Chapitre

Ouai

Arquillière Mathieu - Zangla Jérémy

28 février 2020

Table des matières

Table des figures

1 Lexique

GPS : *Global Positioning System*, système de géolocalisation mondiale par satellite.

Smartphone : *Téléphone intelligent*, un téléphone mobile doté de fonctionnalités similaires à un ordinateur (internet, vidéos, musiques, jeux) et de capteurs (gyroscope, GPS).

Application mobile : Logiciel développé pour des supports semblables à des smartphones.

Serveur : Service décentralisé permettant à des clients l'accès à des services ou des données.

Base de données : Ensemble d'informations structurées accessible via un service.

Java : Langage informatique.

Kotlin : Langage informatique.

XML : Langage informatique de structuration de données par balises.

Android : Système d'exploitation créé par *Google*, le plus souvent déployé sur des smartphones.

Processus (informatique) : Programme en cours d'exécution sur un système.

Activité : Composante métier d'une application android.

Fragment : Activité liée à seulement une partie de l'écran.

Google Map : Carte électronique de *Google*

Marker : Affichage d'une position sur une Google Map.

Polyline : Affichage d'un ensemble de traits sur une Google Map.

2 Introduction

L'outil numérique a envahi progressivement toutes les sphères de notre vie quotidienne. Cet outil nous semble de plus en plus indispensable et c'est tout naturellement que, pratiquant une activité de vélo tout terrain en forêt, nous avons souhaité developper une application en lien.

En effet, en explorant différents chemins on souhaite pouvoir s'en souvenir et les réemprunter. Le projet serait de proposer la possibilité de repérer, baliser et répertorier ces chemins, puis de les partager avec une communauté amateur de cyclisme.

Notre problématique est donc de répondre à ce besoin en prenant en compte les restrictions liées au sport. La solution doit être portable, utilisable pour un cycliste. Elle doit utiliser la géolocalisation et une base de données.

Le smartphone est un support qui convient parfaitement, muni le plus souvent d'un GPS, d'une connexion internet et d'une taille permettant d'être transporté en vélo.

Notre projet se déroule dans le cadre d'un projet de deuxième année d'école d'ingénieurs en informatique à ISIMA. C'est dans ce contexte que nous proposons une solution.

Nous verrons tout d'abord dans ce rapport notre démarche pour analyser la problèmatique, puis ce que nous avons mis en oeuvre pour développer une solution et enfin quel est le résultat produit et ses possibles améliorations.

Nous tenons à remercier toutes les personnes qui nous ont aidés dans ce projet :

M. Bruno Guillon, notre tuteur de projet qui a toujours été présent pour nous conseiller et nous aider, notemment sur les problèmes de VPN rencontrés.

L'ensemble de l'équipe du Centre de Ressources Informatiques qui s'est montrée compréhensive et patiente face à nos problèmes liés à leur service.

Nos camarades du local Isilab, qui nous ont aidés au diagnostic du problème de VPN.

Mme Murielle Mouzat, pour son très utile livret de consignes pour le rapport de projet.

3 Le développement de l'application

3.1 Les outils

3.1.1 Machine virtuelle

Concernant la partie serveur, nous avons demandé un support pour héberger la base de données. Isima nous a donné en conséquence un machine virtuelle sur leurs serveurs. Pour travailler dessus, nous utilisons le *ssh*. Le système d'exploitation installé est *CentOS*. Pour le savoir il faut lire le contenu du fichier */proc/version* :

> Linux version 3.10.0-1062.4.3.el7.x86_64 (mockbuildkbuilder.bsys.centos.org) (gcc version 4.8.5 20150623 (Red Hat 4.8.5-39) (GCC)) #1 SMP Wed Nov 13 23 :58 :53 UTC 2019

Cela nous permet de savoir comment installer les programmes dont nous auront besoin. En effet pour utiliser et construire notre base de données et notre serveur il nous faut quelques outils qu'on installe avec le gestionnaire de paquets de CentOS *yum* :

```
sudo yum install java-1.8.0-openjdk
sudo yum install mysql-server
sudo yum install screen
sudo yum install nc
sudo yum install nmap
```

3.1.2 Base de données

Nous avons tout d'abord dû choisir quelle technologie nous allions utiliser pour notre base de données. Nous en connaissions un nombre suffisant pour ne pas en chercher d'autres. Notre choix s'est finalement porté sur MySQL car il répondait à nos besoins, et sa mise en place était plus accessible.

Technologies que nous connaissions

Nom	Niveau de connaissance	Facilité d'intégration	License	
PostGreSQL	Inconnu	Oui	Oui	
MySQL	Bon	Oui	Oui	GPLv2
Oracle Database	Bon	Oui	Oui	
sqlite3	Bon	Non	Oui	

Pour installer le *Système de Gestion de Base de Données*. Il faut tout d'abord mettre un mot de passe administrateur pour s'y connecter.

```
sudo mysqladmin variables -u root -p
```

On peut ensuite modifier le contenu du SGBD en utilisant l'interpréteur de commandes mysql :

```
MySQL -u root -p
```

Il faut enfin créer et sélectionner la base de données que l'on veut utiliser :

```
CREATE DATABASE <nom de la base de données>
USE DATABASE <nom de la base de données>
```

3.1.3 Le serveur Java

Pour l'exécution du serveur java, on utilisera Screen. Screen est un logiciel qui permet de créer des terminaux (appelés *sessions*), de s'y déconnecter puis reconnecter autant que désiré. Ces terminaux continueront leur exécution en arrière plan. On peut donc utiliser l'entrée et la sortie standard pour communiquer avec le programme. Pour se connecter à une session ou la créer si elle n'existe pas, on utilise l'option -R.

```
screen -R <nom_du_terminal>
```

3.1.4 Côté client

Pour développer la partie client de l'application, le choix s'est porté naturellement sur une application android puisque nous possèdons des smartphones sous android 9 et android 7. Nous avons donc utilisé android studio, un environnement de développement intégré conçu pour générer des applications androids. Ce logiciel utilise le langage XML pour la partie "statique", visuelle, et nous laisse le choix entre le langage Kotlin et le langage Java pour la partie exécution de code. Au départ, nous avions la volonté de profiter de ce projet pour apprendre le Kotlin, un langage qui semble très intéressant avec son paradigme fonctionnel. Cependant, nous n'avions également que très peu d'expérience dans le développement mobile qui est aussi très riche avec beaucoup d'aspects et fonctionnements propres à apprendre. Il s'est très vite révélé qu'il était très difficile d'avancer le projet en apprenant en parallèle le développement mobile et le Kotlin. De plus, lorsqu'un problème survient, il est beaucoup plus aisé de trouver de la documentation ou de l'aide avec le langage Java puisque le Kotlin est beaucoup plus récent. Il a donc été convenu de reprendre le projet avec le langage Java afin de se concentrer sur l'apprentissage du développement mobile.

3.2 Base de données

Le développement de la base de données s'est fait rapidement. Nous avons décidé de

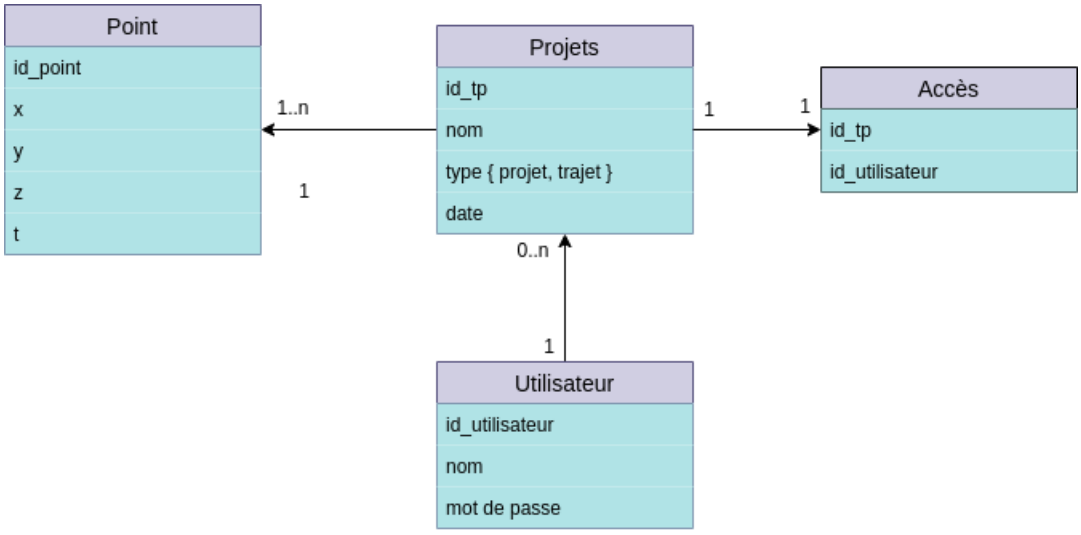

FIGURE 1 – Schéma représentant la base de données

3.3 Le serveur

3.3.1 Processus

Le serveur utilise une architecture avec plusieurs processus. Le processus principal attend une connexion provenant d'un client (application). A chaque connexion un nouveau processus est créé pour intéragir avec le client. Ce processus attend donc un message du client, exécute la commande SQL nécessaire pour obtenir une réponse et répond au client en formatant les données. Un processus supplémentaire existe pour pouvoir travailler directement avec le serveur, sans passer par un client. Pour cela, l'entrée et la sortie standard sont utilisées et il faut donc un accès direct à la machine.

3.3.2 Classes

Le processus principal n'utilise qu'une seule classe *Serveur*, tout comme le processus de communication direct *LocalCommand*. Cependant le processus de communication avec le client est séparé en plusieurs objets : *Client*, *SQLHandler* et *CommunicationHandler*. Le premier est le processus en lui même et utilise les deux autres pour effectuer les tâches qui lui sont assignées. L'objet *SQLHandler* accède à la base de données et formate les réponses. Enfin le *CommunicationHandler* gère la communication réseau avec le client, il permet la réception et l'envoi de chaîne de caractères. Enfin deux classes sont utilisées pour faciliter le travail de conversion des formats pour les points : *Point3D* et *Point4D* qui correspondent respectivement à une coordonnée dans l'espace : (x, y, z) et à une coordonnée dans l'espace et le temps : (x, y, z, t).

3.3.3 Echange type

TODO : Diagramme de séquence

3.3.4 Implémentation

La création du serveur devait à l'origine être rapide car on le considérait comme étant annexe. C'est pourquoi on a choisi l'utilisation d'une technologie que nous connaissions déjà : *Java*. Ce langage contient dans sa bibliothèque standard tout ce qu'il faut pour développer un serveur. De plus il existe une bibliothèque Java développée par Oracle pour la communication avec les bases de données MySQL.

Le *Java* est un langage compilé particulier. Le code est tout d'abord compilé dans un langage intermédiaire appelé *Bytecode*. Puis, il est exécuté dans une machine virtuelle appelée la *Java Virtual Machine* en utilisant une nouvelle étape de compilation *Just In Time*.

Les programmes développés et compilés en Java peuvent être exécutés sur toutes les machines possédant la JVM installée sans avoir aucun changement de code source ou bien de paramètre de compilation.

3.3.5 Fonctionnement des processus

L'organisation des processus en java est particulière. En effet, il n'y a qu'un seul processus lourd (à la différence des *forks* du langage C). Ce processus lourd unique est la JVM. Tous les autres processus, y compris le processus principal de notre programme, ne sont que des processus légers. Il y a donc un partage des ressources, et l'échange de données est plus simple. Cependant, il faut faire attention aux accès simultanés aux ressources.

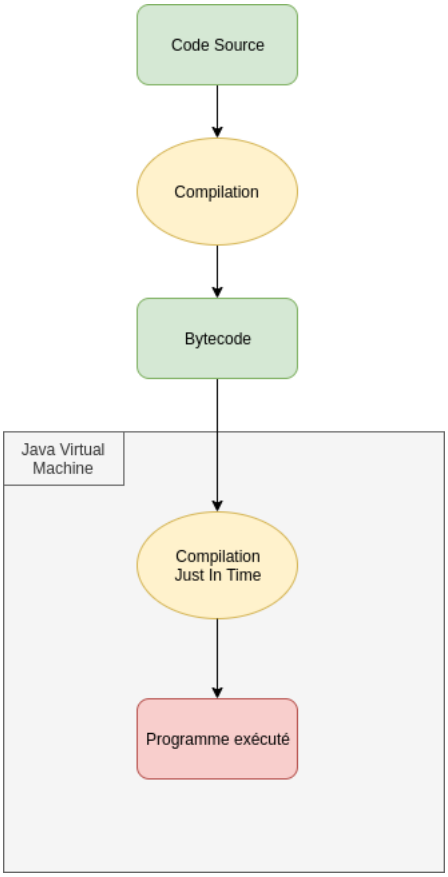

FIGURE 2 – Schéma représentant la compilation en Java

3.3.6 Création de processus

Le processus principal de notre programme est créé automatiquement par la JVM et il exécute le code de la fonction *public static void main(String[] args)* qui est le point d'entrée de notre programme. Pour construire d'autres processus, il y a plusieurs manières. On va ici se concentrer sur la classe *Thread* et l'interface *Runnable*.

3.3.7 Utilisation de *Thread*

Un *Thread* est un objet qui exécute du code dans un autre processus. Pour cela, il suffit de créer un Thread et de le lancer en utilisant la méthode *start()*.

```
Thread monThread = new Thread();
monThread.start();
```

Pour changer le code exécuté, il suffit de redéfinir la méthode *void run()* de Thread.

```
Thread monThread = new Thread() {
  @Override
  public void run() {
    System.out.println("Nouveau processus");
  }
}
monThread.start();
```

3.3.8 Communication réseau

En Java, on utilise les objets de type *Socket* pour faire de la communication en réseau. Ces objets utilisent le protocole TCP pour communiquer et permettent donc de s'assurer de l'état de la connexion. Un premier *socket* (*ServerSocket*) permet d'attendre qu'un client se connecte et de créer un *Socket* pour communiquer avec lui.

```
ServerSocket socket = new ServerSocket(PORT);
while (true) {
  Socket clientSocket = socket.accept();
}
```

Ensuite pour échanger avec le client, on utilise les flux d'entrée et de sortie fournis par le Socket. Les *BufferedReader* et *PrintWriter* sont des objets qui permettent de traiter les flux plus simplement, grâce à des chaînes de caractères.

```
InputStream input = socket.getInputStream();
OutputStream output = socket.getOutputSteam();

BufferedReader reader = new BufferedReader(new InputStreamReader(input));
PrintWriter writer = new PrintWriter(output);
```

3.3.9 Protocole de communication

Liste des messages possibles dans le sens clients $\rightarrow$ serveur

Commande	Utilisation
Subscribe :$< id >:< mdp >$	Permet de s'inscrire
Connect :$< id >:< mdp >$	Permet de se connecter à son compte
History :$< debut >:< fin >$	Permet de récupérer x trajets entre début et fin (en id)
Projects :$< debut >:< fin >$	Permet de récupérer les x projets entre début et fin
NewP :$< nom >:< x+y+z;... >$	Ajoute un nouveau projet
NewJ :$< nom >:< x+y+z+t;... >$	Ajoute un nouveau trajet
EditP :$< id >:< x+y+z;... >$	Modifie un projet

Liste des messages possibles dans le sens serveur $\rightarrow$ client

Commande	Utilisation
Subscribed :$< id >$	Confirme l'inscription
Unsubscribed	Erreur lors de l'inscripton
Connected :$< id >$	Valide la connexion à son compte
Unconnected	Erreur lors de la connection
Project :$< id >:< nom >:< x+y+z;...;x+y+z >$	Envoie des informations sur un projet
Journey :$< id >:< nom >:< d >:< x+y+z+t... >$	Envoie des informations sur un projet

3.4 Développement de l'application android

3.4.1 Squelette de l'application

Afin de comprendre les mécanismes du développement mobile, la première phase a été de simplement créer une application très basique, contenant uniquement les différentes sections qu'on voudrait développer par la suite, sans leur contenu. Il a donc fallu comprendre le système des "activités" et des "fragments" qu'utilise android.

— Une activité est une composante métier d'une application android et possède une "View" (un partie graphique).
— Un fragment s'apparente grandement à une activité. La différence est qu'un fragment est lié à une partie d'écran et non pas à un écran entier.

Ainsi pour créer les différents onglets, on utilise une activité principale qui contient la barre d'outils en haut avec le nom de l'onglet dans lequel on se trouve et le bouton permettant d'afficher le menu de navigation. Ce menu est également contenu dans l'activité principale. Chaque élément de ce menu change le fragment situé en dessous de la barre d'outils. On a donc un fragment pour chaque onglet.

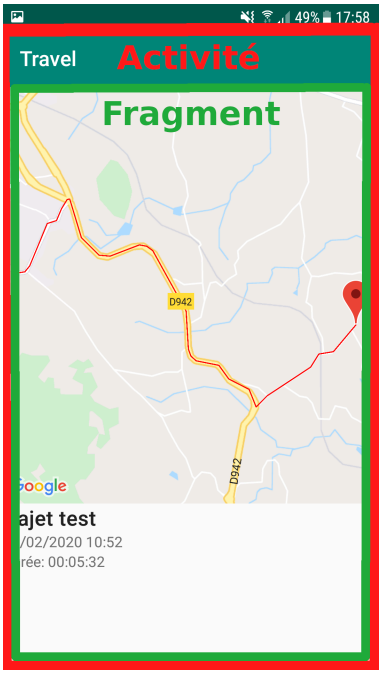

FIGURE 3 – Schéma de l'imbrication d'un fragment dans l'activité principale

3.4.2 Création d'un trajet

Une fois plus à l'aise avec Android Studio, notre objectif premier était de pouvoir créer un simple trajet. Pour ce faire, on a utilisé un fragment qu'on affiche lorsqu'on clique sur l'onglet "Nouveau trajet". Dans ce fragment, on a placé deux objets : une carte et un bouton. Android Studio met à disposition des éléments complexes déjà faits et nous permet de les utiliser à travers de multiples fonctions. C'est le cas pour les boutons et la carte électronique qui est une "Google Map". Ces éléments, que l'on place statiquement grâce au langage XML, sont ensuite accessibles dans le code Java avec leur identifiant. On utilise ces objets en récupérant leurs informations (par exemple quand le bouton est cliqué) ou en chageant leur apparence (par exemple en changeant le texte du bouton ou en ajoutant un trait sur la carte). La première étape a donc été de prendre en main les fonctionnalités d'une Google Map. Celles qui nous ont servi pour ce projet sont :

— Le placement de la caméra (position et zoom)
— Les *Markers*, qui permettent de pointer sur une position précises
— La *Polyline*, un outil qui permet de dessiner sur la carte avec une suite de positions

Le point suivant a été de comprendre le fonctionnement d'android pour obtenir la localisation géographique du téléphone. Le GPS n'est pas le seul moyen d'obtenir une position. En effet il existe trois façons d'obtenir une localisation :

1. le *GPS_PROVIDER* (Global Positionning System) utilise les satellites
2. le *NETWORK_PROVIDER* utilise les wifis et le antennes téléphoniques que détecte le télephone
3. le *PASSIVE_PROVIDER* reçoit les positions passivement lorsque d'autres applications en font la demande

Le *NETWORK_PROVIDER* et le *PASSIVE_PROVIDER* ne donnent qu'une localisation globale, peu précise, en se servant de wifis et d'antennes. Dans notre cas, l'utilisateur sera potentiellement éloigné de ce genre d'appareil. De plus, pour tracer un chemin réalisé en vélo, il faut priviligier la précision des positions. Ainsi nous avons utilisé essentiellement le *GPS_PROVIDER*.

Ainsi la création d'un trajet se fait de la manière suivante :

1. On clique sur l'onglet "Nouveau trajet" qui fait apparaître le fragment contenant une carte Google et un bouton "Nouveau trajet". Si l'application est lancée pour la première fois, une fenêtre contextuelle nous indique qu'il faut autoriser l'application à utiliser le GPS. Cette fenêtre permet d'ouvrir les paramètres du smartphone.
2. On clique sur le bouton "Nouveau trajet". A ce moment l'application prend la localisation acutelle comme point de départ et fait apparaître un point sur la carte à cet endroit. De plus, la carte se grossit et se centre autour de ce point avec une vision d'une centaine de mètres de rayon. Le texte du bouton change et devient "Arrêter le trajet".
3. On se déplace. Lorsque le GPS détecte un changement de position, l'application créer un nouveau point dans le trajet. Le *Marker* se déplace sur cette nouvelle position et un trait se dessine entre le précédent point et le nouveau.
4. On clique sur le bouton "Arrêter le trajet". Un fenêtre contextuelle permettant de rentrer le nom du trajet apparaît.
5. On rentre le nom du trajet dans la zone de texte.
6. On clique sur le bouton "Valider". Le trajet est créé et on peut recommencer.

Avec cette première version de la création d'un trajet, il a fallu tester pour se rendre compte sur un vrai trajet si l'application était assez précise. Pour réaliser des tests, le problème est l'utilisation du GPS, puisque contrairement à ce dont on a l'habitude de développer, ici l'utilisateur bouge pour utiliser l'application. Il devient donc difficile de tester et corriger de manière répétée puisqu'il faut bouger un minimum pour observer les changement du GPS, puis revenir sur son poste de travail pour corriger le problème éventuel et remettre l'application sur le téléphone.

Cependant il existe d'autres manières de tester les applications mobiles. Par exemple Android Studio propose un smartphone virtuel qu'on crée sur notre poste de travail afin de le manipuler et faire des tests. Pour la partie GPS, on peut injecter à cette machine virtuelle de fausses coordonnées. Le problème lié à cette méthode est que la virtualisation d'un smartphone coûte très cher en ressources. Ainsi android studio en plus de la machine virtuelle demandent un ordinateur puissant.

On peut également émuler les coordonnées GPS sur un smartphone réel grâce à des applications (disponibles sur le marché *Play Store*). On indique quelles doivent être nos coordonnées et notre GPS fait comme si il y était. Le problème soulevé par cette émulation, que ce soit sur une machine virtuelle ou avec une application sur un smartphone, est que ça ne prend pas en compte les erreurs de positionnement GPS. En effet lors d'un test réel, on a pu observer que le tracé du trajet se fait bien mais que par moment le GPS indique des positions fausses, ce qui a pour effet de tracer deux traits entre ce point et le chemin qu'on parcourt. Pour pallier à ce problème, la solution pourrait être de capturer la vitesse que nous fournit le GPS et calculer si le nouveau point qu'on veut ajouter au tracé est atteignable avec cette vitesse. Cependant la

vitesse communiquée n'est pas fiable lorsqu'on bouge lentement. On a donc décidé de faire ce traitement avec une vitesse maximale, assez haute pour ne pas être atteinte par un cycliste.

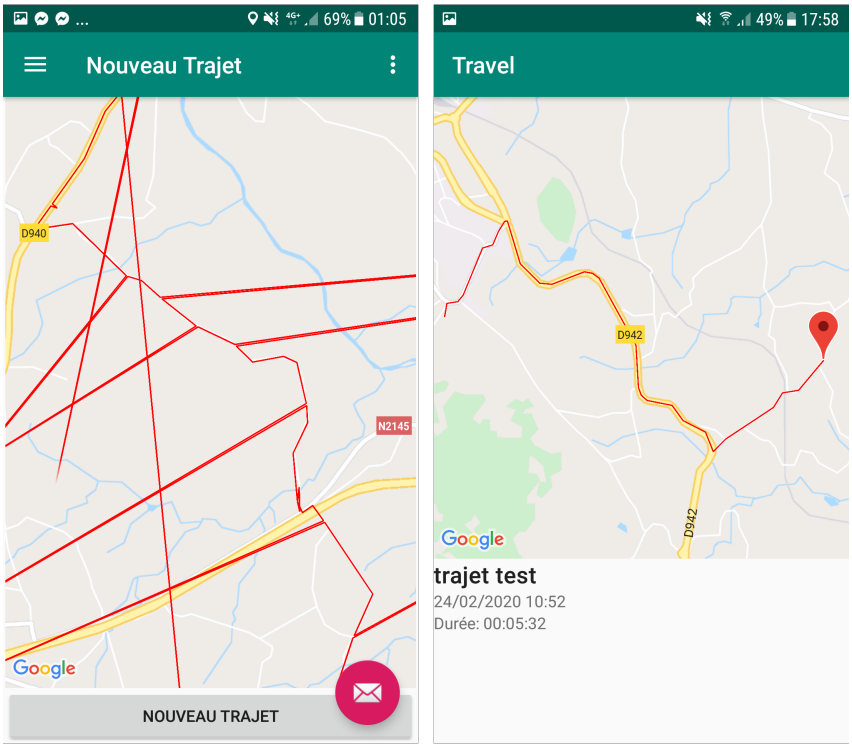

FIGURE 4 – Différence entre un trajet avec le problème d'imprécision du GPS et un trajet avec la correction apportée

3.4.3 Sauvegarde des trajets

Au début du projet, les trajets effectués par l'utilisateur étaient destinés à être enregistrés dans une base de données. Ainsi, les utilisateurs pouvaient utiliser plusieurs appareils sans perdre leurs trajets et cela aurait permis plus tard le partage de ceux-ci. Cependant, après avoir rencontré certains problèmes pour relier l'application smartphone au serveur, il a été décidé de sauvegarder les trajets dans des fichiers, au moins temporairement.

Pour simplifier la communication potentielle avec le serveur, une bibliothèque Java a été réalisée pour représenter un trajet. Cette bibliothèque contient deux objets essentiels *Journey* et *JourneyHistory*. L'un représentant un trajet et l'autre un ensemble de trajets (utilisé plus tard pour l'historique). L'objet trajet n'est qu'une abstraction d'un tableau de localisations. Ces localisations possèdent quatres composantes :

- la latitude de la position
- la longitude de la position
- l'altitude de la position (pour le dénivelé du trajet)
- la date à laquelle le point a été pris

Ainsi à chaque nouveau point lors d'un trajet, c'est dans cet objet qu'on le stocke. Puis lorsque l'utilisateur finit son trajet, on l'ajoute à l'objet *JourneyHistory* et on écrit dans un fichier le contenu de l'objet (son nom, sa date de création et chaque point qu'il contient).

3.4.4 Historique

L'objet *JourneyHistory* contient en mémoire tous les trajets effectués par l'utilisateur tant que l'application est en fonctionnement. Ainsi, lorsque l'application se démarre, on charge dans cet objet tous les trajets sauvegardés dans des fichiers. De cette manière, on peut afficher le contenu de l'onglet "Historique" sans lire des fichiers à chaque fois.

L'affichage de cet onglet se fait via des "MaterialCards". Google met à disposition un certain nombre de composants préfaits. Ceux-ci sont plus "design" et permettent de garder un visuel cohérent simplement. Ici nous avons utilisé les "Cards" qui sont des conteneurs pour d'autres objets comme des images ou du texte. On ajoute simplement dans ce conteneur les éléments qui le compose et la mise en forme se fait quasi automatiquement.

Dans l'historique, chaque "Card" représente un trajet. Une "Card" est composée d'une capture d'écran de la carte prise à la fin du trajet, du nom du trajet, de sa date de création et de sa durée. Ainsi, lorsqu'on affiche le fragment "Historique", on crée une "Card" pour chaque trajet dans le *JourneyHistory*. Ceux-ci sont triés par date de création, on les affiche donc du plus récent au plus vieux.

Afin de pouvoir consulter les trajets de manière plus précise, le dernière fonctionnalité ajoutée a été de pouvoir cliquer sur chaque "Card" de l'historique. Cela a pour effet d'amener vers une autre page, une autre activité permettant de consulter un trajet. Cette page contient une Google Map avec le trajet dessiné dessus. En dessous on retrouve le nom, la date et la durée du trajet. A l'origine, cette page devait être plus complète, avec par exemple le dénivelé du trajet, la météo ou des statistiques. De plus la carte affichée n'est ici que pour permettre à l'utilisateur d'avoir une vue globale du trajet en manipulant la carte. Elle ne remplit pas la fonction voulue à l'origine qui était de fonctionner comme un GPS de voiture, guidant l'utilisateur au cours de son trajet.

A Manuel d'utilisation