UNIVERSITÉ
CÔTE D'AZUR

RAPPORT DE CONCEPTION LOGICIELLE SMARTPHONE

Mia Swery - Mokhtar Samy - Eva Radu

SOMMAIRE

1. Introduction

Au cours de notre projet, nous avons réalisé une application destinée à être déployée sur une smartwatch. Nous avons décidé d'implémenter une application permettant de récolter les données des battements cardiaques de l'utilisateur. Une fois que les données sont collectées, elles sont stockées dans une base de données et l'utilisateur a alors la possibilité d'accéder à différents types d'informations, à la fois sur l'application présente sur la montre et sur un dashboard.

Nous allons à présent vous présenter en détails le travail que nous avons effectué et la manière dont nous l'avons organisé, les difficultés rencontrées et les éventuelles améliorations que nous trouvons potentiellement utiles d'implémenter.

2. Organisation du travail

Nous avons utilisé les outils suivant tout au long de notre travail :

- Kotlin pour l'application sur smartwatch
- Angular pour la partie du dashboard
- Firebase pour le stockage des données

Voici notre répartition du travail :

- **Mokthar** :
  - Mise en place de la récupération des données cardiaques
  - Dashboard
- **Eva** :
  - Création, gestion et mise à jour de la base de données
  - Dashboard
- **Mia** :
  - Design et enchaînement des activités sur l'application
  - Communication avec la base de données

L'ensemble de notre travail est disponible sur le répertoire GitHub suivant :

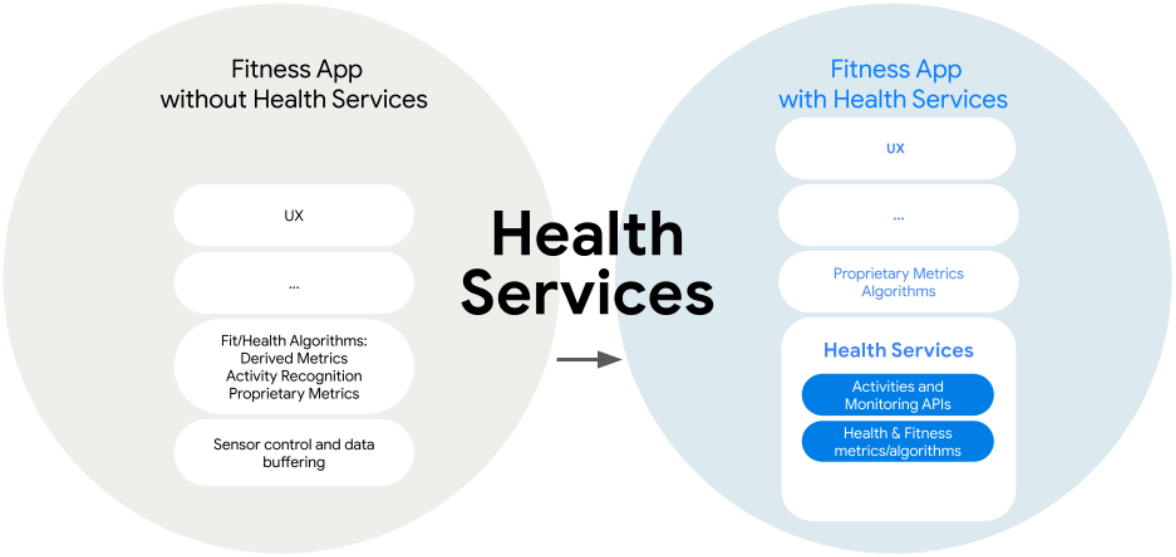
https://github.com/MokhtarSamy/hearts

3. Travail effectué

3.1. Récupération des données

Health Services API

Wear OS 3 offre un service appelé Health Services qui agit comme un intermédiaire entre les capteurs et les applications sur la montre. Nous avons utilisé son API pour récupérer les données du capteur cardiaque de la montre.



Google vient de sortir une version beta de cette API. Malheureusement, elle est très récente, il n'y a donc que très peu de documentation et de tutoriels. Nous avons alors rencontré énormément de difficultés pour mettre en place cette API.
De plus, cette API n'est pas toujours fonctionnelle et compatible avec l'émulateur présent dans Android Studio. Il a alors été compliqué d'avancer étant donné que nous ne disposions que d'une seule smartwatch dans le groupe.

Health Services API fournit 3 types de clients, measure client, exercise client et passive monitoring client. Nous avions tout d'abord utilisé un client "*Measure client*", mais celui-ci ne nous permettait pas de stocker les données même quand l'application était en veille. Nous avons alors changé notre code pour utiliser un client "*Passive monitoring client*".

En effet, voici le détail des fonctionnalités de chaque client :

3.1.1 Measure client

En implémentant ce client, notre UI doit être toujours actif et non pas dans le background pour que les données soient collectées. Ce client sert à récupérer les données très rapidement (plusieurs fois par seconde) d'une manière régulière. Cependant, il arrête d'envoyer les données dès que l'utilisateur arrête l'application ou que la montre se met en veille.

3.1.2 Exercise client

Ce client offre plus de fonctionnalités pour des applications destinées aux activités sportives. En effet, ce client offre une fonctionnalitée "*Exercise goals*" qui sert à surveiller le progrès d'un utilisateur et vérifier qu'il a bien atteint son but et aussi à être à jour avec l'état de l'exercice.

3.1.3 Passive monitoring client

Ce type de client permet que les applications soient actives en background et que les données soient récupérées en continu. Cependant, les données sont collectées de façons moins fréquente que le Measure Client. Cela l'avantage de réduire l'échange de données qui consomme beaucoup d'énergie et aussi réduire le volume des données stockées dans la base de données. Contrairement à measure client, il ne demande pas qu'un exercice soit en cours, il ne fait que surveiller les données venant des capteurs.

Pour éviter la perte des données, par exemple au cas où la connexion internet est coupée, le passif client intègre un data repository où les données récupérées sont stockées et on peut les récupérer même après avoir fermé l'application ou bien après que le système reboot.

Dans les 3 types de clients, il y a plusieurs types de données. Nous avons utilisé le type *HEART_RATE_BPM* qui est un objet qui contient les battements cardiaques.

3.2. Base de données

Afin de stocker toutes les données, nous avons mis en place une base de données placée dans Firebase.
Firebase est un service qui nous a permis d'héberger une base de données NoSQL en temps réel. Les informations stockées sont alors organisées en collection contenant des documents.

Dans notre base nous avons créé deux collections :

- la collection "**hearts**" : cette collection permet de stocker les enregistrements de chaque battements cardiaques collecté par la montre
- la collection "**stats**" : cette collection stocke toutes les données statistiques des informations de la collection "hearts".

Les documents de la collection “hearts” disposent des attributs suivant :

- **id** : un identifiant unique généré automatiquement par Firebase
- **rate** : le battement cardiaque qui a été collecté par la montre
- **date** : la date à laquelle le battement cardiaque a été collecté
- **time** : l’heure à laquelle le battement cardiaque a été collecté
- **moment** : le moment de la journée à laquelle le battement cardiaque a été collecté (AM ou PM)

Les documents de la collection “stats” disposent des attributs suivant :

- **id** : un identifiant unique généré automatiquement par Firebase
- **date** : la date associée au statistique du document
- **avg** : la moyenne des battements cardiaques ayant été récolté à la date spécifiée dans l'attribut “date”
- **min :** le plus petit battement cardiaque ayant été récolté à la date spécifiée dans l'attribut “date”
- **max** : le plus grand battement cardiaque ayant été récolté à la date spécifiée dans l'attribut “date”

Voici un exemple de document de la collection “hearts” :

id : 2B3DRRuCmsyBiItqtH8i
date : “12-12-2022”
moment : “PM”
time : 72
rate : “22:33:10”

Voici un exemple de document de la collection “stats” :

id : M2eDMa4EsjWIIZIJJNN8
avg : 78.76315789473684
date : "12-12-2022"
max : 91
min : 51

Les deux collections sont mises à jour dès qu’un battement cardiaque est détecté et collecté par la montre. En effet, d’une part on rajoute le nouveau battement dans la collection *“hearts”*, et d’autre part on recalcule les éléments statistiques pour la date actuelle et on met à jour le document correspondant dans la collection “*stats*”. Pour recalculer les éléments statistiques, on récupère les documents de la collection “hearts” de la date actuelle et on calcule la moyenne, le minimum et le maximum. Ainsi, dans la collection “*stats*”, une date n’a qu’un seul document associé.

Notre base de données a pour but de pouvoir rendre accessible les données depuis l’application sur la smartwatch et depuis le dashboard.

3.3. Fonctionnalités de l’application sur la montre

Par la suite, il a également fallu s’occuper de l’interface graphique de notre application sur la montre.
Il faut rappeler que notre objectif est de permettre à l’utilisateur de :

- Connaître son rythme cardiaque actuel
- Connaître son nombre de battement par minute moyen à l'échelle de la journée
- Connaître son nombre de battement par minute maximum atteint au cours de la journée
- Connaître son nombre de battement par minute minimum atteint au cours de la journée
- Obtenir un commentaire personnalisé sur les différentes valeurs cardiaques

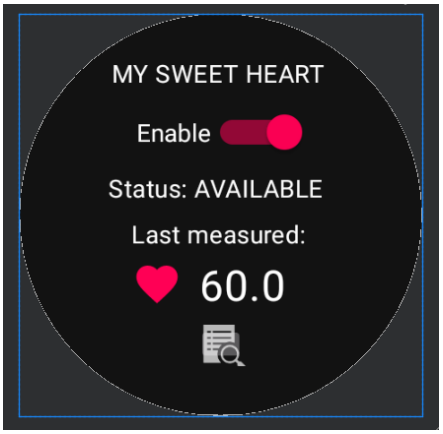
Au départ, nous pensions créer une seule activité. Mais au vu du nombre d’information à afficher à l’utilisateur et par souci de clarté, c’est tout naturellement que nous avons décidé de créer 4 activités :

3.3.1 Une activité Main

Il s’agit en quelque sorte de l’écran d’accueil de notre application.
On y trouve le nom de l’application.
Un bouton de type switch permettant à l’utilisateur de décider lui-même si l’application est en droit de collecter ses données cardiaques même lorsque la montre est en mode veille.
Un textView contenant le rythme cardiaque en temps réel de l’utilisateur récupéré depuis la base de données. La valeur affichée se met à jour si jamais le rythme cardiaque change.
Tout en bas de l’écran se trouve un bouton permettant à l’utilisateur de passer à la vue suivante contenant des détails sur le rythme cardiaque de l’utilisateur.

Voici une capture d’écran de cette activité :

Design / Sur la montre:

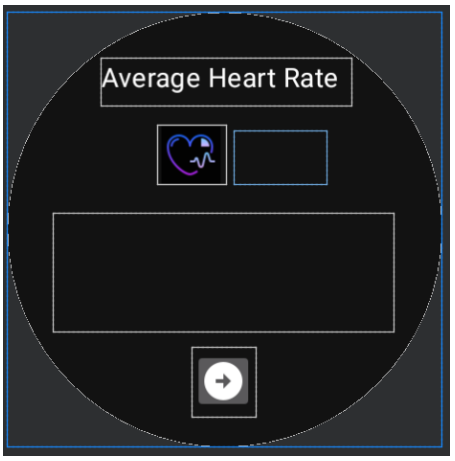


3.3.2 Une activité Average

On trouve dans cette activité le titre : *Average Heart Rate*. L'utilisateur comprend ainsi ce qu'on cherche à lui communiquer.
On trouve ensuite un textView contenant la valeur moyenne du battement cardiaque par minute de l'utilisateur au cours de la journée. Cette valeur se met à jour lorsque de nouveaux battements cardiaques sont enregistrés dans la base de données.
En dessous de ce textView se trouve un nouveau textView destiné à contenir le commentaire associé au nombre de battements par minute moyen de l'utilisateur. Ce commentaire se met à jour lors d'un changement de valeur pour la moyenne.
En bas de l'écran se trouve un bouton permettant à l'utilisateur de passer à l'activité suivante.
L'utilisateur a toujours la possibilité de retourner à l'activité principale en swipant vers la gauche.

Voici des captures d'écran de l'activité détaillé ci-dessus :

Design :



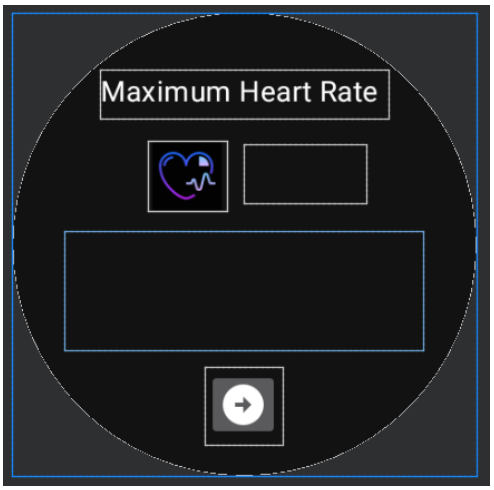
Sur la montre :

3.3.3 Une activité Maximum

Cette activité est similaire à celle décrite ci-dessus.
Tout en haut de la montre apparaît le titre contenant le nom de la statistique affichée, à savoir le nombre de battement cardiaque le plus élevé enregistré dans la journée.
On affiche la valeur maximale obtenue par une requête vers notre base de données dans un TextView .
Un commentaire s'affiche à l'écran pour analyser la bpm obtenue dans un autre TextView.
En bas de l'écran, il y a un bouton permettant de passer à l'activité suivante. Nous avons également la possibilité de swiper vers les deux activités précédentes.

Voici des captures d'écran de l'activité détaillé ci-dessus :

Design :



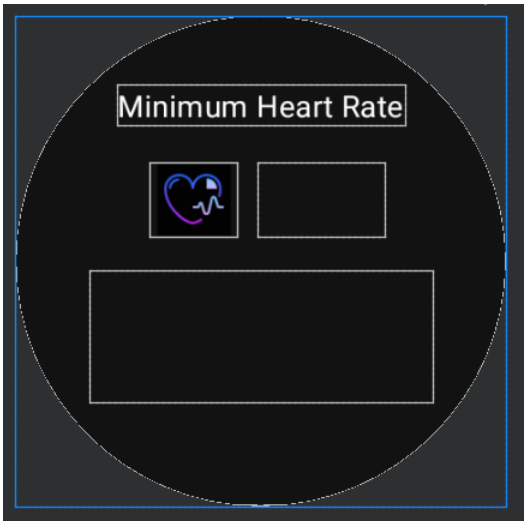
Sur la montre :

3.3.4 Une activité Minimum

C'est la dernière activité que nous avons conçue et implémentée. Là encore, la structure est très similaire aux activités précédentes.
Tout en haut de la montre apparaît le titre contenant le nom de la statistique affichée, à savoir le nombre de battement cardiaque le moins élevé enregistré dans la journée.
On affiche la valeur minimale obtenue par une requête vers notre base de données dans une TextView .
Un commentaire s'affiche à l'écran pour analyser la bpm obtenue.
En revanche, il n'y a pas besoin d'un bouton ici étant donné que c'est la dernière activité de notre application. On peut swiper vers toutes les activités précédentes.

Voici des captures d'écran de l'activité détaillé ci-dessus :

Design :



Sur la montre :

3.3.5 Organisation du code

En termes de code, on remarque bien la similarité entre nos activités : "*Average*", "*Maximum*" et "*Minimum*". C'est pourquoi nous avons décidé de construire une classe abstraite "*MyActivity.kt*" dont chaque classe associée à nos activités hériteront.

Dans notre classe abstraite nous trouvons :

- 3 champs abstraits :
    - **stat : String** → en fonction de l'activité qu'on instancie, on attribuera à stat la valeur "*max*", "*min*" ou "*avg*". On utilise cette variable dans la requête qu'on envoie vers la base de données
    - **id : Int** → c'est l'id de la TextView (déjà présente dans le layout) dans laquelle on écrira la valeur des battements de coeur récupérée dans la base de données
    - **idComment : Int** → c'est l'id de la TextView (déjà existante dans le layout) dans laquelle on écrira le commentaire sur la valeur obtenue dans la base de données

- 3 méthodes :
    - **getTime()** : cette méthode permet de récupérer la date d'aujourd'hui. Elle est très utile lors de notre requête vers la base de données.
    - **setTextView()** : cette méthode récupère tout d'abord les textView, destinées à afficher les battements de cœur et les commentaires, à partir de leur id. Puis elle change le texte des différentes vues : soit avec la valeur récupérée dans la base de donnée, soit avec le commentaire généré avec l'appel à la méthode *"comment"*.
    - **comment(heartBeats)** : en fonction de la valeur heartBeats passée en paramètre, on génère une String. Nous avons fait le choix d'afficher "ATTENTION !\n Votre rythme cardiaque est plutôt faible!" si on a une bpm < 50, "ATTENTION !\n Votre rythme cardiaque est plutôt élevé!" si on a une bpm > 100 ou "Vous avez un rythme cardiaque normal." sinon.
    - **getData()** : c'est à cette méthode que nous faisons appel dans les classes filles dans la fonction "*onCreate*". La fonction getData envoie une requête à la base de données : elle demande le champ *stat* avec la date donnée par *getTime()*. Quand la requête s'est bien passée, on récupère un nombre de battement par minute sous la forme d'un double. On fait ensuite appel à la méthode *setTextView* avec la valeur récupérée en paramètre. Cette méthode fait ensuite elle-même appel à la méthode *comment* pour générer un commentaire.

Pour pouvoir passer d'une activité à l'autre (et plus précisément de *AvgActivity* à *MaxActivity* puis de *MaxActivity* à *MinActivity*), on utilise les boutons présents dans nos layouts et dans le code on ajoute un listener sur le click.

En revanche, notre activité *Main* a été programmée quelque peu différemment étant donné qu'elle ne fait pas exactement la même chose.

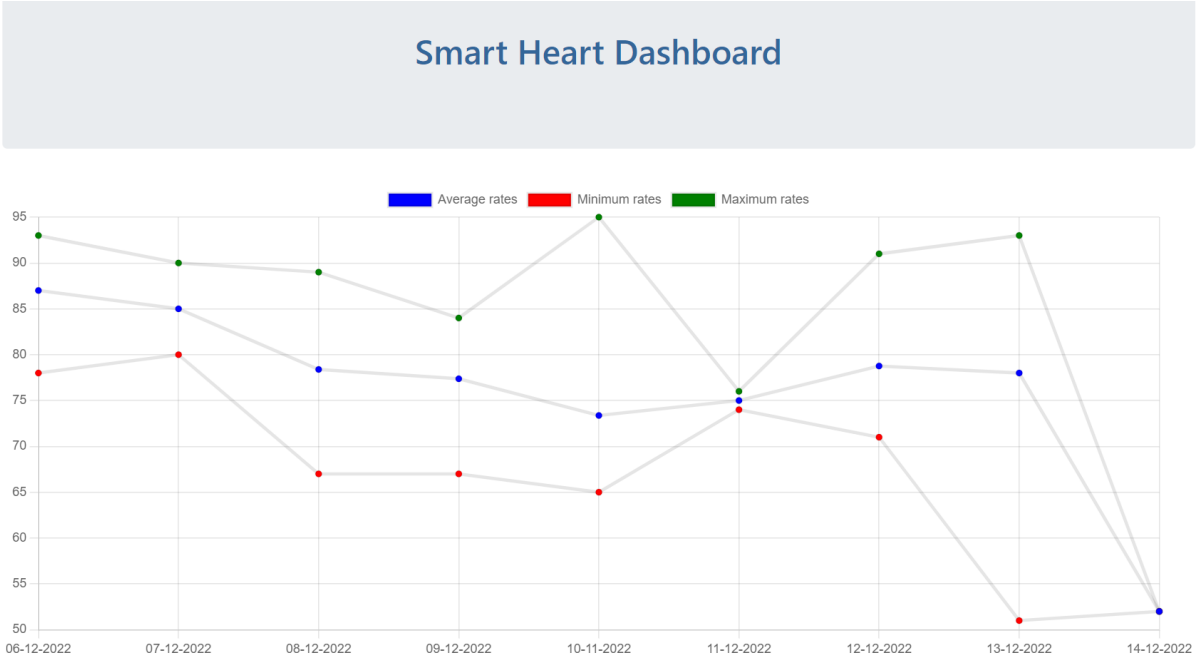
3.4. Dashboard

Enfin, nous avons implémenté un dashboard très simple permettant d'afficher un graphique des statistiques stockées dans la base de données (c'est-à-dire les informations de la collection "stats").

Voici les données constituant le graphique :

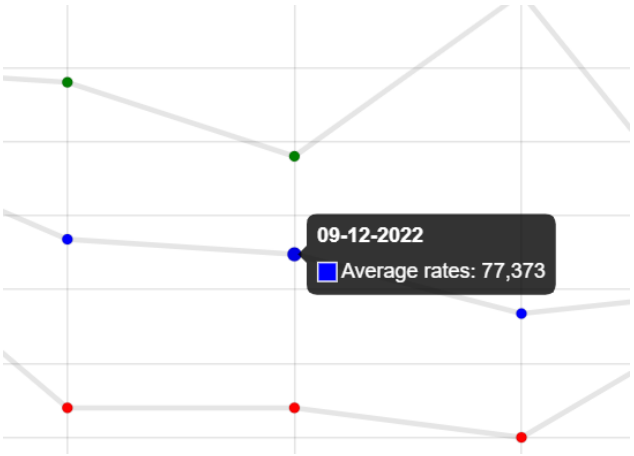
- Sur l'axe des abscisses il y a les dates
- Sur l'axe des ordonnées il y a les valeurs des battements cardiaques
- La courbe bleue représente la moyenne des battements cardiaques pour chaque jours
- La courbe verte représente la plus haute valeur des battements cardiaques pour chaque jours
- La courbe rouge représente la plus basse valeur des battements cardiaques pour chaque jours

Voici une capture d'écran de notre graphique présent dans le dashboard :



De plus, l'utilisateur a la possibilité de filtrer le graphique en n'affichant que la ou les courbes qui l'intéresse grâce aux trois boutons disponibles au-dessus du graphique.

Enfin, en passant la souris sur un des points, il est possible d'obtenir les informations associées de la sorte :



4. Difficultés rencontrées

Au début, nous avons implémenté le "*Measure client*", puisque nous pensions créer une application qui mesure le battement cardiaque pendant une courte durée. Cependant, avec ce client la montre se mettait en veille automatiquement après quelques secondes et l'envoie des données s'arrêtait. L'utilisateur devait alors toucher l'écran régulièrement pour ne pas arrêter l'enregistrement des données.
Nous avons essayé de désactiver la mise en veille automatique par plusieurs moyens, mais cela n'était pas possible.

Nous avons donc décidé d'utiliser le "*Passive client*". Cependant, son fonctionnement est très différent de celui du "*Measure client*". Nous avons donc été obligés de refactoriser notre code.

5. Ce qu'il faudrait améliorer ou ajouter

Nous avons pensé à plusieurs éléments qui seraient judicieux d'ajouter ou d'améliorer dans notre application et dans notre dashboard.
Premièrement notre dashboard pourrait contenir davantage d'informations : on pourrait afficher également les heures de battements les plus hauts et des battements les plus bas.
De plus, nous aurions voulu mettre en place un système d'authentification à la fois sur la montre et sur le dashboard. En effet, les utilisateurs pourraient soit créer un compte soit s'authentifier s' ils possèdent déjà un compte. Cela aura permis d'une part le partage d'une montre pour plusieurs utilisateurs et d'autre part la possibilité d'accéder à ses données personnelles depuis deux montres différentes.
Par ailleurs, de manière inexplicable, les activités Maximum et Minimum affichent les battements de cœur à la mauvaise position alors qu'elles disposent des mêmes contraintes que l'activité Average (où l'affichage est tout à fait correct). Après des heures de debugging et de tests, nous n'avons malheureusement pas réussi à résoudre ce problème. Il s'agirait donc là d'une piste pour une amélioration de notre application.
Enfin, nous pensons qu'il aurait été intéressant d'afficher davantage d'informations médicales pour que les utilisateurs aient une meilleure analyse de leur fréquence cardiaque.

6. Conclusion

En conclusion, même si nous avons rencontré énormément de difficultés pour récolter les données cardiaques, nous avons finalement réussi à implémenter une application fonctionnelle, nos données sont collectées de manière passive et sont mises à jour automatiquement dans la base de données et l'utilisateur dispose d'un dashboard.
En outre, ce projet a été l'occasion pour nous de développer des compétences dans le langage Kotlin qu'aucun de nous ne connaissait et d'aborder quelques problématiques liées au développement mobile et à l'utilisation de smartwatch.

Merci pour votre lecture.