le fichier string.xml dans une autre langue. L'application s'occupe alors de chercher s'il existe un fichier langue dans la même langue que le téléphone, si c'est le cas il utilisera ce fichier, autrement il utilisera le fichier string.xml par défaut. Exemple de chaîne de caractères :

```
<string name="cancel">Annuler</string>
```

Les menus : Les menus sont des fichiers XML qui contiennent deux balises <menu></menu> et des <items> entre deux. Ces items sont les éléments du menu et seront affichés dans l'interface graphique. Un item peut être représenté par du texte ou par un drawable.

Les couleurs : Le fichier color.xml contient les lignes avec le nom de la couleur et la couleur en hexadécimal. Ce qui permet d'appeler toujours la même couleur dans le code, grâce à son nom. Exemple :

```
<color name="red_press">#990000</color>
```

4.5 Modifications

- 26.05.2015 - Modification de l'interface pour l'activité TrainingRowActivity.java. Une barre de résumé en bas de la liste avec le nombre de séquences dans l'entraînement, la moyenne des BPM et le temps total a été rajoutée. Cela permet à l'utilisateur d'avoir un résumé visuel de son entraînement.
- 26.05.2015 – Modification de l'interface pour l'activité TrainingRowDetailsActivity.java. Le bouton « Valider » a été remplacé suite à la demande du mandataire par un bouton permettant d'indiquer le nombre de fois que l'on désire répéter une séquence d'entraînement. Le bouton pour valider la séquence se trouve maintenant dans l'ActionBar en haut à droite.
- 28.05.2015 – Modification de la classe Training.java servant à la sauvegarde des entraînements par rapport au schéma de base proposé au point 3.14. Ajout d'un Booléen bln_isVtt étant égal à « vrai » si c'est un entraînement de Vtt et « faux » s'il s'agit d'un entraînement sur route. Cet ajout permet de connaître le type d'entraînement et de modifier l'image du vélo (bleu pour Vtt et rouge pour la route) en conséquence lors de l'affichage.
- 29.05.2015 – Modification de la classe Training.java et ajout d'une variable int_recup de type Integer permettant l'enregistrement de l'indice de récupération calculé en fin d'entraînement.
- 29.05.2015 – Modification du layout pour l'affichage des entraînements. Maintenant chaque entraînement affiche une ligne renseignant l'utilisateur sur l'indice de performance. L'indice de performance est propre à chaque entraînement et se modifie avec la dernière valeur calculée une fois un entraînement terminé. Si l'entraînement n'a pas encore été fait, le chiffre est remplacé par un trait « - »
- 29.05.2015 – Modification demandée par le chef de projet. Lorsque l'utilisateur doit/peut remplir une EditText, par exemple lors de l'insertion d'une remarque pour une séquence ou lors de l'insertion du nom d'un entraînement, faire en sorte que le clavier apparaisse directement sur l'écran. En effet, dans la version précédente c'était à l'utilisateur de cliquer dans la zone de texte pour faire apparaître le clavier.
- 29.05.2015 – Modification du layout pour l'ajout d'un entraînement (figure 4.17). Les deux radios boutons sont passés au-dessus de l'EditText et le focus se fait sur l'EditText permettant de faire apparaître le clavier directement.
- 01.06.2015 – Modification de l'interface pour l'ajout d'une séquence d'entraînement. Le bouton permettant de choisir le nombre de répétitions laisse sa place à un bouton « - » et à un bouton « + » avec une TextView au milieu affichant le nombre de répétitions. Le mandataire a demandé cette modification plus facile à comprendre pour l'utilisateur.
- 03.06.2015 – Modification des méthodes de contrôle pour l'ajout des fréquences cardiaques et le calcul de l'indice de récupération. Contrôle que le nombre soit compris entre une limite et non pas que sa longueur soit inférieure à trois comme précédemment. En contrôlant seulement la longueur on autorise l'utilisateur à entrer de valeurs comme 999 qui sont des valeurs aberrantes et qui peuvent causer des erreurs dans l'application.
- 04.06.2015 – Modification de l'affichage des items du Spinners. Sur l'image 4.30, les items sont très proches les uns des autres. Pour modifier cela, un layout contenant uniquement une TextView et des marges suffisantes a été utilisé pour augmenter l'espace entre les éléments des Spinners.

5 Tests

5.1 Dossier des tests

- Une grande partie des tests de l'application ont été effectués en cours de développement. Chaque fonctionnalité a été testée avant la création et l'implémentation de la suivante.

- Le test final de l'application a été effectuée par le mandataire qui a joué le rôle de bêta testeur. Il a mis en avant un certain nombre de problèmes et d'améliorations possibles. Les améliorations nécessaires au bon fonctionnement du projet, comme l'apparition et la disparition intuitive du clavier, la modification des boutons pour le nombre de répétitions, la disposition de certains éléments graphiques ou layout, ont été mises à jour directement. Les remarques concernant l'aspect esthétique plus que le bon fonctionnement de l'application ont dû être mises de côté et sont des points qu'il est possible de retrouver dans le chapitre améliorations possibles.
- La partie des tests et Debugging prévue dans la planification initiale a été utilisée pour corriger et améliorer les points selon les remarques du bêta testeur. Une grande partie du temps imparti aux tests et aux corrections des erreurs a été utilisé en cours de programmation. Cela est principalement dû au fait qu'il est inutile de coder toute une application puis de vérifier s'il elle fonctionne. Il est plus intéressant de développer une fonctionnalité et de la tester directement pour voir si elle est correctement implémentée ou non.
- Le manque de temps n'a pas permis de tester l'application en conditions d'utilisation réelles. Cela aurait pu être intéressant et faire ressortir quelques points à améliorer.

6 Conclusion

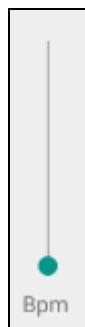
6.1 Bilan des fonctionnalités demandées

Liste des fonctionnalités demandées.

- Créer un entraînement de façon simple
Tâche de difficulté moyenne. La complexité et la mixité des données ont créé une tâche supplémentaire à savoir la sélection des informations récurrentes contenues dans les exemples d'entraînement afin de limiter les choix de l'utilisateur sans pour autant le priver de s'entraîner correctement.
- Afficher un résumé de la séance d'entraînement.
L'entraînement s'affiche sous forme de liste et permet de visualiser toutes les séances contenues dans ce dernier. Une partie de l'écran (en bas) offre à l'utilisateur un résumé et affiche le nombre de séquences, la moyenne de la fréquence cardiaque et le temps total de l'entraînement.
- Démarrer l'entraînement et afficher chaque partie de façon claire au moyen de contrôles graphiques, de barres de défilement, de boutons etc.
L'entraînement peut démarrer en cliquant sur le bouton « DEMARRER ENTRAINEMENT ». Les séquences d'entraînement s'affichent les unes après les autres. C'est le chronomètre qui gère le défilement des séquences et qui n'affiche la séquence suivante qu'une fois la précédente terminée (ainsi de suite).
- Mettre en pause l'entraînement et le redémarrer au moment souhaité.
Il est possible de stopper, mettre en pause et reprendre l'entraînement lorsque celui-ci a démarré. Cependant, si l'on quitte l'entraînement, celui-ci est arrêté et il faudra alors le recommencer du début. Il n'y a aucun sens à reprendre un entraînement par une séquence sans avoir effectué les autres auparavant.
- Sauvegarder une séquence d'entraînement créée.
Par défaut toutes les séquences d'entraînement créées sont sauvegardées. Cela prend du temps de remplir et programmer un entraînement, il n'y a donc aucun intérêt à ce

qu'elle soit temporaire et qu'elle disparaisse une fois le téléphone ou l'application éteinte.

- Charger un entraînement précédemment sauvegardé.
Comme tous les entraînements et les séquences sont sauvegardés, il est possible de refaire un entraînement précédemment créé ou de refaire un entraînement déjà effectué.
- Insérer sa fréquence cardiaque de repos et maximale.
Si le cycliste souhaite utiliser l'application, il doit avant toute chose insérer sa fréquence cardiaque maximale et de repos. Ces deux valeurs sont utilisées pour définir les valeurs minimums et maximales, de la barre des Bpm, lors de l'ajout d'une séquence et permettent, en fin d'entraînement, de calculer un indice de récupération pour l'entraînement que l'on vient de terminer.



6.1 les fréquences min et max définissent les limites de la barre BPM

- Calculer un indice de récupération
Le calcul de l'indice de récupération se fait en fin d'entraînement. L'utilisateur doit patienter une minute et demie et entrer sa fréquence cardiaque. Le calcul s'effectue et l'indice est enregistré dans l'entraînement, cependant seul le dernier indice de performance est enregistré. Le point concernant l'enregistrement de l'historique des indices de récupération n'était pas clair et n'a pas été mis en avant lors des discussions avec le chef de projet. La sauvegarde se fait par entraînement alors que le mandataire aurait souhaité un historique par date afin de pouvoir évaluer son niveau de fatigue. Le point est en partie rempli, car l'indice de récupération s'affiche en dessous de chaque entraînement de la liste. Cela permet d'avoir, de manière moins explicite, l'état de forme de cycliste/utilisateur, s'il effectue les entraînements dans l'ordre.

6.2 Bilan de la planification

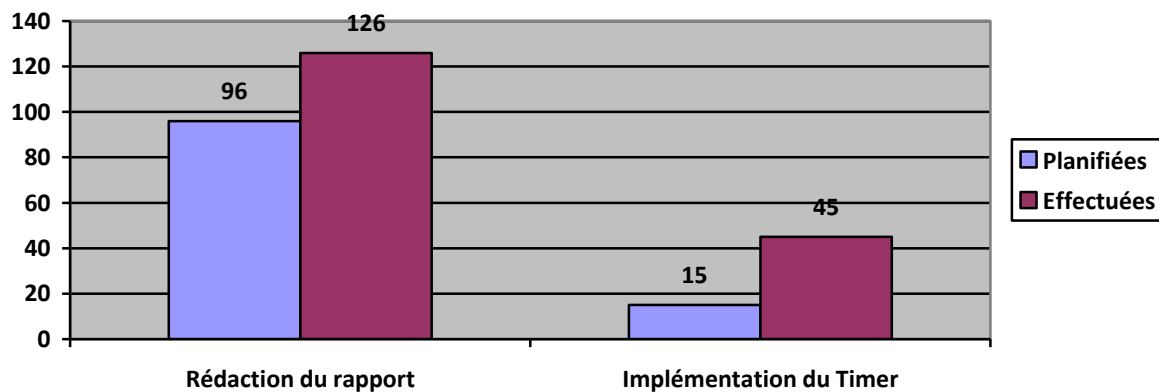
Le projet comptait 102 heures de travail, mais avec le temps pris pour compléter et relire la documentation, le projet s'approche des 110 heures effectives. En passant en revue les différentes tâches de mon projet, j'ai pu en ressortir quelques points qui me semblent important à prendre en compte.

Ma connaissance de l'environnement Android et le fait que ce n'était pas mon premier essai de développement d'une application, m'ont permis de gagner du temps sur les tâches de documentation et d'analyse des besoins matériels. Ce temps gagné c'est avéré utile pour comprendre clairement les besoins de l'utilisateur et mettre en avant les potentielles difficultés auxquelles le projet allait me confronter.

Ma mésaventure avec la machine virtuelle est Visio, m'a fait perdre du temps en début de projet, mais les zone tampons et la facilité d'implémentation de la base de données Realm ont permis de rééquilibrer la balance.

La tâche ayant causé le plus de retard par rapport au temps planifié est la rédaction et la mise en page des documents pour le TPI. Beaucoup de temps consacré à la mise en page, l'insertion d'image, la rédaction et la correction. Le retard est principalement dû à la mise en page. Beaucoup d'images qui doivent s'insérer correctement dans le texte, de gestion des sauts de pages ou encore des styles du texte ou des listes à puces. Une bonne gestion du temps est une planification plus large du temps pour le rapport sont des éléments utiles pour éviter d'accumuler du retard en fin de projet.

Tâches en quart d'heure



La tâche concernant les Timers a demandé beaucoup de temps elle aussi. Elle a créé un trou dans le planning et a pris une part du temps réservé à d'autres tâches. Pour cela des tâches comme le comportement de l'application (affichée vs caché) n'ont été que testées de manière sommaire. Certaines tâches planifiées pour les tests et le débbuging ont aussi été réduites pour rattraper le retard et terminer le projet.

6.3 Bilan personnel

Je suis content d'avoir réussi à répondre entièrement au cahier des charges et d'avoir pu rendre une application utilisable et terminée. Les progrès que j'ai réussi à faire depuis mon début avec le langage Java et l'environnement Android sont une plus-value non négligeable. Le projet n'était pas mon idée, mais j'ai apprécié découvrir le monde du vélo et concevoir cette application.

J'apprécie particulièrement le développement et pouvoir faire quelque chose de concret qui sera utilisé (au moins par le chef projet) est quelque chose de très gratifiant.

Avec ce projet j'ai pris conscience de l'importance de la rédaction d'un rapport de qualité surtout lorsque le projet peut être repris par une autre personne ou que le projet est une application pour un client. L'analyse est un point qui demande du temps à mettre en place, mais qui permet de prévoir et d'anticiper un grand nombre de problème. Bien évidemment tous les problèmes ne sont pas évitables et savoir trouver les solutions sur les forums ou les tutoriaux est quelque chose que j'apprécie aussi. Passer d'une application qui ne fonctionne pas et réussir à débbuger correctement jusqu'à quelque chose d'utilisable est une sensation très grandissante et constructive tant au niveau personnel que professionnel.

Si cela était à refaire, j'utiliserais de nouveau Realm pour la base de données, car c'est une solution simple à mettre en place et facile à utiliser. Concernant la planification, je prendrais plus de temps pour implémenter les Timers et je laisserai des zones tampons plus grandes, afin de mieux prévoir les éventuelles anicroches. Prévoir au minimum un jour à un jour et demi par semaine pour la planification et maintenir le dossier à jour au fur et à mesure, cela permet d'éviter de prendre trop de temps à la fin du projet pour la rédaction.

6.4 Améliorations possibles

L'application est réalisée du mieux possible dans le temps imparti. Mais certaines fonctionnalités ou améliorations sont encore possibles.

L'amélioration du design est un point important pour l'utilisateur qui apprécie que le contenu soit bien présenté et lisible. Le but du projet n'était pas de faire une application design, mais fonctionnelle. Cependant, avec un peu de temps supplémentaire, il pourrait être intéressant de repenser l'interface graphique afin qu'elle soit plus colorée et lisible.

Un thème pourrait être appliqué sur tous les éléments sélectionnables. On peut imaginer utiliser une couleur ou un bouton personnalisé qui se retrouverait sur tous les contrôles, SeekBar, TextView, boutons ou EditText qu'il est possible d'utiliser ou de modifier, dans l'application. Il est vrai que certaines zones qui permettent d'interagir avec l'application ne sont parfois pas claires et on se demande si l'on peut cliquer dessus ou non.

Au niveau de l'application, une activité avec tous les indices de récupération et la date d'enregistrement de ces indices pourrait répondre de manière mieux adaptée aux exigences du mandataire. Il suffirait d'enregistrer les indices dans un tableau de la base de données et de les afficher lorsque l'utilisateur ouvre l'activité créée à cet effet.

Il pourrait être intéressant, pour répondre au point supplémentaire du cahier des charges, de pouvoir générer une séquence aléatoirement en fonction d'un thème choisi (vélocité, force, rythme etc.) avec le choix de la difficulté et de la durée. Cela demanderait cependant beaucoup de temps, car il faudrait soit enregistrer des séquences de base et créer l'entraînement à partir de celles-ci ou créer un algorithme qui permettrait de créer aléatoirement ces séquences.

L'utilisation de la mémoire pourrait être améliorée. En effet, les téléphones Android sont des appareils limités en mémoire et une application qui en consomme trop risque de se faire tuer, pour libérer de l'espace, si cette dernière consomme trop de ressources. Améliorer l'utilisation de la mémoire pourrait être un point intéressant, même si une méthode, permettant de libérer un peu de mémoire, est déjà présente dans le code, elle n'est pas suffisante pour avoir une application moins consommatrice.

La gestion de matériel externe comme une montre Gps ou une montre cardio-fréquencemètre, pourrait permettre au cycliste d'avoir des informations, comme sa fréquence cardiaque ou sa vitesse (dans le cas d'une montre Gps), directement dans l'application. On pourrait alors imaginer d'enregistrer les routes parcourues en vélo et d'afficher le tracé sur une carte ou de calculer les Rpm à partir de la vitesse, affichant ensuite les vues en rouge quand l'utilisateur a un rythme cardiaque supérieur au rythme demandé.

6.5 Remerciements.

Remerciements à mon chef de projet pour son aide et le projet qu'il m'a proposé ainsi qu'aux personnes qui ont pris le temps de relire et corriger mon dossier.

7 Annexes

7.1 Bibliographie

- CookBook Développement Android 4 – 60 recettes de pros de Damien Gosset, Fabrice Impérial, Marc Pybourdin et Nicolas Zinovieff. Édition DUNOD 2013.
- Android 4 - Les fondamentaux du développement d'applications Java de Nazim Benbourahla. Édition Eni 2012.

7.2 Webographie

- <http://mikithemaus.deviantart.com/art/Android-Cycling-360351098> - logo du titre
- <https://realm.io/news/realm-for-android/> - Librairie Realm
- <https://realm.io/docs/java/latest/> - Documentation Realm
- <http://www.lahc.edu/classes/nursing/currenttrnstudents.html> - icône fréquence cardiaque
- <http://www.wikipedia.fr>
- <http://developer.android.com>
- <http://www.veryicon.com/icons/transport/transport/bike.html> - icône du vélo
- <https://github.com/daimajia/AndroidSwipeLayout> - librairie Swipe
- <https://github.com/AndroSelva/Vertical-SeekBar-Android>
- <http://www.stackoverflow.com> – forum d'entraide
- <http://gpsoft.dip.jp/graphic/android/looper.png> - Image du loopier
- <https://github.com/daimajia/AndroidViewAnimations> - Librairie des animations
- <http://openclassrooms.com/courses/programmez-en-orientee-objet-en-php/uml-presentation-1-2> - Cours sur les UML
- <https://bitbucket.org> – Serveur de sauvegarde git
- <https://www.sourcetreeapp.com/> - logiciel SourceTree

7.3 ANNEXE 1 - LEXIQUE

7.4 ANNEXE 2 - TABLEAU DES VERSIONS ANDROID

7.5 ANNEXE 3 – JOURNAL DE TRAVAIL.

7.6 ANNEXE 4 - PLANIFICATION INITIALE

7.7 ANNEXE 5 - PLANIFICATION DÉTAILLÉE

7.8 ANNEXE 6 – CODE SOURCE