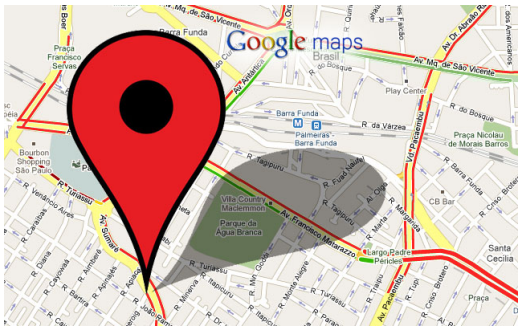
École Polytechnique Montréal

INF8405 : Informatique Mobile

TP2 : Intégration de Maps



Philippe TROCLET 1815208
Alexandre MAO 1813566
Fabien BERQUEZ 1800325

Soumis à : M. Aurel Josas RANDOLF
Soumis le : 23 Mars 2016
Session Hiver 2016

1 Introduction

L'objectif du TP n°2 était de développer une application destinée à faciliter les rencontres entre membres d'un groupe. Plus précisément, les requis de l'application étaient centrés autour de l'utilisation de différentes ressources, comme le GPS, le calendrier des membres, des API externes comme Google Maps ou Google Places. L'application devait permettre à chaque personne du groupe de définir son profil (nom, courriel, lieux de rencontre préférés) et s'il était organisateur ou non. Une fois ceci fait, et le groupe constitué, l'application, en se basant sur les lieux de rencontre préférés et la position géographique des membres du groupe, proposait trois options de rendez-vous. Les membres du groupe devaient alors choisir chacun l'option qui leur convenait le mieux. Le résultat était transmis à l'organisateur qui pouvait alors définir un lieu et une heure, d'après les disponibilités des membres du groupes, ainsi qu'un petit texte présentant le choix qu'il a retenu. Les informations étaient alors transmises à tous les membres du groupe afin que ceux-ci disposent des informations nécessaires pour aller au rendez-vous.
D'un point de vue technique, l'application a pour requis d'être exécutable sur les plateformes Android supportant l'API 18 et supérieures (jusqu'à la version 23 pour la plus récente), c'est à dire des versions d'Android 4.3 à Android 6.0. L'application doit être capable d'être utilisable, avec des fonctionnalités réduites, même lorsque la connexion au réseau cellulaire ou Wi-Fi n'est pas possible, ce qui a eu un impact sur les choix d'implémentation réalisés lors du développement de cette application.

2 Présentation générale

Une première fonctionnalité, qui nous semblait importante est la capacité de s'authentifier sur n'importe quel mobile. C'est-à-dire que l'utilisateur peut posséder plusieurs appareils mobiles (plusieurs portables ou encore un portable et un tablette), et il pourra s'authentifier avec le même compte sur tous ces appareils. L'utilisateur peut, sur la page de login, s'identifier via un mot de passe et l'adresse mail associée à son compte. Lors de la création du compte, ces derniers sont stockés sur le serveur. Au moment du login, on interroge la base de données pour savoir si le mot de passe rentré par l'utilisateur est bien celui qui est associé à ce compte. Si l'utilisateur utilise un nouvel appareil, ses données seront téléchargées depuis le serveur.

Il est également possible de créer, sur la page associée au groupe, un nouveau groupe et de le sauvegarder sur le serveur. Ce serveur contient toutes les données relatives aux utilisateurs et aux groupes. Cela permet potentiellement à chaque utilisateur de récupérer les préférences de tous les autres et donc d'organiser des rencontres en fonction des préférences de chacun.

Afin, dans une certaine mesure, de permettre le fonctionnement hors-ligne, une base de données SQLite est également présente. Elle permet de récupérer les données des autres utilisateurs afin de potentiellement consulter leur profil ou encore la fiche de description des groupes. De même, elle permettrait de consulter les événements déjà prévus pour un groupe, ainsi que de consulter leur description (si celle-ci a été rentrée). Cette fonctionnalité permettrait de mieux se préparer pour les événements à venir.

2.1 Fonctionnalités disponibles

Les fonctionnalités disponibles sont :

— La possibilité de s'identifier avec un login (adresse courriel) et un mot de passe si nous avons un compte déjà créé.
— La possibilité de créer un compte, en passant dans la partie inscription qui nous permet de créer un compte avec en rentrant différentes informations (nom, prénom, adresse courriel, nom du groupe, les préférences, la possibilité de mettre une photo ou d'en prendre une), en sachant que si le groupe n'existe pas un nouveau groupe est créé.
— Toutes les fonctionnalités liées aux groupes : inscription des utilisateurs dans un groupe, affichage des informations etc. On peut voir dans la partie Localisation une carte présentant notre position, celle des utilisateurs du groupe (un marqueur de couleur différente par personne).
— Dans le profil, les informations sont disponibles et modifiables : nom, courriel, préférences et photo.
— Sur l'écran d'un événement, on voit la description de l'événement, le nom des participants, et un bouton est présent pour que l'organisateur puisse créer et modifier la description de l'événement
— Les cartes sont cliquables sur le groupe.
— Pour se déconnecter, un bouton est disponible dans le profil.
— Le système de vote est accessible en cliquant sur un événement. Un bug que nous n'avons pas réussi à résoudre est présent. Nous avons donc créé un affichage par défaut pour le moment.

3 Présentation Technique

3.1 Gestion de la batterie

La gestion de la batterie a été prise en compte, nous avons mesuré l'impact énergétique d'une requête (Dans notre cas, nous nous sommes intéressés à la récupération d'un groupe depuis le serveur). Il en résulte que le dit impact n'est pas mesurable, la variation de la batterie pour une seule requête est trop infime pour être mesuré. Cela vient du fait que les échanges de données entre les utilisateurs et le serveur représente de petits paquets. On est loin du traffic lié au visionnage d'une vidéo sur internet par exemple, aussi, pour que l'impact soit perceptible il faudrait récupérer beaucoup plus de données. Ce qui n'est pas réaliste dans le cadre de notre application.

3.2 Partie Serveur

Les différents utilisateurs doivent communiquer entre eux afin de pouvoir voter, mais aussi pour connaître la position de chacun des utilisateurs. Une telle communication nécessite la présence d'un serveur connu de tous. Pour mettre en place une serveur nous avons choisi d'utiliser le service cloud *Firebase*. Les données sont stockées sous le format JSON sur le serveur. On peut alors les récupérer via leur id (générés par l'API Firebase).

Cette solution a plusieurs avantages :

1. la présence d'une API déjà testée et éprouvée, ce qui limite le nombre de bugs. Et permet de ne pas réinventer la roue.
2. la présence de mécanismes pour écouter des données sur le serveur et être informé en cas de modifications.
3. une grande communauté d'utilisateurs, ce qui facilite la recherche d'information en cas de déboguage.

Ainsi, les coordonnées (longitude/latitude) de chaque utilisateur peuvent être mises à jour sur le serveur en temps réel (ou presque). Il suffit pour cela d'envoyer ces nouvelles coordonnées sur le serveur à chaque fois que le *callback* associée au changement de position dans *Google Maps* est appelée (l'envoi se ferait dans le dit *callback*). Les autres membres du groupe n'auraient qu'à écouter ces coordonnées sur le serveur afin d'être informé de tout changement de position.

Cette approche est un peu gourmande en terme de réseau, aussi il est possible de mettre à jour les données uniquement à des moments jugés opportuns. On obtient alors une meilleure utilisation du réseau mais on perd le temps réel.

Nous voulions abstraire au maximum le fonctionnement du serveur ainsi que la récupération des données géographiques. Aussi, nous encapsulons ces mécanismes dans une classe *ConnectedMap*. Cette classe se charge d'implémenter les *callback* récupérant les informations de position. Elle se charge également d'instancier la classe *firebaseBD*.

Cette classe, se charge d'interfacer l'API *Firebase* afin de fournir simplement toutes les fonctionnalités nécessaires pour ce projet. Elle permet entre autres, de récupérer un utilisateur ou un groupe depuis le serveur. Les réponses de la base de données étant asynchrones, le comportement attendu lors de la récupération des données doit être encapsulé dans une *callback* (qui sera appelée après récupération des données). Par exemple, si on veut récupérer un utilisateur, si on a une classe *User* qui contient tous les attributs nécessaires pour identifier l'utilisateur. On encapsulera sûrement cette classe dans une autre, disons *LocalUser*. La récupération se fera alors en plusieurs étapes :

1. création d'un objet *LocalUser*
2. création et assignation d'une *callback* à l'objet précédemment créée.
3. appel d'une méthode du type *getUser(id, localUser)*
4. lorsque l'objet est récupéré du serveur, la *callback* est appellée (on peut par exemple afficher une fenêtre qui indique que les données ont bien été sauvegardées)

3.3 Gestion de la Base de données SQL

Si la partie serveur utilise Firebase pour stocker les différentes données, il a fallu réfléchir à un moyen alternatif de disposer des dernières données à jour en local, sur l'appareil mobile de l'utilisateur, au cas où celui ci ne disposerait pas d'une connexion Internet à un moment donné de son utilisation de l'application. Nous avons opté pour une solution commune à ce problème sous Android, à savoir l'existence d'une base de données SQLite sur l'appareil.
On peut notablement remarquer que la majorité des objets encapsulant les données de notre application sont doublés d'une autre classe gérant la table associée à ces objets dans la base de données SQLite utilisée. Ces classes ont pour objectifs, entre autres, la définition des tables, la définition des requêtes de récupération, d'insertion et de mise-à-jour des données contenues, ainsi que de créer les objets associés pour les récupérer sous une forme plus adaptée à nos besoins.
Lors de la conception de ces bases de données, il est important de noter quelques points pris particulièrement en compte :

— La création de colonnes numériques destinées à être des identifiants pour lier les éléments de différentes tables et permettre plus facilement de rattacher les éléments allant ensemble dans chacune des tables. On peut noter par exemple que la table Groupe réfère les utilisateurs en faisant partie par leur user_id, ce qui permet de réaliser des jointures et récupérer en une seule requête les informations sur les différents membres d'un même groupe.

— Cette base de données doit nous permettre de représenter fidèlement le dernier état connu de la base de données en ligne. Elle est donc mise à jour régulièrement et permet de réaliser de manière assez transparente les traitements que nous voulons effectuer à un moment donnée, que la base de données en ligne soit accessible ou non.

4 Difficultés rencontrées

Comme tous les projets intéressants, celui-ci n'a pas été sans écueils. Ce projet comporte de nombreuses facettes, il doit posséder une base de données internes, un serveur et pouvoir communiquer avec les API google afin de récupérer des données sur la position de l'utilisateur courant et sur les lieux (cafés, restaurants, bars) proches d'une certaine position. Il y a donc des domaines différents à explorer. Le plus simple est alors de répartir ces domaines entre les membres du groupe. Mais s'il chacun travaille trop de son côté, il devient compliquer d'intégrer les différents travaux respectifs. Le problème est donc avant tout un problème de communication.

Cette difficulté est d'autant plus accentuée que ces facettes ne sont pas indépendantes : la base de données doit connaître les positions afin d'informer les autres utilisateurs lorsqu'ils en font la demande. De ce fait, Les différentes facettes seront très liées. Et leurs codes respectifs vont, à terme, s'entrecroiser. Ce qui rend leur intégration beaucoup plus compliquée.

Un autre problème rencontré a été beaucoup plus contraignant, car il a impacté fortement notre travail et nous en a même fait perdre une partie non négligeable (environ une journée de travail). Il s'agit du dépôt Git utilisé, qui a rencontré un problème majeur nous ayant fait perdre les derniers commits réalisés au cours d'une journée relativement chargée. Alexandre a notamment été fortement handicapé par ce problème car il disposait des dernières versions testées des classes traitant de la base de données. Le retard accumulé dans ce domaine a fini par se répercuter sur le développement d'autres fonctionnalités qui se retrouvaient en suspens, car dépendantes de la présence d'une base de données locales fonctionnelle. Nous avons essayé au maximum de limiter les dommages occasionnés sur notre travail, mais la marge de temps était assez restreinte. Certaines fonctionnalités des requis en ont probablement pâti.

Enfin, nous avons rencontré des difficultés quand au choix de la solution pour le serveur. *Google Drive* était par exemple proposé ou en tout cas mentionné dans le sujet, mais ce service ne nous paraissait pas approprié pour un tel usage. L'API Android ne permet tout simplement pas de récupérer des fichiers qui ont été créés par le programme faisant appel à l'API. De plus, le partage des fichiers entre différents utilisateurs est laborieux. Et l'utiliser comme serveur demanderait de réimplémenter de nombreux mécanismes. en particulier il faudrait réimplémenter des mécanismes d'écoute et de mise à jour. Nous avons donc eu du mal ensuite à nous décider parmi les nombreuses alternatives gratuites disponibles en ligne pour effectuer ce genre de tâche. Nous avons finalement opté pour Firebase, mais nous avons perdu une certaine quantité de temps qui nous aurait été utile dans la dernière ligne droite de ce TP.