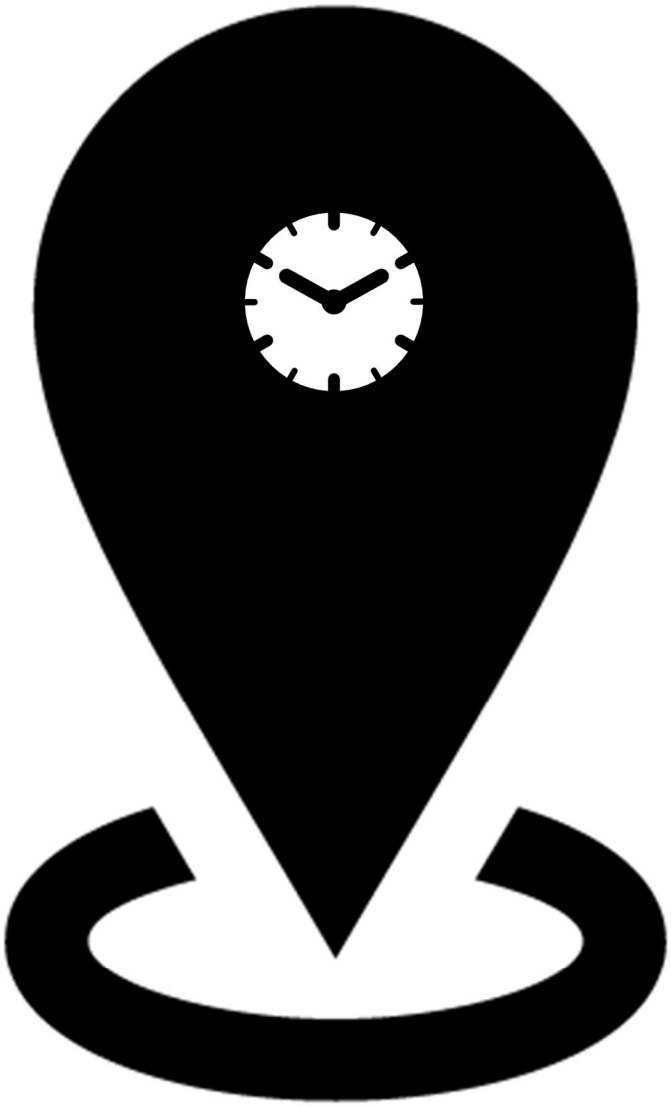


GEOSYS

Luy Karim, Serex Alexandre et Jeanneret Cyril

Rapport Android

haute école arc ingénierie
neuchâtel berne jura saint-imier le locle delémont

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

1. Introduction

Geosys est une application mobile android permettant la configuration de paramètres systèmes selon certaines régions du globe terrestre.

1.1 But

Lors de différents trajets en transport en commun, il nous arrive de créer une alarme afin de pouvoir s'endormir lors du trajet et se réveiller à destination. Le problème est qu'il ne nous est pas possible de prévoir une avance ou un retard des transports. Ainsi, rater son arrêt est tout à fait possible. De là est née l'idée de Geosys. Le principe consiste à régler une alarme non plus en fonction d'une donnée temporel mais en fonction d'une zone géographique définie par un point et un rayon. Ainsi, lorsque nous nous trouvons dans cette zone, une alarme s'active automatiquement afin de nous avertir de l'arrivée dans cette région.

1.2 Objectifs

Nous devions développer une application comprenant au moins deux des 4 points suivants.

1. Mesure de performance (temps, délai au démarrage, de réponse, de téléchargement, nombre maximal d'images par seconde, fréquence sonore maximale, etc.),

2. Persistance (stockage de données sur le téléphone).

3. Gestion de l'utilisation d'un moyen de communication : Wifi, 3G, Bluetooth. Par exemple, gérer la connexion RF (activer/désactiver, choix de canal de communication, etc.), indiquer à l'utilisateur qu'il est passé de 3G à EDGE, etc.

4. Utilisation d'au moins deux capteurs : GPS, accéléromètre, magnétomètre, camera, microphone, contexte (Wifi/ BT/ 3G/ ModeAvion/ TimeZone/ BatteryLevel/ Proximity/ Light/ etc.).

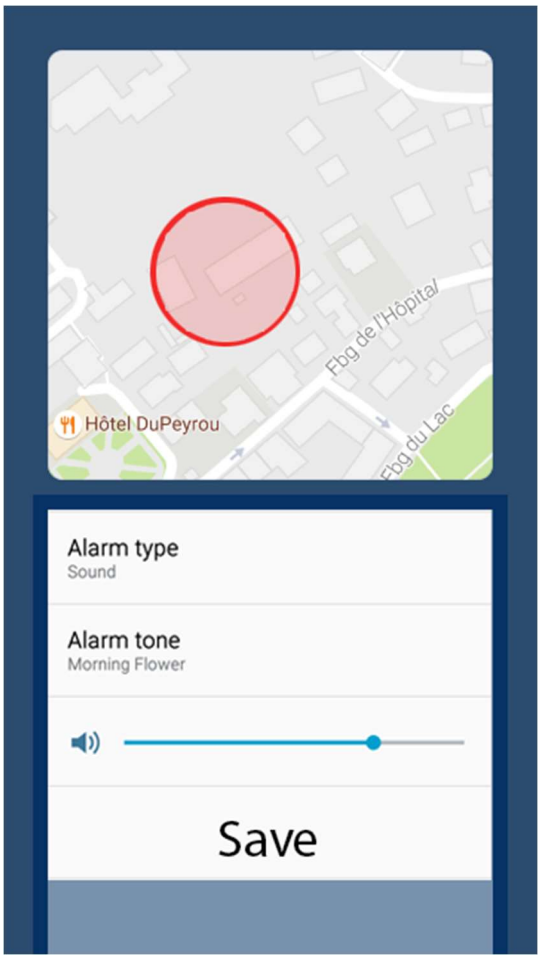
Geosys prend en compte la persistance des données au moyen de la sauvegarde des préférences de nos alarmes, dans un fichier sauvegardé sur le téléphone. L'utilisation des coordonnées GPS ainsi que des alarmes système en font l'utilisations des 2 capteurs imposés.

2. Manuel d'utilisateur

Lors du premier lancement de l'application, un message d'alerte s'affiche afin de demander à l'utilisateur d'accepter l'utilisation de sa géolocalisation. Une fois cette condition acceptée, une Google map avec un point central bleu est affichée. Ce point correspond à la situation géographique de l'utilisateur.

Pour commencer, l'utilisateur peut sur la partie inférieure de l'écran, choisir une alarme qu'il veut configurer. En cliquant dessus, une page contenant le rayon de l'alarme va s'afficher. A cet instant, l'utilisateur peut cliquer sur la map afin de définir le point de géolocalisation central de la zone. Ensuite en modifiant le slider du rayon, il définit la limite de la zone en question.

3. Structure et interface



L'interface se présente de la manière suivante :

- Un fragment pour la map occupe toute la partie supérieure de l'écran, en tout temps. L'utilisateur peut donc toujours savoir quels lieux sont occupés par un paramètre, comment ces derniers sont disposés, etc.

- Un fragment qui liste les alarmes créées qui occupe la partie inférieure de l'écran, invisible sur la maquette ici.

- Deux fragments présents dans un Bottomsheet, un élément qui apparait depuis le bas de l'écran et qui se dresse jusqu'à la moitié de celui-ci en hauteur. Le Bottomsheet alterne entre 3 états : caché, étendu, ou minimisé. Caché, le Bottomsheet est invisible et supprime les données potentiellement entrée aux préalable dans les fragments qu'il contient. Etendu, il recouvre le fragment des alarmes afin de permettre l'édition des paramètres, et minimisé, il permet de visualiser les alarmes sans perdre les données rentrées. Un morceau du bottomsheet reste visible afin de le retirer vers le haut et continuer l'édition des paramètres. C'est pratique car il permet de revoir le fragment des alarmes sans perdre les données potentiellement entrées dans les fragments qu'il contient.

4. Problèmes rencontrés

Cette section regroupe tous les problèmes que nous avons rencontrés ainsi que les solutions et les choix que nous avons mis en place pour y remédier.

4.1. Géolocalisation

4.1.1. Location android vs latlng google map

La classe location d'android définit un point de localisation en longitude latitude, qui peut être donnée en plusieurs format :

- En degrés « [+-] DDD.DDDDD » ou D indique les degrés
- En minutes « [+-]DDD:MM.MMMMM » ou D indique les degrés et M les minutes

- En secondes « DDD:MM:SS.SSSSS » attention identique plus les secondes mais ainsi, le signe disparait car les précisions apportés aux coordonnées sont déjà précise.

Ainsi en fonction du type de format que l'on choisit, les données devrons être convertie.

Afin de pouvoir visualiser une carte cohérente et pouvoir afficher notre géolocalisation nous avons utilisé Google maps. Le problème c'est que la classe « LatLng » qui nous permet de trouver un point sur notre map ne prend pas les mêmes formats que le point retourné par notre application. Il nous a fallut convertir ceux-ci au moyen des format Android. Une fonction « UpdateLocation() » nous permet de changer le format et ainsi le convertir afin que la longitude / latitude corresponde à celles demandé par le fragment GoogleMap.

Un dernier point nous a posé problème, c'est l'inversion de la latitude/longitude entre les objets, ce qui nous a empêché de voir notre point un certain temps. Il se situait en Ethiopie, ce qui nous indiquait que le point existait mais plutôt qu'il n'était pas au bon endroit.

4.1.2. Rafraichissement des points sur la map

Nous devions rafraichir le point de l'utilisateur, soit sa coordonnée, ainsi que les points qui représentent les alarmes et le tout en temps réel. Lorsque les points sont sauvegardés, il est facile de récupérer ceux-ci afin de les actualiser sur la Google Map, Mais à la création d'un point, nous ne le sauvegardons pas car il n'est attribué à aucune alarme. Le problème étant qu'il nous faut quand même le rafraichir. Pour résoudre ce problème, nous avons créé une méthode qui dessine les marqueurs avec celui de l'utilisateur et on l'appel quand la location de l'utilisateur change et quand il appuis sur la map pour créer un marqueur. Ainsi il reste temporairement sauvé à cet instant. Puis si l'utilisateur désire en faire une alarme, nous le sauverons dans le fichier qui sera chargé lors du démarrage de l'application.

4.2. Persistance des données

Un problème nous a fait perdre un peu de temps. Lorsque nous sauvegardions le fichier sur un émulateur, celui-ci se sauve avec des droits de lecture seul. Ainsi lorsque nous essayons de le récupérer afin de le modifier, il nous était impossible de le faire. Ceci à fait cracher notre application à de nombreuse reprise mais il s'avère que sur un appareil physique, le problème disparait.

4.3 List fragment, Bottomsheet et logique d'interaction

La logique d'interaction définit le flux d'évènements que l'utilisateur doit générer pour pouvoir utiliser l'application. Le Bottomsheet a posé quelques problèmes de compatibilité et surtout de transitions entre les états étendu, caché et minimisé. Le souci était que les états minimisé et cachés n'étaient pas distincts parce que le XML décrivant le Bottomsheet n'était pas conforme à ses besoins. Pour fonctionner, le Bottomsheet est contenu dans un FrameLayout directement enfant de la racine du XML, mais ceci implique que l'on doit limiter la hauteur maximale du Bottomsheet à la moitié de l'écran par le biais du Java.

Un autre problème rencontré est la gestion des évènements, de la transaction de ceux-ci entre un ListFragment (un élément indépendant contenu dans un fragment à part entière) et l'activité principale, et des paramètres qu'on peut y passer. Les ListFragment générés par Android Studio sont très verbeux, et plusieurs fichiers distincts sont créés pour faire fonctionner le fragment. Ceci pose les difficultés suivantes :

- Comprendre d'où les objets du fragment sont générés et comment y passer des arguments
- Comment gérer le clic d'un élément du ListFragment et repasser l'évènement plus loin

- Comment retourner des données d'un élément unique au GoogleFragment

Ces questions sont tout autant de problèmes qui sont venus prendre pas mal de temps au développement de l'application. Pour y répondre, nous avons procédé respectivement ainsi :

- Les arguments sont passés au éléments du fragment par le biais des Bundle, à leur instanciation
- Un ListFragment peut, s'il possède une référence sur le contexte parent (l'activité), récupérer une référence sur le GoogleFragment
- Un ListFragment peut retourner des données par le biais d'une HashMap présente dans tous les ListFragment, récupérée à la gestion de l'évènement par le biais d'une interface implémentée dans l'activité principale. Cette dernière profite du polymorphisme et peut récupérer les données de la HashMap.

5. Avantages et inconvénients de la structure du code

Avantages :

- Le code fait beaucoup usage de polymorphisme et d'abstraction. On peut donc ajouter facilement d'autres types de paramètres gérés par le GPS sans tout recoder.
- L'interface est très simple à prendre en main. L'usage du Bottomsheet est naturel et facile à utiliser, tout en laissant à l'utilisateur un maximum d'informations facilement accessibles à l'écran.

Inconvénients :

- Le code généré pour la gestion de plusieurs ListFragment par Android Studio est très verbeux, et on se retrouve avec beaucoup de classes qu'on touche à peine et qui rendent le pipeline d'interactions difficile à suivre entre les différents composants de l'application.
- L'ajout de nouvelles options systèmes (vibreur par exemple) activables par GPS pourrait être amélioré en récupérant automatiquement les différents paramètres de l'option en question plutôt que de les hard coder dans les fragments et l'interface.
- La GoogleMap efface tous les éléments pour tous les réafficher lors de mise à jour de ceux-ci. Plus que les soucis de performance qui sont négligeables pour l'utilisation qu'on a de l'application, ça pose problème lors de la mise à jour en temps réel d'un rayon sur la map par l'usage d'un slider par exemple. On voit que les positions affichées s'effacent et se réaffichent, ce qui est bien évidemment un problème pour l'interface.

6. Conclusion

L'application n'est pas parfaite. Il pourrait être plus facile d'ajouter de nouveaux paramètres pour faire avancer l'application plus loin, et le pipeline des données et des évènements entre les composants de l'application est difficile à suivre, et donc à tenir à jour, à débugger, à faire scale, etc.

Malgré ces quelques défauts, l'application fonctionne bien en terme d'interface, et reste relativement facilement adaptable pour la suite. Elle fait bien usage de son support mobile également, par l'utilisation du GPS.

Dans l'ensemble, ce projet était pour nous un moyen d'entrevoir l'étendue des possibilités offerte par la plateforme mobile d'Android, ses capacités comme ses limites.