Table des matières

Table des illustrations

Glossaire

Activity	Il s'agit d'un composant propre à Android.
API	Application Interface programming. En programmation, c'est une interface offrant diverses méthodes.
Class	En programmation, une class est un « plan » permettant de créer un objet.
E-health	Cybersanté.
Framework	Infrastructure logicielle (« Le grand dictionnaire terminologique », s. d.).
HTML	HyperText Markup Language. Il s'agit d'un langage permettant de créer des vues et des interfaces graphiques.
Javascript	Langage de script.
JSON	Javascript Object Notation. Il s'agit d'un format de données texte (« JSON », s. d.).
Layout	Il s'agit d'un composant graphique propre à Android. Un Layout représente un écran affiché à l'utilisateur.
Listener	Un listener est un écouteur en Français. Il va écouter un évènement particulier (par exemple, la réception de données ou le clique sur un bouton).
Manifeste	Fichier Android utilisé pour spécifier les caractéristiques d'une application.
Service	Il s'agit d'un composant Android. Un service est un processus exécuté en arrière-plan et n'est pas dépendant de l'interface graphique.

Sprint	En terminologie Scrum, un sprint caractérise une période de temps.
Textview	Composant graphique permettant d'afficher du texte.
Thread	Il s'agit d'un sous-processus lancé en parallèle du processus principal. (« ISO/IEC 2382:2015(en), Information technology — Vocabulary », s. d.).
User Story	Une User Story est une fonctionnalité.
Web service	Service proposé par un serveur distant utilisant le protocole http ou https.
Webview	Composant graphique permettant d'afficher une page internet au sein d'une application Android.

1 Introduction

1.1 Motivation personnelle

Arrivé au terme de notre formation en informatique de gestion, nous devons mettre en pratique nos connaissances acquises par le biais d'un travail de Bachelor. Ayant choisi mon thème parmi une liste, je me suis naturellement dirigé vers le thème « monitorer sa santé via une montre connectée ». La technologie étant relativement nouvelle (Android Wear 2.0), il s'agit d'un challenge des plus stimulants qui, de plus, apporte de nouvelles connaissances et compétences. Un des aspects qui m'intéresse particulièrement est le domaine de la santé et du sport. Il s'agit d'un domaine particulièrement riche et complexe.

1.2 Contexte

Avec l'arrivée de l'internet des objets, de nouveaux périphériques intelligents et connectés sont apparus. Le Smartphone ayant été le précurseur sur le marché, a apporté son lot d'innovations en termes d'utilisation et de développement. Ces nouveaux périphériques étant plus puissants et disposant de leur propre système d'exploitation, permettent désormais aux développeurs de créer des applications via des environnements de développement spécialement conçus.

Le développement d'applications devient plus accessible. Les développeurs peuvent alors proposer leurs applications via des plateformes mises à disposition, comme par exemple Apple Store ou Google Play Store.

Ces dernières années, de nouveaux accessoires portables intelligents ont émergé sur le marché. Les montres connectées, également nommées Smartwatch, accompagnent désormais les téléphones portables. Ces nouveaux périphériques disposent de capteurs permettant la collecte de données. Les données capturées ouvrent la voie à de nouvelles approches pour des applicatifs dans le domaine de la santé (mesure de soi).

1.3 Objectif

En collaboration avec l'institut AISLab (Applied Intelligent Systems Lab) l'objectif de ce travail est d'étudier le potentiel des nouveaux dispositifs de montres connectées Android Wear, dans le domaine de la santé par le biais d'une application mobile.

1.4 Résumé

La première étape sera l'analyse des fonctionnalités du système d'exploitation Android Wear, notamment son architecture, ses spécificités et les nouveautés apportées par la version 2.0.

En vue de développer une application, nous analyserons les caractéristiques techniques des montres disponibles sur le marché afin de pouvoir sélectionner un modèle approprié. Nous définirons les critères d'analyse selon nos besoins. Ensuite, nous choisirons un modèle en fonction des critères sélectionnés.

Une fois la montre acquise, l'étape suivante de notre travail consistera à nous familiariser avec le Framework MAGPIE. Il s'agit d'un Framework spécialement conçu pour monitorer et traiter des événements dans le cadre d'application *E-health*.

L'étape finale sera le développement d'une application *E-health* Android Wear à l'aide de MAGPIE. Cette application sera développée de manière itérative en suivant les principes Scrum. Elle aura pour but d'informer l'utilisateur sur son état de santé et sera axée sur cinq catégories : les pulsations, la pression sanguine, le nombre de pas journaliers, le poids, le taux de sucre dans le sang.

Mots-clés : Android Wear, E-health, Smartwatch, Application mobile.

2 Smartwatch

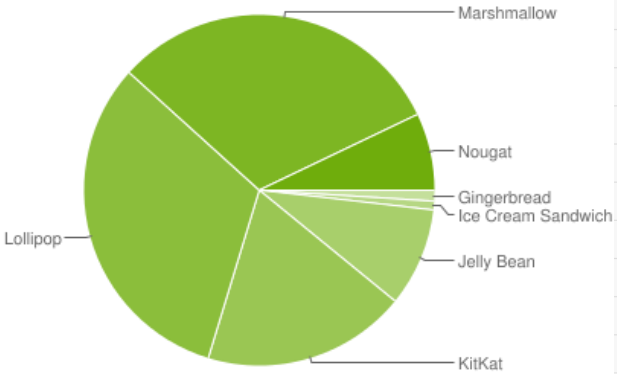
2.1 Définition

Une Smartwatch, ou montre intelligente en français, est une montre avec des fonctionnalités supérieures aux montres standard. De nos jours, ces montres sont équipées des mêmes composants que les ordinateurs, leur permettant d'exécuter des applications, de se connecter aux réseaux, de lire des fichiers audios, etc… (« Smartwatch », 2017) .

Il existe plusieurs systèmes d'exploitation pouvant équiper les montres intelligentes comme Android Wear, Watch OS (le système d'exploitation d'Apple), et Tizen.

2.2 Système d'exploitation Android

Android est un système d'exploitation développé par Google. Basé sur le noyau Linux, Il est conçu principalement pour les Smartphones et les tablettes. La première version (1.0) a été annoncée le 23 septembre 2008. Le premier téléphone équipé avec le système d'exploitation Android était le T-Mobile G1, créer par HTC (Morrill, s. d.). A partir de cette date, de nombreuses versions ont été publiées. La dernière version disponible au public est la version Nougat (version 7).



Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.8%
4.1.x	Jelly Bean	16	3.2%
4.2.x		17	4.6%
4.3		18	1.3%
4.4	KitKat	19	18.8%
5.0	Lollipop	21	8.7%
5.1		22	23.3%
6.0	Marshmallow	23	31.2%
7.0	Nougat	24	6.6%
7.1		25	0.5%

Figure 1 : Distribution des versions
https://developer.android.com/about/dashboards/index.html

Android est un système d'exploitation Open-Source. Cela signifie que le code source est accessible au public. Ainsi, chacun a la possibilité de modifier et adapter le code à ses besoins. Il pourra créer son propre système d'exploitation (« Android Open Source Project », s. d.) .

2.3 Système d'exploitation Android Wear

Android Wear est un système d'exploitation utilisé pour équiper les montres intelligentes. Concrètement, il s'agit du système d'exploitation Android, modifié et adapté aux montres. Annoncé le 18 mai 2014, Android Wear équipe la première montre LG Watch le 5 septembre 2014. La dernière version (2.0) a été publiée en février 2017. Cette version propose plusieurs nouveautés qui seront détaillées dans les chapitres suivants (« Android Wear », 2017).

2.3.1 Compatibilité avec les téléphones

Les montres Android Wear peuvent être associées à un téléphone portable en utilisant une connexion Bluetooth. Une fois associés, ils permettent d'offrir une expérience optimale, notamment via les notifications partagées, la prise en charge des appels (pour autant que la montre possède un microphone), etc.

Les montres Android Wear supportent les téléphones possédant le système d'exploitation Apple iOS (Iphone) et Android. Les IPhones doivent posséder au minimum la version iOS 9. En ce qui concerne l'association avec les Smartphones Android, ils doivent posséder au minimum la version Android 4.3(« Android Wear », s. d.-a).

2.3.2 Nouveautés de la version 2.0

Les paragraphes suivants décrivent les nouveautés apportées par la version 2.0 d'Android Wear (« Android Wear 2.0 : Ultimate guide to the major smartwatch update », s. d.).

I Application autonome

Il n'est plus nécessaire d'associer la montre au téléphone en permanence. Désormais, la montre peut se connecter directement à internet via une connexion wifi ou en utilisant une mini carte Sim. Les applications peuvent être totalement indépendantes du téléphone.

On dénombre trois types d'applications pouvant être exécutées sur la montre :

- Complètement indépendante du téléphone : l'application ne requiert pas la présence d'un téléphone.
- Semi-indépendante du téléphone : l'utilisation du téléphone est facultative. Toutefois, le téléphone peut fournir des fonctionnalités additionnelles.
- Dépendante du téléphone : l'application nécessite la présence d'un téléphone pour fonctionner.

Les développeurs doivent spécifier le type d'application en introduisant une métadonnée à l'intérieur du *manifeste* (« Standalone Apps | Android Developers », s. d.).

La documentation ne spécifie pas si la montre est en mesure de se connecter à un *webservice* de manière autonome. Nous allons donc tester la connectivité de la montre en utilisant un programme écrit en Java. Le programme va initier une connexion au serveur et récupérer des données en format JSON. Il va ensuite afficher les données dans une *textView*. Le code suivant est utilisé pour accéder au serveur.

```
private void getContentFromWebService() {
//create the connection to the URL via the library "Volley"
RequestQueue queue = Volley.newRequestQueue(this);
   //url of the web service
String url ="http://echo.jsontest.com/Key1/Value1/Key2/value2";
// Request a string response from the provided URL.
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
      new Response.Listener<String>() {
         @Override
         public void onResponse(String response) {
            // Display the response of the web server

            try {
               //parse the JSON object into an object
               JSONObject jsonObject =new JSONObject(response);
               //put the response into the view
               text_value_1.setText(jsonObject.getString("Key1"));
               text_value_2.setText(jsonObject.getString("Key2"));
```

```
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }, new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {
        text_value_1.setText("That didn't work!");
    }
});
// Add the request to the RequestQueue.
queue.add(stringRequest);
```

Figure 2 : Code permettant l'accès à un webservice. Source personnelle

L'URL utilisée dans notre programme nous retourne l'objet *JSON* suivant :

```
{
  "Key1": "Value1",
  "key2": "value2"
}
```

Le résultat du test est illustré par l'image suivante.

Figure 3 : Résultat du webservice.
Source personnelle

Nous pouvons constater que la montre parvient à récupérer les données provenant du *webservice*, de les formater et de les afficher. La connexion peut être effectuée soit par wifi, soit en utilisant le Bluetooth connecté avec le téléphone. Cela signifie que la montre est autonome en matière de connexion et supporte la connexion aux *webservices*.

II Play Store

Play store est la plateforme de Google qui permet de télécharger des applications sur Android. Les montres Android Wear 2.0 possèdent l'application Play Store. Cela signifie qu'il n'est pas nécessaire de devoir installer l'application sur le téléphone au préalable.

III Interface graphique

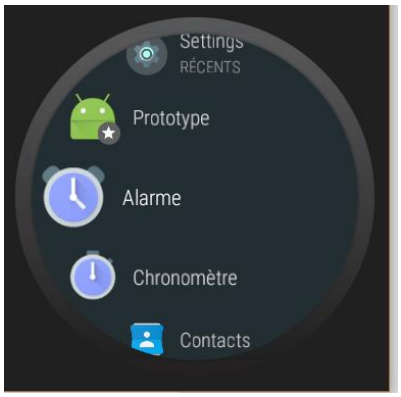


Figure 4 : Interface graphique
Source personnelle

L'interface graphique a été optimisée pour les écrans arrondis. Les applications sont affichées sous forme de liste, et non plus en balayant l'écran de gauche à droite.

IV Watch face

Figure 5 : Cadran principal
Modifié par l'auteur. Source : http://www.androidauthority.com/best-android-wear-watch-faces-581582/

Il est désormais possible d'afficher plusieurs éléments directement dans le cadran principal (l'horloge). Ces informations peuvent provenir d'application tierces, comme Google Fit et Spotify. Ainsi, l'utilisateur disposera d'un « tableau de bord » au travers duquel il pourra consulter ses informations facilement.

La façon de changer de cadran a également été simplifiée. Pour se faire, il suffit de balayer l'écran et de sélectionner le cadran désiré.

V Android Pay

Android Wear 2.0 possède la fonctionnalité Android Pay, qui permet d'effectuer ses paiements. Une fois les cartes de crédit ou débit ajoutées, il sera possible d'utiliser son périphérique pour payer dans les magasins, via des applications ou directement en ligne (« Android – Android Pay », s. d.).

Seuls les périphériques équipés d'une puce NFC seront en mesure d'utiliser Android Pay pour effectuer des paiements

VI Traitement de texte

Android wear 2.0 propose plusieurs façons d'écrire du texte :

- Réponses préenregistrées : Lorsque l'on doit répondre à un message texte, la montre propose des réponses préenregistrées.

Figure 6 : Réponses préenregistrées
Source personnelle

- Ecriture manuscrite : Il est possible de dessiner les lettres qui seront ensuite converties en texte par la montre.

Figure 7 : Écriture manuscrite
Source personnelle

- Clavier tactile : comme pour les smartphones, la Smartwatch Android propose un clavier tactile. La taille de l'écran étant réduite, il s'avère difficile de presser la touche désirée. Toutefois, cette difficulté est compensée par un système de prédiction de mots qui permet de gagner en rapidité et en précision.

Figure 8 : Clavier tactile
source personnelle

- Reconnaissance vocale : Android wear est équipé d'une reconnaissance vocale. Elle va être en mesure de transcrire les mots prononcés et de les retourner sous forme de texte.

VII Notifications

Le système de notifications a été repensé et amélioré. Désormais, il est plus facile de naviguer entre les notifications.

2.3.3 Architecture & API

L'architecture de l'Android Wear est relativement similaire à celle d'Android. La structure générale (*Activity*, Service, *Layout*, Ressources) est identique. (« Creating Wearable Apps | Android Developers », s. d.) . On distingue toutefois certaines spécificités.

I Mode ambient

Lorsque la montre est en mode ambient, elle n'émet pas de lumière et elle consomme moins de batterie. Quand l'utilisateur tourne son poignet ou touche l'écran de la montre, le mode ambient passe en mode interactif et la lumière s'active sur le périphérique. (« Creating Wearable Apps | Android Developers », s. d.)

II Data layer API

L'*API* Data Layer permet la communication entre le téléphone et la montre. Pour cela, il faudra associer la montre avec le téléphone en utilisant l'application Android Wear sur le téléphone. Il sera possible d'associer un téléphone avec plusieurs montres en même temps. La figure ci-dessous illustre le fonctionnement de transmission de données entre deux périphériques, en utilisant l'*API* Data Layer.

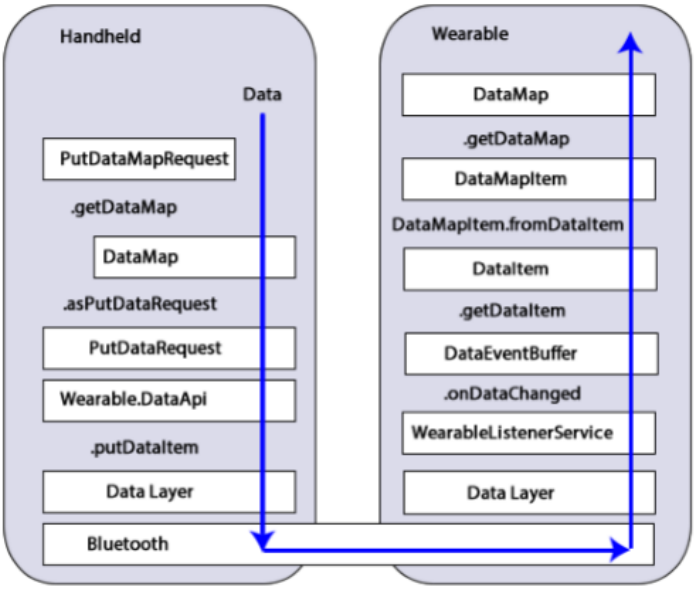


Figure 9 Schéma représentant la transmission de données au sein de la couche Data Layer
http://android-wear-docs.readthedocs.io/en/latest/data.html

Cette *API* va se charger de la communication entre les différents périphériques en utilisant des objets spécifiques que le système peut transmettre aux périphériques appareillés (connecté).

La synchronisation est assurée par un serveur cloud hébergé par Google. Il n'est ainsi pas nécessaire d'être directement connecté aux périphériques par Bluetooth. (« Network Access and Syncing | Android Developers », s. d.).

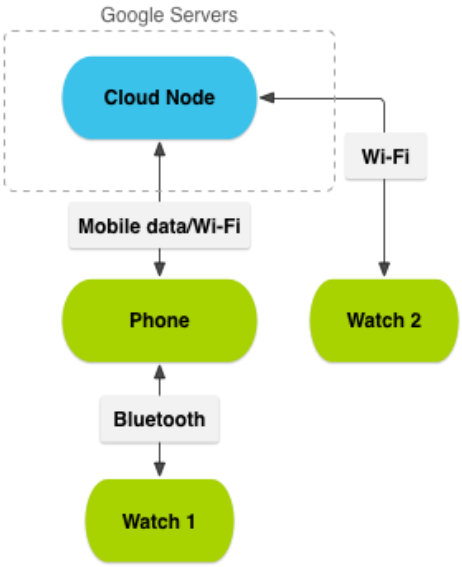


Figure 10 : Architecture Data Layer
https://developer.android.com/training/wearables/data-layer/index.html

III Limitations des *API*

Android wear possède certaines limitations par rapport au système d'exploitation Android standard. Les *API* suivantes ne sont pas supportées par les montres Android Wear :

- android.webkit : Cette *API* est utilisée pour naviguer sur internet. Par exemple, elle permet de récupérer et d'afficher une page internet dans une *Webview*.
- android.print : Cette *API* met à disposition plusieurs *class* qui permettent de gérer les impressions directement au sein des applications.
- android.app.backup : L'*API* backup contient des outils permettant de restaurer les données des applications. Si un utilisateur supprime des données, les applications intégrant ces outils seront en mesure de restaurer les données supprimées.

- android.appwidget : Contient les composants nécessaires pour créer des Appwidget. Les appwidget permettent aux utilisateurs d'intégrer une activité à une autre application, comme par exemple, d'afficher un élément sur l'écran d'accueil.

- android.hardware.usb : permet de communiquer au périphérique connecté via le câble USB[1].

(« Creating Wearable Apps | Android Developers », s. d.)

2.3.4 Framework & langages de programmation

Il existe plusieurs *Frameworks* et langages permettant de développer des applications Android.

I Framework basé sur les technologies web

Se servant de l'*HTML* et du *Javascript*, ces *Frameworks* permettent de développer des applications multi-plateforme, incluant Android. Ainsi, il n'est plus nécessaire de développer une application pour chaque système d'exploitation. Les technologies les plus connues sont PhoneGap et Cordova(« Présentation - Apache Cordova », s. d.).

En ce qui concerne la compatibilité avec Android Wear, Il semblerait que, nativement, les technologies basées sur le web ne soient pas supportées, principalement en raison de l'absence du support des *Webviews* (Maximiliano, 2015).

Toutefois, Meteor, un Framework basé sur Cordova semble pouvoir être déployé sur une montre Android Wear.(« Using Meteor to develop for Android Wear - CodeProject », s. d.). En l'absence de documentation crédible quant aux fonctionnalités supportées par cette technologie (accès aux capteurs, Data Layer, etc…), il est difficile d'évaluer si ce Framework est une option viable pour développer des applications Android Wear.

II Xamarin

Il s'agit d'un environnement de développement qui permet de créer des applications mobiles multiplateforme en utilisant le langage C# (vincent@nextinpact.com, 2015). L'environnement de développement Xamarin offre la possibilité de créer des applications pour Android wear. Les nouvelles fonctionnalités de la version 2.0 sont également supportées, comme par exemple l'accès à l'*API* Data Layer. (« Introduction to Android Wear - Xamarin », s. d.).

[1] Universal Serial Bus. Il s'agit d'une norme pour connecter les périphériques

III Unity

Il s'agit d'un moteur de jeux multiplateforme utilisant C# comme langage de programmation.(« Unity (moteur de jeu) », 2017).

Il n'est pas spécifié officiellement que Unity est supporté par Android Wear 2.0. Toutefois, il existe des tutoriaux expliquant comment déployer une application vers une montre Android (pastilleadmin, 2016).

IV Java

Java est le langage de programmation officiel sur Android (« I want to develop Android Apps – What languages should I learn? », s. d.).

Le code est compilé sous forme de bytecode. Ensuite, il est exécuté par une machine virtuelle Java (JVM) (« The JVM Architecture Explained - DZone Java », s. d.). Sur les périphériques Android, la machine virtuelle ART (Android RunTime) permet d'exécuter le code (Frumusanu, s. d.). Java permet de développer des applications Android pour Smartphones, Smartwatches, véhicules (Android Auto) et objets connectés (Android Things).

V C++

En utilisant le NDK (Native Development Kit) il est possible développer des applications Android en C++. (« Getting Started with the NDK | Android Developers », s. d.)

VI Kotlin

Kotlin est un nouveau langage de programmation Open source. Il a été créé par JetBrains. Ce langage est décrit comme étant concis, puissant et facile à lire. De plus, il est interopérable avec Java, ce qui permettra de mixer Kotlin et Java au sein du même programme Android. Il est ainsi envisageable de convertir une application JAVA vers Kotlin.(« Kotlin and Android | Android Developers », s. d.).

Les deux extraits de code suivants exécutent la même action, à savoir l'ajout d'un *listener* (évènement) dans un bouton, une fois en Java et une fois en Kotlin :

Java :

```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {
    ...
  }
});
```

Figure 11 : Extrait de code Java
https://developer.android.com/kotlin/get-started.html

Kotlin :

```
val fab = findViewById(R.id.fab) as FloatingActionButton
fab.setOnClickListener {
  ...
}
```

Figure 12 : Extrait de code Kotlin
https://developer.android.com/kotlin/get-started.html

2.3.5 IDE

Les IDE (Environnement de développement intégré) sont des logiciels qui soutiennent le développeur et l'aident à concevoir des applications. Le choix de l'IDE dépend principalement du langage de programmation utilisé. Les IDE suivants permettent de développer et déployer des applications sur Android Wear.

I Visual Studio

Visual Studio est un IDE développé par Microsoft. Il supporte de nombreux langages de programmation notamment C++, C#, HTML, CSS, Python, Javascript, etc… Concernant le développement d'application Android, Visual Studio supporte les technologies suivantes (Webster, 2016) :

- Xamarin en utilisant C#
- C++
- Cordova, via HTML et Javascript
- Unity, utilisant le C#

II Android Studio

Android studio est l'IDE officiel recommandé par Android. Il intègre plusieurs modules pour le développement (téléphone, montre, TV). En outre, il possède également ses propres émulateurs pour chaque module. (« Download Android Studio and SDK Tools | Android Studio », s. d.).

Android Studio est en mesure de compiler et interpréter le Java, le C++ et le Kotlin. (« Add C and C++ Code to Your Project | Android Studio », s. d.).

2.3.6 Sélection de la montre connectée

Pour développer notre application *E-health*, nous allons avoir besoin d'un périphérique Android Wear 2.0. Dans un premier temps, nous allons définir les critères de recherche. Ensuite, il faudra évaluer et comparer les différentes montres disponibles sur le marché. Une fois terminé, nous procéderons à la sélection de la montre qui sera utilisée pour le développement de l'application.

I Délai de livraison

Le délai de livraison est un élément à prendre en compte lors du choix du périphérique. La période de travail étant limitée, nous allons privilégier les modèles disponibles rapidement.

II Capteurs

Les capteurs permettent de collecter diverses informations sur l'utilisateur et son environnement.

- **Accéléromètre**

Ce capteur permet de monitorer le changement de position et l'inclinaison du périphérique (Goodrich, 1, & ET, s. d.). Il sera possible d'analyser non seulement le mouvement, mais également la vitesse de celui-ci et d'utiliser ces valeurs dans les applications (« Sensors Overview | Android Developers », s. d.).

- **Baromètre**

Il s'agit d'un instrument permettant de mesurer la pression atmosphérique. La pression est un indicateur météorologique (on associe une basse pression à un temps nuageux). La pression de l'air va également diminuer lorsque l'on se trouve à une altitude plus élevée (Society & Society, 2014).

- **Altimètre**

Comme son nom l'indique, il permet de mesurer l'altitude du périphérique.

- **Gyroscope**

Ce capteur permet de mesurer la variation d'angle d'un objet. Il peut être utilisé en conjonction avec l'accéléromètre afin d'améliorer la précision et la pertinence des résultats (« Sensors - Mobile terms glossary - GSMArena.com », s. d.) .

- **Boussole**

La boussole est utilisée pour détecter la position du périphérique par rapport au nord. (« Sensors Overview | Android Developers », s. d.).

- **Moniteur de fréquence cardiaque**

Permet de détecter le rythme cardiaque des utilisateurs. Il s'agit d'un capteur intéressant dans le domaine de la santé, notamment dans l'analyse de son état physique.

III Hardware

Il s'agit des composants matériels (disque dur, RAM, processeur, batterie). Cela nous permettra de définir la puissance, la capacité de stockage et l'autonomie de la montre.

IV Comparaison

Les modèles sélectionnés sont proposés sur le site officiel d'Android Wear (« Android Wear », s. d.-) . Nous n'allons pas intégrer les modèles de luxe à notre recherche.

Le tableau suivant liste les modèles disponibles sur le marché avec leurs caractéristiques techniques.

COMPARAISON ANDROID WEAR 2.0

Modèle	Date de sortie	Accéléromètre	Baromètre	Altimètre	Gyroscope	Boussole	Cœur	Proximité	Processeur	Ram	Batterie	Lien
Motorola Moto 360 (1st gen)	sept.14	oui	non	non	non	oui	non	non	1GHZ	512MB	Non-removable Li-Ion 320 mAh battery.	http://www.gsmarena.com/motorola_moto_360_(1st_gen)-7682.php
Motorola Moto 360 46mm (2nd gen)	sept.15	oui	non	non	oui	non	oui	non	Quad-core 1.2 GHz Cortex-A7	512 MB RAM	Non-removable Li-Ion 400 mAh battery.	http://www.gsmarena.com/motorola_moto_360_46mm_(2nd_gen)-7683.php
LG Watch Style	févr.17	oui	non	non	oui	non	non	non	1.1GHz	512MB	Non-removable Li-Ion 240 mAh battery.	http://www.pocket-lint.com/news/140202-lg-watch-sport-vs-lg-watch-style-what-s-the-difference
LG Watch Sport	févr.17	oui	oui	non	oui	non	oui	oui	Quad-core 1.1 GHz	768 MB RAM	Non-removable Li-Ion 430 mAh battery.	http://www.gsmarena.com/lg_watch_sport-8552.php
Motorola Moto 360 Sport (1st gen)	janv.16	oui	non	oui	oui	non	oui	non	Quad-core 1.2 GHz	512 MB	Non-removable Li-Ion 300 mAh battery.	http://www.gsmarena.com/motorola_moto_360_sport_(1st_gen)-7685.php
Casio WSD-F20-RG	avr.17	oui	oui	non	oui	oui	non	non	non spécifié	non spécifié	Lithium-ion battery.	http://wsd.casio.com/us/en/wsd-f20/products/
Huawei Watch 2	avr.17	oui	oui	non	oui	oui	oui	non	Quad-core 1.1 GHz	768 MB RAM	Non-removable Li-Ion 420 mAh battery.	http://www.gsmarena.com/huawei_watch_2-8585.php
ASUS ZenWatch 3	déc.16	oui	non	non	oui	non	non	non	Qualcomm® Snapdragon™ Wear 2100	512 MB RAM	Non-removable Li-Ion 341 mAh battery.	http://www.gsmarena.com/asus_zenwatch_3_wi503q-8271.php
LG Watch Urbane 2nd Edition LTE	mars.16	oui	oui	non	oui	oui	oui	oui	Quad-core 1.2 GHz Cortex-A7	768 MB RAM	Non-removable Li-Ion 570 mAh battery.	http://www.gsmarena.com/lg_watch_urbane_2nd_edition_lte-7607.php http://www.lg.com/us/smart-watches/lg-W200V-lg-watch-urbane-2nd-edition-verizon http://www.ubergizmo.com/products/lang/en_us/devices/lg-urbane-2/
ZTE quartz	avr.17	oui	oui	non	oui	non	non	non	1.1 GHz Quad-Core MSM8909w Qualcomm Snapdragon Wear 2100	768 MB RAM	500 mAh, non-removable	http://smartwatchspecifications.com/Device/zte-quartz/ https://www.engadget.com/2017/04/10/zte-quartz/
Casio WSD-F10 smartwatch	mars.16	oui	oui	oui	oui	oui	non	non	non spécifié	non spécifié	Lithium-ion battery.	http://www.androidcentral.com/casio-wsd-f10-smartwatch-specs https://www.gs-central.com/casio-smart-outdoor-watch-wsd-f10/

Tableau 1 : Tableau comparatif des montres Android Wear 2.0

Le choix final s'est porté sur la montre « Huawei Watch 2 ». Cette montre possède les avantages suivants :

- Présence de nombreux capteurs.
- La taille de la mémoire vive est conséquente.
- Elle est disponible rapidement.

3 MAGPIE

Figure 13 : Logo MAGPIE
https://www.hevs.ch/en/minisites/projects-products/aislab/projects/magpie-4826

Un des impératifs de ce travail est l'utilisation du Framework MAGPIE. Nous allons intégrer ce Framework à l'application que nous allons développer.

3.1 Introduction

MAGPIE est un sigle signifiant « Mobile computing with AGents and Publish/subscribe for Intelligent u-hEalthcare ». Il s'agit d'un Framework développé en Java par Albert Brugués durant sa thèse de doctorat, dans le cadre de l'institut AISLab. Ce Framework est utilisé dans les applications *E-health* (cybersanté).

Les paragraphes suivants vont décrire son architecture. Les informations quant aux spécificités techniques ont été détaillées et décrites par l'auteur dans sa thèse. Il est supposé que le lecteur dispose d'une certaine connaissance en architecture Android.

3.2 Architecture et composants

Ce Framework multi-agent a été conçu pour traiter des évènements. Un évènement est caractérisé par une valeur physiologique, autrement dit, une mesure (par exemple la pression sanguine, ou le taux de glucose dans le sang).

Des agents vont ensuite se charger de traiter ces évènements, selon des règles prédéfinies par le programmeur. Ils vont émettre des alertes en fonction des données traitées. Le Framework est composé de trois éléments : les MagpieActivity, les Events et les Agents.

3.2.1 MagpieActivity

Concrètement, les MagpieActivity sont des activités comportant des fonctionnalités supplémentaires propres au Framework. Ces activités permettent, entre autres, d'instancier et souscrire des agents. Une fois l'agent souscrit à l'activité, il sera possible de lui transmettre des évènements au travers de la MagpieActivity.

3.2.2 Les Events (évènements)

Les évènements sont représentés par la class MagpieEvent au sein du Framework MAGPIE. Il s'agit de mesures. Chaque événement peut comporter une ou plusieurs valeurs. Par exemple, un évènement concernant la pression sanguine va comporter deux mesures : la pression systolique et la pression diastolique.

Chaque évènement est composé des éléments suivants :

- Un type : permet d'identifier le type d'événement (pression sanguine, taux de glucose dans le sang, etc..). Il s'agit d'une chaîne de caractères.
- Un horodatage : permet d'indiquer quand l'évènement s'est déclenché. Il désigne le nombre de secondes ou millisecondes écoulées depuis le 1er janvier 1970. Il est stocké sous forme d'entier.
- Valeur : Il s'agit de la valeur mesurée. Par exemple, un taux de sucre dans le sang de 6,9. Chaque évènement peut comporter une ou plusieurs valeurs. Les valeurs sont enregistrées dans un tableau de chaînes de caractère. Cela offre la possibilité aux développeurs de traiter la valeur comme ils le souhaitent (sous forme nombre entier ou de nombre décimal).

3.2.3 Les agents

Les agents sont des entités qui permettent de traiter les données provenant d'évènements, à l'aide de règle. Les règles émanent généralement du corps médical. Elles informent le patient sur son état de santé et sa condition physique actuelle.

Les agents sont composés de deux éléments :

- Body : Il s'agit de la partie qui reçoit/produit les évènements.
- Mind : Il s'agit du traitement des données. Il sera possible de définir les règles en intégrant un comportement (Behaviour) à l'agent. Les données peuvent être traitées de deux façons différentes, soit en utilisant les « Prolog Mind », soit en utilisant les « Java Mind ».

I Prolog Mind

Prolog, également appelé langage déclaratif, est un langage de programmation haut niveau, basé sur des formules logiques. Il est constitué de règles et de faits (« What is Prolog? Webopedia Definition », s. d.).

MAGPIE possède la capacité de lire les règles développées en langage Prolog. De facto, il est possible d'intégrer ces règles dans un programme Android. La figure suivante illustre la structure d'une règle développée en Prolog, au sein d'une application MAGPIE.

```
initiates_at(alert(first)=situation('Brittle diabetes'),T):-
        hours_ago(6,Tago,T),
        not query_kd(happens_at(alert(first,'Brittle diabetes'),Tev0), [Tago, T]),
        query_kd(happens_at(glucose(Value1),Tev1), [Tago, T]),
        query_kd(happens_at(glucose(Value2),Tev2), [Tago, T]),
        Value1 =< 3.8,
        Value2 >= 8,
        Tev2 > Tev1.
```

Figure 14 : Exemple de règle Prolog - Règle écrite par Albert Brugués

La règle déclenche une alerte si :

- Aucune autre alerte n'a été émise durant une période de six heures auparavant.
- Il existe au minimum deux évènements durant une période de six heures.
- La valeur de glucose du premier évènement doit être inférieur à 3,8.
- La valeur de glucose du second évènement doit être supérieur à 8.
- Le second évènement doit apparaître après le premier évènement.

II Java Mind

Il s'agit de règles programmées directement en Java. Elles permettent notamment de compléter les règles écrites en Prolog, comme par exemple effectuer des statistiques sur le nombre d'événements.

3.3 MAGPIE sous Android Wear 2.0

Le système d'exploitation Android Wear est un dérivé d'Android. De ce fait, il est important de savoir s'il supporte le Framework MAGPIE. Les prochaines sections décrivent les tests de compatibilités effectués.

I Test

Nous allons baser nos tests sur l'application « Sample-sensor », développée par Albert. Pour être en mesure de déployer l'application sur la montre, nous avons modifié le fichier *manifeste*. La ligne suivante permet de spécifier le périphérique de destination.

```
<uses-feature android:name="android.hardware.type.watch" />
```

Figure 15 : Extrait de code du manifeste Android
Source personnelle

Cela force le déploiement de l'application sur une montre Android Wear. L'IDE nous empêche de déployer l'application sur un autre type de périphérique.

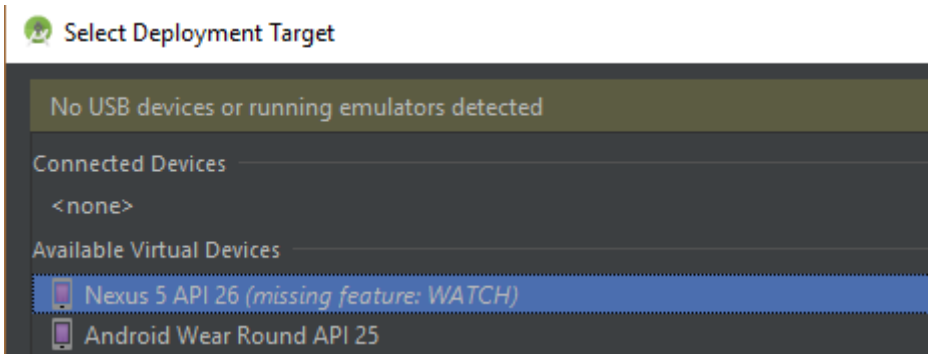


Figure 16 : Capture d'écran de la sélection du périphérique Android Studio
Source personnelle

L'application est composée de *class* de tests unitaires. Il existe deux tests : LogicTupleTest et PrologAgentMindTest. Albert Brugués nous a confirmé que ces tests permettent de déterminer si le périphérique exécute correctement le Framework MAGPIE. Les *class* sont les suivantes :

- **LogicTupleTest**

Cette *class*, composée de trois tests unitaires, va tester la création et la génération d'évènements. En addition, elle va également tester si les informations retournées par les évènements (le type, les valeurs et la catégorie) sont correctes.

- **PrologAgentMindTest**

Cette *class* comporte également trois tests. Elle va contrôler la viabilité des agents développés en Prolog. Chaque test unitaire va déterminer si l'agent déclenche une alerte suite à un évènement particulier. Ces tests permettent de définir si le périphérique est en mesure d'exécuter des règles Prolog.

II Résultats et conclusion

Les deux tests décrits précédemment ont été exécutés sur un émulateur Android Wear. Ils donnent les résultats suivants :

- **Résultat LogicTupleTest**

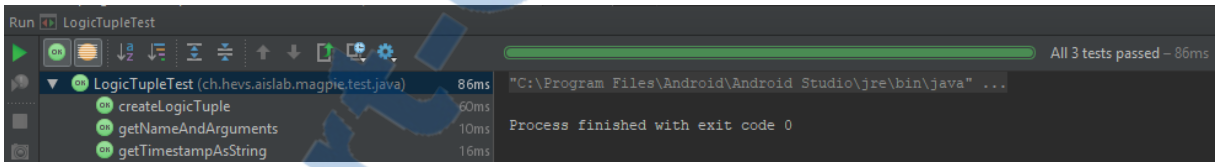


Figure 17 : Résultat du test LogicTupleTest
Source personnelle

- **Résultat PrologAgentMindTest**

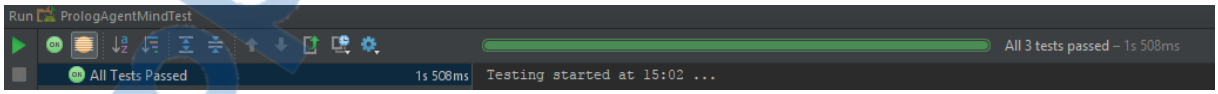


Figure 18 : Résultat du test PrologAgentMindTest
Source personnelle

Les résultats indiquent que le *Framework* est compatible avec le système d'exploitation Android Wear 2.0. Ainsi, Il est envisageable de développer une application Android Wear avec le *Framework* MAGPIE.

4 Prototype d'application

Afin de démontrer les possibilités applicatives offertes par une montre Android Wear, nous allons développer une application *E-health*, à l'aide du Framework MAGPIE. Cette section va décrire le prototype réalisé et ses fonctionnalités.

4.1 Environnement de travail

4.1.1 Langage de programmation

Le Framework MAGPIE a été développé en Java. De facto, nous allons nous tourner vers ce langage pour créer notre application. Nous n'allons pas utiliser le langage Kotlin. Son intégration à Android est récente, il est donc difficile d'avoir un retour quant à sa fiabilité. Nous allons utiliser l'IDE officiel proposé par Android, Android Studio.

4.1.2 VCS

Nous allons utiliser un système de contrôle de version. Ces systèmes sauvegardent chaque version de l'application sur un serveur externe. Le développement sera ainsi plus sécurisé (possibilité de revenir à une version précédente en cas de corruption des fichiers, sauvegarde du code en cas de panne d'un ordinateur, etc.). (« Git - About Version Control », s. d.).

Android Studio propose un outil qui permet de synchroniser le code vers un serveur GIT[2]. Nous nous sommes dirigé vers la plateforme GitHub[3], qui propose un service gratuit sans restriction de taille.

4.2 Scénario

Nous allons développer une application *E-health* qui permettra de donner un retour à l'utilisateur concernant son état de santé.

4.2.1 Objectif du projet

L'objectif principal de ce projet est le développement d'un système personnel de santé (PHS) embarqué sur un périphérique portable à l'aide du Framework MAGPIE.

Initialement, il était prévu de capturer les données sur la montre, de les transmettre vers le téléphone et finalement, de les traiter à l'aide du Framework MAGPIE, au sein du téléphone même.

Toutefois, au vu des conclusions positives concernant la capacité autonome de la montre, et son aptitude à exécuter le Framework MAGPIE, nous avons décidé d'un commun accord avec Monsieur Michael Schumacher et Monsieur Davide Calvaresi d'intégrer directement le Framework à la montre. Cela apporte plus de souplesse à l'utilisateur et permet à la montre d'être totalement indépendante du téléphone.

[2] Logiciel de control de version

[3] Plateforme offrant des services GIT gratuitement. https://github.com/

Quant au téléphone, il sera utilisé pour stocker les données enregistrées par la montre et les afficher sous forme de graphique. Il permettra en outre de libérer de l'espace mémoire sur la montre.

4.2.2 Objectif de l'application

Cette application a pour objectif d'informer l'utilisateur sur son état de santé. Elle est axée sur cinq catégories médicales :

- **Glucose :** il s'agit du taux de sucre dans le sang. Il est exprimé en mmol/l (Millimoles par Litre).
- **Pulse :** Il s'agit du rythme cardiaque, exprimé par des battements de cœur par minute.
- **Pressure :** La pression sanguine. La pression sanguine est caractérisée par deux valeurs : la pression systolique et la pression diastolique.
- **Weight :** le poids de l'utilisateur exprimé en Kilo.
- **Steps :** le nombre de pas effectués durant une journée.

Des mesures liées à chaque catégorie vont être introduites au sein de l'application. Ces mesures seront ensuite traitées par les agents de MAGPIE, en utilisant des règles et déclencheront des alertes en fonction de l'état de santé de l'utilisateur.

4.3 Méthodologie

Pour développer l'application, nous avons utilisé la méthodologie agile Scrum. Cela nous permettra de travailler par itération et nous procurera plus de souplesse. La durée des *sprints* varie entre une et deux semaines, selon le nombre de *User Stories*.

4.3.1 Product Backlog

Le *Product Backlog* disponible en annexe comporte toutes les *User Stories* de l'application. Elles définissent les fonctionnalités que doit posséder notre application. Les *User Stories* ont été approuvées et validées par Monsieur Michael Schumacher et Monsieur Davide Calvaresi.

Les sept premières *User Stories* sont dédiées à l'analyse du système d'exploitation Android Wear et à l'apprentissage du *Framework* MAGPIE. Les suivantes sont spécifiques au prototype. Le prototype est découpé en deux grandes sections :

- Le développement de l'application sur la montre.
- Le développement de l'application sur le téléphone.

4.4 Développement des fonctionnalités de la montre

Les outils et les technologies ont été définis. Nous allons maintenant passer au développement du prototype. Les paragraphes suivants décrivent les fonctionnalités développées sur l'application propre à la montre.

4.4.1 Fondation et structure de l'application

Nous allons définir les fondations et l'architecture générale de notre application.

I Structure de la base de données

La base de données utilisée par Android est SQLite (« Saving Data in SQL Databases | Android Developers », s. d.). Elle est également présente sur les périphériques Android Wear et sera utilisée pour stocker les données.

La figure suivante illustre la structure de notre base de données :

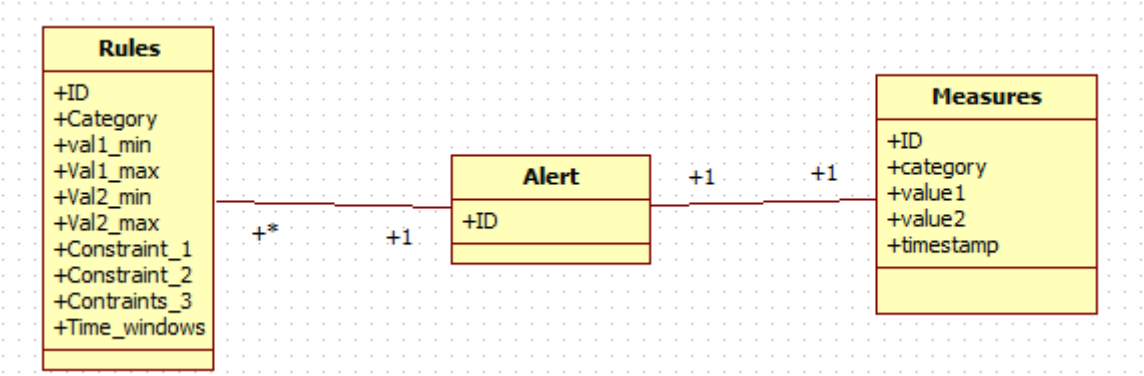


Figure 19 : Schéma de la base de données
Source personnelle

- **Measures**

Une mesure représente une donnée physiologique. Elle est associée à un évènement au sein du Framework MAGPIE (par exemple, l'ajout d'une nouvelle valeur de pression sanguine, l'ajout d'un taux de sucre dans le sang). Une mesure est constituée des éléments suivants :

- Une catégorie.
- Deux valeurs physiologiques.
- Un horodatage indiquant la date et l'heure de l'événement.

➢ **Rules**

Représente la table des règles. Les agents vont utiliser les règles pour contrôler les données. Les règles sont définies par :

- Une catégorie (Pression sanguine, taux de sucre dans le sang, etc…).
- Des valeurs limites, autrement dit, des bornes.
- Des contraintes liées aux valeurs.
- Une fenêtre de temps (par exemple six heures ou sept jours).

➢ **Alert**

Une alerte est déclenchée si une mesure est supérieure ou inférieure aux valeurs limites définies dans les règles. Une alerte est donc liée à une mesure et à une règle.

II ORM

Comme la base de données est souvent cible de changement, elle doit s'adapter aux besoins métiers. Pour cela, nous allons utiliser un ORM (Object Relational Mapping).

En programmation, un ORM est un outil qui permet de faire le lien entre les tables de la base de données et la programmation objet. Chaque table sera associée à un objet java (« Dictionnaire des développeurs », s. d.). Ainsi nous n'aurons plus à écrire nous-mêmes la majorité des requêtes. Cette tâche sera déléguée à l'ORM. Cela nous apporte un gain de souplesse, de temps et réduit l'impact d'une restructuration de la base de données.

➢ **GreenDAO**

Il existe de nombreux ORM pour Android. Chacun dispose de sa propre syntaxe et de ses propres caractéristiques. Nous allons en choisir un parmi les plus connus. Les critères suivants ont été définis pour la sélection : la rapidité d'apprentissage, la taille de la communauté et la vitesse d'exécution. La figure suivante nous montre le nombre de recherches Google des ORM les plus connus.

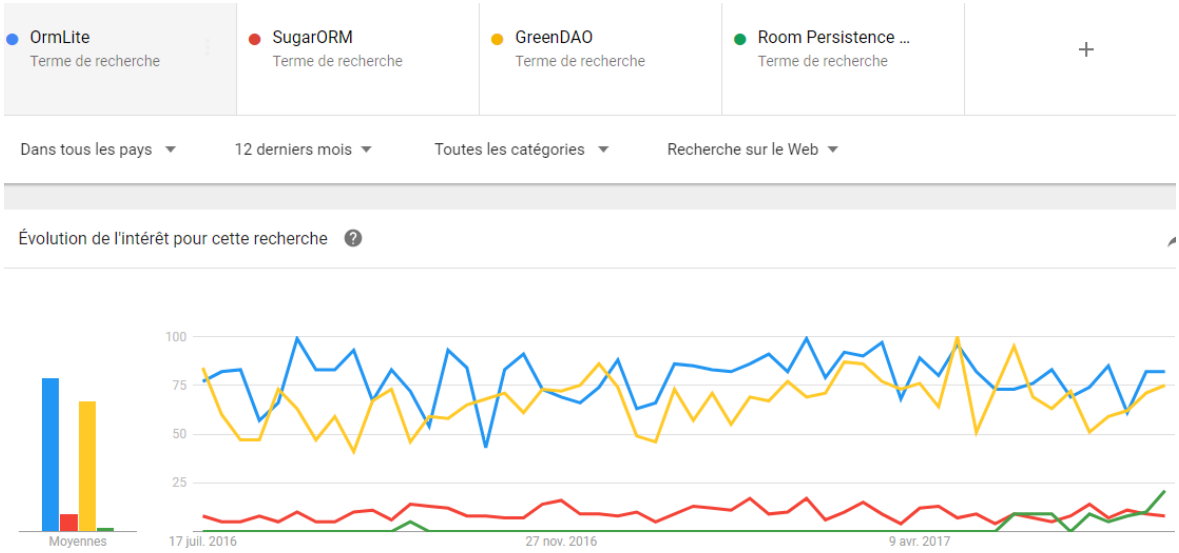


Figure 20 : Comparaison des recherches Google d'ORM
https://trends.google.ch/trends/explore?q=OrmLite,SugarORM,GreenDAO,Room%20Persistence%20Library

Il apparait que GreenDao[4] et ORMLite[5] sont les plus utilisés. Par conséquent, ils possèdent une communauté importante. En ce qui concerne la vitesse d'exécution, GreenDao est supérieur à son concurrent ORMLite, comme le montre la figure suivante, trouvée sur le site officiel de GreenDao.

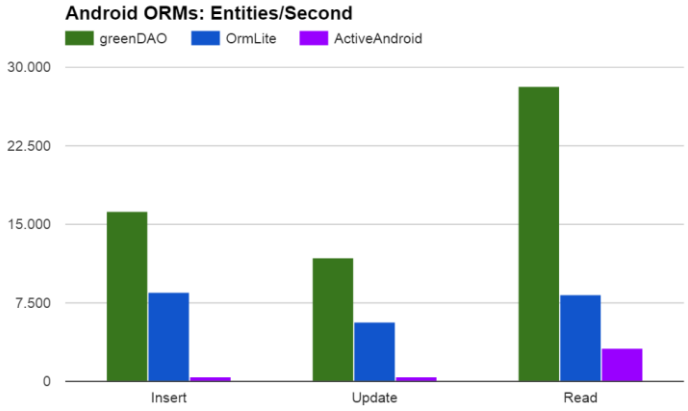


Figure 21 : Comparaison des performances des ORM
http://greenrobot.org/greendao/features/

GreenDAO s'identifie comme l'ORM le plus rapide sur le marché. (« greenDAO Features », s. d.). Sa syntaxe est claire et il est facile à prendre en main. Il possède également une bonne documentation ce qui facilite le développement. GreenDAO a été utilisé lors du développement de grande application comme Pinterest.

[4] ORM Android. http://greenrobot.org/greendao/

[5] ORM Java. http://ormlite.com/

III Création du module Android Wear

Android Studio offre la possibilité d'intégrer un module en spécifiant le type de périphérique ciblé. Nous allons ajouter un module « Android Wear » au *Framework* MAGPIE.

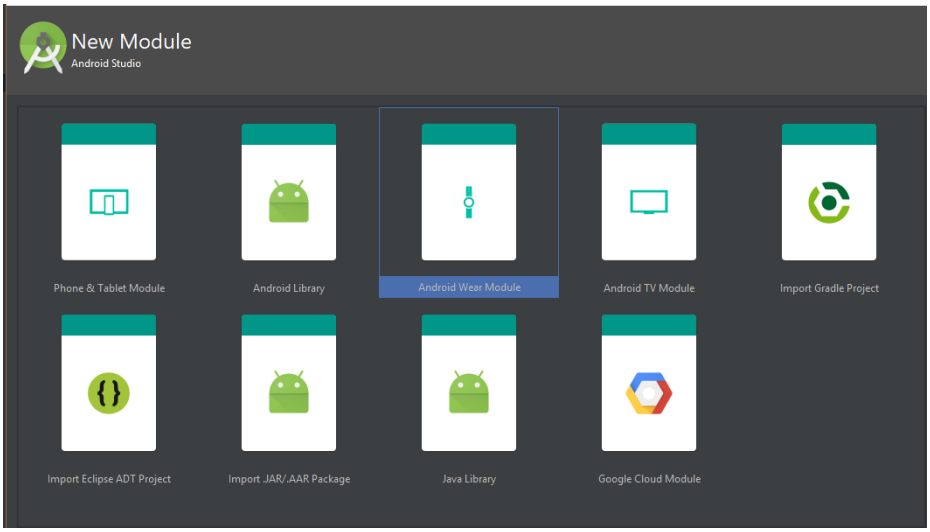


Figure 22 : Création du module Android Wear sur Android Studio
Source personnelle

L'IDE se charge ensuite de créer l'activité de départ. Il ajoutera également la ligne dans le *manifeste* spécifiant le périphérique de destination.

IV Activités et fragment

Notre application sera constituée de deux activités et plusieurs fragments.

Les activités sont le point central des applications Android. Elles permettent de gérer les interactions avec l'utilisateur et la navigation. Il s'agit en quelque sorte d'un container pouvant accueillir plusieurs éléments, comme par exemple des fragments.

Un fragment est une portion d'interface utilisateur. Le cycle de vie des fragments, l'affichage et l'interaction avec l'utilisateur sont délégués à l'activité (« Fragments | Android Developers », s. d.).

Le schéma suivant illustre l'organisation des activités et des fragments au sein de notre application.

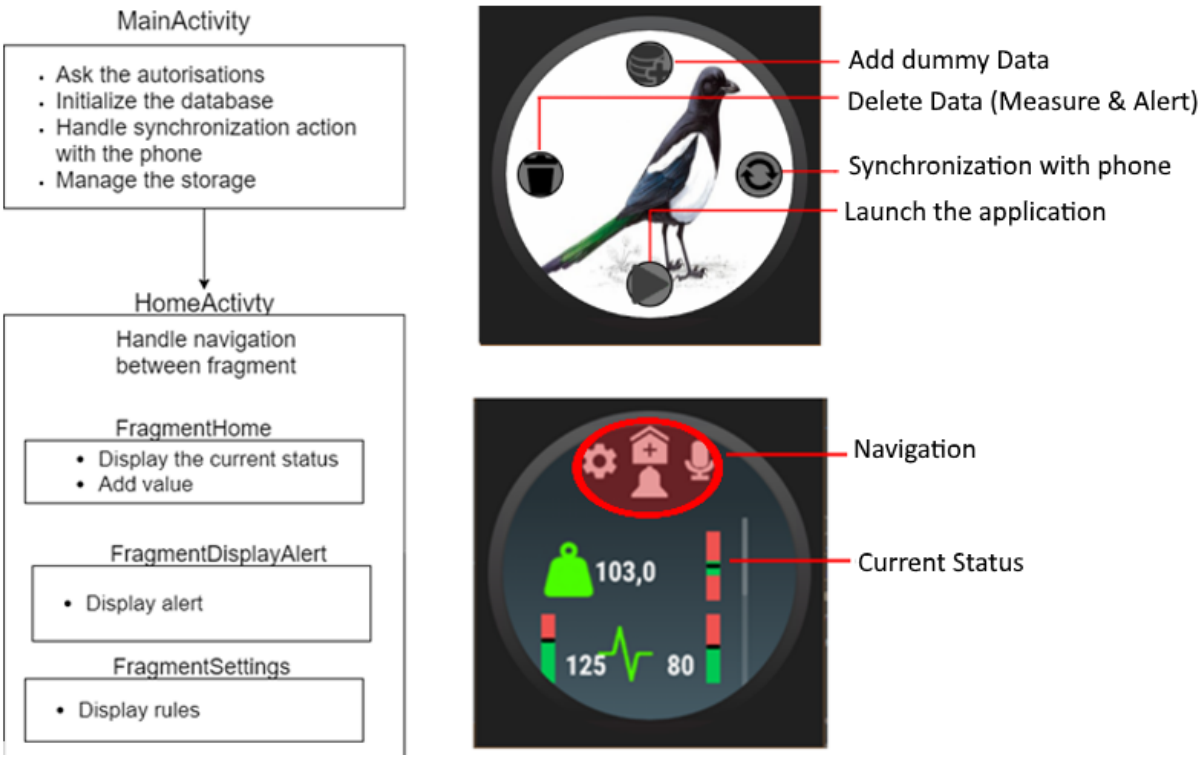


Figure 23 : Schéma général de l'application
Source personnelle

V Implémentation de la base de données et de l'ORM GreenDao

L'ajout de la librairie GreenDao se fait au sein des fichiers Gradle[6]. Pour cela, il suffit d'y ajouter la dépendance. Les *class* seront ensuite automatiquement générées.

Les tables dans la base de données sont créées via les entités. Il s'agit d'objets java avec des annotations spécifiques qui permettent de les lier aux tables. Le code suivant représente l'entité « Alertes ».

```
@Entity(
        nameInDb = "Alertes",
        indexes = {
        @Index(value = "measure_id, rule_id", unique = true)
    }
public class Alertes  {

    @Id(autoincrement = true)
    private Long id;
    @NotNull
    private long measure_id;
```

[6]Gradle est un outil utilisé par Android Studio qui permet de gérer les dépendances et la construction des applications. https://gradle.org/

```
    @NotNull
    private long rule_id;

    @ToOne(joinProperty = "measure_id")
    private Measure measure;


    @ToOne(joinProperty = "rule_id")
    private CustomRules rule;

  public Alertes() {
    }
  }
 public Long getId() {
    return id;
  }

  public void setId(Long id) {
    this.id = id;
  }


  public long getMeasure_id() {
    return measure_id;
  }

  public void setMeasure_id(long measure_id) {
    this.measure_id = measure_id;
  }

  public long getRule_id() {
    return rule_id;
  }

  public void setRule_id(long rule_id) {
    this.rule_id = rule_id;
  }
}
```

Figure 24 : Exemple de code d'une entité
Source personnelle

Une fois l'application assemblée par Android Studio, l'ORM va générer des annotations et des méthodes au sein de chaque entité. Ces méthodes sont utilisées en interne par l'ORM.

VI Repository

Nous allons utiliser une adaptation personnalisée du pattern repository[7] pour centraliser les accès à l'ORM. Les repository sont des *class* Java où sont stockées les méthodes d'accès à l'ORM. Ces méthodes vont manipuler les tables dans la base de données.

[7] Pattern permettant de centraliser les accès base de données

Chaque table possède son propre repository. La figure suivante illustre le cheminement des accès à la base de données :

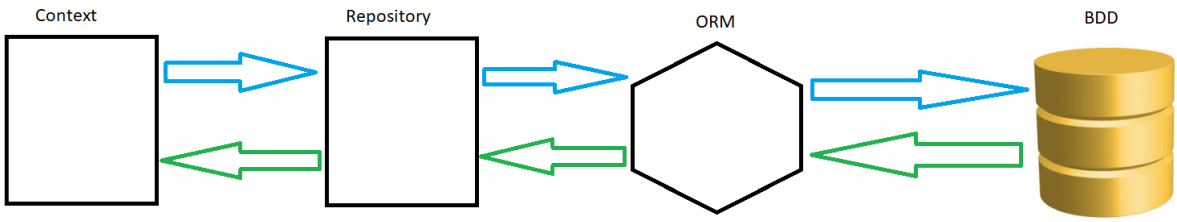


Figure 25 : Schéma illustrant l'accès aux repositories
Source personnelle

Les repositories utilisent le pattern Singleton[8] pour éviter une instanciation multiple.

VII Support multilingue

Le support multilingue permet de proposer plusieurs langages au sein d'une application. Chaque texte affiché à l'utilisateur, via l'interface graphique, sera traduit en plusieurs langues. Android propose une gestion multilingue simplifiée en utilisant des fichiers ressources XML [9]pour chaque langage. La figure suivante nous montre les différents fichiers texte de l'application.

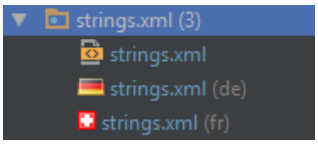
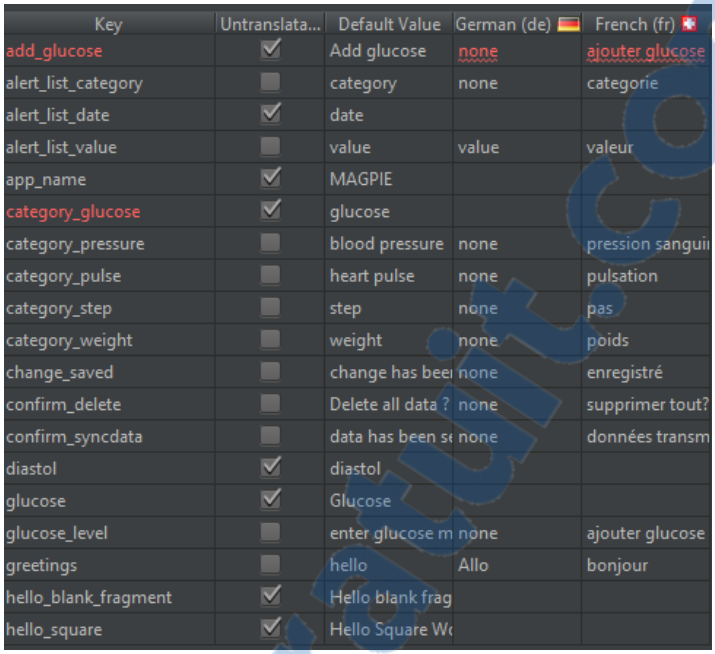


Figure 26 : Capture d'écran des fichiers de langues sous Android Studio
Source personnelle

Android Studio possède un outil graphique permettant l'affichage et la traduction des textes via un tableau. La capture suivante nous montre le tableau de traduction.

[8] Pattern permettant d'instancier une seule fois un objet

[9] Extensible Markup Language. Il s'agit d'un format de données



Key	Untranslata...	Default Value	German (de)	French (fr)
add_glucose	☑	Add glucose	none	ajouter glucose
alert_list_category	☐	category	none	categorie
alert_list_date	☑	date		
alert_list_value	☐	value	value	valeur
app_name	☑	MAGPIE		
category_glucose	☑	glucose		
category_pressure	☐	blood pressure	none	pression sanguii
category_pulse	☐	heart pulse	none	pulsation
category_step	☐	step	none	pas
category_weight	☐	weight	none	poids
change_saved	☐	change has bee	none	enregistré
confirm_delete	☐	Delete all data ?	none	supprimer tout?
confirm_syncdata	☐	data has been s	none	données transm
diastol	☑	diastol		
glucose	☑	Glucose		
glucose_level	☐	enter glucose m	none	ajouter glucose
greetings	☐	hello	Allo	bonjour
hello_blank_fragment	☑	Hello blank frag		
hello_square	☑	Hello Square W		

Figure 27 : Tableau de traduction Android Studio Source personnelle

Notre application est traduite en français et en anglais. Le support pour l'allemand est également présent, mais les textes ne sont pas traduits.

4.4.2 Insertions des mesures

Nous allons maintenant nous concentrer sur la création et l'insertion des mesures. Elles sont créées de deux manières. De manière automatique à l'aide des capteurs de la montre et de manière manuelle.

I Insertion par les capteurs

Notre application utilisera deux capteurs : le capteur de fréquence cardiaque et le capteur de pas. L'utilisateur n'aura donc pas besoin d'insérer lui-même ces deux valeurs.

- **Initiation et instanciation**

Les capteurs s'instancient à l'intérieur des activités à l'aide de l'objet SensorManager. Le code suivant permet d'instancier et d'activer le capteur de pas.

```
sensorManager=(SensorManager) getSystemService(SENSOR_SERVICE);
sensor_step =sensorManager.getDefaultSensor(Sensor. TYPE_STEP_DETECTOR);
sensorManager.registerListener(this,sensor_step,SensorManager.SENSOR_DELAY_NORMAL);
```

Figure 28 : Code permettant d'initialiser le capteur de pas
Source personnelle

Le capteur va ensuite déclencher deux types d'événements :

- Un changement de précision, qui va être représenté par un nombre allant de -1 à 3 (la valeur -1 correspond à un capteur non fiable, et 3 signifie que le capteur est fiable). Dans notre cas, nous allons enregistrer les changements de précision du capteur de pulsations. Il sera utilisé par la suite pour déterminer si les valeurs sont fiables et peuvent être transmises aux agents. L'exemple suivant illustre le *listener* de précision des pulsations.

```
public void onAccuracyChanged(Sensor sensor, int i) {
      //set the accuracy of the pulse snsors
      if (sensor.getType()==Sensor.TYPE_HEART_RATE)
      {
         accuracySensorPulse=i;
      }
   }
```

Figure 29 : Code Permettant l'accès au changement de précision du capteur
Source personnelle

- Un changement de la valeur du capteur, représenté par un nombre. Par exemple, le capteur de pulsations passe de 70 pulsations par minute à 80. Le code suivant est utilisé pour traiter les changements de valeurs des capteurs.

```
public void onSensorChanged(SensorEvent sensorEvent) {

      switch (sensorEvent.sensor.getType())
      {  //handle heart event
         case Sensor.TYPE_HEART_RATE :
            //add a value to the array, but only if the accuracy is at least at 1 (-1 == no contact, 3 ==
best contact)
            if (accuracySensorPulse<1)
               return;
            double value=sensorEvent.values[0];
            listPulse.add(value);
            break;
         case Sensor.TYPE_STEP_COUNTER :
            //for this sensor, we juste update the value. the value is processed only once a week
            addStep(1);
            break;

      }
   }
```

Figure 30 : Code permettant d'enregistrer le changement de valeur d'un capteur
Source personnelle

Le capteur de pas et de pulsations ont chacun un cycle de vie et un fonctionnement bien spécifique.

- **Capteur de pulsations**

Le capteur de pulsations enregistre des données chaque seconde. Pour éviter d'avoir une multitude de données et pour éviter une utilisation excessive de la batterie, nous allons déléguer son cycle de vie à un *thread* qui va activer et stopper le capteur. La figure suivante illustre son cycle de vie.

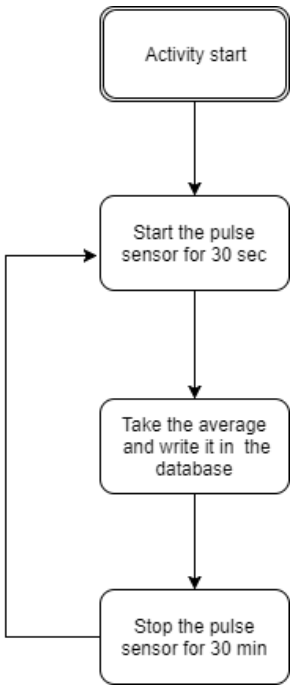


Figure 31 : Schéma illustrant le cycle de vie du capteur de pulsations
Source personnelle

Le capteur de pulsations sera initialisé au lancement de l'activité. Il va enregistrer les données pendant 30 secondes et sauvegarder chaque valeur de pulsations dans une liste. Si une de ces valeurs n'est pas précise, elle sera ignorée et ne sera pas introduite. Une fois ce laps de temps écoulé, la valeur moyenne de la liste sera calculée. Cette valeur sera ensuite enregistrée dans la base de données sous forme de mesure.

Le capteur sera ensuite désactivé pendant 30 minutes et va recommencer. Il ne sera stoppé que lorsque l'on quitte l'activité.

➢ **Capteur de pas**

Nous allons stocker le nombre de pas effectués chaque jour. L'image suivante représente la manière dont fonctionne l'enregistrement des pas au sein de la base de données.

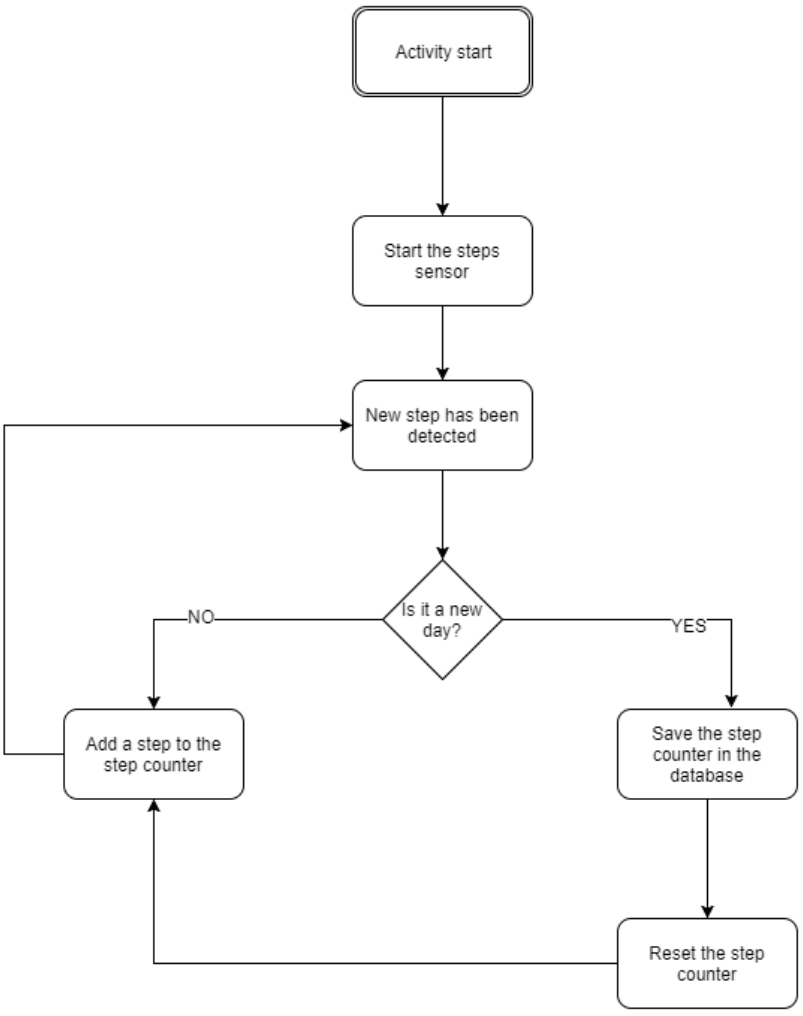


Figure 32 : Schéma illustrant la gestion du capteur de pas
Source personnelle

Le capteur de pas sera activé au démarrage de l'application. A chaque nouveau pas, nous allons vérifier s'il s'agit d'un nouveau jour par rapport au dernier pas enregistré. Si tel est le cas, nous allons sauvegarder le nombre de pas effectués dans la base de données. Le compteur sera ensuite réinitialisé à zéro et nous allons rajouter le nouveau pas effectué.

II Insertion manuelle

Le taux de sucre dans le sang (glucose), la pression sanguine (systole et diastole) et le poids devront être insérés manuellement par l'utilisateur. Il pourra le faire de deux façons.

- **Via l'interface graphique**

En cliquant sur une des icônes de catégorie glucose, pression sanguine ou poids.

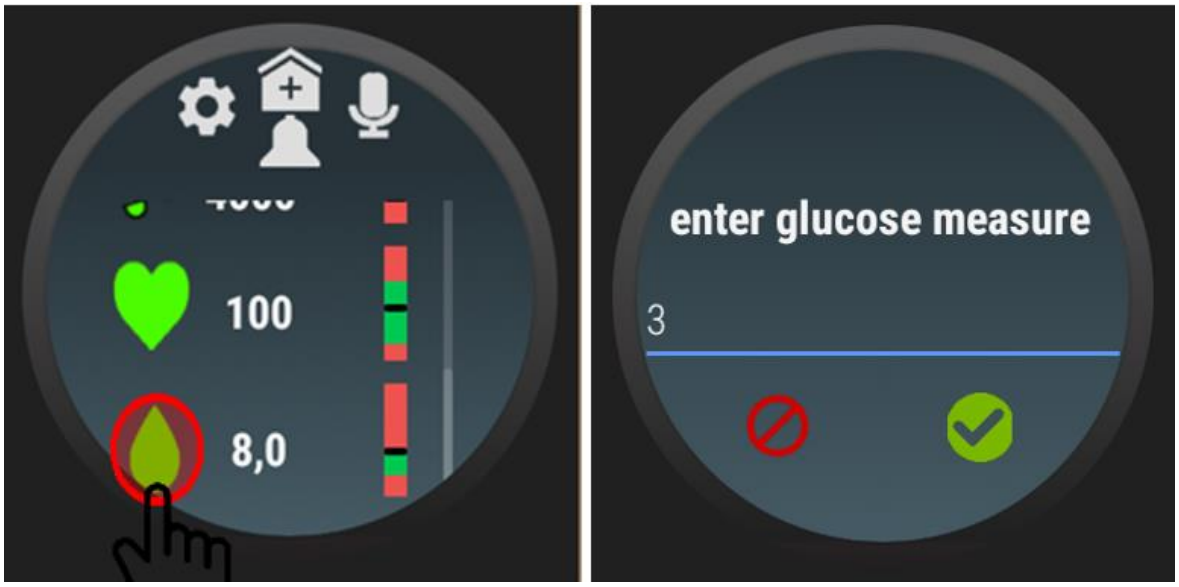


Figure 33 : Illustration de l'option permettant d'entrer des valeurs manuellement
Source personnelle

- **Via la reconnaissance vocale**

L'utilisateur pourra également introduire des données en utilisant la reconnaissance vocale. Une fois activée à l'aide du bouton, l'utilisateur va dicter sa commande. L'image ci-dessous illustre l'activation de la commande vocale.

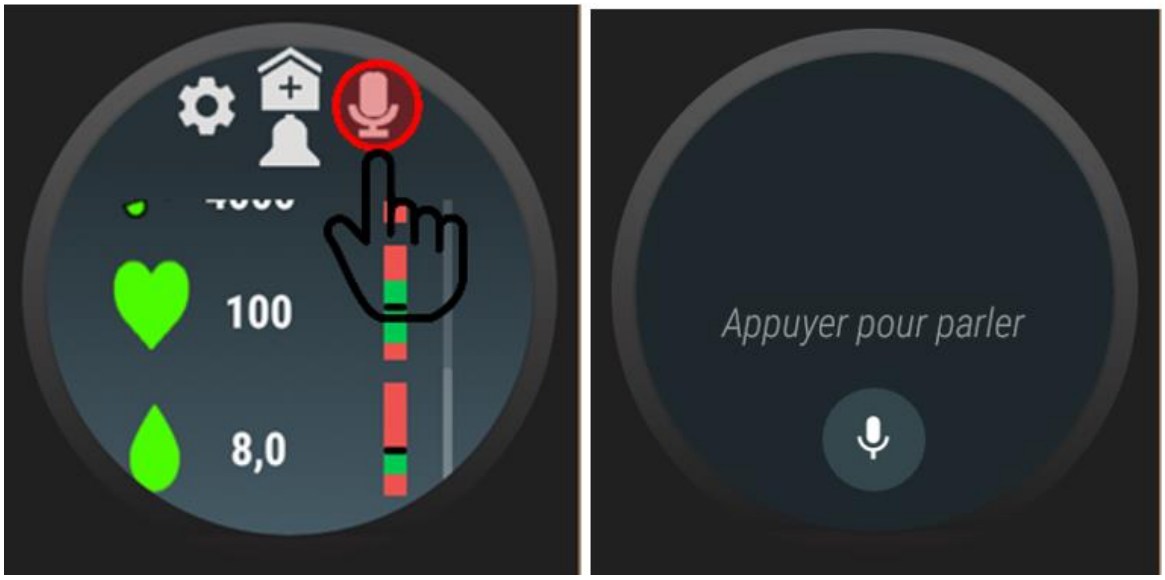


Figure 34 : Illustration de la commande vocale
Source personnelle

Android wear va traduire les mots en chaîne de caractère. Elle sera ensuite renvoyée à l'activité. Les commandes vocales disponibles au sein de l'application sont les suivantes :

- **Glucose {valeur} :** permet d'ajouter une valeur de taux de sucre dans le sang.
- **Poids {valeur} :** Permet d'ajouter son poids.
- **Pression {valeur systole} sur {valeur diastole} :** permet d'insérer sa pression sanguine.

Le code suivant est utilisé pour traiter la commande vocale « ajout de glucose ».

```
private void handleVoiceEvent(Intent data) {
    List<String> results = data.getStringArrayListExtra(
            RecognizerIntent.EXTRA_RESULTS);
    //get the spocken language in a string
    String spokenText = results.get(0).toLowerCase();
    //replace eventual comma by "."
    spokenText = spokenText.replace(",", ".");
    String glucose = getString(R.string.voice_glucose);
    //split the text spoken into an array. The first value will be category, the second the value
    String[] arraySpocken = spokenText.split(" ");
    //ad the glucose by voice
    if (arraySpocken[0].toLowerCase().equals(glucose))
    {
        if (arraySpocken.length >= 1)
        {
            voiceAction_addValue(Const.CATEGORY_GLUCOSE, arraySpocken[1]);
            return;
        }
        // no number has been specified, so command is not completed
        CustomToast.getInstance().warningToast(getString(R.string.voice_incomplet_number), this);
        return;
    }
}
```

Figure 35 : Exemple de code permettant de traiter une commande vocale
Source personnelle

4.4.3 Fonctionnement des agents

Une fois les mesures introduites, elles doivent être traitées par les agents de MAGPIE. Ce chapitre présente l'implémentation de chaque agent et leur règle. Pour garantir plus de souplesse, nous avons développé les règles en Java, et non en Prolog. MAGPIE ne permet pas la modification des règles de manière dynamique en l'état actuel.

Chaque agent possède une règle qui est stockée dans la base de données. Cette règle va déterminer sous quelles conditions et, en fonction de quelles valeurs, l'agent doit déclencher une alerte. Les règles sont créées et insérées dans la base de données lorsque l'application est lancée pour la première fois. Des tests manuels ont également été effectués pour chaque agent afin de vérifier leur fonctionnement. Les tests sont effectués durant la même fenêtre de temps.

I Agent glucose

La règle concernant le taux de sucre dans le sang a été traduite par rapport à une règle Prolog existante. Le diagramme suivant illustre la règle appliquée par l'agent.

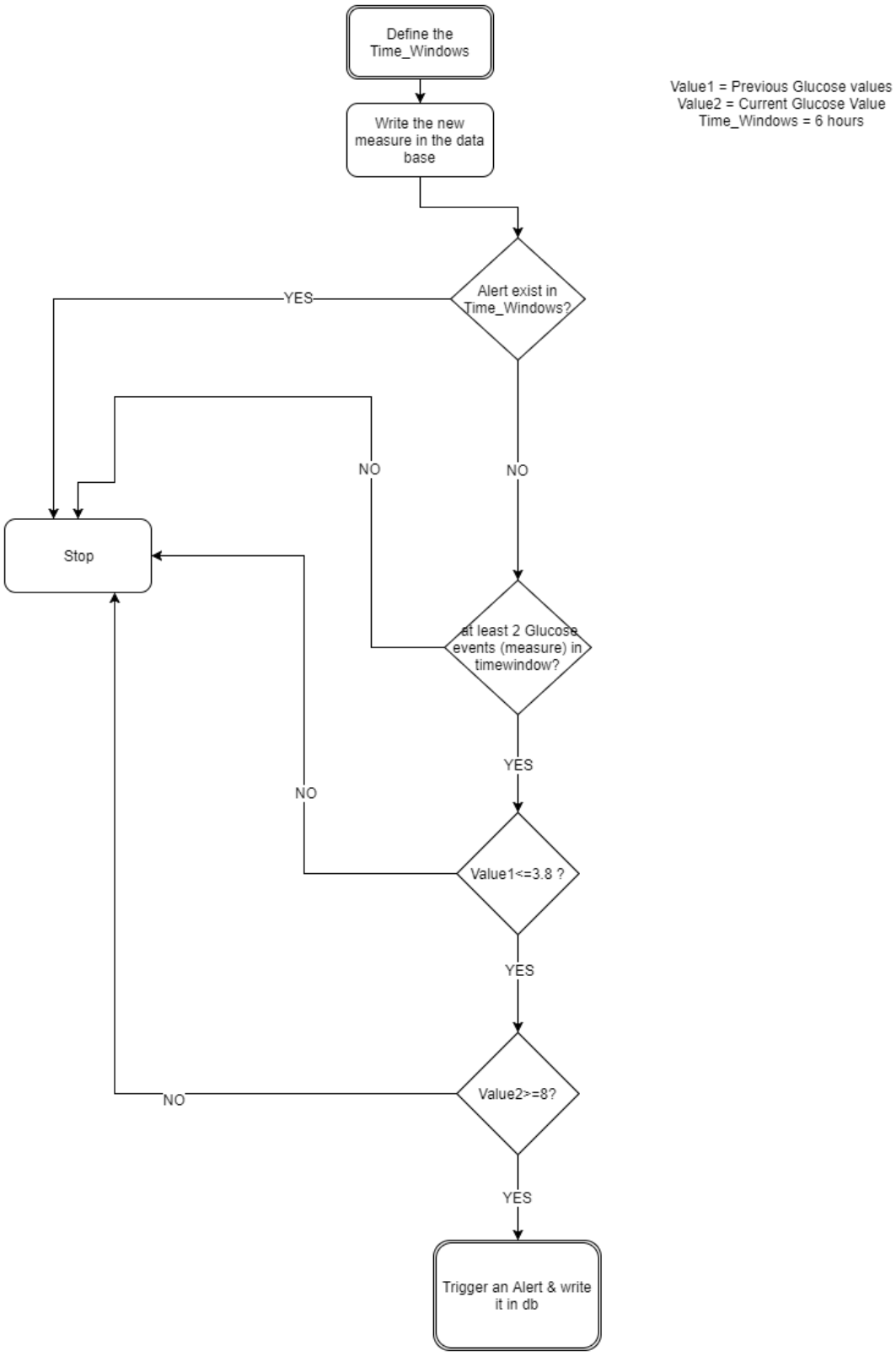


Figure 36 : Schéma de fonctionnement de l'agent Glucose
Source personnelle

L'agent va enregistrer la mesure dans la base de données. Ensuite, il va déclencher une alerte si les conditions suivantes sont vérifiées :

- Il n'existe pas d'alerte dans la fenêtre de temps définie (6 heures).
- Il doit y avoir au minimum deux mesures durant la fenêtre de temps dans la base de données.
- L'une des mesures précédentes doit être inférieure ou égale à 3,8.
- La valeur actuelle doit être supérieure ou égale à 8.

➢ **Test**

Voici les tests effectués pour l'agent glucose.

TEST 1	
Values	Status
2	no alert
9	alert triggered
10	no alert
1	no alert
25	no alert
21	no alert

TEST 2	
Values	Status
10	no alert
2	no alert
1	no alert
4	no alert
4	no alert
4	no alert
9	alert triggered

TEST 3	
Values	Status
3	no alert
20	alert triggered
3	no alert
20	no alert

TEST4	
Values	Status
1	no alert
2	no alert
3	no alert
1	no alert
4	no alert
10	alert triggered

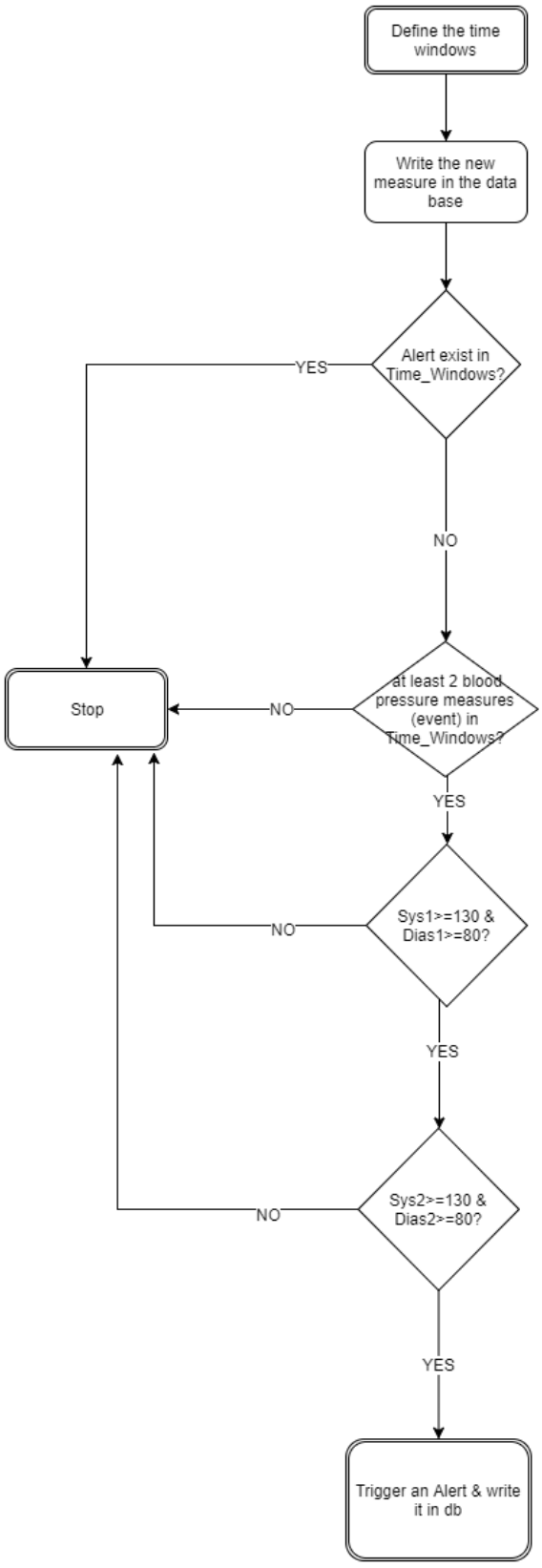
Figure 37 : Tests de l'agent Glucose
Source personnelle

II Agent pression sanguine

Cet agent sera en charge de traiter les mesures liées à la pression sanguine. Ces mesures sont composées de deux valeurs :

- La pression systolique
- La pression diastolique

La règle concernant la pression sanguine est basée sur une règle Prolog existante. L'illustration suivante nous montre les étapes exécutées par l'agent dès qu'une nouvelle mesure est ajoutée.



Time_Windows= 1 week
Sys1= Previous Systol Values
Dias1= Previous Diastol Values
Sys2= Current Systol value
Dias2 = Current Diastol Value

Figure 38 : Schéma de fonctionnement de l'agent Pression Sanguine
Source personnelle

Une fois la mesure introduite dans la base de données, l'agent va déclencher une alerte si toutes les conditions suivantes sont respectées :

- Il n'existe aucune alerte dans la base de données concernant la pression sanguine, durant un laps défini dans la fenêtre de temps (une semaine)
- Il existe au moins deux mesures dans la fenêtre de temps
- Il doit y avoir au moins deux mesures dont la pression systolique est supérieure à 130 et la pression diastolique est supérieure à 80.

➢ **Test**

La figure suivante illustre le résultat des tests effectués pour cet agent.

TEST 1		
Systol	Diastol	Status
140	90	no alert
140	90	alert triggered
140	90	no alert

TEST2		
Systol	Diastol	Status
120	70	no alert
120	70	no alert
120	70	no alert
125	80	no alert
140	90	no alert
140	90	alert triggered

TEST 3		
Systol	Diastol	Status
120	80	no alert
125	80	no alert
120	90	no alert
140	90	no alert
120	80	no alert
131	81	alert triggered
140	90	no alert

Figure 39 : Tests de l'agent Pression Sanguine
Source personnelle

III Agent poids

L'agent en charge de gérer les mesures de poids a un fonctionnement un peu particulier. Il ne va pas vérifier directement la valeur, mais la variation par rapport à la valeur précédente. Ainsi, les mesures de poids vont contenir deux valeurs : le poids et la variation. Le schéma suivant défini le processus exécuté par l'agent lorsqu'un nouveau poids est ajouté.

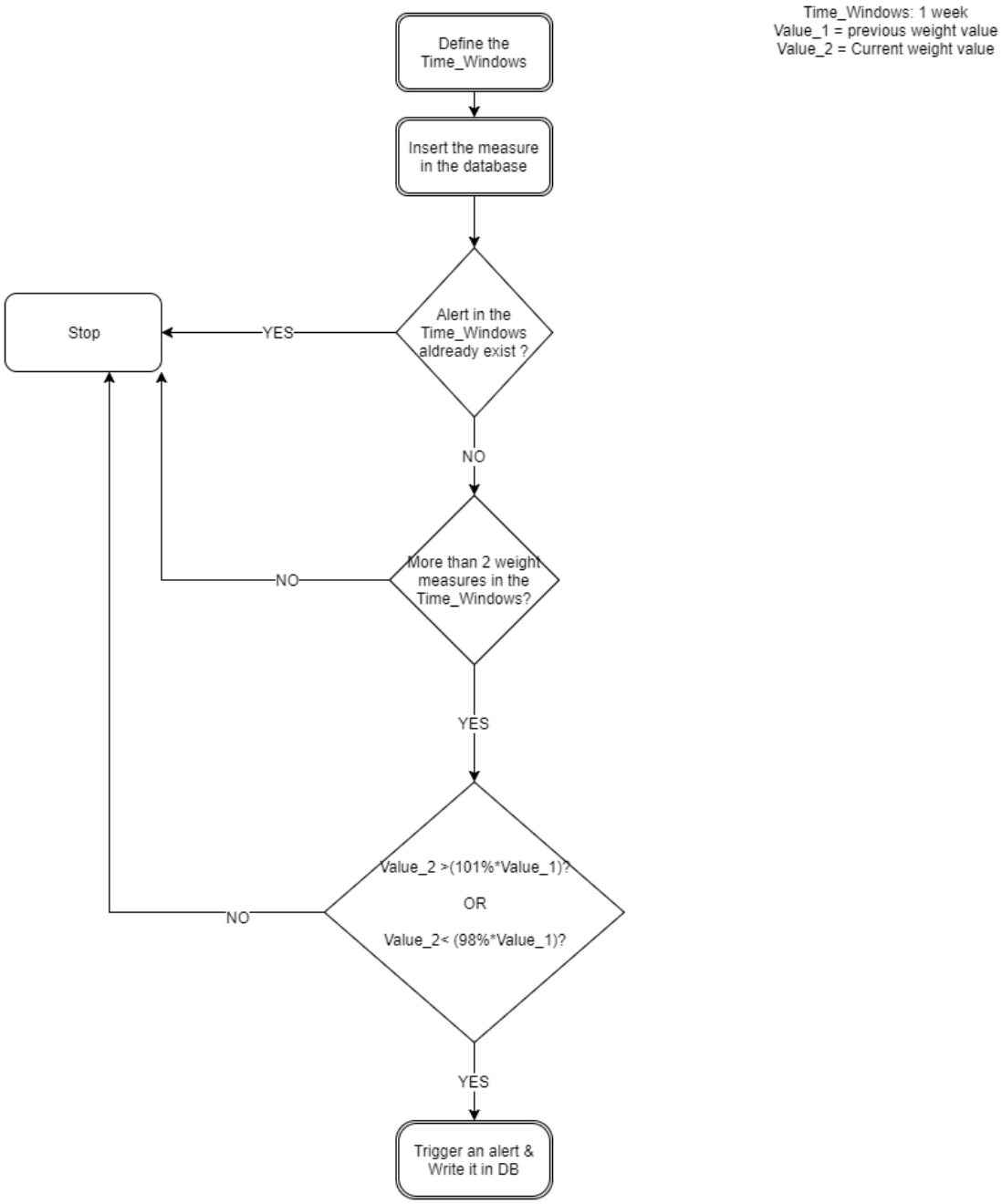


Figure 40 : Règle de l'agent Poids
Source personnelle

L'agent va ajouter le poids dans la base de données, avec la variation de poids par rapport à la mesure précédente. Ensuite, une alerte sera déclenchée si toutes les conditions suivantes sont remplies :

- Il n'existe pas d'alerte de poids dans la fenêtre de temps définie (une semaine).
- Il existe au moins deux mesures dans la fenêtre de temps.

- Il y a une augmentation de poids par rapport à la valeur précédente de plus de 1% ou une perte de poids par rapport à la valeur précédente de plus de 2%.

➢ **Test**

Pour vérifier si l'agent fonctionne correctement, les tests suivants ont été effectués :

TEST 1	
Values	Status
100	no alert
99	no alert
99,8	no alert
100,50	no alert
99	no alert
101	alert triggered
97	no alert

TEST2	
Values	Status
100	no alert
101,1	alert triggered
98	no alert
100	no alert

TEST3	
Values	no alert
100	no alert
100,5	no alert
99	no alert
97	alert triggered

Figure 41 : Tests de l'agent Poids
Source personnelle

IV Agent de pulsations

Il sera en charge de traiter les mesures acquises via le capteur de pulsations sanguine. Il va insérer la valeur dans la base de données. Ensuite, il va contrôler les valeurs via une règle. La règle relative aux pulsations est une règle simple, elle n'a pas été basée sur une règle Prolog. Elle pourra être modifiée directement par l'utilisateur, via l'interface graphique. Cette fonctionnalité est décrite en détail dans une prochaine section.

La règle est composée de deux bornes (valeur minimum et valeur maximum). Si la mesure est inférieure à la borne minimum ou supérieure à la borne maximum, une alerte sera déclenchée. Il n'y a pas de vérification par rapport au temps, étant donné qu'il s'agit d'une règle simple.

V Agent de pas

Chaque fois que le périphérique détecte un nouveau pas, il sera ajouté au compteur de pas qui est stocké dans l'application. Au début du jour suivant, l'agent va se charger d'enregistrer le nombre pas dans une nouvelle mesure. Il va ensuite, via une règle, définir s'il faut déclencher une alerte ou non.

Comme pour les pulsations, la règle pour les pas est une règle simple qui repose uniquement sur deux bornes (nombre minimum de pas effectués dans la journée et nombre maximum de pas). Si la valeur dépasse les bornes, l'agent va déclencher une alerte. Cette règle pourra également être modifiée par l'utilisateur.

4.4.4 Alertes

I Notification des alertes

Lorsque les agents émettent une alerte, une notification sera générée pour informer l'utilisateur. Elle comportera le nom de la catégorie ainsi qu'une description. De plus, la montre va émettre un son et une vibration.

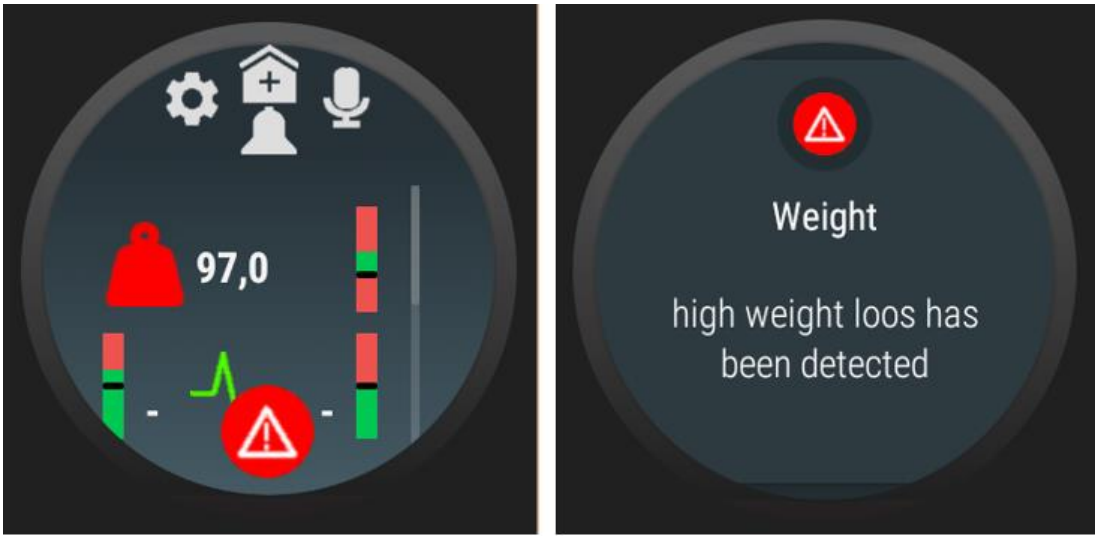


Figure 42 : Déclenchement d'une notification d'alerte
Source personnelle

II Affichage des alertes

L'utilisateur a la possibilité d'afficher la liste de toutes les alertes émises par le périphérique. Pour cela, il doit cliquer sur le bouton « Alertes ». Il pourra ensuite filtrer par catégorie. Chaque alerte affichera les informations suivantes :

- La catégorie de l'alerte, représentée par une icône.
- L'heure et la date de l'alerte.
- Une ou deux valeurs. Les catégories de poids et pression sanguine afficheront deux valeurs.

Figure 43 : Affichage des alertes
Source personnelle

4.4.5 Modification et affichage des règles

L'utilisateur peut afficher les règles liées à chaque agent. Pour cela, il doit cliquer sur le bouton « paramètres » comme illustré dans l'image suivante.

Figure 44 : Affichage des règles
Source personnelle

L'exemple suivant affiche la règle concernant la pression sanguine.

Figure 45 Affichage de la règle pression sanguine
Source personnelle

Les boutons situés en bas de l'écran permettent de changer de catégorie. La règle des pulsations et la règle des pas peuvent être modifiées via des zones de texte, comme l'illustre l'image suivante.

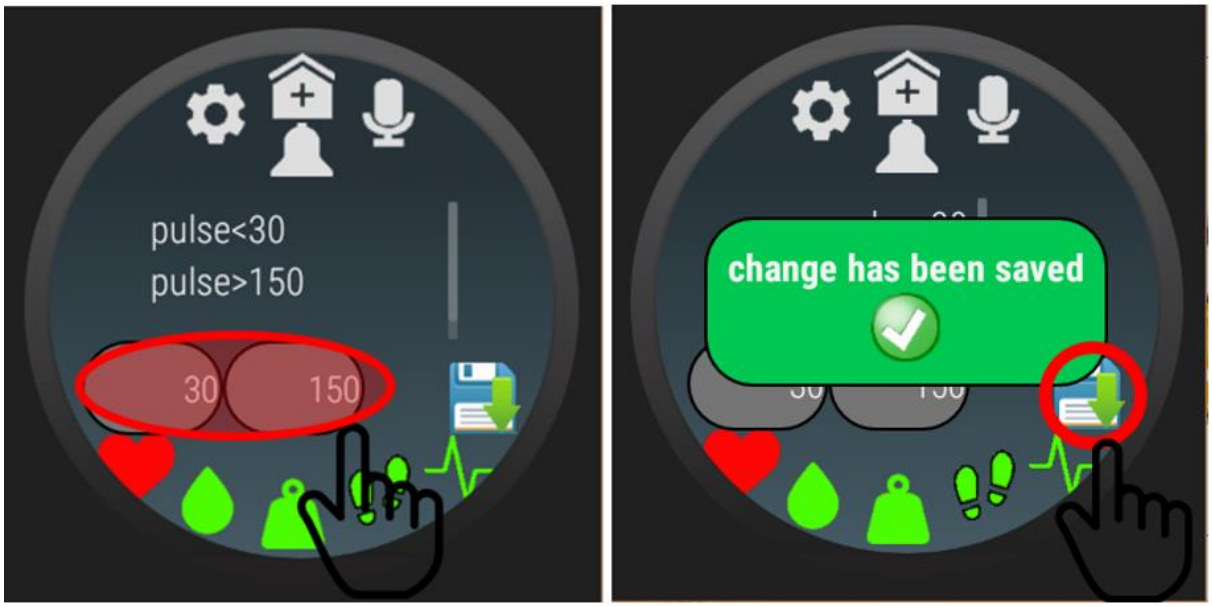


Figure 46 : Modification d'une règle
Source personnelle

Une fois la règle modifiée, elle doit être sauvegardée à l'aide du bouton situé à droite de l'écran.

4.4.6 Affichage des statuts de l'utilisateur

Les barres de statut sont des indicateurs permettant à l'utilisateur de situer son état de santé. Elles contiennent un curseur qui permet de positionner l'utilisateur. Elles sont placées à côté de chaque icône, dans le menu général.

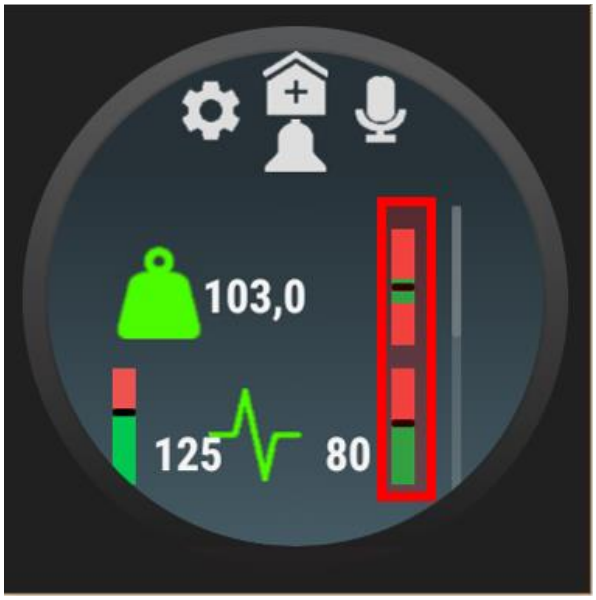


Figure 47 : Barres de statut
Source personnelle

I Fonctionnement

Si le curseur se trouve dans la zone verte, cela indique que la dernière mesure est dans les bornes définies au sein des règles. Si le curseur se trouve dans la zone rouge, cela signifie que la dernière mesure a dépassé les bornes.

La taille des zones est générée dynamiquement en fonction des règles. Si une règle est modifiée par l'utilisateur, le programme adaptera la taille de chaque zone automatiquement. L'exemple suivant illustre l'ajustement de la barre de statut.

Voici les valeurs de la règle des pulsations :

Figure 48 : Ajustement de la barre de statut - règle initiale
Source personnelle

La barre de statut concernant les pulsations sera la suivante:

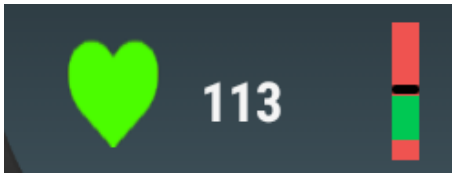


Figure 49 : Ajustement de la barre de statut - barre de statut initiale
Source personnelle

Nous allons maintenant modifier la règle :

Figure 50 : Ajustement de la barre de statut - règle modifiée
Source personnelle

La barre de statut est modifiée et s'adapte en fonction des nouvelles valeurs, comme illustré dans l'image suivante.

Figure 51 : Ajustement de la barre de statut - barre de statut modifiée
Source personnelle

II Calcul des zones

Les zones à l'intérieur des barres de statut sont calculées en fonction de chaque règle. Chaque barre de statut est spécifique à une catégorie.

Poids : Calculé par rapport à la variation. Les zones rouges correspondent à une variation de poids trop importante. Les valeurs vont de -10% (perte de poids de 10%) à +10% (gain de poids de 10%).

Figure 52 : Barre de statut de poids
Source personnelle

Pression sanguine : La zone rouge correspond à une pression sanguine trop élevée. La pression systolique et la pression diastolique ont été traitées séparément. Les valeurs vont de 0 à 200 pour la pression systolique (à gauche) et 0 à 150 pour la pression diastolique (à droite)

Figure 53 : Barre de statut de pression sanguine
Source personnelle

Compteur de pas : les zones rouges correspondent aux nombres de pas qui sortent des bornes (minimum et maximum) définies par l'utilisateur. Les valeurs vont de 0 à 20'000.

Figure 54 : Barre de statut de pas
Source personnelle

Pulsations : les zones rouges correspondent aux nombres de pulsations par minute qui sortent des bornes (minimum et maximum) définies par l'utilisateur. La valeur va de 0 à 220.

Figure 55 : Barre de statut de pulsations
Source personnelle

Glucose : la zone verte correspond à un taux de sucre dans le sang qui se situe entre 3,8 et 8 mmol/l. La valeur va de 0 à 20.

Figure 56 : Barre de statut de glucose
Source personnelle

4.4.7 Gestion de la base de données

L'écran initial nous offre plusieurs options permettant de gérer l'espace de stockage de la montre. L'image ci-dessous illustre les fonctionnalités de gestion de la base de données.

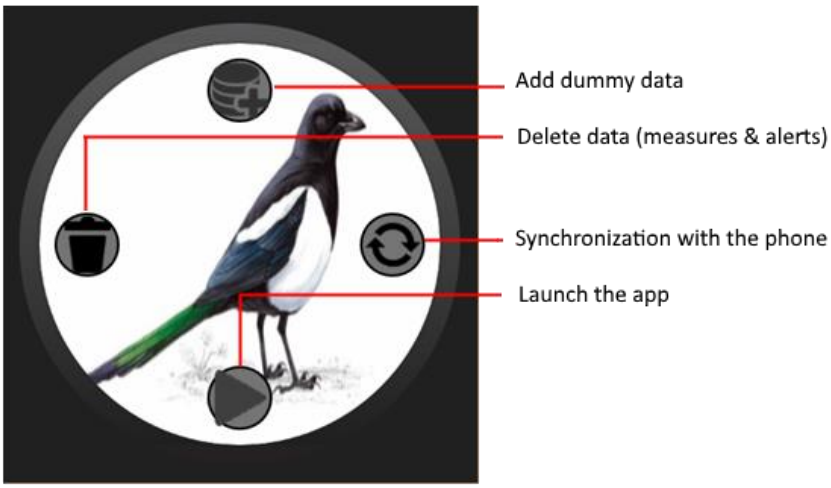


Figure 57 : Gestion de l'espace de stockage
Source personnelle

I Ajout de données factices

Permet d'ajouter des données factices. Cette fonction est utilisée à des fins de test et de démonstration. L'insertion des données comprend l'ajout de mesures et l'ajout d'alertes.

II Suppression des données

Permet de libérer de l'espace mémoire. Les mesures et les alertes seront supprimées. Il est conseillé d'effectuer une synchronisation avec le téléphone avant de procéder à la suppression des données.

III Synchronisation avec le téléphone

Permet de transmettre l'ensemble des données au téléphone. Cette fonctionnalité est expliquée en détail dans le chapitre suivant.

4.5 Développement des fonctionnalités sur le téléphone

Il s'agit d'une nouvelle application spécifique aux Smartphones. Cette application va utiliser les données provenant de la montre, via un processus de synchronisation.

4.5.1 Structure et architecture

L'architecture de l'application Smartphone est identique à l'application développée sur la montre. Elle est composée des mêmes tables de la base de données, du même ORM ainsi que de la même organisation des fragments et des activités. La figure suivante illustre l'interface utilisateur.

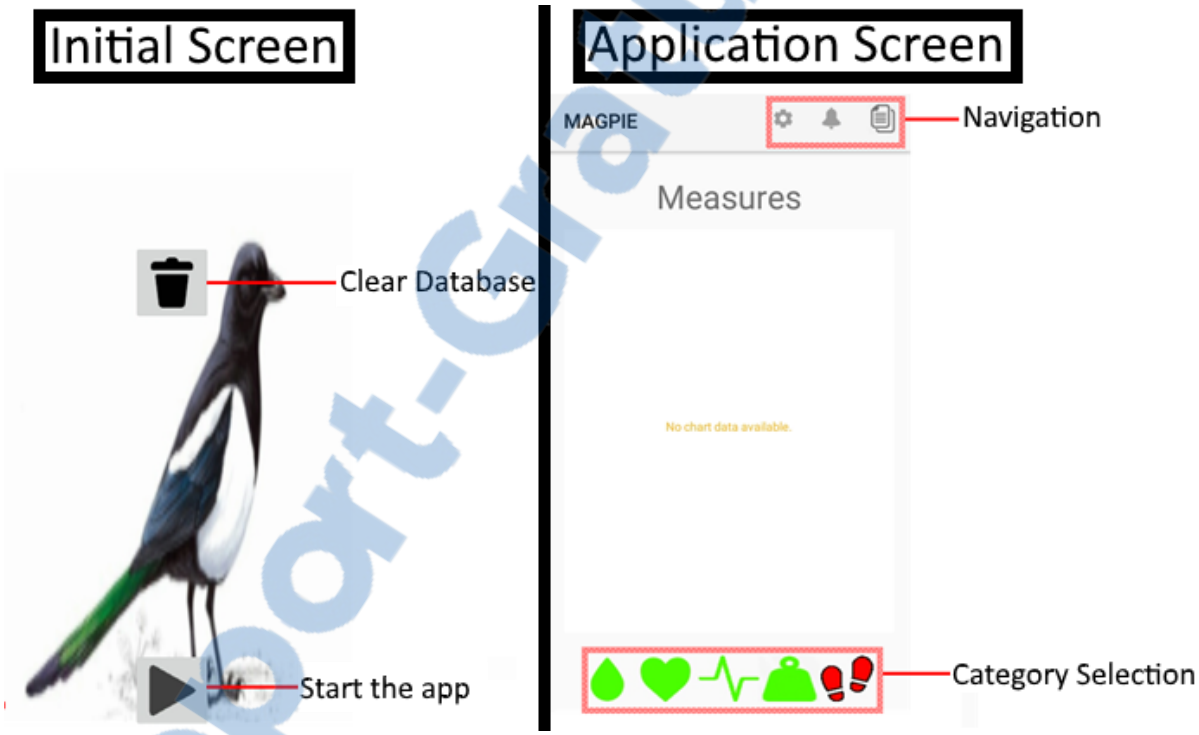


Figure 58 : Activités de l'application sur Smartphone
Source personnelle

La première activité affiche une option permettant de supprimer toutes les mesures et les alertes de la base de données. La seconde activité permet de visualiser les données.

4.5.2 Synchronisation entre la montre et le téléphone

Durant ce chapitre, nous allons expliquer et détailler le processus de synchronisation. La synchronisation permet de transférer l'ensemble des données (règles, mesures et alertes) de la montre vers le téléphone. Les données sont transmises en utilisant l'*API* Data Layer, décrite plus haut dans ce document.

I Transmission des données

La transmission des données se fait en pressant le bouton « synchronisation » de la montre.

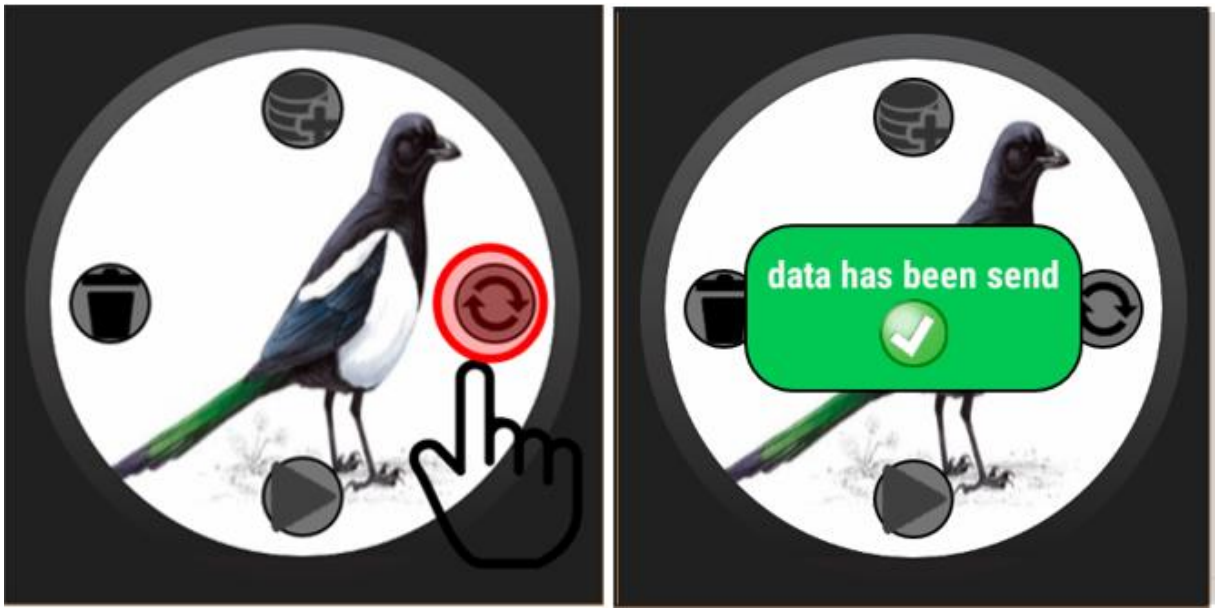


Figure 59 : Synchronisation depuis la montre
Source personnelle

Afin de ne pas ralentir l'exécution de l'application, les données sont transmises en utilisant un *thread*. La figure suivante illustre le processus d'envoi des données par la montre.

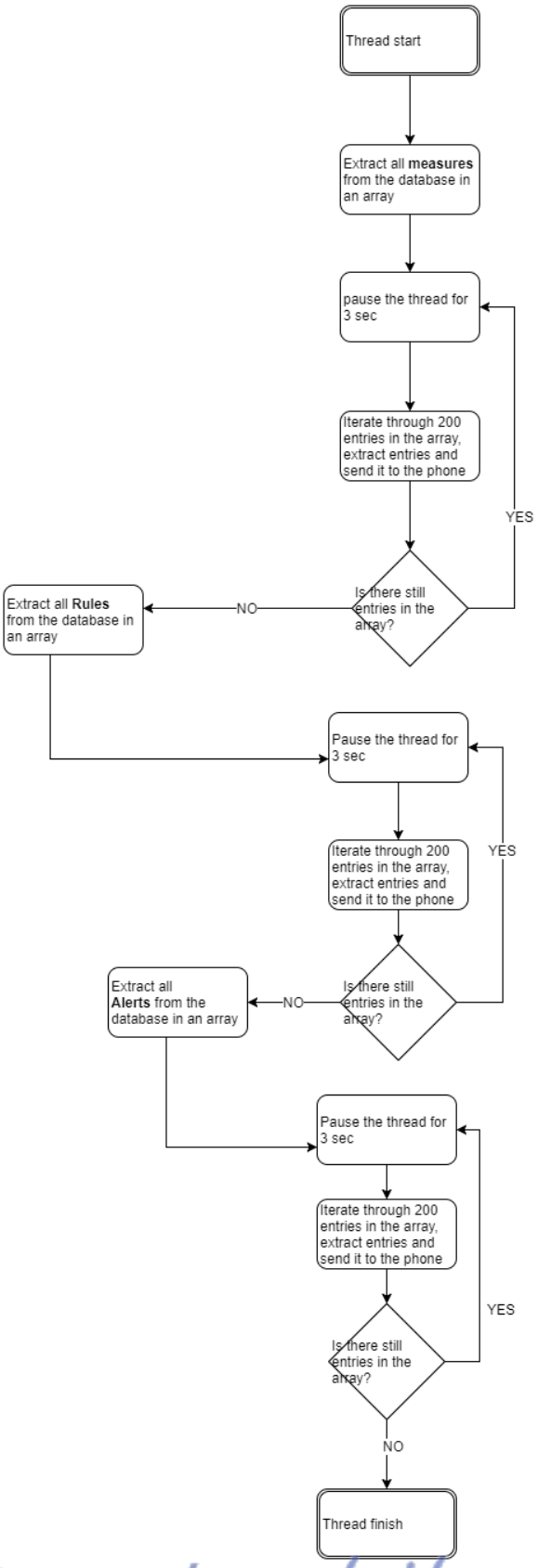


Figure 60 : Illustration du processus de synchronisation
Source personnelle

Comme les données peuvent être conséquentes, elles sont transmises par paquet de 200. Après chaque paquet transmis, le *thread* est stoppé pendant trois secondes pour permettre au smartphone de traiter et d'enregistrer les données dans sa base de données. Ce même fonctionnement est utilisé pour traiter les mesures, les règles et les alertes.

Il convient de noter que le téléphone ne va pas transmettre un accusé de réception à la montre. Ainsi, lorsque les données sont transmises, la montre ne sait pas si le récepteur (dans notre cas le téléphone) va correctement recevoir les données.

II Réception les données

La réception et l'enregistrement des données sont gérées à l'aide de *service*, composé d'un *listener*. Les *services* ne sont pas soumis au cycle de vie des activités. Ils seront donc actifs en permanence, dès le lancement de l'application.

Les services qui s'occupent de la réception des données sont les suivants :

- Listener_data_alert : traite et sauvegarde les alertes reçues par la synchronisation.
- Listener_data_measure : traite et sauvegarde les mesures reçues par la synchronisation.
- Listener_data_rule : traite et sauvegarde les règles reçues par la synchronisation.

Le code suivant est utilisé dans la *class* Listener_data_alert. Il permet de réceptionner les données et les enregistre dans la base de données.

```
  public void onDataChanged(DataEventBuffer dataEvents) {
      DataMap dataMap;
      for (DataEvent event : dataEvents) {
          // Check the data type
          if (event.getType() == DataEvent.TYPE_CHANGED) {
              // Check the data path
              String path = event.getDataItem().getUri().getPath();
              dataMap = DataMapItem.fromDataItem(event.getDataItem()).getDataMap();
              //get the data array list
              ArrayList<DataMap>
containerList=dataMap.getDataMapArrayList(Const.KEY_MEASURE_DATA);

              //extract data and save each value
              for (DataMap aData : containerList)
              {
                  Alertes alerte= extractAlertFromDatamap(aData);
                  AlertesRepository.getINSTANCE().insertOrReplace(alerte);
              }

          }
      }
  }
```

Figure 61 : Extrait de code du Listener_data_alert
Source personnelle

Lorsque le téléphone a synchronisé des données, il affichera un message notifiant l'utilisateur. L'image suivante illustre la notification de réception de données du téléphone.

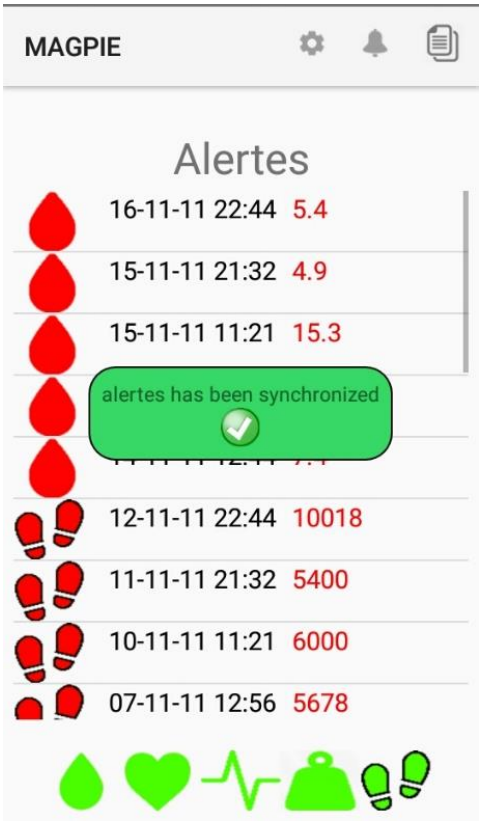


Figure 62 : Notification de la synchronisation sur le téléphone
Source personnelle

III Datamap

Lors de synchronisation, les données sont transmises sous forme de Datamap. Il s'agit en quelque sorte d'un paquet qui permet d'encapsuler différents objets. Une Datamap peut contenir les éléments suivants :

- Des données primitives (int, float, etc…).
- Des listes ou des tableaux de types primitifs.
- Des Datamap.
- Des listes de Datamap.

Lorsque l'on insère des données dans une Datamap, elles contiendront une clé. Cette clé sera ensuite utilisée pour extraire les données.

IV Test

Les images suivantes illustrent le résultat du test de synchronisation effectué. Nous pouvons constater que la taille des paquets est identique des deux côtés. Cela signifie qu'il n'y a pas eu de perte durant la synchronisation.

```
07-08 14:32:49.806 2272-2272/hevs.aislab.magpie.watch D/size_of_data_send: 200
07-08 14:32:52.863 2272-2272/hevs.aislab.magpie.watch D/size_of_data_send: 200
07-08 14:32:55.897 2272-2272/hevs.aislab.magpie.watch D/size_of_data_send: 200
07-08 14:32:58.909 2272-2272/hevs.aislab.magpie.watch D/size_of_data_send: 43
```

Figure 63 : Affichage du test des données transmises
Source personnelle

```
07-08 14:32:51.345 6007-8613/hevs.aislab.magpie.watch D/size_of_data_recive: 200
07-08 14:32:54.205 6007-8614/hevs.aislab.magpie.watch D/size_of_data_recive: 200
07-08 14:32:57.065 6007-8615/hevs.aislab.magpie.watch D/size_of_data_recive: 200
07-08 14:32:59.745 6007-8616/hevs.aislab.magpie.watch D/size_of_data_recive: 43
```

Figure 64 : Affichage du test des données réceptionnées
Source personnelle

4.5.3 Affichage des mesures

Les mesures sont affichées sous forme de diagramme. Elles permettent à l'utilisateur de voir son évolution au fil du temps.

I MPAndroidChart

Il existe plusieurs librairies Android permettant de dessiner des schémas. Nous avons opté pour la librairie MPAndroidChart disponible sur GitHub. Il s'agit de la plus connue. Elle propose huit types de graphiques différents. (Jahoda, 25 avril 2014/2017).

II Affichage des graphiques

Pour afficher les mesures, l'utilisateur doit cliquer sur le bouton « mesure ». Il pourra ensuite sélectionner la catégorie qu'il souhaite consulter. La figure suivante illustre les données affichées lorsque l'utilisateur clique sur la catégorie « pulsations ».

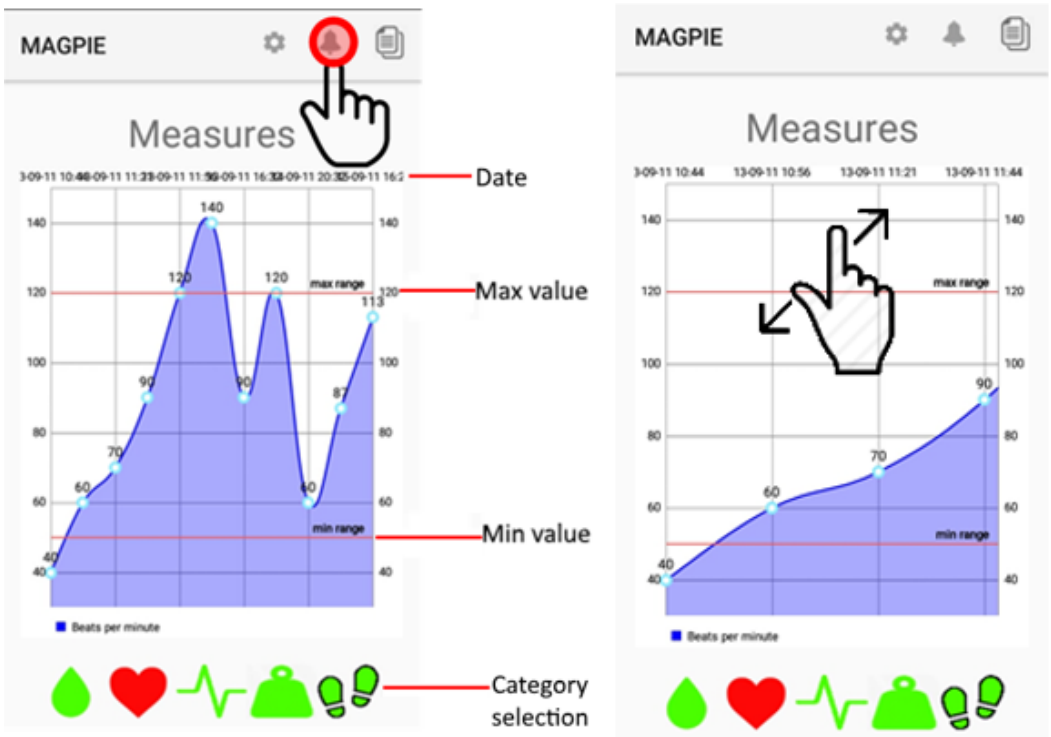


Figure 65 : Affichage des mesures
Source personnelle

Le diagramme affiche les informations suivantes :

- La date et l'heure de la mesure.
- La valeur de chaque mesure.
- Les bornes limites, générées en fonction des règles.

4.5.4 Affichage des alertes

A l'instar de la montre, l'application sur le téléphone nous offre également la possibilité d'afficher la liste des alertes. L'image suivante illustre la fonctionnalité.

Figure 66 : Affichage des alertes
Source personnelle

L'option permettant de filtrer les alertes par catégorie est également présente.

4.5.5 Affichage des règles

Le programme Smartphone propose également l'affichage et la modification des règles. A chaque modification, la règle sera synchronisée automatiquement sur la montre.

Le mécanisme utilisé pour la synchronisation du téléphone vers la montre est identique à celui décrit précédemment. Il se fait à l'aide de Datamap. S'agissant d'un seul objet à transmettre, il n'a pas été nécessaire d'utiliser un *thread*. L'image suivante nous montre l'affichage et la modification des règles sur le téléphone.

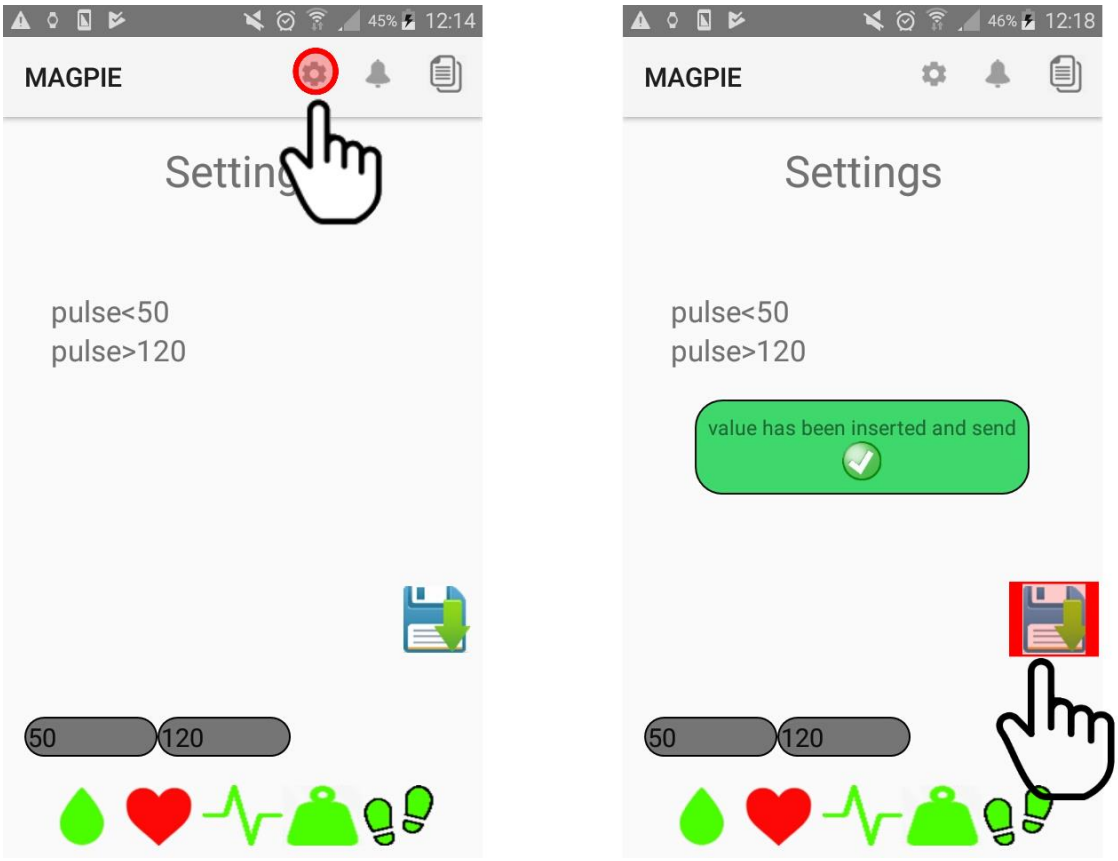


Figure 67 : Affichage et modification d'une règle
Source personnelle

Si la montre et le téléphone sont tous deux connectés à internet, la règle modifiée sur le téléphone sera également mise à jour sur la montre, comme le montre la figure ci-dessous.

Figure 68 : Synchronisation d'une règle
Source personnelle

5 Défis rencontrés

Lors du développement du prototype, nous avons rencontré plusieurs difficultés.

5.1 Panne de la montre

Le 27 juin 2017, la montre est tombée en panne. Elle ne pouvait plus être chargée et de ce fait, il était impossible de l'allumer. Nous avons donc dû utiliser exclusivement l'émulateur pour développer notre application. Par conséquent, Il n'a pas été possible de tester certaines fonctionnalités, comme par exemple les capteurs de pas et de pulsations, ainsi que le rendu de l'interface graphique.

5.2 Ecran de la montre

La taille de l'écran est très limitée sur la montre. Il a été préférable de ne pas afficher trop d'informations sur la même fenêtre dans le but d'éviter la confusion de l'utilisateur. Pour cette raison, l'affichage d'éléments complexes, comme par exemple le diagramme de mesures, a été délégué au smartphone.

L'utilisation de fragments carrés avec un écran rond a apporté quelques surprises. A chaque modification de l'interface utilisateur, il a fallu relancer l'émulateur et vérifier que le résultat soit conforme aux exigences.

6 Conclusion

6.1 Amélioration

Nous avons développé l'ensemble des fonctionnalités définies à l'intérieur du *Product Backlog*. Par manque de temps, certaines fonctions n'ont pas été optimisées. Les points suivants décrivent les améliorations qui pourraient être effectuées au sein de l'application dans un développement futur.

6.1.1 Synchronisation

A l'heure actuelle, la synchronisation entre la montre et le téléphone se fait manuellement via un bouton. Il serait intéressant de développer un système de synchronisation automatique. Ainsi, les données seraient synchronisées dynamiquement.

6.1.2 Création de vues responsives sur le téléphone

Par manque de temps, l'interface graphique de l'application smartphone n'a pas été adaptée aux différentes tailles d'écran. L'ajustement des dimensions permettrait de proposer une expérience optimale sur tout type de périphérique Android (Smartphone et tablette).

6.2 Conclusion du projet

Durant ce travail, nous avons pu observer les fonctionnalités et les capacités offertes par le système d'exploitation Android. La phase de recherche nous a permis d'identifier les spécificités et les limitations du système d'exploitation Android Wear 2.0.

La phase de développement nous a permis de distinguer les nombreux outils et librairies proposés aux développeurs. Par exemple, l'accès aux capteurs et la transmission des données ont été grandement facilités par la présence d'*API*. L'application développée nous a permis d'évaluer le potentiel de ces nouveaux périphériques en termes d'utilisation et de développement logiciel.

Il apparait évident que les montres Android Wear offrent de nombreuses fonctionnalités dans le domaine de la santé et de la mesure de soi. Véritables compagnons autonomes, ces montres permettent de guider l'utilisateur et de l'informer sur son statut actuel. Elles sont un outil idéal pour les sportifs, les personnes âgées ou les personnes atteintes de certaines maladies comme par exemple le diabète.

6.3 Conclusion personnelle

Véritable challenge, l'implémentation de l'application m'a permis d'améliorer mes compétences en programmation Android et en Java. Le prototype a demandé une attention toute particulière de par la nouveauté des *API* utilisées, notamment l'*API* Data Layer et l'accès aux capteurs, étant donné qu'il s'agit de fonctionnalités propres à Android Wear.

Ce travail conclut trois années d'études auprès de la HES-SO. De la conception à l'implémentation d'un projet, il m'a permis de mettre en pratique toutes les connaissances que j'ai pu acquérir durant ma formation.

7 Références

1) Add C and C++ Code to Your Project | Android Studio. (s. d.). Consulté 13 juillet 2017, à l'adresse https://developer.android.com/studio/projects/add-native-code.html
2) Android – Android Pay. (s. d.). Consulté 14 juillet 2017, à l'adresse https://www.android.com/intl/fr_ca/pay/
3) Android Open Source Project. (s. d.). Consulté 4 mai 2017, à l'adresse https://source.android.com/
4) Android Wear. (2017, avril 30). In *Wikipedia*. Consulté à l'adresse https://en.wikipedia.org/w/index.php?title=Android_Wear&oldid=778063603
5) Android Wear. (s. d.-a). Consulté 14 juillet 2017, à l'adresse https://www.android.com/wear/
6) Android Wear. (s. d.-b). Consulté 2 mai 2017, à l'adresse https://www.android.com/wear/
7) Android Wear 2.0: Ultimate guide to the major smartwatch update. (s. d.). Consulté 14 juillet 2017, à l'adresse https://www.wareable.com/android-wear/android-wear-update-everything-you-need-to-know-2735
8) Creating Wearable Apps | Android Developers. (s. d.). Consulté 26 mai 2017, à l'adresse https://developer.android.com/training/wearables/apps/index.html
9) Dictionnaire des développeurs. (s. d.). Consulté 17 juillet 2017, à l'adresse http://dico.developpez.com/html/3091-Conception-ORM-Object-Relational-Mapping.php
10) Download Android Studio and SDK Tools | Android Studio. (s. d.). Consulté 13 juillet 2017, à l'adresse https://developer.android.com/studio/index.html
11) Fragments | Android Developers. (s. d.). Consulté 18 juillet 2017, à l'adresse https://developer.android.com/guide/components/fragments.html
12) Frumusanu, A. (s. d.). A Closer Look at Android RunTime (ART) in Android L. Consulté 13 juillet 2017, à l'adresse http://www.anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l
13) Getting Started with the NDK | Android Developers. (s. d.). Consulté 14 juillet 2017, à l'adresse https://developer.android.com/ndk/guides/index.html

14) Git - About Version Control. (s. d.). Consulté 17 juillet 2017, à l'adresse https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

15) Goodrich, R., 1, L. C. | O., & ET, 2013 07:07pm. (s. d.). Accelerometers: What They Are & How They Work. Consulté 2 mai 2017, à l'adresse http://www.livescience.com/40102-accelerometers.html

16) greenDAO Features. (s. d.). Consulté 6 juin 2017, à l'adresse http://greenrobot.org/greendao/features/

17) I want to develop Android Apps – What languages should I learn? | AndroidAuthority. (s. d.). Consulté 14 juillet 2017, à l'adresse http://www.androidauthority.com/want-develop-android-apps-languages-learn-391008/

18) Introduction to Android Wear - Xamarin. (s. d.). Consulté 13 juillet 2017, à l'adresse https://developer.xamarin.com/guides/android/wear/intro-to-wear/

19) ISO/IEC 2382:2015(en), Information technology — Vocabulary. (s. d.). Consulté 23 juillet 2017, à l'adresse https://www.iso.org/obp/ui/fr/#iso:std:iso-iec:2382:ed-1:v1:en

20) Jahoda, P. (2017). *MPAndroidChart: A powerful Android chart view / graph view library, supporting line- bar- pie- radar- bubble- and candlestick charts as well as scaling, dragging and animations*. Java. Consulté à l'adresse https://github.com/PhilJay/MPAndroidChart (Original work published 25 avril 2014)

21) JSON. (s. d.). Consulté 23 juillet 2017, à l'adresse http://www.json.org/json-fr.html

22) Kotlin and Android | Android Developers. (s. d.). Consulté 13 juillet 2017, à l'adresse https://developer.android.com/kotlin/index.html

23) Le grand dictionnaire terminologique. (s. d.). Consulté 20 juillet 2017, à l'adresse http://www.gdt.oqlf.gouv.qc.ca/ficheOqlf.aspx?Id_Fiche=8872480

24) Maximiliano, F. (2015, octobre 12). Getting Started With Wearables: How To Plan, Build And Design – Smashing Magazine. Consulté 23 juillet 2017, à l'adresse https://www.smashingmagazine.com/2015/10/getting-started-wearables-plan-build-design/

25) Morrill, D. (s. d.). Announcing the Android 1.0 SDK, release 1. Consulté 4 mai 2017, à l'adresse https://android-developers.googleblog.com/2008/09/announcing-android-10-sdk-release-1.html

26) Network Access and Syncing | Android Developers. (s. d.). Consulté 11 mai 2017, à l'adresse https://developer.android.com/training/wearables/data-layer/network-access.html

27) pastilleadmin. (2016, décembre 13). How to make Unity games that run on Android Wear. Consulté 14 juillet 2017, à l'adresse http://pastille.se/2016/12/13/make-unity-games-run-android-wear/

28) Présentation - Apache Cordova. (s. d.). Consulté 13 juillet 2017, à l'adresse https://cordova.apache.org/docs/fr/latest/guide/overview/index.html

29) Saving Data in SQL Databases | Android Developers. (s. d.). Consulté 17 juillet 2017, à l'adresse https://developer.android.com/training/basics/data-storage/databases.html

30) Sensors - Mobile terms glossary - GSMArena.com. (s. d.). Consulté 2 mai 2017, à l'adresse http://www.gsmarena.com/glossary.php3?term=sensors

31) Sensors Overview | Android Developers. (s. d.). Consulté 3 mai 2017, à l'adresse https://developer.android.com/guide/topics/sensors/sensors_overview.html

32) Smartwatch. (2017, mai 4). In *Wikipédia*. Consulté à l'adresse https://fr.wikipedia.org/w/index.php?title=Smartwatch&oldid=137047894

33) Society, N. G., & Society, N. G. (2014, juin 19). barometer. Consulté 3 mai 2017, à l'adresse http://www.nationalgeographic.org/encyclopedia/barometer/

34) Standalone Apps | Android Developers. (s. d.). Consulté 30 mai 2017, à l'adresse https://developer.android.com/training/wearables/apps/standalone-apps.html

35) The JVM Architecture Explained - DZone Java. (s. d.). Consulté 13 juillet 2017, à l'adresse https://dzone.com/articles/jvm-architecture-explained

36) Unity (moteur de jeu). (2017, juillet 7). In *Wikipédia*. Consulté à l'adresse https://fr.wikipedia.org/w/index.php?title=Unity_(moteur_de_jeu)&oldid=138772657

37) Using Meteor to develop for Android Wear - CodeProject. (s. d.). Consulté 13 juillet 2017, à l'adresse https://www.codeproject.com/Articles/1035846/Using-Meteor-to-develop-for-Android-Wear

38) vincent@nextinpact.com. (2015, novembre 18). Xamarin 4.0 veut encore simplifier le développement pour Android et iOS. Consulté 13 juillet 2017, à l'adresse https://www.nextinpact.com/news/97373-xamarin-4-0-veut-encore-simplifier-developpement-pour-android-et-ios.htm

39) Webster, L. (2016, septembre 13). Développement d'applications mobiles. Consulté 14 juillet 2017, à l'adresse https://www.visualstudio.com/fr/vs/mobile-app-development/

40) What is Prolog? Webopedia Definition. (s. d.). Consulté 16 juillet 2017, à l'adresse http://www.webopedia.com/TERM/P/Prolog.html

Annexes

Annexe I Product Backlog

N.	Theme	User Stories			Priority	Story Points	Acceptance Criteria	Sprint	Stat
		As a…	I want to…	So that…					
1	Watch selection	developer	Know the characteristic of smartwatches	Define the pertinent characteristics	1	7	create a criteria list	1	3
2	Watch selection	developer	Get the list of the watch available with their characteristics	Create a benchmark with the specificity of each watch	1	7	Benchmark is created	1	3
3	Watch selection	developer	Choose the watch according to the benchmark	Being able to create the prototype	1	1	The watch is selected	1	3
4	Programming Android Wear	developer	Analyze the sensor usage and the accessibility	Understand the sensor usage	1	10	Create a simple application with sensors	2	3

5	Programming Android Wear	developer	Analyze the standalone capacity of the watch	Determine if the watch need to be paired with a phone to be able to connect though internet	1	2		2	3
6	Programming Android Wear	developer	Analyze and understand the communication system between the wear and the phone	Being able to make the phone and the watch communicate	1	20	Create a simple application that will communicate between the watch and the phone	2	3
7	MAGPIE	developer	Understand the framework MAGPIE	Being able to use it in an app	1	25	Know how to create agent and interact with it	2	3
8	Prototype	developer	Define a scenario	Create the prototype	1	2	The scenario has been defined and the user stories are created	3	3
9	Prototype	developer	Create a repository a folder on GITHUB	To be able to share and save my code	1	2	The folder has been created	3	3
10	Prototype	developer	Create the UML schema of the database	Create the structure of the apps	1	10	UML is created	3	3
11	Prototype	User - Watch	Measure my pulse	Apply the rules on my pulse level	1	2	Pulse are shown on the screen	3	3

12	Prototype	User - Watch	Measure my step	Apply the rules on my step	1	2	Step is shown	3	3
13	Prototype	User - Watch	Set my actual glucose level, either by voice or by hand	Apply the rules on my glucose level	1	10	Glucose level is shown	4	3
14	Prototype	User - Watch	Set my actual blood pressure measures, either by voice or by hand	Apply the rules on my blood pressure measures	1	10	Blood pressure level is shown	4	3
15	Prototype	User - Watch	set my actual weight either by voice or by hand	Apply the rules on my weight	1	10	Weight is shown	4	3
16	Prototype	User - Watch	Create the rules for the glucose according to the prolog rules established	Know if my glucose value is bad	1	5	Glucose agent trigger an alert depending on the value	4	23
17	Prototype	Developer	Create the rules for the blood pressure according to the prolog rules established	Know if my blood pressure status value is bad	1	5	Blood pressure agent trigger an alert depending on the value	4	3
18	Prototype	User - Watch	Create the rules for the weight according to the prolog rules established	Know if my weight status is bad	1	5	Weight agent trigger an alert depending on the value	5	3
19	Prototype	User - Watch	Create the rules for the blood and steps	Know if my current status is bad	3	15	Steps and pulse agent trigger an	5	3

							alert depending on the value		
20	Prototype	User - Watch	Show historic about my previous alert	Being able to display my alert	2	15	All alerts are displayed, we can can display alert by category	6	3
21	Prototype	User - Watch	Get my current status based on color for each rules & including a progress bar	Know my current state	2	15	Current status is displayed	6	3
22	Prototype	User - Watch	Being able to show all the rules and update the current pulse, step	To adapt the rules to my current status	2	20	Color and progress bar change based on the rules	6	3
23	Prototype	Developer	Have a nice graphical interface	Update the current GUI and make it prettier	3	10	Interface is prettier	7	3
24	Prototype	Developer	validate the entry of the user	Disallow the user to save false data	2	2	User cannot enter bad data	7	3
25	Prototype	User - Phone	Synchronize the data from the from the watch to the phone	Free space on the watch	2	15	User can push data from the watch to the phone	7	3
26	Prototype	User - Phone	Display all Alerts based on category	user can see the alert triggered by the watch	2	2		7	3

27	Prototype	User - Phone	Display all the measures based on the category	User can see a graph of his measures	2	15		7	3
28	Prototype	User - Phone	Display and update rules from the phone to the watch	To show the bidirectional communication	2	2		7	3
29	Prototype	Developer	Create dummy data for the watch	Be able to show data immediately	1	3		7	3

Déclaration de l'auteur

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

- Monsieur Michael Schumacher
- Monsieur Davide Calvaires
- Monsieur Albert Brugués