Développement D'une Application Android

Location Hunter

http://www.locationhunter.ch

Maxime ZAMMIT 302-402

Tm Maturité 2018-2019

Catégorie : Travail Technique

Maître : M. Ischi

TABLE DES MATIÈRES

Nota Bene : Ce document se réfère aux codes contenus dans le dossier annexe. Les mots surlignés en jaune sont expliqué dans le lexique en fin de travail.

Introduction

Dans le cadre de mon travail de maturité, j'ai imaginé et développé un jeu pour téléphone mobile, qui tourne sur le système d'exploitation Android. Ce jeu, sous forme d'application pour smartphone, utilise la technologie GPS (Global Positioning System).

Je suis parti sur l'idée d'un jeu parce que c'est un genre d'application qui permet beaucoup de liberté dans son développement. Le but à atteindre est fixé dès le départ, mais la façon d'y arriver est multiple. J'ai eu plusieurs idées de jeux utilisant la technologie GPS, mais beaucoup se sont révélés infaisables d'un point de vue technique ou trop difficiles pour mes connaissances en programmation. Par exemple, lors des premiers essais, j'ai observé que le GPS ne fonctionne pas à l'intérieur des bâtiments. J'ai aussi remarqué qu'il est difficile de faire communiquer de manière simultanée deux appareils, tout en ayant à gérer les possibles déconnexions d'utilisateurs. Suite à ces différents essais, l'idée de *LocationHunter* m'est venue. Il s'agit d'un jeu avec une interaction entre un téléphone mobile et un serveur, sous forme de requête envoyée par le téléphone au serveur.

Le but du jeu est simple. Chaque jour, une coordonné GPS est tirée au hasard parmi une liste située sur une base de données. Le joueur doit atteindre le plus rapidement possible la destination, pour gagner un maximum de point. Afin de vérifier si l'utilisateur s' est bien rendu aux coordonnées GPS demandées, il utilise l'application qui les confirme au serveur.

Le jeu est composé de plusieurs programmes qui tournent en parallèle et interagissent entre-eux afin de fournir le résultat demandé. Les éléments essentiels du jeu sont : la base de données qui est gérée par le système MariaDB[1], un serveur codé en python qui permet l'interaction entre l'application Android et la base de données, ainsi qu'un site internet pour la gestion des utilisateurs l'affichage des scores.

Dans ce travail, je commencerai par expliquer mes premières expériences en Java et le développement d'une première application Android. Celle-ci sert de test et applique un algorithme qui utilise des techniques de hachage afin de vérifier mes compétences. Puis je détaillerai le jeu et son fonctionnement en l'illustrant par un exemple. Enfin je compléterai en expliquant l'utilité et les parties importantes de chaque sous-programme formant le jeu, et je mentionnerai certaines des difficultés rencontrées. Je conclurai ce travail par un bilan personnel.

1 Disponible à l'adresse : https://mariadb.org/

Développement

Ma première application en Java

Comme expliqué dans l'introduction, je n'avais jamais programmé en Java auparavant, ce fut une nouvelle découverte pour moi. Afin de tester mes compétences et l'IDE, j'ai décidé de reproduire une application utilisant le RFC 6238[2]. Celui-ci est expliqué sur le site internet de l'IETF[3], il permet de générer, à partir de l'heure et d'une clé privée, un code de six chiffres changeant toutes les trente secondes. Ce type d'algorithme est celui utilisé par exemple par l'application Google Authentificator[4], il permet de sécuriser un compte utilisateur en demandant une deuxième vérification. Il s'agit d'un algorithme de création de mot de passe à utilisation unique, qui va générer à partir d'une clé privée et de l'heure, un mot de passe à six chiffres valide pendant 30 secondes. Il servira de supplément au mot de passe officiel. Ce type d'algorithme était d'abord utilisé pour sécuriser les comptes de E-Banking avant de s'être démocratisé et d'être utilisé un peu partout.

Le premier objectif que je me suis fixé, est de recréer une application client en Java, compatible avec celles déjà existantes et reprenant l'algorithme RFC 6238. Pour ce faire, j'ai séparé mon application en deux activités. La première, que j'ai appelée *Main*, permet l'affichage des codes de six chiffres qui doivent être entrés lors de la connexion. La deuxième, *EditKey*, permet de modifier la liste des clés privées qui sont enregistrées dans l'application.

I.Main

Dans l'activité *Main*, je vais analyser la méthode *sixDigitsGen* car elle est intéressante à décortiquer. Avant ceci, j'ai créé deux méthodes respectant la norme RFC 6238, une qui permet de transformer l'hexadécimal en bytes et une qui permet d'appliquer une fonction de hachage.

[CODE-Annexe 4-P.2, l.42 à 63]

Ensuite dans la méthode *sixDigitsGen*, la première étape est de récupérer le temps écoulé depuis le premier janvier 1970 UTC, il s'agit du temps Unix Epoch . Pour ce faire, j'utilise l'instruction : system.currentTimeMillis() que je divise par 1000 afin d'avoir ce temps en seconde, puis que je divise par 30 afin d'avoir une valeur qui augmente toute les trente secondes. Après, je cast la valeur en Integer et je l'enregistre dans une variable. Ensuite, je m'occupe de mettre en forme la clé privée, en supprimant tous les espaces et en mettant toutes les lettres en majuscules, puis dans les lignes suivantes, je transforme ma variable contenant le temps en byte et je convertis la clé privée en

Illustration 1: Capture d'écran de l'activité Main

2 Disponible à l'adresse : https://tools.ietf.org/html/rfc6238

3 Internet Engineering Task Force, disponible à l'adresse : https://www.ietf.org/

4 Google Authentificator. *Wikipédia : l'encyclopédie libre* [en ligne]. Dernière modification de la page le 31 juillet 2018 à 20:31. [Consulté le 1 Novembre 2018]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Google_Authenticator

hexadécimal, que je reconvertis directement en byte. Enfin, j'applique le reste de l'algorithme décrit sur le site de l'IETF et je retourne le code à six chiffres.

[CODE-Annexe 4-P.2 à 3, l.66 à 117]

Pour terminer l'analyse de l'activité *main*, il faut encore étudier la méthode *onCreate*. Dans celle-ci nous retrouvons plusieurs parties indispensables au fonctionnement de l'application. Premièrement, je dois récupérer les clés privées stockées dans un fichier texte. Pour ce faire, je dois utiliser une petite manipulation qui récupère caractère par caractère que je concatène afin d'obtenir une variable en String qui contient toute mes clés. Puis, je crée un objet *timer* qui contient une méthode *onTick* qui s'exécute toute les secondes.

[CODE-Annexe 4-P.3 à 4, l.131 à 163]

La méthode *onTick* est la plus importante de l'application, car c'est elle qui donne l'instruction de calculer le code à six chiffres, et qui le met à jour dans l'interface utilisateur. Elle permet aussi de mettre à jour la barre affichant le temps restant de la validité des codes.

[CODE-Annexe 4-P.4, l.165 à 187]

II.EditKey

Dans la deuxième activité, *EditKey*, j'ouvre le fichier contenant les données avec les clés privées et je met en place une interface afin de pouvoir les modifier.

Au début de la méthode *onCreate*, je récupère la valeur String contenue dans le fichier *rfc6238authentificatorconfig.txt* et je la stocke dans une variable. Celle-ci contient toutes les clés privées, qui sont séparées par des virgules.

[CODE-Annexe 4-P.6, l.41 à 60]

Ensuite, je mets en place la roulette de l'interface utilisateur et crée un tableau vide correspondant.

[CODE-Annexe 4-P.6 à 7, l.63 à 74]

Enfin, je transforme le String en Array de String à l'aide de la méthode split. Puis je vais le transférer dans une ArrayList. Une fois fait, et comme ma roulette s'étend sur l'intervalle [1;99] et mon ArrayList sur [0;98], je récupère la valeur de la roulette à laquelle je soustrais 1 et j'insère dans l'éditeur la valeur correspondante de la ArrayList. Je termine mon opération avec le tableau en remplissant ses cases vides avec la valeur « NewKey ».

[CODE-Annexe 4-P.7, l.82 à 100]

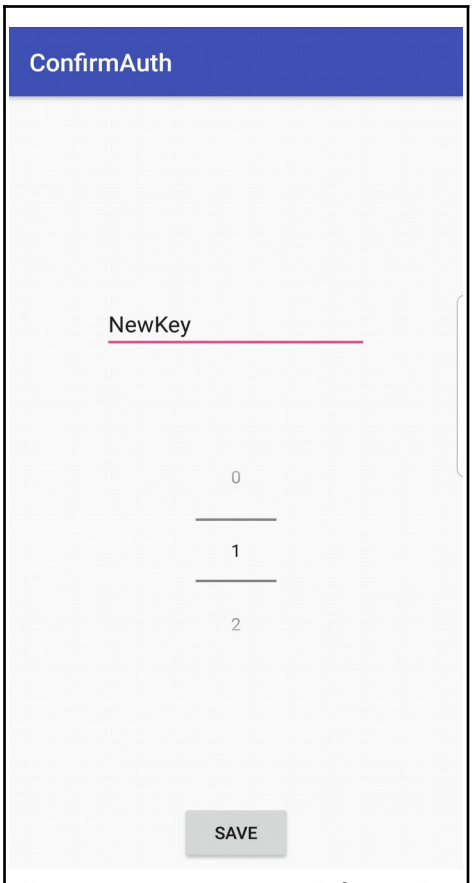

Illustration 2: Capture d'écran de l'activité EditKey

Après, je mets en place un listener, afin de détecter et d'exécuter les modifications conséquentes lors d'un changement de roulette dans le GUI, c'est-à-dire modifier l'éditeur et sauvegarder les changements dans la variable.

[CODE-Annexe 4-P.7, l.103 à 118]

Pour conclure l'étude de cette activité, je crée la méthode *save.* Elle permet de retransformer le ArrayList en un String continu, avec les valeurs séparées par des virgules et de les sauvegarder dans le fichier. De plus, elle va renvoyer l'utilisateur vers l'activité *main*.

[CODE-Annexe 4-P.7 à 8, l.120 à 165]

Coder cette application m'a été très bénéfique. Malgré le fait que certaines parties de la norme restent très abstraites pour moi, j'ai pu découvrir, grâce à celle-ci, l'importation de libraire externe sur Android Studio. J'ai aussi découvert des particularités propres à la programmation en Java. Et surtout, c'est la preuve que chacun peu réaliser un programme fonctionnel en autodidacte en partant de zéro. Ainsi, dès à présent, j'utilise ma propre application de connexion à deux facteurs pour me connecter à mes comptes. Par exemple, pour me connecter à mon compte OVH[5], que j'utilise pour louer le serveur pour l'application principale de ce travail, je dois fournir le code à six chiffres généré par cette première application. La partie la plus compliquée était de rendre l'application compatible avec celles déjà existantes, tel Google Authentificator. Pour ce faire, j'ai essayé plusieurs paramètres différents, dont certains types de variables que j'ai changé plusieurs fois.

5 Disponible à l'adresse : https://www.ovh.com/fr/

II LocationHunter

a) Présentation du projet

Location Hunter est un jeu en ligne, que j'ai inventé, utilisant le positionnement GPS. Il fonctionne grâce à une base de données, un serveur codé en Python utilisant le protocole HTTP et une application client, qui émet des requêtes à celui-ci. Le serveur en python contient deux processus. Le premier permet de faire tourner le protocole HTTP et de répondre aux requêtes de l'application. Le deuxième, tournant en parallèle, sert d'horloge au jeu. En effet, chaque jour à 10h du matin, le deuxième processus récupère la liste des coordonnées GPS enregistrées sur la base de données. Il tire un couple de coordonnée au sort et le sélectionne comme destination du jour à atteindre. Puis, lors de l'ouverture de l'application, celle-ci va récupérer la destination sélectionnée et l'afficher sur une carte avec un waypoint. Plus vite l'utilisateur atteindra l'objectif, plus il aura de points. Ils sont distribués selon la logique suivante : s'il arrive premier, il gagne 20 points, deuxième 10 points, troisième 5 points, puis 3 points pour tous les autres. De plus, si l'utilisateur arrive moins d'une heure après le tirage, alors son gain de points est doublé. Les joueurs peuvent ensuite consulter leur score sur le site internet, ainsi que celui des 5 meilleurs participants depuis la création du jeu.

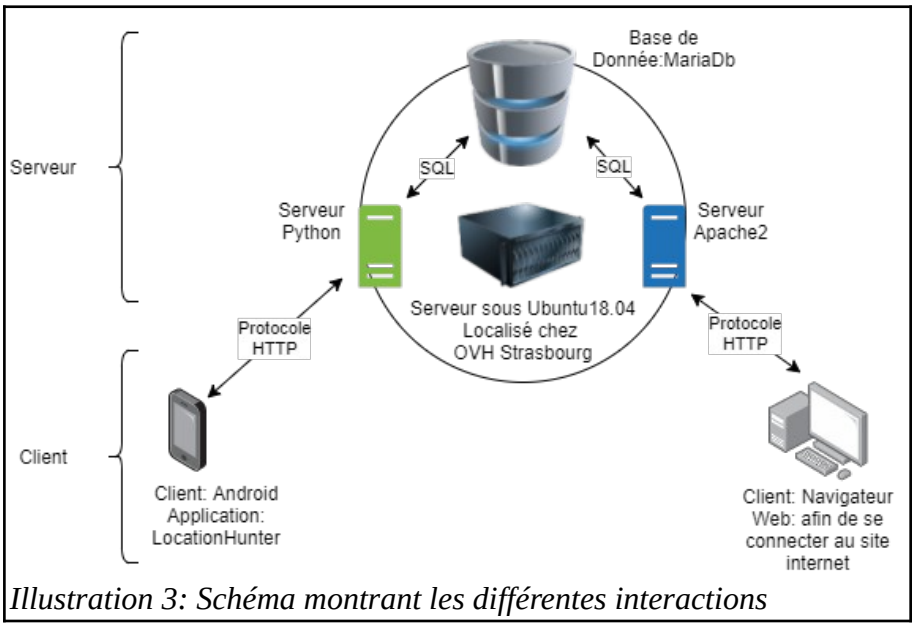

Illustration 3: Schéma montrant les différentes interactions

Je vais tout d'abord expliquer la structure de la base de données, puis je continuerai avec la partie la plus technique de ce travail, l'application en Java. Ensuite, je présenterai l'API que j'ai crée en python afin de faire communiquer l'application et la base de données. Enfin je terminerai brièvement avec les parties les plus intéressantes du site internet et de la publication sur Google Play.

b) La base de données

MariaDB est une application serveur de base de données. Elle m'a permis de créer les deux bases pour le jeu. La première, que j'ai nommée *lhwebsite*, contient des tables relatives au site internet, telles que les nouveautés en français, en anglais ou encore des statistiques. La deuxième, *tmdata*, contient toutes les tables relatives à l'application telles que *location* qui est la liste des coordonnées GPS pouvant être tirées au sort par le serveur Python. La table *history* contient un historique de toutes les informations relatives au jeu pour chaque jour, comme la coordonnée à atteindre, par qui elle a été atteinte et quand elle a été sélectionnée. Il existe aussi

account qui contient toutes les informations sur les comptes utilisateurs. Et finalement, la table *command* qui a une fonction un peu spéciale que je détaillerai plus tard.

c) L'application en Java

L'application Android fut la partie la plus technique à réaliser. D'une part, je n'ai jamais fait un projet d'une telle complexité, et d'autre part, je n'avais que très peu d'expérience en Java. Elle est constituée de quatre activités, que je détaillerai ci-dessous dans l'ordre suivant : le menu, les paramètres, la page de gestion du compte utilisateur et la page de jeu. La dernière contient toute la partie qui permet de récupérer les coordonnées GPS du téléphone, de se connecter à l'API et de calculer la distance entre deux latitudes et deux longitudes. Il s'agit donc de la pièce maîtresse de ce travail.

I. Le Menu

Lorsque l'on démarre l'application, nous tombons sur cette activité. Celle-ci joue un rôle clé dans la vérification de plusieurs paramètres du téléphone, afin que le jeu puisse fonctionner correctement. Mais avant tout, elle s'occupe à chaque démarrage de regarder les paramètres de l'utilisateur afin de charger le fichier de variables correspondant à la langue de celui-ci. En effet, tous les textes de l'application sont réunis dans deux fichiers, l'un d'eux contient ceux en français et l'autre contient l'équivalent en anglais. Ce système me permet d'avoir une compatibilité bilingue, voir plus si je le voulais, car facile à mettre en place. Voici le code correspondant :

[CODE-Annexe 3-P.2, l.52 à 63]

Illustration 4: Capture d'écran de l'activité du Menu

Puis, je m'occupe d'initialiser quelques variables et je vérifie si l'utilisateur a accordé les permissions de géolocalisation à l'application. Si ce n'est pas le cas, je lui envoie une requête. Dans le cas où celle-ci est refusée, je bloque le reste de l'application et j'informe l'utilisateur que s'ils ne les acceptent pas, l'application ne pourra pas fonctionner. Par ailleurs, je lui fournis un lien pour s'informer du fonctionnement de l'application sur le site internet.

[CODE-Annexe 3-P.3, l.67 à 81 et P.3, l.88 à 119]

Une fois que j'ai les permissions, je vais procéder à une autre vérification : est-ce que l'application est à jour ? Pour ce faire, je vais lancer une tâche asynchrone afin d'exécuter une requête HTTP au serveur python et récupérer la version actuelle de l'application. Dès lors, trois cas sont possibles. Le premier est que la valeur de la variable *version* de l'application client est équivalente à celle récupérée sur internet, dans ce cas il n'y a pas de problème. Le deuxième est que la valeur sur internet ne coïncide pas avec celle du client, dans ce cas un message informe le client qu'il doit mettre l'application à jour s'il veut pouvoir jouer. Le dernier est si la requête n'aboutit

pas, dans ce cas, soit le téléphone n'a pas de réseau internet, soit le serveur est indisponible, ce qui dans les deux cas fait apparaître un message d'erreur empêchant l'utilisateur de jouer.

[CODE-Annexe 3-P.3, l.82 et P.4 à 5, l.174 à 242]

Pour finir, je définis trois méthodes qui permettent chacune de changer d'activité. Celle qui permet d'aller à l'activité pour jouer, contient une vérification qui oblige l'utilisateur d'être connecté. Dans le cas contraire, un message d'erreur s'affiche.

[CODE-Annexe 3-P.4, l.140 à 171]

II. Les Paramètres

Cette activité permet de changer la langue de l'application qui est chargée au démarrage. L'utilisateur a trois options, soit la langue par défaut du téléphone, soit le français ou soit l'anglais. La première étape est de récupérer le choix actuel afin de l'afficher dans des radio boutons. Puis, permettre à l'utilisateur de modifier selon son choix, de le sauvegarder et d'effectuer les modifications. La variable *i*, qui est comparée ci-dessous dans le listener, correspond à l'id du radio bouton sélectionné. Ce paramètre est stocké dans le SharedPreferences de l'application.

[CODE-Annexe 3-P.14 à 15, l.23 à 95]

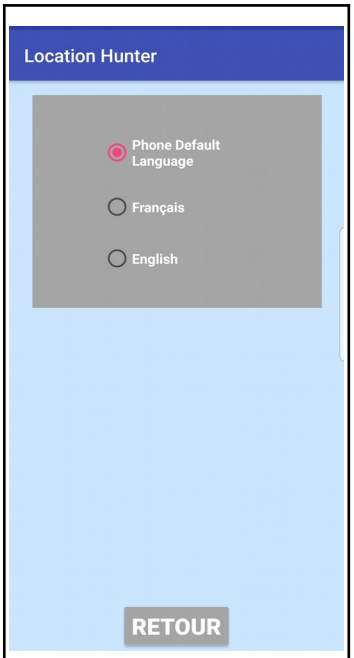

Illustration 5: Capture d'écran de l'activité des paramètres

III. Gestion du compte utilisateur

Nous allons maintenant étudier la manière dont l'application gère la connexion aux comptes utilisateurs. La première étape est de vérifier si l'utilisateur est déjà connecté. Pour ce faire, on récupère, s'il existe, l'e-mail qui est stocké dans les SharedPreferences. Si ce n'est pas le cas, alors l'utilisateur n'est pas connecté. Selon le résultat, la page affiche des options différentes. Si le client est connecté, alors le bouton de connexion sera désactivé et il y aura un bouton de déconnexion. Alors que s'il est déconnecté, il y aura le bouton de connexion et un bouton d'inscription qui renvoie sur le site internet. De plus, le texte indiquant le statut change selon ce dernier.

[CODE-Annexe 3-P.8 à 9, l.53 à 78]

La méthode de connexion passe par une tâche asynchrone, qui va de nouveau questionner le serveur Python sur l'existence et la véracité du couple mot de passe et e-mail. Si c'est le cas, ces valeurs sont enregistrées dans les SharedPreferences, sinon un message d'erreur apparaît.

[CODE-Annexe 3-P.9 à 11, l.120 à 214]

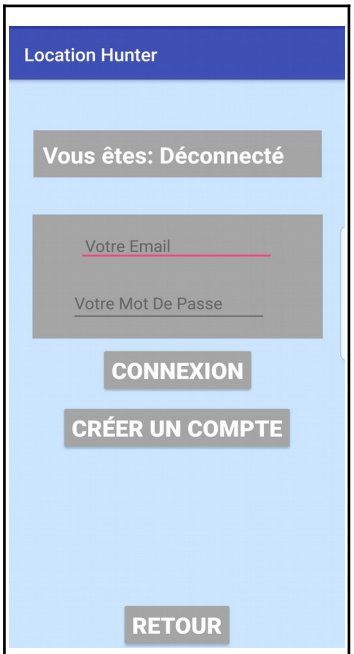

Illustration 6: Capture d'écran de l'activité de connexion

La méthode de déconnexion consiste seulement à supprimer les SharedPreferences contenant les informations de connexion et à reparamétrer la page pour se reconnecter.

[CODE-Annexe 3-P.9, l.95 à 117]

On retrouve a la fin du code deux méthodes permettant le hachage en SHA-1 du mot-de-passe, afin d'avoir un minimum de sécurité et de protection des données dans la base.

[CODE-Annexe 3-P.11, l.216 à 235]

IV. La page du jeu

Pour cette activité, qui est la plus longue et la plus importante, je vais détailler toutes les étapes de son fonctionnement. La première étape est de se connecter à Mapbox avec la clé unique qui m'a été fournie. Il s'agit d'un service qui permet d'intégrer des maps dans des projets numériques. Il me permet d'afficher la map dans le GUI. Puis, j'effectue la première tâche asynchrone, nommée *getToday*, qui va récupérer l'objectif du jour en faisant une requête HTTP au serveur Python. De plus, elle sert aussi de test de connexion. Je continue en vérifiant que l'application a bien les droits de géolocalisation. Si ce n'est pas le cas, je renvoie l'utilisateur sur le menu principal. Autrement, je mets en place le listener pour récupérer les coordonnées GPS et vérifie que l'utilisateur est bien connecté.

[CODE-Annexe 3-P.19, l.79 à 108]

La classe *getToday* fait une requête à l'API Python et récupère la réponse, puis l'analyse. Celle-ci consiste à isoler toutes les informations transmises, répondues en un string séparé par des points-virgules. Si elle ne récupère aucune donnée, cela signifie qu'il y a eu un problème de connexion, soit avec nos services, soit avec le réseau de l'utilisateur, et envoie un message d'erreur. Puis elle va s'occuper de mettre en forme certaines données, comme remettre les accents qui ont été encodés à ma façon dans l'API et qui doivent être décodés. Il s'agit d'un problème rencontré lors de la création. Je n'arrivais pas à transmettre les accents sans qu'il y ait un problème d'encodage avec le serveur python. J'ai donc repris l'encodage HTML existant, c'est-à-dire « é » pour un « é », que j'ai modifié, ce qui donne « /eacute/ ». Il ne me reste plus qu'à l'afficher dans le texte view et sur la map, ce que je fais grâce à une tâche asynchrone.

[CODE-Annexe 3-P.21 à 23, l.244 à 363]

Illustration 7: Capture d'écran de la page de jeu

Je crée aussi une méthode qui me permet de calculer la distance en mètre entre deux couples de coordonnés GPS[6].
[CODE-Annexe 3-P.20, l.181 à 191]

6 VENESS, Chris, 2002. *Calculate distance, bearing and more between Latitude/Longitude points* [en ligne]. Dernière mise à jour de la page en janvier 2015. [Consulté le 1 Novembre 2018]. Disponible à l'adresse : http://www.movable-type.co.uk/scripts/latlong.html

Il ne me reste plus qu'à analyser ce qui se passe lorsqu'on appuie sur le bouton : « J'y suis ». Pour ce faire, j'ai créé deux méthodes, la première, qui se nomme *ImHere*, est déclenchée lorsque l'on presse sur le bouton. Celle-ci va vérifier que le service de localisation fonctionne bien. Si ce n'est pas le cas, elle envoie un message d'erreur. Autrement, elle exécute la deuxième qui est une tâche asynchrone, nommée *verification*.

[CODE-Annexe 3-P.21, l.219 à 239]

Cette dernière, qui fonctionne comme toute les autres tâches asynchrones déjà analysées précédemment, va envoyer une requête à l'API python, en fournissant la latitude, la longitude, la distance calculée, l'e-mail et le mot de passe de l'utilisateur. Puis, toujours s'il n'y a pas eu de problème de connexion, elle va analyser la réponse, en la séparant en trois. Soit l'utilisateur a gagné, dans ce cas l'application affiche le nombre de points gagnés. Sinon, soit il n'est pas aux bonnes coordonnées et un message le lui indiquant s'affiche, ou soit il a déjà validé la mission du jour et un message d'erreur correspondant s'affiche.

[CODE-Annexe 3-P.23 à 25, l.365 à 477]

Voilà comment fonctionne l'application Android, de paire avec l'API python que je vous décrirai dans la partie suivante. Le développement de cette application a été un vrai challenge. Récupérer les coordonnés GPS n'a pas été simple, c'était la première fois que je manipulais le système de permission du téléphone. La gestion des permissions ainsi que les différentes manières d'enclencher le listener étaient plus complexes que prévu. J'ai aussi rencontré des difficultés avec la mise en place des tâches asynchrones qui n'étaient pas un concept facile.

d) L'API en python

L'API en python est composé de deux fichiers. Le premier, *Server.py,* est un programme composé de deux Threads qui fonctionnent en boucle sur le serveur Linux. Ils servent à deux tâches différentes et auraient pu être séparés mais j'ai trouvé que c'était un bon défi de les regrouper, sachant que lors du cours de programmation python, nous n'avions pas encore vu le concept de classe et de processus. De plus, il aurait fallu gérer un deuxième programme au démarrage du serveur, ce qui n'aurait pas été simple à mettre en place. Le deuxième fichier de scripte, *api.py*, est exécuté lors de chaque requête à l'API. Dans la partie qui suit, je développerai le fonctionnement des deux fichiers.

I. Server.py

Comme introduit précédemment, ce fichier contient deux Threads. Le premier ouvre un serveur HTTP sur le port 8080 qui permet à l'application Java de communiquer avec L'API. Pour ce faire j'utilise le module « http.server ».

[CODE-Annexe 2-P.2, l.32 à 52]

Le deuxième Threads est une « horloge » qui va vérifier toute les 0.9 secondes s'il est 10h du matin. Si c'est le cas, le scripte suivant s'enclenche :

[CODE-Annexe 2-P.2 à 3, l.56 à 98]

Celui-ci permet de récupérer la liste des coordonnées existantes sur la base de données, d'en choisir un couple au hasard, qui n'est pas le même que celui choisi précédemment, et le sélectionner comme destination du jour. Tout ceci est principalement de la manipulation de commande SQL et de la mise en forme.

Précédemment, j'avais mentionné une table nommée *command*, celle-ci intervient dans ce fichier car elle permet d'enclencher ce processus manuellement.

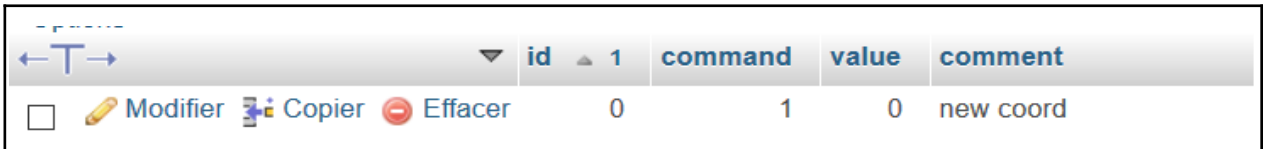

Illustration 8: La table command, le processus s'enclenche lorsque value vaut 1

II Api.py

Ce fichier est exécuté à chaque requête de l'application Java à l'API. Il utilise le module « cgi » qui est complémentaire au module « http.server ». Les informations sont passées par la méthode GET, c'est-à-dire dans l'URL de la requête. Voici un exemple de requête que tout le monde peut exécuter dans un navigateur web: http://www.locationhunter.ch:8080/api.py?command=getToday. Celle-ci permet à l'application de connaître la coordonné du jour, elle renvoie la réponse sous ce format :
« -[ServerApp]- ; *ID* ; *LATITUDE, LONGITUDE* ; *NOM_DE_L_OBJECTIF* ; *QUI_A_DEJA_REUSSI* ; *DATE + HEURE_DU_TIRAGE* ; »
Ce qui donne :
« -[ServerApp]- ;236;46.205204, 6.203843;Aula Candolle;,user@email.ch;2018:10:22:10:00:00; »

Chaque commande a sa fonction qui est exécutée lorsqu'elle est appelée, sauf *sendIt*, qui est implémentée directement dans le corps du scripte. Les commandes sont :

-*getToday* : Elle permet d'obtenir les coordonnées du jour.

[CODE-Annexe 2-P.5, l.79 à 94]

-*connexion* : Elle permet de vérifier la véracité d'un couple e-mail et mot-de-passe.

[CODE-Annexe 2-P.4 à 5, l.63 à 77]

-*getVersion* : Elle permet d'obtenir le numéro de la dernière version de l'application.

[CODE-Annexe 2-P.5, l.96 à 107]

-*sendIt* : Elle permet de vérifier que l'utilisateur a bien atteint l'objectif. Elle va premièrement récupérer toutes les informations de l'utilisateur passées dans l'URL, c'est-à-dire la latitude, la longitude, la distance de l'objectif, l'e-mail et le mot-de-passe. Puis elle va vérifier que le couple e-mail et mot-de-passe existe bien. Si c'est le cas, elle va calculer la distance entre les deux couples de coordonnées, c'est-à-dire celle de l'utilisateur et celle de l'objectif. Elle va vérifier que la distance obtenue n'a pas une différence plus grande que ± 1 mètre par rapport à celle donnée lors de la requête dans l'URL et va aussi vérifier que sa valeur vaut moins de 50 mètres. Dès que toutes ces conditions sont vérifiées, elle va effectuer la modification dans les différentes tables de la base de données.
[CODE-Annexe 2-P.5 à 6, l.130 à 187]

e) Le site internet

Je ne détaillerai pas tout le fonctionnement du site internet, car il pourrait faire l'objet d'un travail à part entière. Je développerai uniquement quelques parties qui sont relatives à l'application, comme la gestion des comptes utilisateur, la page d'administration et les statistiques.

I. Les comptes utilisateurs

Les utilisateurs doivent créer leur compte sur le site internet à l'adresse http://www.locationhunter.ch/account.php. Une fois fait, ils doivent l'activer en cliquant sur le lien envoyé par e-mail contenant un code à dix chiffres généré aléatoirement. Dès lors, ils pourront utiliser leur compte et se connecter. Le mot de passe est haché en SHA-1 et toutes les informations sont passées dans des fonctions d'assainissement afin d'éviter tout problème.

[CODE-Annexe 1-P.10]

id	email	username	password	enabled	points	code
1	contact.locationhunter@gmail.com	Maxime	2bf2d48e26b21dedb61886082cabc70fb5367ffb	1	100	8821199683

Illustration 10: La table account qui contient tout les comptes utilisateurs

La capture d'écran ci-dessus montre l'architecture de la base de données pour les comptes utilisateurs. Nous pouvons remarquer le mot-de-passe haché, la colonne *enabled*, qui lorsqu'elle vaut 1, signifie que le compte est activé, *points* qui stocke le nombre de points et *code*, le nombre à dix chiffres confirmé par e-mail.

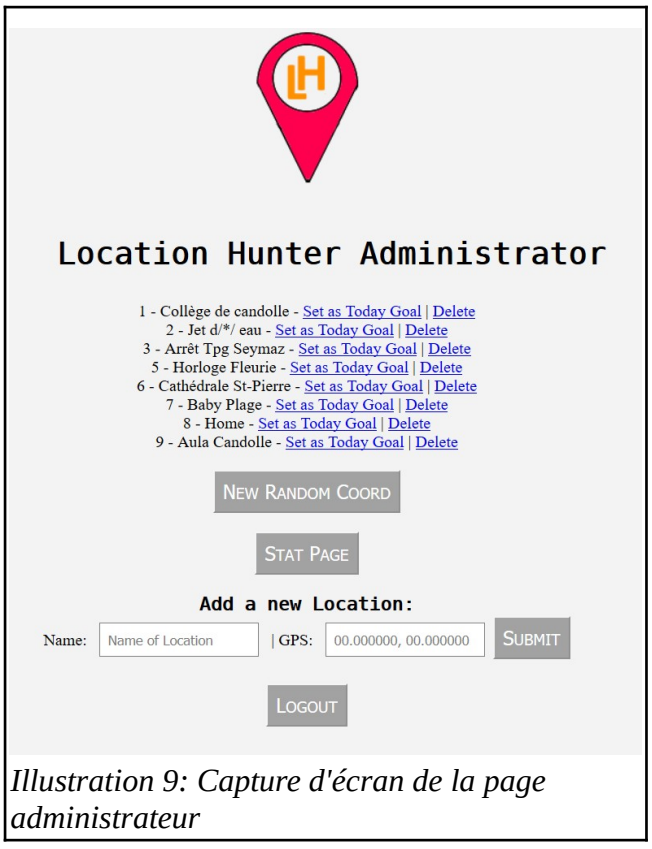

Illustration 9: Capture d'écran de la page administrateur

II. La page d'administration

Cette page contient majoritairement des manipulations de base de données, elle permet d'exécuter beaucoup d'actions comme :

- Rajouter des objectifs pour le jeu

[CODE-Annexe 1-P.12, l.51 à 54]

- En supprimer

[CODE-Annexe 1-P.12, l.45 à 49]

Et de changer l'objectif journalier de deux façons :
- De manière aléatoire avec la table *command* citée ci-dessus

[CODE-Annexe 1-P.12, l.27 à 30]

- En le choisissant parmi la liste

[CODE-Annexe 1-P.12, l.33 à 43]

III. Les statistiques

Les statistiques sont récoltées par un petit module que j'ai créé et que j'importe sur toutes les pages clients. Celui-ci récupère l'IP des clients, et grâce à une API disponible en ligne[7], je récupère la ville d'où provient la connexion. Je collecte aussi le temps que l'utilisateur a passé sur chaque page et dans quelle langue il consulte le site. Voici ce que cela donne une fois stocké pour les dix premières entrées :

id ▲ 1	ip	location	sessionID	page	pageTime	pageNext	langue	date
1	188.60.118.127	Switzerland,Geneva	bnls1kkgo9hoko0tk0luefg1c7	index	3	score	en	2018:08:20:11:25:22
2	188.60.118.127	Switzerland,Geneva	bnls1kkgo9hoko0tk0luefg1c7	score	1	account	en	2018:08:20:11:25:23
3	188.60.118.127	Switzerland,Geneva	bnls1kkgo9hoko0tk0luefg1c7	account	2	download	en	2018:08:20:11:25:25
4	188.60.118.127	Switzerland,Geneva	bnls1kkgo9hoko0tk0luefg1c7	download	2	instruction	en	2018:08:20:11:25:27
5	188.60.118.127	Switzerland,Geneva	bnls1kkgo9hoko0tk0luefg1c7	instruction	2	score	en	2018:08:20:11:25:29
6	188.60.118.127	Switzerland,Geneva	bnls1kkgo9hoko0tk0luefg1c7	score	6	error	en	2018:08:20:11:25:35
7	188.60.118.127	Switzerland,Geneva	bnls1kkgo9hoko0tk0luefg1c7	error	4	download	en	2018:08:20:11:25:39
8	188.60.118.127	Switzerland,Geneva	bnls1kkgo9hoko0tk0luefg1c7	download	2	download	fr	2018:08:20:11:25:41
9	188.60.118.127	Switzerland,Geneva	bnls1kkgo9hoko0tk0luefg1c7	download	1	instruction	fr	2018:08:20:11:25:42
10	188.60.118.127	Switzerland,Geneva	bnls1kkgo9hoko0tk0luefg1c7	instruction	1	download	fr	2018:08:20:11:25:43

Illustration 11: Capture d'écran de la table contenant toutes les données récoltées à des fins de statistique

[CODE-Annexe 1-P.32 à 33]

Le tableau de statistique ci-dessus nous permet de dessiner les diagrammes présent à la page suivante. Ceux-ci sont générés à partir des données récupérées de la base de données, analysées avec du PHP et affichées avec GoogleChart[8], une libraire de diagramme en JavaScript. Les statistiques calculées sont :
- Le nombre de pages visitées par langue
- Le nombre de pages visitées par session d'utilisateur créée.
- Le nombre de pages visitées par pays
- Le temps en moyenne passé par page
- L'évolution du nombre de visite par jour (Le seul problème avec ce graphique c'est qu'il ne prend pas en compte les jours où il n'y a pas eu de visite)

Tout ces calculs sont exécutés dans le code suivant :

[CODE-Annexe 1-P.15 à 19]

7 À l'adresse : http://www.geoplugin.net
8 Bibliothèque téléchargeable à l'adresse : https://developers.google.com/chart/

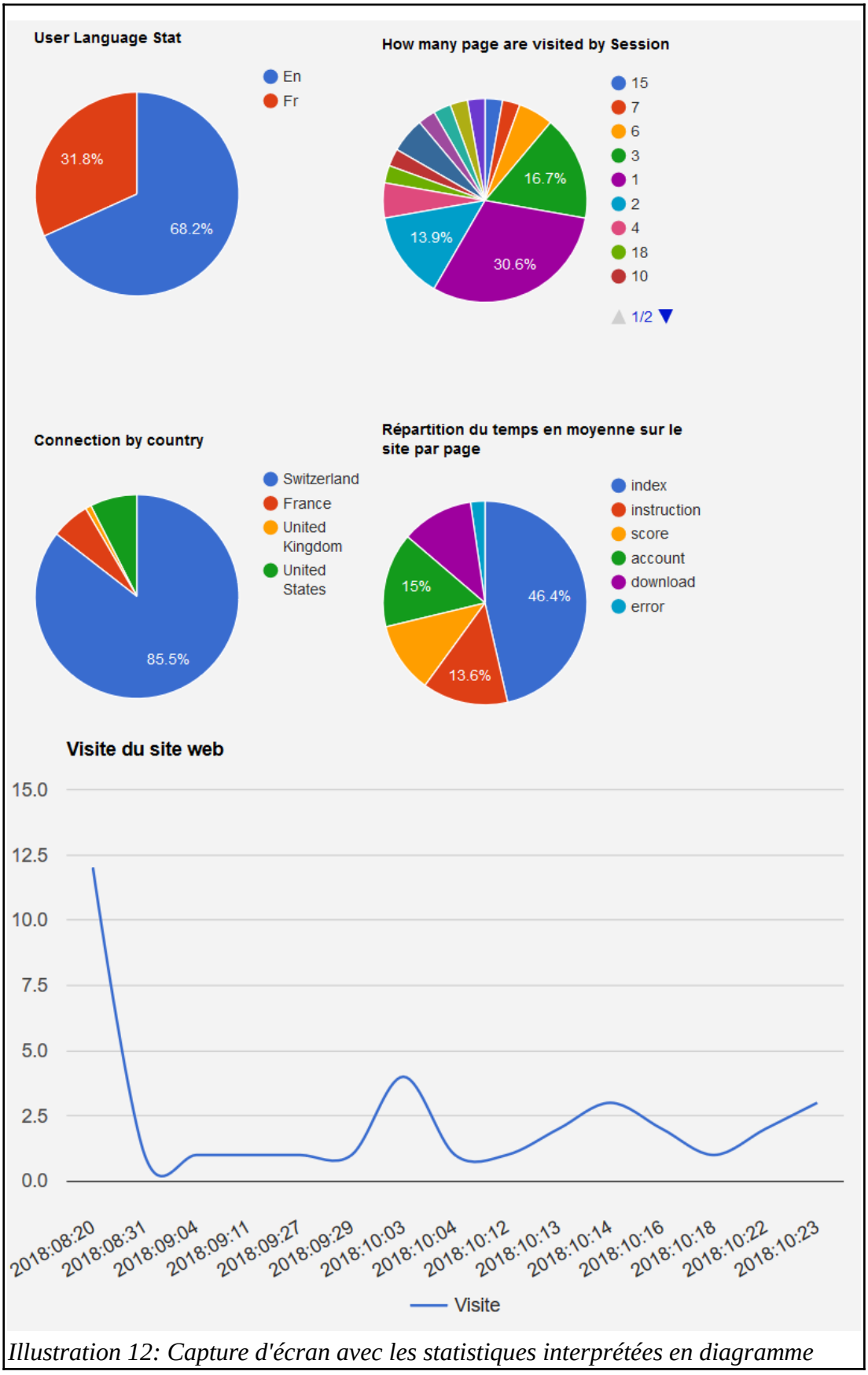

Illustration 12: Capture d'écran avec les statistiques interprétées en diagramme

f) La publication de l'application

Comme aboutissement de mon travail de programmation, j'ai décidé de publier mes applications sur le Play Store. La première étape était de créer un compte utilisateur Google et de payer les frais d'inscription au programme de développement Android qui s'élève à 25 Dollars. Ensuite, il faut ajouter les deux applications sur le site en remplissant toutes les informations et les fiches de descriptions disponibles pour le public. Il faut publier les icônes des programmes, des captures d'écran et s'il l'on veut, d'autres médias tels que des vidéos. La prochaine étape est de compiler l'application en mode release (et non en debug) et d'uploader le fichier .apk obtenu sur le store. Il ne reste plus qu'à remplir le formulaire de rating pour définir l'âge d'utilisation et de soumettre le produit à Google. Peu de temps après, nous retrouvons notre travail achevé, téléchargeable par le monde entier sur un des plus grand store d'applications mobiles au monde[9].

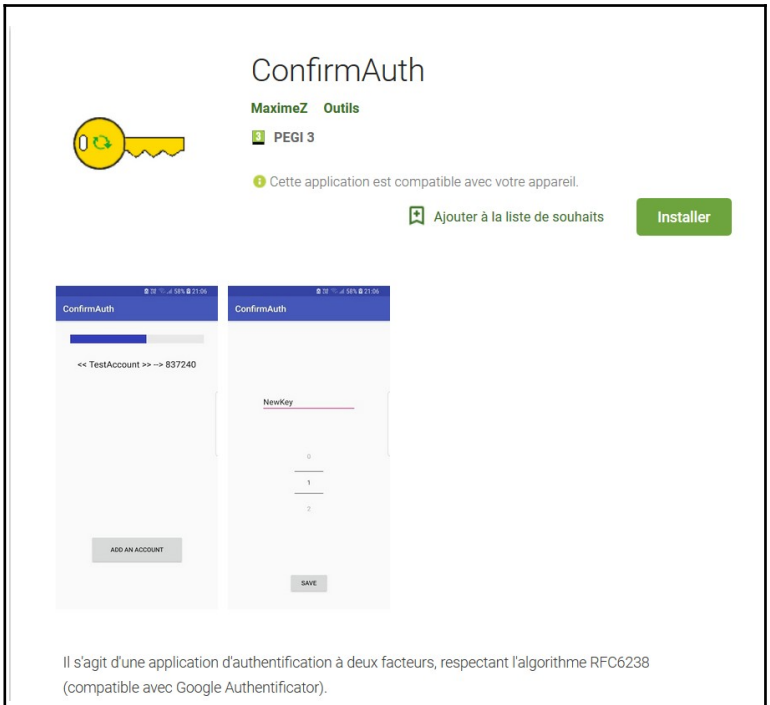

Illustration 13: Capture d'écran de la fiche du PlayStore de ConfirmAuth

Illustration 14: Capture d'écran de la fiche du PlayStore de LocationHunter

9 Disponible à l'adresse : https://play.google.com/store/apps/dev?id=7503591740588323695

Conclusion

J'ai beaucoup apprécié de créer ce projet de A à Z. Ce fut un vrai projet entrepreneurial car je suis passé de la phase de brainstorming, aux essais, à la planification, à la réalisation et pour finir à la publication. Les parties techniques les plus difficiles furent de choisir et mettre en place la communication entre l'application et la base de données. J'aurais sûrement pu me passer de l'API en python et directement faire communiquer l'application et la base de données, mais en terme de sécurité cela aurait été plus difficile à gérer car les identifiants de celle-ci auraient été stockés chez l'utilisateur et pas côté serveur. J'ai passé beaucoup de temps à la création de l'application principale et surtout l'activité de jeu, car j'ai dû réaliser de multiples tests pour récupérer les coordonnées GPS. Lors de la phase de planification, j'ai même été jusqu'à créer une petite application indépendante afin d'essayer toutes les fonctionnalités possibles avec les capteurs GPS.

Bilan Personnel

Ce travail m'a appris beaucoup de notion en programmation. Il m'a permis d'apprendre le Java, et quelques base en JavaScript. De plus, j'ai pu revoir mes acquis en PHP, HTML, CSS et approfondir mes connaissances en Python. Il m'a aussi permis d'apprendre à gérer un projet d'une grande envergure, de m'organiser afin de terminer la création dans le temps imparti.

Lexique

- Une Application Client : Il s'agit d'un logiciel qui est développé pour des utilisateurs. Elle communique généralement avec un logiciel serveur qui s'occupe de gérer tous les différents clients existant.
- Une Application Serveur : Il s'agit d'un logiciel qui gère la communication entre tous les clients.
- Une Requête : Il s'agit d'une demande d'un client vers un serveur, ce dernier lui donnera une réponse en retour.
- Hachage : Il s'agit d'une fonction qui transforme une chaîne de caractère de manière à ce qu'il soit théoriquement impossible à retrouver la valeur initiale.
- IDE : Integrated Devlopement Environement, il s'agit du terme technique pour parler d'une interface de développement. C'est un logiciel qui comprend un éditeur de texte, un compilateur et un débogueur. (Par Exemple : Android studio est un IDE Android)
- RFC : Request for comment, il s'agit d'une série de documents publiés sur le site internet de l'IETF décrivant le fonctionnement et les interactions entre des systèmes informatiques. Ceux-ci peuvent être adoptés par la suite comme norme. (Par Exemple : RFC 2068 est le standard HTTP 1.1)
- Une Activité : Chaque écran est une activité différente. (Par Exemple, dans LocationHunter il y a les activités suivantes : Menu, Paramètre, Mon Compte, et Jeu). Chaque activité contient une méthode onCreate qui permet de l'initialiser et d'appeler d'autres méthodes par la suite.
- Cast : Il s'agit d'une conversion d'ajustement, la valeur est tronquée selon le type de variable voulue. (Par Exemple i = 0.7 est une variable float, si on la cast en int, cela donne i=0)
- Concatène : Il s'agit de l'action de mettre à la suite plusieurs chaînes de caractères. (Par Exemple : a = 'Hello' ; b=' World' ; ⇒ c = a+b = 'Hello World')
- Une Méthode : Il s'agit du terme technique que l'on utilise pour désigner une fonction dans une classe.
- La Méthode Split : Il s'agit d'une méthode en Java qui permet de créer une liste en séparant un String à chaque occurrence d'un caractère.
- Un Listener : Il s'agit d'un objet qui va continuellement vérifier la réalisation d'une action. Si elle se réalise, cela engendre l'exécution d'une méthode.

- GUI : Graphic User Interface, il s'agit de l'interface graphique.
- Librairie Externe : Fichier contenant des méthodes déjà programmées. (Par exemple : la méthode pour hacher en SHA-1 est disponible dans la librairie Apache Commons Codec[10])
- SHA-1 : Secure Hash Algorithm 1, il s'agit d'une norme de hachage créée par la NSA.
- HTTP: Hypertext Transfer Protocol, il s'agit d'un protocole, utilisé pour les sites internet, qui permet à un client, un navigateur web, d'accéder au contenu d'un fichier stocké sur un serveur.
- Waypoint : Il s'agit de la notation d'un but à atteindre sur une carte.
- API : Application Programming Interface, il s'agit de méthode, de fonction et de commande, disponibles grâce à une librairie externe ou à un service web, permettant d'exécuter une action sans la programmer soi-même. (Dans le cas de LocationHunter, j'utilise le terme d'API pour le serveur python car il s'agit d'une couche programme qui m'évite de programmer et de stocker les identifiants de la base de données sur l'application Android.)
- Base de données : Il s'agit d'un endroit où l'on peut stocker des données de manière triée dans des tables.
- Table : Elles permettent de regrouper plusieurs même informations dans un tableau. (Par Exemple dans LocationHunter, il y a la table *account* avec la colonne E-mail qui contient les adresses de tout les utilisateurs)
- Radio bouton : Il s'agit de plusieurs boutons dont on ne peut en sélectionner qu'un seul à la fois.
- SharedPreferences : Il s'agit de variables internes à l'application permettant de stocker des données. Elles persistent même lorsqu'on ferme l'application.
- Tâche Asynchrone : Il s'agit d'un bout de code exécuté en parallèle du reste du programme. (Le concept équivalent en Python est le Threads)
- Threads : Les processus, sont des bouts de code exécutés en parallèle du programme principal.
- Text View : Il s'agit du nom du widget (=objet graphique) permettant d'afficher du texte dans une application Android.
- CGI : Common Gateway Interface, Il s'agit d'une autre vision des sites internet. À la place d'envoyer un fichier HTML lors d'une requête, le serveur HTTP exécute un programme et renvoie la réponse de celui-ci.
- Méthode Get : Il s'agit d'une manière de passer des données dans l'URL d'une requête.

10 Disponible à l'adresse : http://commons.apache.org/proper/commons-codec/

- Fonction d’assainissement : Il s’agit de méthode permettant d’enlever toutes les parties d’une chaîne de caractère pouvant empêcher le fonctionnement correcte d’un programme. (Par Exemple, sur les sites internets, chaque formulaire est passé dans des fonctions d’assainissement afin d’éviter l’injection de code dans celui-ci.)