
**I**nstitut **S**upérieur d'**I**nformatique de **M**odélisation et de leurs **A**pplications

Campus des Cézeaux
24 avenue des Landais
BP 10125
63173 AUBIERE Cedex.

**L**aboratoire d'**I**nformatique, de **M**odélisation et d'**O**ptimisation des **S**ystèmes

Complexe scientifique des Cézeaux
63173 AUBIERE cedex - FRANCE

Rapport d'ingénieur
**Projet de 3ème année**
Filière : Systèmes d'Information et Aide à la Décision.

# Réalisation d'un GPS communautaire

*Présenté par* : **Oussama SEDRATI** et **Radouane ROUFID.**

*Responsables ISIMA* : **Philippe LACOMME**
**Benjamin VINCENT**
**Libo REN**

Date de soutenance : **05 mars 2014.**


**I**nstitut **S**upérieur d'**I**nformatique de **M**odélisation et de leurs **A**pplications

Campus des Cézeaux
24 avenue des Landais
BP 10125
63173 AUBIERE Cedex.

**L**aboratoire d'**I**nformatique, de **M**odélisation et d'**O**ptimisation des **S**ystèmes

Complexe scientifique des Cézeaux
63173 AUBIERE cedex - FRANCE

Rapport d'ingénieur
**Projet de 3ème année**
Filière : Systèmes d'Information et Aide à la Décision.

# Réalisation d'un GPS communautaire

*Présenté par* : **Oussama SEDRATI** et **Radouane ROUFID.**

*Responsables ISIMA* : **Philippe LACOMME**
**Benjamin VINCENT**
**Libo REN**

Date de soutenance : **05 mars 2014.**

# ***Remerciements***

Nous tenons tout d'abord à remercier nos tuteurs de projet, Monsieur Philippe LACOMME, Monsieur Benjamin VINCENT et Monsieur Libo REN qui ont toujours été disponibles pour répondre à nos interrogations et pour nous avoir encadrés tout au long de la réalisation du projet.

# ***Résumé***

Afin de faciliter la gestion de rendez-vous, il nous a été proposé un projet intitulé « Réalisation d'un GPS communautaire ».

Ce projet a comme objectif final la création d'une application Android permettant à ses utilisateurs de se rencontrer de manière rapide et optimale en utilisant un système de gestion intelligent guidant chaque utilisateur avec un point de rendez-vous calculé automatiquement par l'application à partir des positions des utilisateurs.

Pour réaliser le projet, nous avons conçu dans un premier temps un modèle UML et un MPD (Modèle Physique de Données) afin de modéliser la base de données. Ensuite, nous avons développé des fonctions dans notre Web Service, puis nous nous sommes focalisés sur la partie mobile plus précisément les parties permettant d'appeler les fonctions du Web Service ainsi que les interfaces que nous avons intégrées sur Android.

La développement de cette application a été faite en utilisant les environnements de développement NetBeans, Eclipse et Java, SQL, JavaScript, HTML, CSS comme langage de programmation utilisant la spécification Java EE. En ce qui concerne la gestion de la base de données nous avons utilisé MySQL Workbench.

**Mot clés :** UML, Android, Java, SQL, JavaScript, HTML, CSS, Java EE, Eclipse, NetBeans, MySQL Workbench.

# *Abstract*

We often need to meet family or friends and to do that, in first we have to prevent the people we want to see, then we have to find the ideal place where we need to be and stay in touch to track the progress of the movement of each of us. All these steps can demotivate people and influence them to renounce to the appointment. A project was proposed by Mr. Lacomme aimed to implement a solution that will allow people to easily meet.

The aim was to provide a management interface that allows users to handle their own properties, to create and manage meeting groups and to be guided to a meeting point automatically calculated by the application using a Java centroid algorithm. For the server side, we modeled and developed a MySQL database using JPA associated to DAO as access and we used the SOAP protocol to create the link with the client side. On the Android part, we combined Java, HTML and JavaScript associated to Google Maps API. Because of the inconsistency between the Android layout and the Google Maps API, we opted to perform the display with HTML that provides a better rendering and a fast calculation time, in addition to that we improved the calculation of the centroid which provides better accuracy and the management interface has been improved to be easily usable.

About 80% of the application has been developed and can be used by users to organize appointments and meet friends further improvements in attractiveness of the application might be by adding a chat-messaging for the users.



**Keywords:** GPS, Centroid, Meeting, Android, SOAP web service, MySQL, JPA

# Table des matières

## Contenu

# Tables de figures

# Introduction

Dans le cadre des projets proposés en troisième année à l'ISIMA, nous avons choisi le projet consistant à réaliser un GPS communautaire consistant à calculer dynamiquement le barycentre d'un groupe d'utilisateur et faire converger chacun d'eux à ce point-là tout en offrant une interface ergonomique de gestions d'utilisateurs et de meeting (rendez-vous).

L'application en question devait offrir plusieurs fonctionnalités : l'inscription et la connexion à l'application et la gestion du profil utilisateur et de ces groupes et bien sûr l'affichage de la carte dans laquelle le guidage s'effectuera.

De plus, il fallait gérer la sécurité de l'application et plus exactement le Web Service. Mais avant, il était primordial d'établir le périmètre de développement en réalisant un cahier de charge et de faire une analyse précise pour connaitre les besoins des différents utilisateurs. Et ensuite, nous devrions concevoir la solution tout en essayant de satisfaire le cahier de charge.

Dans un premier temps, nous présenterons l'objectif du projet, ensuite nous argumenterons notre choix de modélisation de la base de données, puis nous détaillerons les différentes étapes de réalisation du projet, pour finir par exposer les résultats obtenus ainsi que l'interface graphique.

# I. Présentation du projet

Nous présenterons dans cette première partie le problème auquel s'intéresse notre projet, puis nous exposerons nos analyses du projet.

## 1. Présentation du Limos

Le LIMOS : Informatique, Modélisation et Optimisation des Systèmes, créé en 1995, en accompagnement de l'ISIMA : Institut d'Informatique, de Modélisation et des Applications, a alors rassemblé 23 enseignants chercheurs, dont 7 nouvellement recrutés, autour des STIC pour l'organisation. Réparti sur 3 établissements (UBP : porteur, IFMA et UdA : secondaires), reconnu EA en 1996 et en expansion rapide, il s'est alors fixé quelques grands objectifs :

- Fédérer les STIC sur le site Clermontois et être partie prenante du développement socio-économique du site
- Acquérir un rayonnement au plan national et international
- Afin de parvenir à ces objectifs, il a mis en place une politique de
- Incitation à la publication en revues, soutien aux jeunes vers l'HDR, mutualisation des équipements lourds
- Ouverture des recrutements vers l'extérieur et ouverture internationale
- Forte articulation avec les entités de formation professionnelle
- Equilibre entre fondamental et appliqué et mise en place de liens forts avec le tissu socio-économique local

Le LIMOS a été associé au CNRS en 2000, relevant successivement des départements SPI, STIC, MIPPU, ST2I et INS2I.

Depuis janvier 2012, le LIMOS a une antenne à l'Ecole Nationale Supérieure des Mines de Saint-Etienne (ENSM.SE, 13 enseignants chercheurs permanents). Maintenant il y a 4 tutelles suivantes : le CNRS, l'Université Blaise Pascale, l'Université d'Auvergne et l'ENSM.SE, et un partenaire institutionnel : IFMA.

[Présentation LIMOS]

## 2. Contexte et objectifs

Supposons que nous voulions rencontrer des collèges de travail à un lieu donné, pour ce faire tout participant à cette réunion utilisera normalement son smartphone pour localiser le lieu de rendez-vous. Malheureusement, il se peut que tous les collèges soient au lieu de rendez-vous sans se rencontrer et donc se trouvent obligé de se rappeler pour déterminer exactement la position de chacun et donc pour se rencontrer toutes ces personnes ont perdu beaucoup de temps et le temps c'est l'argent ! Et afin d'optimiser ce temps, nos tuteurs Monsieur LACOMME, Monsieur VINCENT et Monsieur REN nous ont proposé de créer une application mobile qui va faciliter à ces utilisateurs.

L'objectif de cette application est de pouvoir après identification à l'application ajouter des amis dans un groupe et l'application calcule le barycentre de la position de tous les membres et ensuite affiche l'itinéraire que doit chaque utilisateur suivre afin d'accéder à ce point de rencontre.

Ce qui est aussi intéressant dans cette application, c'est la possibilité de voir la position de vos amis sur la carte et donc vous permettra d'appeler un ami dans le cas où il s'éloigne du lieu de rendez-vous. Le travail à faire donc résider dans le fait de créer à la fois une application Android qui permettra de gérer les utilisateurs et les groupes et aussi visualiser la position de l'ensemble des membres d'un groupe dans le cas d'une prise de rendez-vous.

## 3. Analyse du problème

Pour créer notre application nous avons choisi d'utiliser comme environnement de développement : *ECLIPSE* et *NETBEANS*. Le premier nous a permis de *développer* sur Android et *NETBEANS* pour créer notre *Web Service* qui contient les fonctions nécessaires au développement de l'application.

Nous avons aussi utilisé *MySQL Workbench* qui est un logiciel de gestion et d'administration de bases de données MySQL qui permet via une interface graphique intuitive, de créer, modifier ou supprimer des tables, des comptes utilisateurs et d'effectuer toutes les opérations inhérentes à la gestion d'une base de données.

Dans le cadre de ce projet nous avons programmé notre partie mobile grâce aux langages Java, JavaScript, JQuery Mobile, Html et CSS alors que pour le *Web Service* nous avons utilisé la spécification J2EE.

La figure 1 présente les interactions entre les différents outils utilisés.

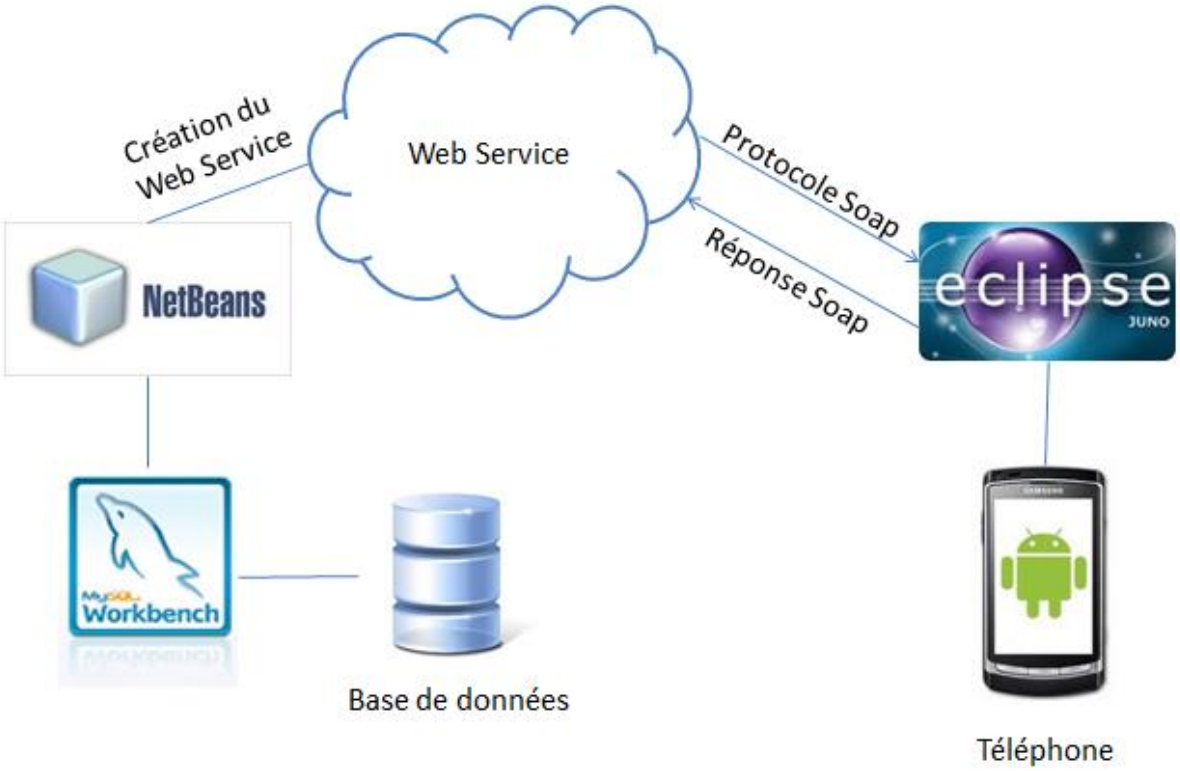



Figure 1 : Interactions entre les différents outils utilisés

En effet, à partir de cette figure nous remarquons que l'IDE *NetBeans* est relié à la base de données via *MySql Workbench* qui nous permettra de visualiser et modifier manuellement la base de données. Grâce à *NetBeans* nous allons donc créer un *Web Service* contenant un grand nombre de fonctions. D'autre part le protocole *Soap* nous permettra de faire des appels aux fonctions implémentées sur le Web Service et ainsi les utiliser au sein du téléphone.

Afin de mieux comprendre l'utilité de notre application nous proposons les diagrammes suivants :

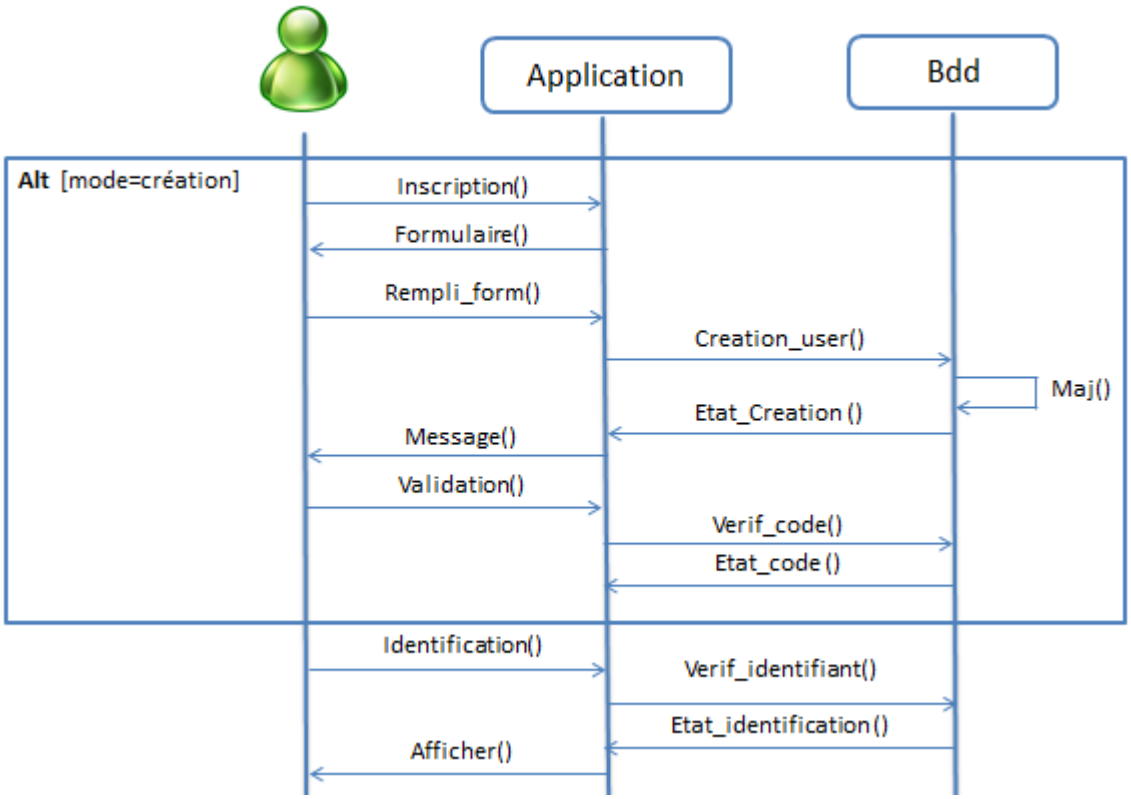



Figure 2 : Diagramme de séquence de la création d'un compte

*Explication du diagramme* :

Lorsqu'un utilisateur installe l'application pour la première fois il devra s'inscrire. Le processus d'inscription est assez simple puisqu'en cliquant sur le bouton d'inscription l'utilisateur devrait remplir un formulaire, si le formulaire est bien renseigné l'utilisateur va être inséré à la base de données et recevra un message lui indiquant un code secret qu'il devra saisir dans l'application pour activer son compte. Après validation du compte l'utilisateur pourra après se connecter et accéder à la page d'accueil de l'application. Le principe de ce diagramme de séquence est bien illustré par la maquette de la figure 3 :

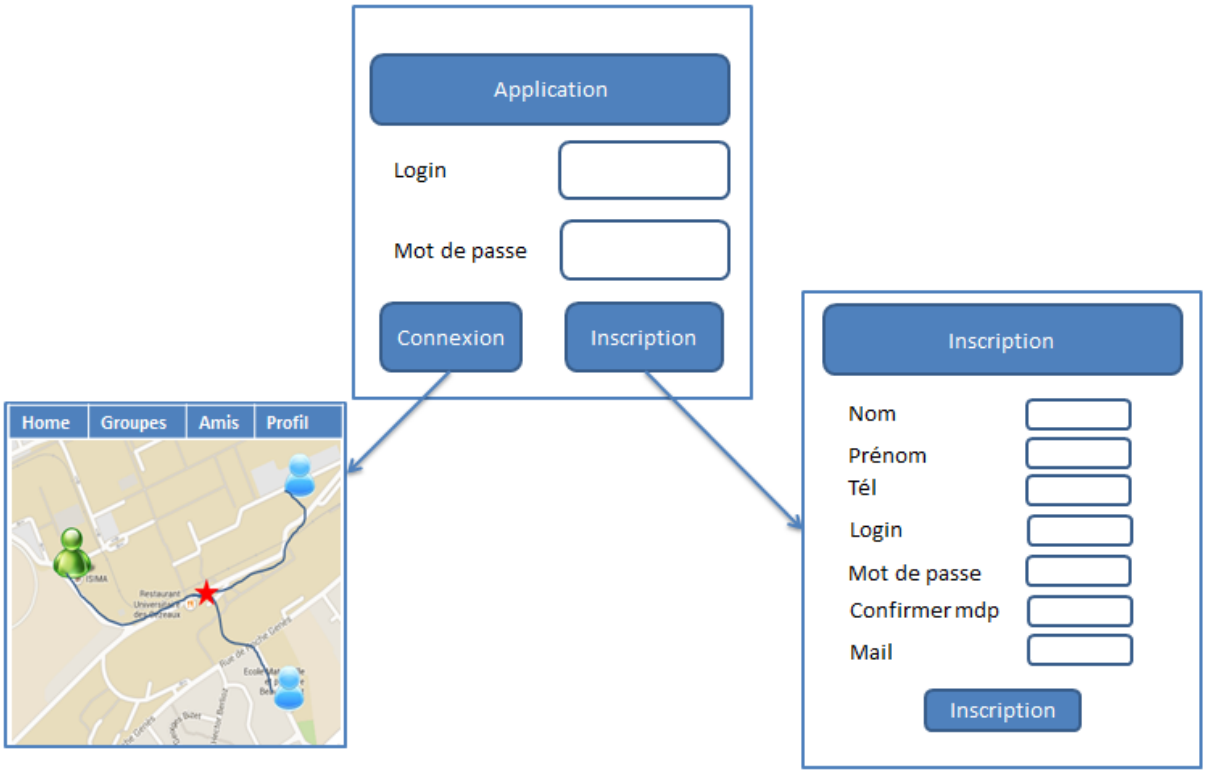



Figure 3 : Maquette approximative de l'écran d'accueil

Comme nous pouvons le voir sur la figure 3, la page d'accueil est composé d'un menu de quatre onglets :

- Home pour accéder à la page d'accueil.
- Groupes pour gérer les groupes.
- Amis pour la gestion des amis.
- Profil pour gérer son profil.

***« Home » :***

Nous pouvons illustrer la page d'accueil par l'image de la figure 4 :


Figure 4 : Principe générale de l'application

En effet, la page d'accueil représente exactement le principe général de l'application. Puisque l'utilisateur pourra visualiser la position de ses amis qui appartiennent au même groupe que lui ainsi que leurs positions et bien sûr la position du point de rendez-vous. La carte affiche aussi le trajet que devra suivre chaque membre du groupe pour se retrouver.

***« Groupes » :***

Lorsque l'utilisateur de l'application clique sur l'onglet « Groupes » il accédera à une page selon son choix comme le montre la maquette de la figure 5 :

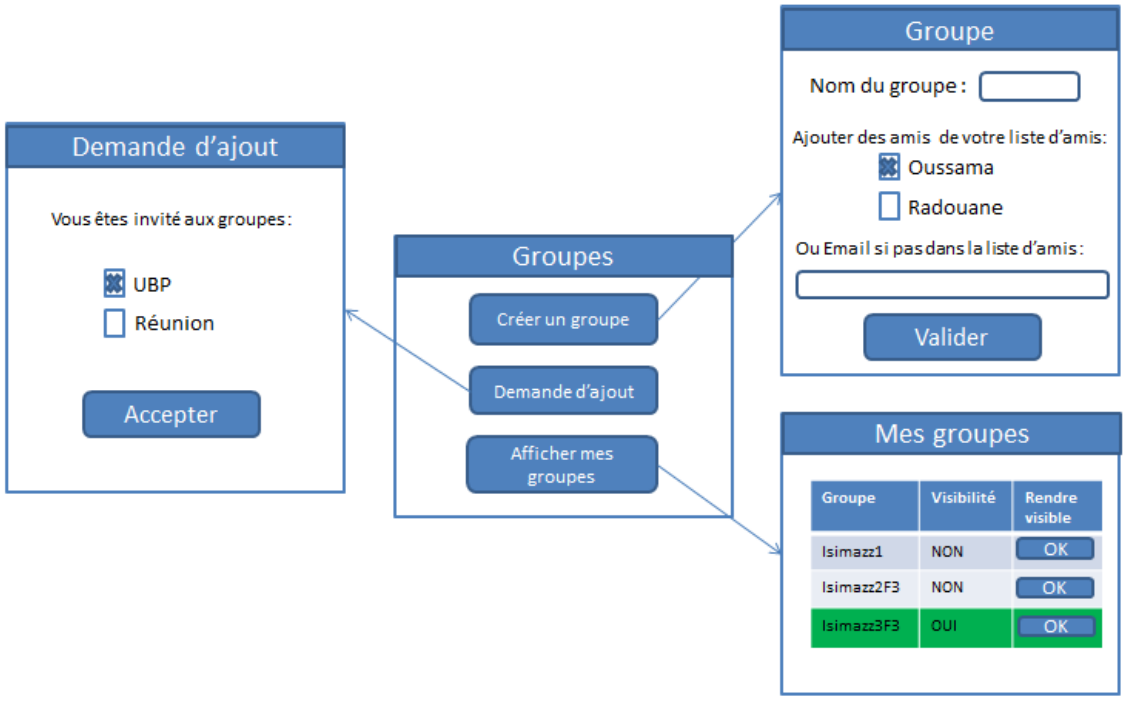



Figure 5 : Maquette de l'application lors de l'appui sur l'onglet "Groupes"

Comme vous pouvez le constater nous avons trois choix à faire :

- Soit nous créons un nouveau groupe et dans ce cas nous accéderons à un formulaire dans lequel nous devrions saisir le nom du groupe ainsi que les amis que nous voulions mettre dans le groupe.

- Le deuxième choix est de voir la liste des demandes d'ajouts c'est-à-dire l'ensemble de groupe dans lequel nous sommes invités et nous pourrions accepter ou non ces différentes invitations.

- Le dernier choix et le plus important puisque nous pourrions voir sa liste de groupe et le groupe dans lequel nous sommes visibles et nous avons le droit de changer la visibilité de nos groupes.

Dans le cas où nous cliquons sur un groupe nous aurions la possibilité de changer le nom du groupe, ajouter ou supprimer des amis au groupe ou même supprimer le groupe si nous sommes administrateur. Ceci est résumé en figure 6:

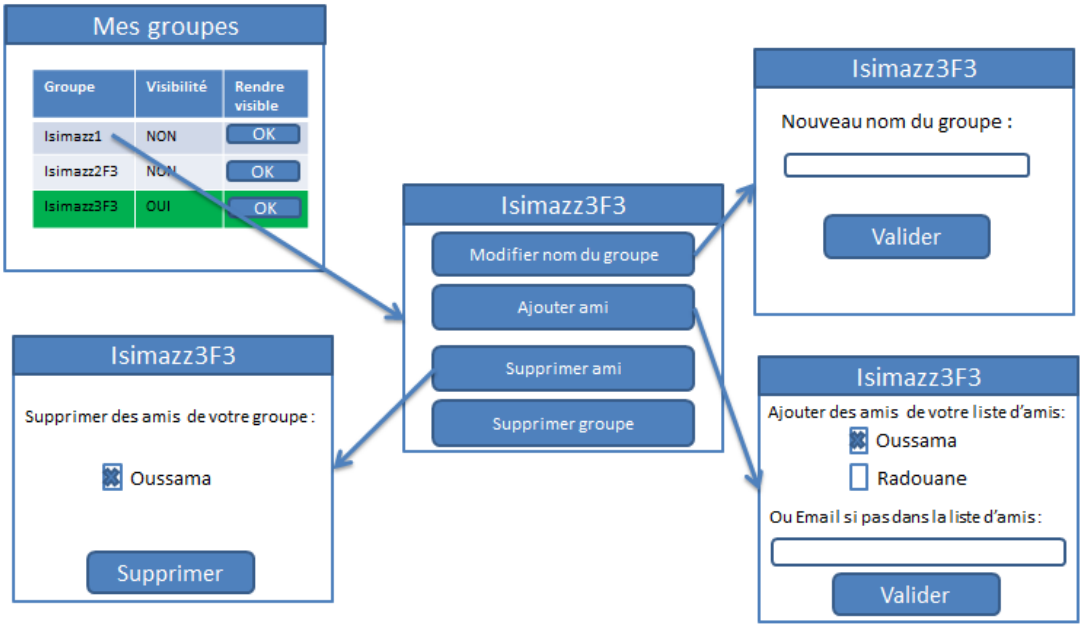



Figure 6 : Maquette de l'application lors de l'appui sur un groupe

## ***« Amis » :***

L'application selon le cahier de charge devra nous permettre de gérer les amis (ajouter, supprimer, modifier) et accepter les demandes d'ajouts.

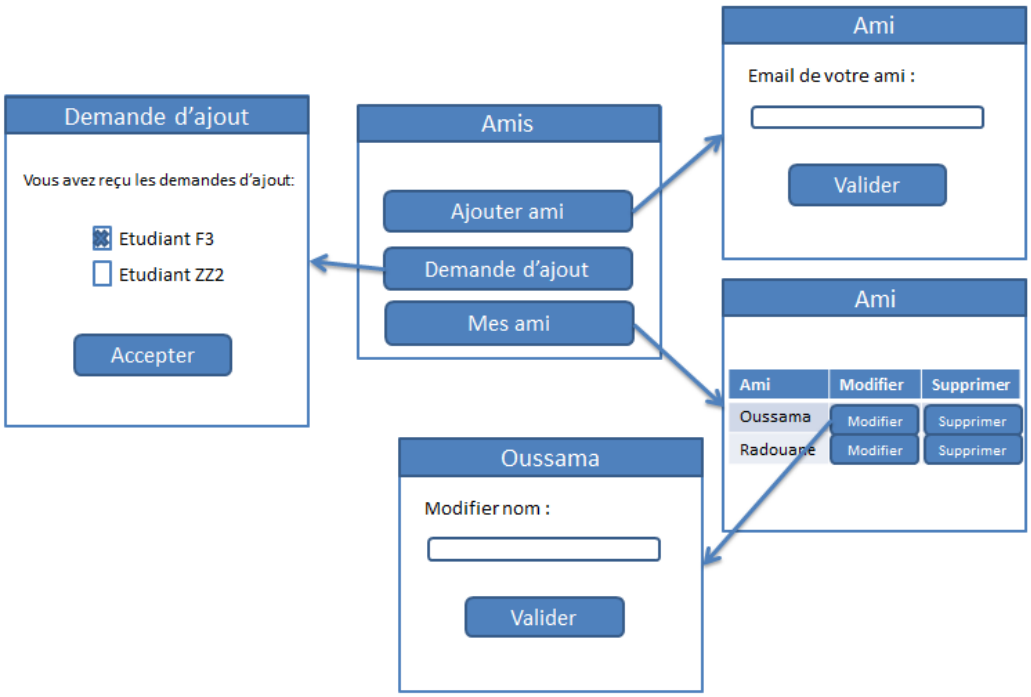



Figure 7: Maquette de l'application pour la gestion des amis

Le principe pour ajouter un ami est de saisir son adresse mail ans le cas où il n'a pas de compte sur l'application il recevra un mail lui indiquant un lien pour télécharger l'application, sinon l'utilisateur recevra une demande d'ajout qu'il pourra accepter ou non.

Dans le cas où il accepte de devenir notre ami, ce dernier sera affiché dans notre liste d'amis et donc nous pourrions le supprimer ou modifier son nom pour mieux le reconnaitre.

***« Profil » :***

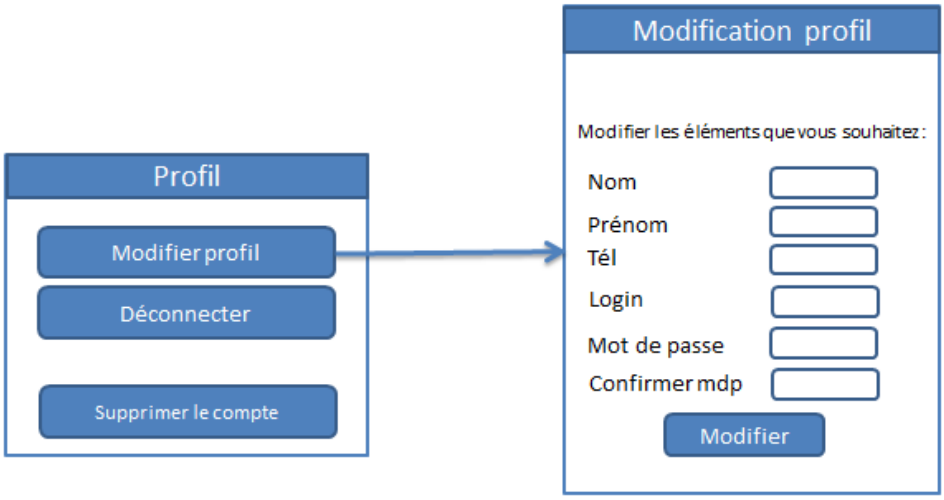



Figure 8 : Maquette de l'application pour la gestion du profil

La figure 8 résume à peu près la gestion du profil d'un utilisateur qui pourra supprimer son compte ou se déconnecter ainsi que de modifier ses données personnelles telles que le nom, prénom ou mot de passe via un formulaire.

Après avoir compris les objectifs de l'application et établi le cahier de charge durant nos réunions avec nos tuteurs il fallait donc commencer à concevoir la solution.

# II. Conception de la solution

Dans cette partie, nous présenterons les étapes de construction de la solution finale, nous commencerons dans un premier temps par le modèle physique de données, ensuite nous détaillerons les différentes parties de la programmation de l'application, puis nous parlerons de l'interface graphique pour finir avec les différents outils que nous avons mis en place pour assurer la sécurité.

## 1. Modélisation BDD

Après notre discutions avec nos tuteurs sur les différentes informations qui peuvent caractériser chacune des entités et sur les relations qui peuvent exister entre elles, nous sommes parvenus à l'aide du diagramme de classe UML de la figure 5 à restituer toutes les informations nécessaires pour réaliser le model physique :

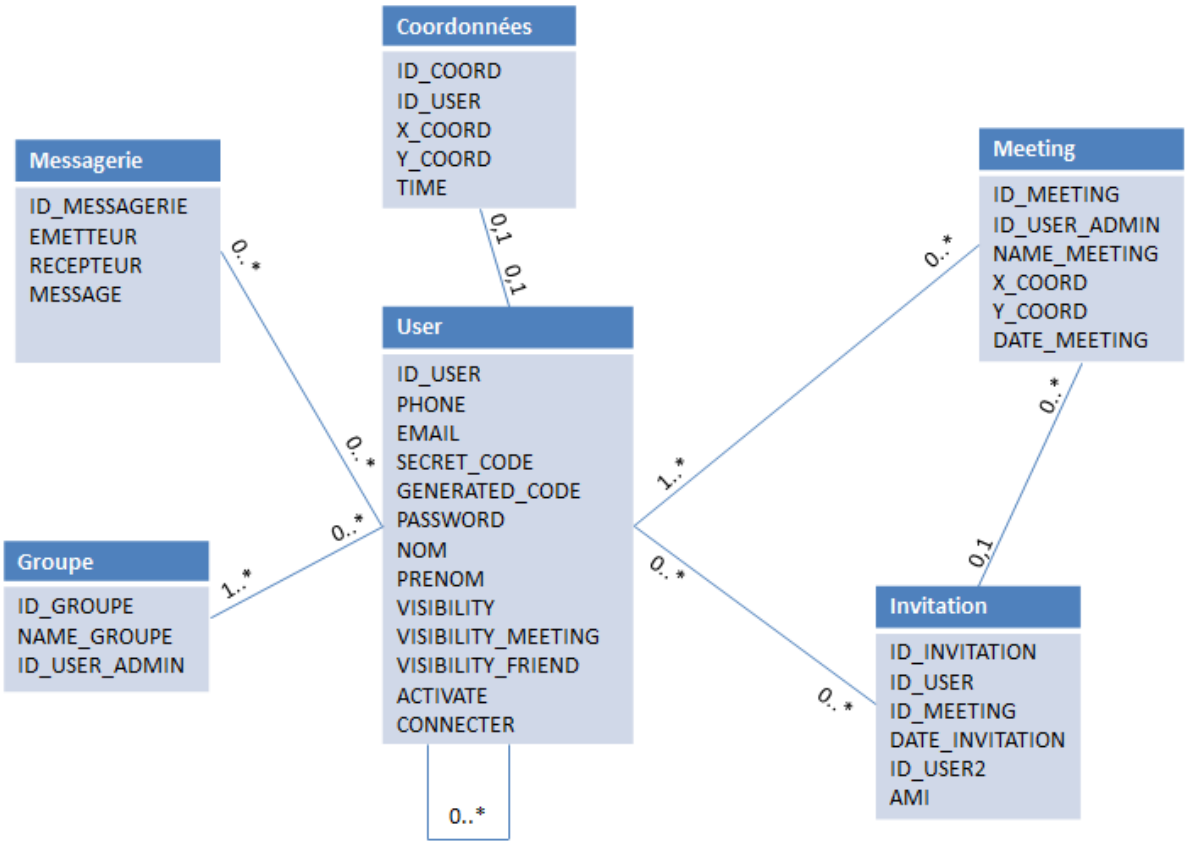



Figure 9: Diagramme de classe UML

Dans ce diagramme de classe nous trouvons les tables suivantes :

***User*** : cette table est utilisée pour stocker les différentes informations relatives à un utilisateur que ce soit son:

- Identifiant (*ID_USER*).
- Numéro de téléphone (*PHONE*).
- Mail (*EMAIL*).
- Code secret (*SECRET_CODE*) qui nous permettra d'identifier son téléphone.
- Code généré automatiquement (*GENERATED_CODE*) que l'utilisateur doit utiliser pour activer son compte.
- Mot de passe (*PASSWORD*) qui doit comporter au moins 6 caractères.
- Nom (*NOM*) et prénom (*PRENOM*).
- La visibilité de l'utilisateur sur toute l'application (*VISIBILITY*).
- Le groupe dans lequel il veut être visible(*VISIBILITY_MEETING*).
- La visibilité par rapport à un ami (*VISIBILITY_FRIEND*).
- Activation de son compte (*ACTIVATE* : 0 si compte non activé et 1 sinon).
- Son statut : connecté ou non (*CONNECTER* : 1 si connecté 0 sinon).

***Coordonnées*** : Table utilisé pour stocker la position exacte (X_COORD, Y_COORD) grâce au GPS d'un utilisateur donné (ID_USER) à un instant t (TIME).

***Meeting*** : Table nous permettant de créer un rendez-vous. Pour chaque rendez-vous nous avons l'identifiant de l'admis (ID_USER_ADMIN) ainsi que la position du point de rencontre (X_COORD, Y_COORD) et la date de création et modification (DATE_MEETING).

***Invitation*** : Table stockant les différentes invitations envoyées par un utilisateur (ID_USER) à un (ID_USER2) pour le rejoindre à un meeting (ID_MEETING) à un instant t (DATE_MEETING). Une variable AMI est égale à 1 si l'utilisateur à accepter l'invitation et 0 sinon.

***Groupe*** : Chaque utilisateur a le droit de créer un groupe selon ses affinités (par exemple : un groupe de travail), c'est le même principe que les cercle de Google +.

***Messagerie*** : Tous les messages qui seront envoyé par les utilisateurs seront dans cette table, par exemple si un utilisateur X ajoute un ami qui n'est pas encore inscrit sur l'application, dès que ce dernier télécharge l'application il recevra un message lui indiquant que l'utilisateur X l'a invité à le rejoindre.

Des tables de jointures ont également été créées :

***Contact*** joue le rôle d'intermédiaire entre les utilisateurs et relie chaque utilisateur à son ami.
**User_meeting** relie un utilisateur à un rendez-vous.

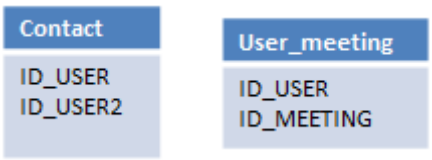



Figure 10 : Tables de jointure

Après avoir modélisé l'ensemble des données du cahier de charge nous avons créé grâce au SGBD *MySQL Workbench* comme le montre la capture d'écran suivante :

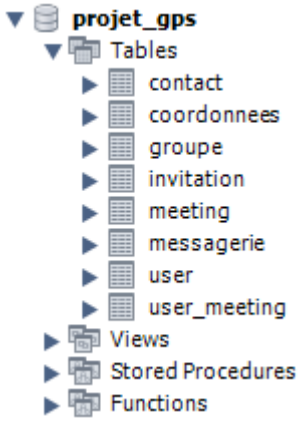



Figure 11 : Base de données à partir du MySQL Workbench

# 2. Développement de l'application

Après que nos tuteurs aient validé la conception proposée, nous nous sommes focalisés sur la partie métier de l'application. A savoir que celle-ci devait être séparée en deux grandes parties :

- La partie Web Service implémentant les fonctionnalités sur le serveur et les échanges avec la base de données.
- La partie Android comportant les fonctionnalités d'affichage et de liaison avec le web service.

Cette partie présentera l'implémentation de chacune des deux parties à commencer par la partie Web Service.

## 1 Web Service

La première chose à faire était de créer la base de données avec les différentes tables évoquées précédemment, pour cela nous avons utilisé ***MySQL Workbench*** qui, avec son interface ergonomique et la possibilité d'importer ou d'exporter des dumps, nous a facilité le travail. La figure suivante présente l'interface de ***MySQL Workbench*** avec les différentes tables de l'application.

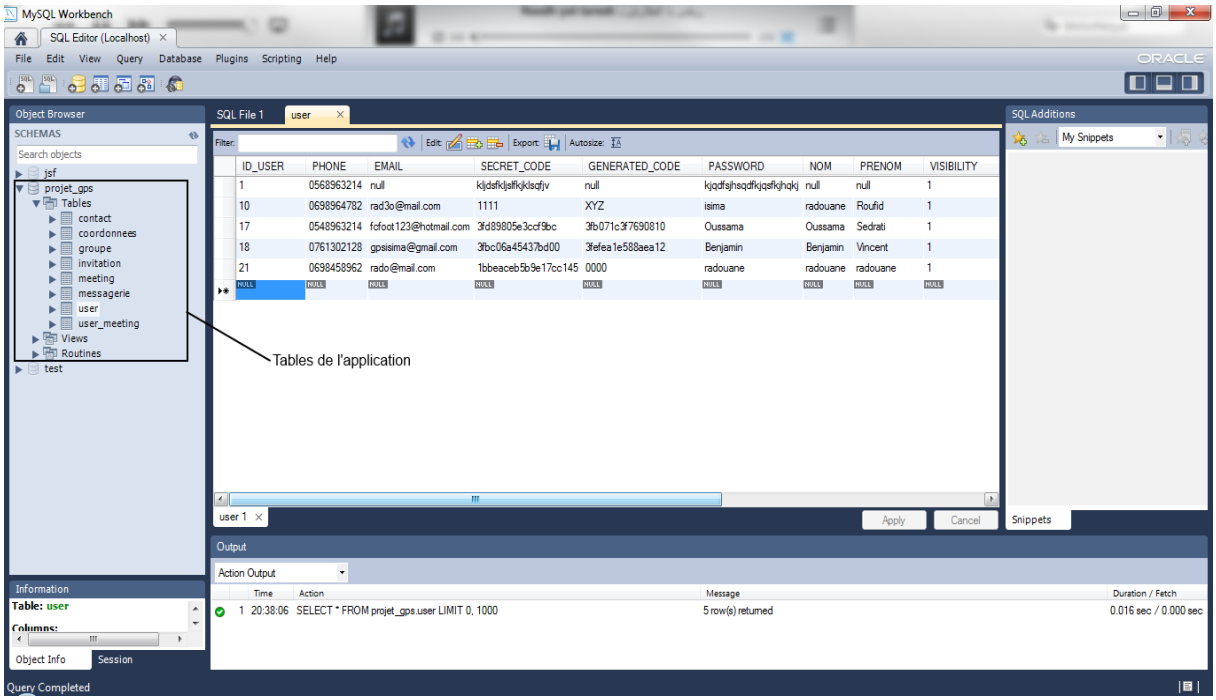



Figure 12: Utilisation de MySQL Workbench.

Nous avons implémenté le Web Service sous ***NetBeans*** pour différentes raisons :

1- NetBeans offre un serveur d'application **GlassFish**
2- NetBeans facilite l'implémentation des EJB
3- Création automatique de la couche JPA *(Java Persistence API)*
4- Faciliter de créer un Web Service type SOAP. [Création d'un EJB]

La figure 13 présente le rôle de chaque couche :

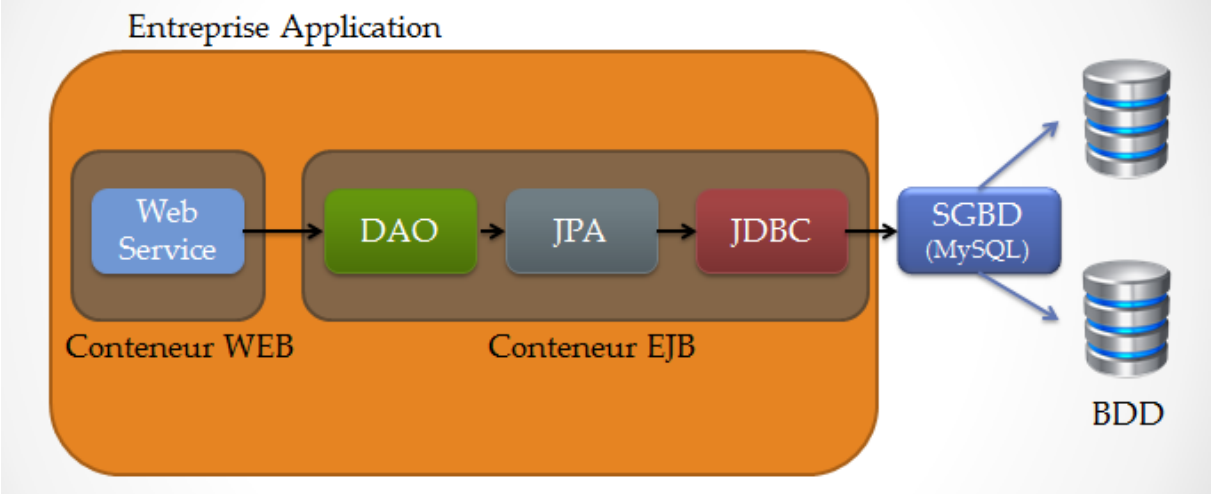



Figure 13: Couches du serveur GlassFish.

Explication du rôle de chaque entité :

- **SGBD :** Représente le système de gestion de base de données utilisé, dans notre application nous avons utilisé ***MySQL***.
- **JDBC :** C'est la couche responsable de la connexion avec la base de données. Dans notre cas, la couche JDBC est la nation de pool de connexion qu'offre Glassfish. Un pool de connexion est un ensemble de connexions avec la base de données déjà instanciées.
- **JPA :** Couche d'abstraction de la couche JDBC, elle permet de faire le Mapping Relationnel-Objet qui consiste à modéliser la base de données sous forme d'objets pour une manipulation plus simple à travers le code Java.
- **DAO :** Une structure spécifique permettant d'accéder aux objets et proposent les méthodes dites CRUD (Create, Read, Update, Delete).
- **Web Service :** Représente l'ensemble des fonctions qui seront appelées selon le protocole SOAP. [Création des web services]

Le choix d'utilisation de JPA est motivé par le gain de temps que cette technologie offre, en effet si nous avions choisi d'écrire à la main les requêtes SQL, nous allions d'une part perdre du temps pour parser les réponses, nous n'aurons pas un système de contrôle côté serveur (contrôle redondance et valeur nulle) et également une difficulté pour les requêtes

complexes. Mais avec JPA, d'une part NetBeans se charge de la création des différentes classes, d'autre part les requêtes sont nettement plus allégées. Et montrer cela, nous donnons l'exemple ci-dessous :

Comme expliqué dans la partie modélisation, un utilisateur a une liste d'amis qui ont chacun des coordonnées (position). Si nous souhaitons récupérer la liste des amis d'un utilisateur donné avec leurs coordonnées en requête SQL  alors nous avions à écrire une jointure en précisant l'identifiant de l'utilisateur principal et ensuite pour chacun de ses amis il faut aller chercher ses coordonnées et implémenter une solution pour mapper un utilisateur avec ses coordonnées pour ne pas perdre l'information. Le développement de cette fonctionnalité est couteux en temps et en performance mais grâce à JPA tout cela peut être fait facilement comme le montre la figure 14 :

```
@Override
public List<Coordonnees> findCoordonneesByUser(int iduser) {
   return em.createQuery("select c from Coordonnees c WHERE c.idUser.idUser = '"+iduser+"'").getResultList();
}
```

Figure 14: Implémentation de la méthode FindCoordonneesByUsers avec JPA

Ainsi dans la classe « ***Coordonnees*** » nous avons déjà la classe « ***User*** » qui est référencée et du coup n'avons pas à nous soucier du problème du mapping car il se fait tout seul.

Pour fournir une bonne qualité de code et que celui-ci soit parfaitement bien lisible pour le ou les utilisateurs qui souhaiterons continuer le projet, nous avons séparé chacune des entités dans un package spécifique comme le montre la figure 15 :

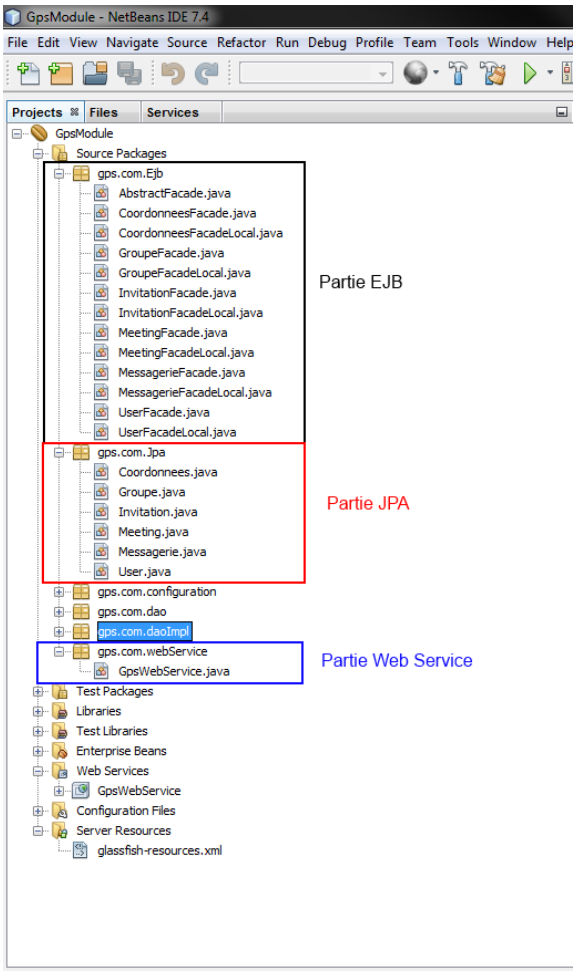



Figure 15: Organisation du code source côté serveur.

Le Web Service comporte les différentes fonctionnalités pour gérer les utilisateurs, les amis et les rendez-vous ainsi que les fonctions permettant de calculer le barycentre d'un rendez-vous. Nous ne présenterons pas ici toutes les fonctions, mais nous donnerons une seule à titre d'exemple pour comprendre le principe, nous choisirons la fonction « ***EditUser*** » permettant de modifier les informations d'un utilisateur :

```
@WebMethod
public void editUser(@WebParam(name="email") String email,
                     @WebParam(name="name") String name,
                     @WebParam(name="prenom") String prenom,
                     @WebParam(name="password") String password,
                     @WebParam(name="phone") String phone,
                     @WebParam(name="secretCode") String secretCode){
    if(!userDao.findUserByMail(email).isEmpty()) {
        User u = userDao.findUserByMail(email).get(0);
        if(u.getGeneratedCode().equals(secretCode)) {
            if (name != null) u.setNom(name);
            if (prenom != null) u.setPrenom(prenom);
            if (phone != null) u.setPhone(phone);
            if (password != null && password.length()>5) u.setPassword(password);

            userDao.edit(u);
        }
    }
}
```

Figure 16: Fonction de modification d'un utilisateur

On précise que cette fonction est une fonction du Web Service pouvant être appelée de l'extérieur grâce à l'annotation *@WebMehtod* dont la spécification des paramètres se fait grâce à l'annotation *@WebParam* où l'attribut « *name* » est spécifié car il servira de faire le référencement lors de l'appel du Web Service à partir de l'Android. Le paramètre « *SecretCode* » assure la sécurité de l'appel du Web Service, ceci sera détaillé dans la section sécurité.

« *userDao* » est l'EJB que nous utilisons pour faire les appels avec la base de données et il est déclaré comme suit :

```
@WebService(serviceName = "GpsWebService")
@Stateless()
public class GpsWebService {
    private static final float distMax = 500;
    private static final float refreshCentroid = 2;

    @EJB
    private UserDao userDao;

    @EJB
    private MeetingDao meetingDao;
```

Figure 17: Déclaration de l'EJB « userDao ».

Vu que les utilisateurs de l'application bougent continument, leur position change ainsi que le barycentre du rendez-vous (lieu de rencontre), il a fallu alors trouver une solution pour mettre à jour les informations directement, pour cela un signale périodique est envoyé à partir de l'application Android via la fonction « *Schedule* » de la classe « *Timer* » dont un « *TimerTask* » doit être donné en paramètre avec le temps de la période. La figure suivante montre l'instanciation de trois « Task », une pour mettre à jour les cordonnées d'un utilisateur, une pour mettre à jour le barycentre d'un « meeting » et la dernière pour récupérer les nouvelles positions des autres utilisateurs.

```
Timer time = new Timer(); // Instantiate Timer Object
ScheduledTask st = new ScheduledTask(); // Task pour les coordonnées d'un utilisateur
MeetingUpdateTask meetingTask = new MeetingUpdateTask(); // Task pour mettre à jour le barycentre
UsersUpdateTask userTask = new UsersUpdateTask(); // Task pour récupérer les nouvelles positions
time.schedule(st, 0, timeRefreshCoord);
time.schedule(meetingTask, 0, timeRefreshMeeting);
time.schedule(userTask, 0, timeRefreshUsers);
```

Figure 18: Déclaration des différentes Task pour mettre à jour les informations.

Nous allons prendre le « *Task* » permettant de rafraichir le barycentre, en effet cette fonction a une période de **3 minutes** pour éviter d'une part de surcharger le serveur et d'autre part permettre à chaque utilisateur de mettre à jour leurs coordonnées (La période d'envoi est d'une minute). Comme indiqué dans la partie modélisation chaque utilisateur ne peut apparaître que dans un seul rendez-vous à la fois, en effet ceci est traduit grâce à

l'attribut « *VISIBILITY_MEETING* » dans la base de données où est stocké l'identifiant du meeting courant, pour optimiser les mises à jour nous ne rafraichissons que les meetings où en moins un utilisateur est connecté. Une fois le signal envoyé au Web Service, ce dernier récupère l'identifiant du meeting et récupère tous les utilisateurs qui lui appartiennent et recalcule le barycentre. Deux règles sont appliquées pour mettre à jour le barycentre :

1- Si le délai entre la dernière mise à jour et l'appel actuel est supérieur à **2min** alors on rafraichit sinon on rejette l'appel.
2- Si la distance entre le barycentre calculé et l'ancien barycentre est plus grande que **100m** alors on met à jour le barycentre avec celui calculé sinon on garde l'ancien. Ainsi nous avons.

Un détail plus approfondi de cette mise à jour est donné par la suite.

Une fois la partie Web Service terminée, nous nous sommes focalisées sur la partie Android qui sera présentée dans la partie suivante.

## 2 Android

La partie Android était la plus importante et comportait des tâches délicates, nous nous sommes répartis les tâches afin de mieux avancer et de pousser le développement du projet au mieux. Au départ nous avions utilisé les « Activity » de Java Android pour basculer d'un onglet à un autre (Inscription, Authentification, Groupe…) mais l'utilisation de cette tâche était compliquée à gérer et le temps de réponse était important, pour remédier à ce problème nous avons choisi de combiner HTML5 avec une seule « Activity » qui est la « MainActivity ». En effet, nous avons créé pour chaque onglet une page HTML comme le montre la figure 19

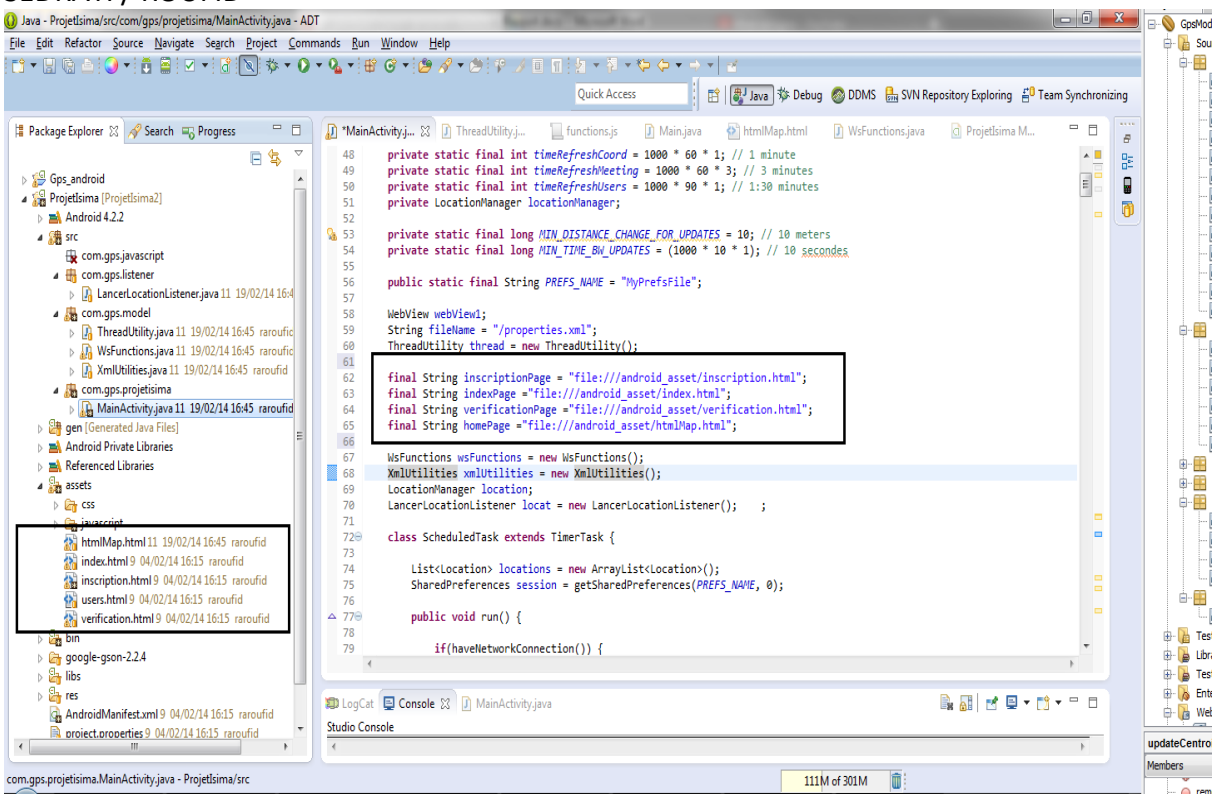
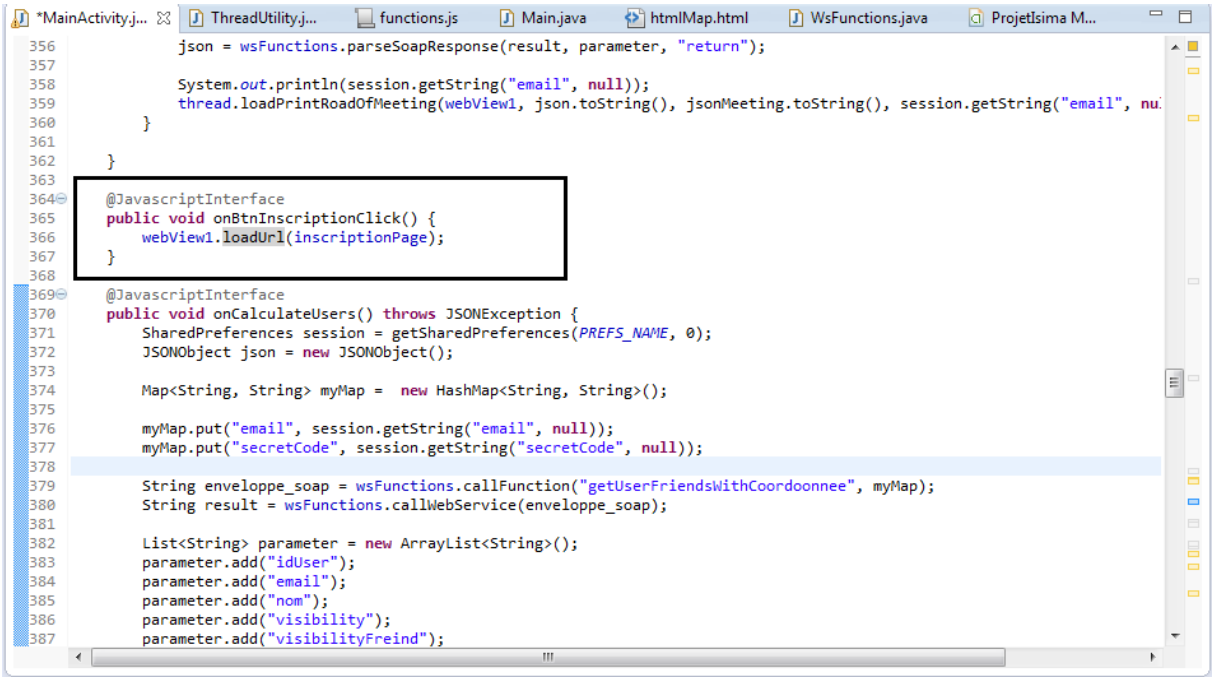


Figure 19: Utilisateur des pages HTML au lieu des « Activity »

Et le basculement entre les différents onglets se fait simplement par l'appel de la méthode « ***LoadUrl*** » sur la « ***WebView*** » principale comme présenté sur la figurer 20 :



```
356             json = wsFunctions.parseSoapResponse(result, parameter, "return");
357
358             System.out.println(session.getString("email", null));
359             thread.loadPrintRoadOfMeeting(webView1, json.toString(), jsonMeeting.toString(), session.getString("email", nu
360         }
361
362     }
363
364     @JavascriptInterface
365     public void onBtnInscriptionClick() {
366         webView1.loadUrl(inscriptionPage);
367     }
368
369     @JavascriptInterface
370     public void onCalculateUsers() throws JSONException {
371         SharedPreferences session = getSharedPreferences(PREFS_NAME, 0);
372         JSONObject json = new JSONObject();
373
374         Map<String, String> myMap =  new HashMap<String, String>();
375
376         myMap.put("email", session.getString("email", null));
377         myMap.put("secretCode", session.getString("secretCode", null));
378
379         String enveloppe_soap = wsFunctions.callFunction("getUserFriendsWithCoordoonnee", myMap);
380         String result = wsFunctions.callWebService(enveloppe_soap);
381
382         List<String> parameter = new ArrayList<String>();
383         parameter.add("idUser");
384         parameter.add("email");
385         parameter.add("nom");
386         parameter.add("visibility");
387         parameter.add("visibilityFreind");
```

Figure 20: Chargement de la page « inscription » dans une WebView.

Le choix de cette solution nous a permis de gagner un temps considérable de développement ainsi qu'en temps de réponse de l'application.

L'un des points importants auquel nous avons bien réfléchi est l'échange de données entre le serveur et la partie Android. A savoir que le serveur accepte et renvoie des fichiers « XML » alors que pour JavaScript le format que nous avons choisi est « Json », nous avons dynamisé les conversions afin de faciliter le travail. Le schéma suivant résume les différentes conversions que nous avons faites :

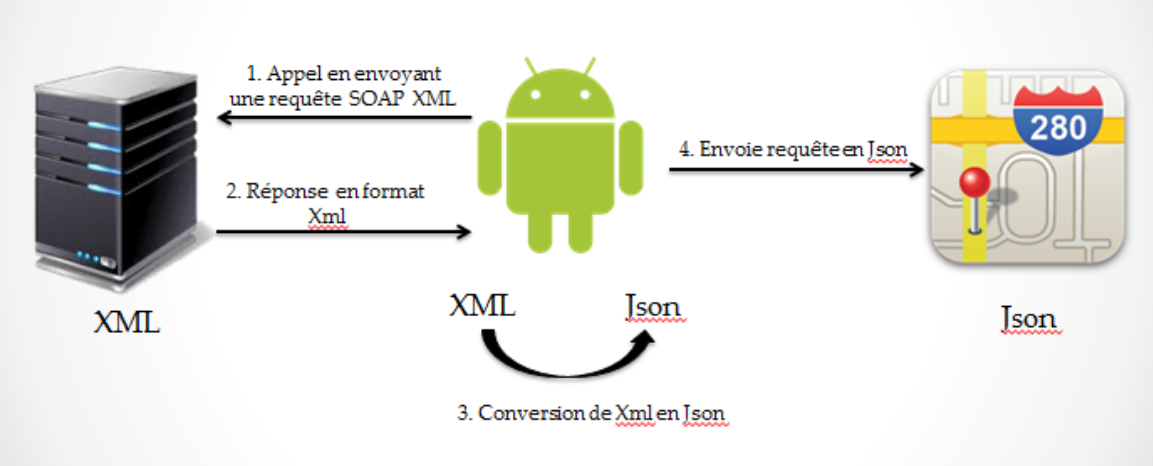



Figure 21: Conversion des requêtes entre le serveur et l'application.

- **Construction du fichier XML pour l'appel du Web Service :** C

Concernant l'appel au Web Service, nous avons créé un model permettant de générer automatiquement le fichier XML pour la requête SOAP à partir de l'objet Java « Map » contenant les paramètres d'appel. A titre d'exemple, voici l'appel de la fonction « ***FindUserByEmail*** » prenant comme paramètre le « mail » et le « secretCode ».

```
Map<String, String> myMap =  new HashMap<String, String>();
myMap.put("email", session.getString("email", null));
myMap.put("secretCode", session.getString("secretCode", null));
```

Paramètres de la requête SOAP

```
String enveloppe_soap = wsFunctions.callFunction("findUserByEmail", myMap);
String result = wsFunctions.callWebService(enveloppe_soap);
```

Construction de l'enveloppe
Appel du Web Service

Figure 22: Construction de l'enveloppe SOAP et appel du Web Service

- **Parsing du fichier XML réponse en fichier Json :**

Un fichier qu'on a la réponse du Web Service et qu'on souhaite la transmettre en JavaScript, il faut la changer en format Json et pour cela nous avons implémenté une méthode « ***ParseSoapResponse*** » qui prend en paramètre le fichier XML réponse, les paramètres du parsing et le champ sur lequel portera le parsing (généralement le champ « return »)

```
enveloppe_soap = wsFunctions.callFunction("getUsersInMeeting ", myMap);
result = wsFunctions.callWebService(enveloppe_soap);

parameter = new ArrayList<String>();
parameter.add("email");
parameter.add("idUser");
parameter.add("nom");
parameter.add("phone");
parameter.add("prenom");
parameter.add("visibility");
parameter.add("XCoord");
parameter.add("YCoord");
json = wsFunctions.parseSoapResponse(result, parameter, "return");

thread.loadUpdateMarkersOnMap(webView1, json.toString(), jsonMeeting.toString(),idUser);
```

Paramètres du Parsing

Parsing de la réponse

Figure 23: Parsing de l'XML au format Json

Nous allons maintenant évoquer l'implémentation de chaque partie de l'application.

## 1. *Authentification*

Nous avons commencé par l'interface authentification via laquelle l'utilisateur à la possibilité de soit se loguer s'il a déjà un compte ou bien d'être dirigé sur l'interface d'inscription. La figure 24 présente cette interface :

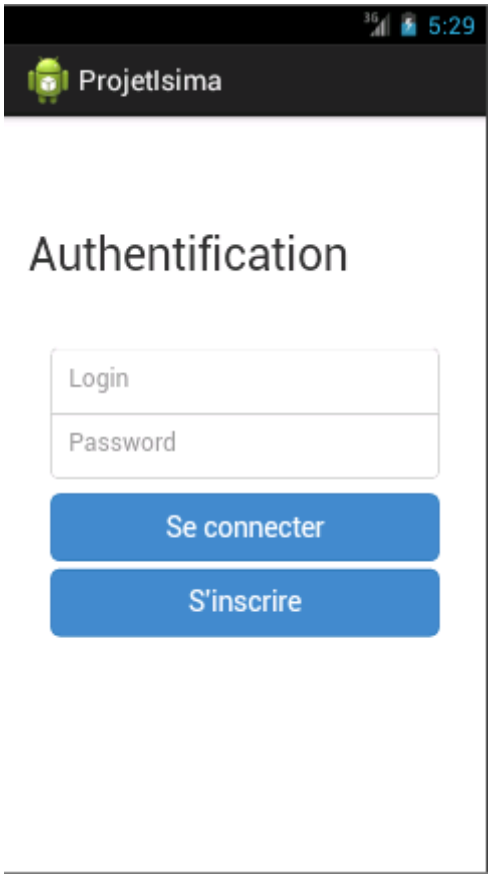



Figure 24: Interface d'authentification.

Comme indiqué dans la partie modélisation, un compte utilisateur peut être soit actif soit inactif et son activation se fait via le code généré envoyé par mail à l'utilisateur. D'où si un utilisateur s'inscrit puis ferme l'application sans entrer le code qui lui a été envoyé, la

prochaine fois qu'il ouvrira la connexion et qu'il rentrera son login et son mot de passe, il sera redirigé vers l'écran de vérification lui demandant de saisir le code qui lui a été envoyé. L'écran de vérification est présenté en figure suivante :

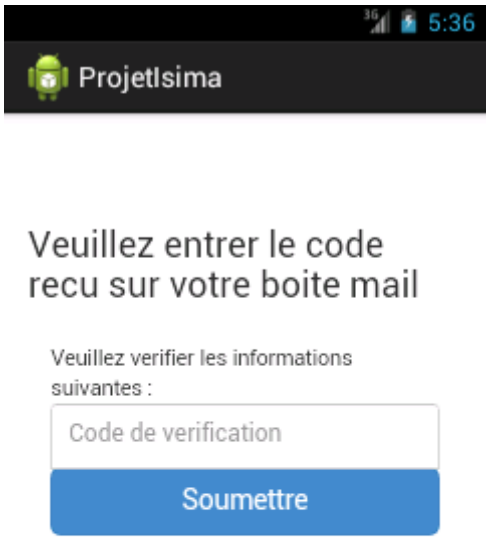



Figure 25: Ecran de vérification.

Tant que l'utilisateur n'aura pas entré son code secret, il ne réussira pas à accéder aux fonctionnalités de l'application.

Si l'utilisateur n'a pas de compte alors il peut en créer un en appuyant sur le bouton « S'inscrire » de l'interface « Authentification » dans le détail est donné dans la section suivante.

### 2. *Inscription*

L'écran d'inscription a été mis en place afin de permettre aux utilisateurs d'avoir un compte et accéder aux fonctionnalités GPS, pour cela l'utilisateur doit fournir certaines informations lui concernant permettant de l'identifier, ces informations sont données ci-dessous :

-Email : Adresse électronique qui doit être unique
-Numéro de téléphone : devant être unique également
-Mot de passe : Devant comporté plus de 5 caractères et une confirmation de mot de passe est demandée également
-Nom

La figure ci-dessous résume l'ensemble :

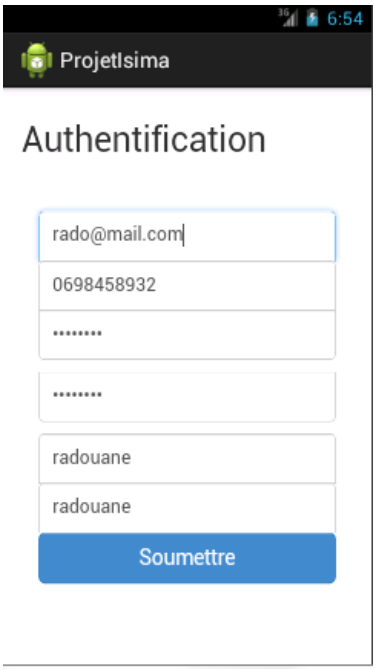



Figure 26: Interface d'inscription.

Il est important d'assurer l'intégrité des données dans la base de données pour cela un système de contrôle a été mis en place à deux niveaux :

1- **Niveau JavaScript :** Des règles de contrôle s'appliquent aux différents champs, par exemple le respect du format du mail, le respect du numéro de téléphone, les valeurs identiques entre le mot de passe et sa confirmation. Dans le cas où le format d'un champ n'est pas respecté on en informe l'utilisateur via un message JavaScript comme le montre la figure 27 :


Figure 27: Erreur JavaScript en saisissant un email ne respectant pas le format.

2- **Niveau serveur :** ce test est également nécessaire pour assurer au maximum l'intégrité des données. En effet, les règles de contrôle sont également appliquées côté serveur sur l'email, le numéro de téléphone, le mot de passe et le nom. Si ces règles ne sont pas respectées alors la création n'a pas lieu.

Si la création réussie alors l'utilisateur est dirigé vers la page de vérification où il lui est demandé d'insérer le code généré qu'il a reçu par mail (voir figure 25).

## *3- Gestion des groupes*

En ce qui concerne la gestion des groupes, comme nous l'avions expliqué dans la partie d'analyse nous avons créé la page suivante :


Figure 28 : Ecran de gestion de groupes

Cette page a été créée en utilisant le langage *HTML* ainsi que le *JQueryMobile* (un *Framework* web tactile optimisée compatible avec une grande variété de smartphones et les tablettes). Chaque bouton de cette page nous renvoie vers une autre page.

Par exemple lorsque nous allons cliquer sur le bouton « Afficher mes groupes » nous passons par les étapes suivantes:

1- Le bouton fait appel à la fonction JavaScript « *openListeGroupPage* »

```
<div data-role="content" data-theme="b">
<button id="GroupUsers" onclick="javascript:openListeGroupPage()">
Afficher mes groupes
</button>
```

2- La fonction JavaScript nous renvoie à la fonction java « *openListeGroupPage* » qui se trouve à la classe *MainActivity*

```
function openListeGroupPage()
{
    Android.openListeGroupPage();
}
```

3- La fonction java charge une *WebView* dont l'url est :

```
final String listegroupe ="file:///android_asset/listegroupe.html";
            @JavascriptInterface
            public void openListeGroupPage() {
                thread.loadUrl(webView1, listegroupe);
            }
```

4- Dans la page « listegroupe.html » nous appelons la fonction getMyGroups()

```
<script>getMyGroups();</script>
```

Dans le fichier JavaScript :

```
function getMyGroups() {
    Android.onbtnAfficherGroupes();
}
```

5- La fonction java onbtnAfficherGroupes() récupère à partir de la fonction créer dans le web Service ***findMeeting*** qui prend en paramètre l'email de l'utilisateur ainsi que son code secret et retourne la liste de ses groupes. Ensuite nous affichons ces groupes sur la *WebView* grâce à la fonction JavaScript *printGroups()* .

```
@JavascriptInterface
public void onbtnAfficherGroupes() throws JSONException {
    SharedPreferences session = getSharedPreferences(PREFS_NAME, 0);
    Map<String, String> map = new HashMap<String, String>();
    //remplissage  des parametres.
    map.put("email",session.getString("email", null));
    map.put("secretCode", session.getString("secretCode", null));
    String enveloppe_soap= wsFunctions.callFunction("findMeeting", map)
    String result= wsFunctions.callWebService(enveloppe_soap);
    List<String> parameter = new ArrayList<String>();
    parameter.add("idMeeting");
    parameter.add("nameMeeting");
    parameter.add("XCoord");
    parameter.add("YCoord");
    JSONObject json = new JSONObject();
    json = wsFunctions.parseSoapResponse(result, parameter,"return");
    webView1.loadUrl("javascript:printGroups('"+json+"')");
}
```

6- Finalement la fonction *printGroups parse* l'objet JSON et ensuite affiche le résultat sur la page HTML

```
function printGroups(par) {
    var json = JSON.parse(par);
    var liste = '<center><table border="1">';
    liste += '<TR style="Background-Color:Blue; Color:White"><TH>Groupe</TH>';
    liste += '<TH>Visible</TH><TH>Oui/Non</TH></TR>';
    for(var i=0; i<json.result.length; ++i) {
    var j = json.result[i].nameMeeting;
    var myVar=isvisible(j);

    if(myVar == "oui")
        {
        liste += '<TR style="Background-Color:Lime; Color:Black"><TH>';
        liste += '<button onclick = passer_parametre(\''+j+'\') >'+json.result[i].nameMeeting+'</button></TH>';
        liste += '<TH>'+myVar+'</TH><TH><button onclick = makevisible(\''+j+'\') >OUI</button>';
        liste += '</TH></TR>';
        }
        else
        {
        liste += '<TR><TH><button onclick = passer_parametre(\''+j+'\') >'+json.result[i].nameMeeting+'</button></TH>';
        liste += '<TH>'+myVar+'</TH><TH><button onclick = makevisible(\''+j+'\') >OUI</button>';
        liste += '</TH></TR>';
        }

    }
    liste += '</table></center>';
    document.getElementById("info2").innerHTML +=liste;
}
```

Après ces six étapes nous obtenons un tableau  avec le nom du groupe et la visibilité du groupe que nous pouvons changer en cliquant  sur le bouton « OUI » :


Figure 29 : Capture d'écran des groupes

Ensuite, lorsque nous cliquons sur un groupe nous trouvons :


Figure 30 : Capture d'écran de la page du groupe "Groupeoussama"

Afin de faire passer le nom du groupe entre deux WebView :

- Nous envoyons le nom par l'url

```
thread.loadUrl(webView1, mongroupe+"?nom="+name_meeting+"");
```

- Nous récupérons le nom en JavaScript:

```
var url = window.location.search;
var j = url.substring(url.lastIndexOf("=")+1);
```

Lorsque nous voulions ajouter des amis au groupe, nous aurons la fenêtre suivante :

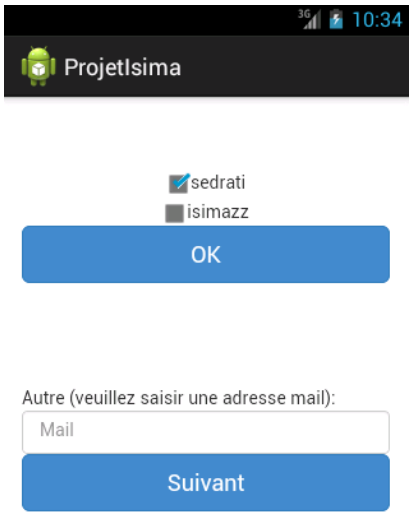



Figure 31 : WebView pour l'ajout d'amis dans un groupe

Cette page respecte ce que vous avions établit lors de la phase d'analyse, nous avons utilisé les mêmes principes que les six étapes pour afficher la liste de nos amis il fallait juste choisir la bonne fonction du *WebService* à utiliser.
La particularité de cette page est l'utilisation des *checkboxs*, nous avons récupéré les valeurs cochés grâce au script :

```
$(window).load(function(){
(function( $ ){
      $.fn.valList = function(){
            return $.map( this, function (elem) {
                  return elem.value || "";
            }).join( "," );
      };
})( jQuery );
});
var tab = $("input:checked").valList();
var elem = tab.split(',');
```

Le résultat de ce script est un tableau des éléments cochés.

Tous les autres boutons (« Supprimer des amis du groupe, Modifier nom du groupe, Supprimer le groupe » pour chaque groupe, « Demande d'ajout, créer un nouveau groupe ») suit le même principe que les six étapes la différence réside dans la fonction du *WebService* utilisé.

## *4- Gestion des utilisateurs*

Cette partie a été implémentée par M. Damien Brugiere, un stagiaire au sein de l'ISIMA. Et elle suit exactement le même principe que les gestions des groupes.

## 5- *Affichage des amis sur la carte*

Une fois connecté l'utilisateur est redirigé vers une page principale lors de laquelle il peut consulter la position de ses amis qui ont accepté d'être visible sur une carte. Pour cela, on fait un appel au Web Service en lui envoyant le mail de l'utilisateur connecté qui nous servira à extraire tous ses amis de la base de données puis les renvoyer pour l'affichage. L'affichage se fait sous Google Map et chaque utilisateur est présenté par un « Marker ». La figure ci-dessous présente l'écran d'accueil de l'utilisateur « radouane ROUFID » ayant 2 amis,

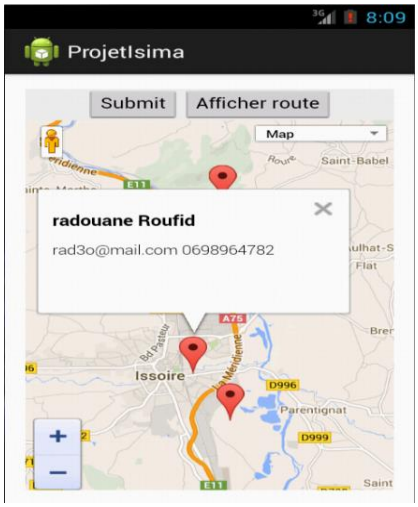



Figure 32: écran d'accueil d'un utilisateur.

Comme le montre la figure précédente, les « Markers » de Google Map sont cliquables où l'utilisateur peut consulter les informations de ses amis (mail, téléphone…).

## 6- *Affichage d'un rendez-vous*

Cette partie constitue le cœur du projet, c'est dans cette partie que devait être implémenté les fonctions de guidage de l'utilisateur, d'affichage des listes des amis qui appartiennent au même groupe que l'utilisateur et de rafraichir périodiquement ses données.

La première chose à faire était d'afficher les données statiquement (sans se préoccuper du rafraichissement), il a fallu alors distinguer l'affichage de l'utilisateur principale de l'affichage de ses amis, puisque les amis ne seront représentés qu'avec des markers sur la carte alors que l'utilisateur devra être en mode guidage entre un point de départ et un point d'arrivé. Pour cela nous avons consulté la documentation de Google Map et nous avons trouvé l'outil «*DirectionRender* » qui permet de gérer le guidage. Il est important de rappeler ici qu'un utilisateur ne peut être affiché que dans un « meeting » où il a choisi d'être visible, ce choix-là nous a permis de récupérer le « meeting » de l'utilisateur ensuite récupérer la liste des utilisateurs abonnés à ce meeting avec leur cordonnées puis transmettre l'ensemble au model JavaScript qui s'occupe de l'affichage. La bonne conception de l'application nous a permis de minimiser les appels Web Service. [Documentation - Google Map Api]

Une fois les informations sur le meeting récupérées ainsi que les coordonnées des utilisateurs abonnés à ce meeting, le model JavaScript gère l'affichage selon 2 modes :

### 1- Affichage de l'utilisateur de l'application :

Cet affichage doit être un guidage entre un point de départ et un point d'arrivé, le code JavaScript permettant de faire cela est donné ci-dessous :

```
var request = {
    origin: new google.maps.LatLng(route.XCoord, route.YCoord),
    destination: end,
    travelMode: google.maps.TravelMode.WALKING
};
```

Spécification départ et arrivée

```
var directionsDisplay = new google.maps.DirectionsRenderer();
directionsDisplay.setMap(map);
```

Affichage de la route sur la Map

```
directionsService.route(request, function(result, status) {
if (status == google.maps.DirectionsStatus.OK) {
        directionsDisplay.setDirections(result);
    }
});
```

Démarrer le guidage

Figure 33: Affichage de la route de guidage pour l'utilisateur principal

### 2- Affichage pour un ami :

Cet affichage ne se fait qu'avec des « markers » de Google Map permettant à l'utilisateur de l'application de connaitre l'avancement de ses amis. Le code JavaScript est donné ci-dessous :

```
var marker = new google.maps.Marker({
    id: route.idUser,
    map: map,
    position: new google.maps.LatLng(route.XCoord,route.YCoord),
    title: route.email,
    zIndex: i
});
```

Instanciation du Marker

```
var contentHtml = "<div style='width:300px;height:200px'><h3>"+route.nom +
" " +route.prenom+"</h3>"+route.email + " " + route.phone+"</div>";
var infowindow = new google.maps.InfoWindow({
    content: contentHtml
});
```

Instanciation de la Pop-Up d'information

```
google.maps.event.addListener(marker, 'click', function() {
    infowindow.open(map,marker);
});

marker.locid = route.idUser;
marker.infowindow = infowindow;

markers.push(marker);
```

Figure 34: Affichage des listes d'amis sur la Map.

Ainsi nous avons pu afficher un meeting avec sa liste d'utilisateur et faire converger l'utilisateur principal vers son barycentre, et voici une prise d'écran reflétant le résultat de cette première partie

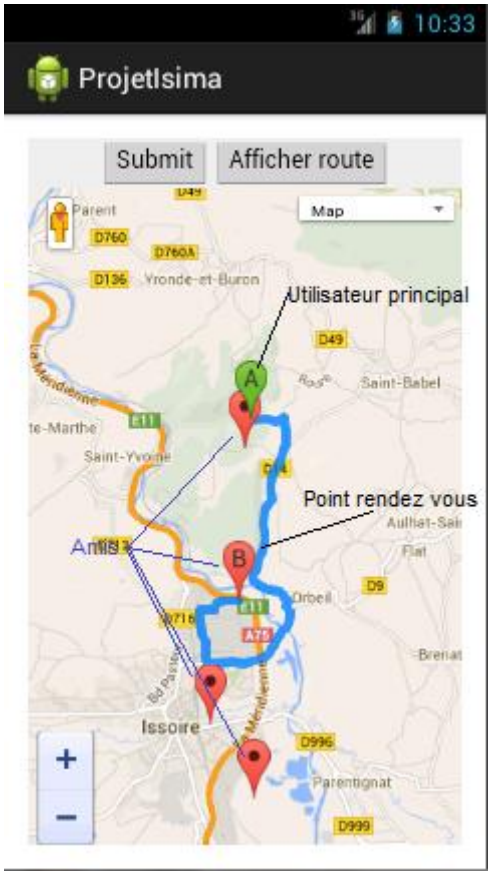



Figure 35: Rendu d'un meetinge

Comme indiqué précédemment, les « Markers » sont cliquables et contiennent les informations sur les utilisateurs. Le chemin en bleu représente le trajet que l'utilisateur principal a à parcourir pour arriver au point de rendez-vous.

Remarque : on peut également faire en sorte que le trajet des amis soit un guidage pour cela il suffit de changer le type « *Marker* » de Google en « *DirectionRender* »

Une fois cette partie terminée, il nous a alors fallu dynamiser d'une part la prise de position de l'utilisateur principal, recalculer du barycentre et mettre à jour les positions des utilisateurs si elles ont changé tout en veillant à ce que cela ne soit pas couteux pour l'application ainsi que pour le serveur, il a fallu trouver une optimisation pour réussir au mieux l'application. Trois points majeurs avaient besoin d'être dynamisés :

1- **La position de l'utilisateur :**

Nous devions prendre en permanence la position courante de l'utilisateur afin de suivre son avancement, pour cela nous avons utilisé la propriété « ***LocationListner*** » de Java Android. Cette classe nous permet de récupérer les meilleures dernières positions GPS de l'utilisateur avec un intervalle que nous avons défini de 10 secondes. Une fois la meilleur position prise, on doit l'envoyer au serveur à fin de mettre à jour la base de données. Cet appel se fait périodiquement chaque minute pour éviter d'une part de surcharger le serveur et d'autre part ça permet d'avoir plusieurs prises de positions. La figure 36 résume l'ensemble :

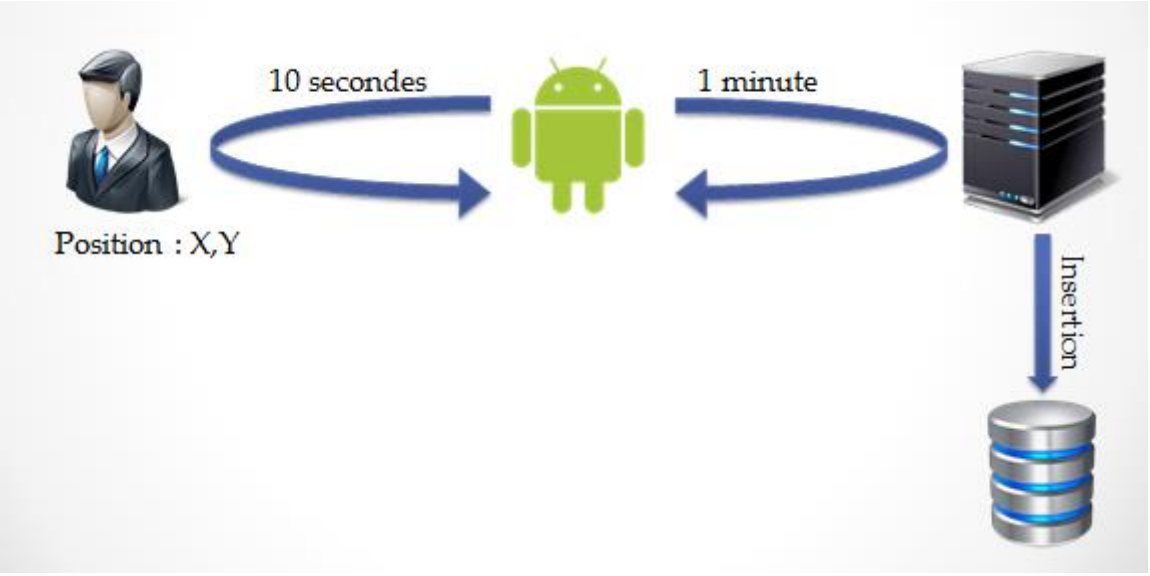



Figure 36: Rafraichissement de la position de l'utilisateur

Ainsi nous n'avons pas à nous soucier de la position des autres amis puisque chacun mettra à jour sa position automatiquement.

## 2- La position du barycentre :

Le barycentre d'un meeting change également continument selon la position des utilisateurs appartenant au meeting. Avec nos tuteurs, nous avons vu qu'il était judicieux d'implémenter une solution qui évite le changement continue du barycentre, en effet avec les erreurs de réception des satellites il se peut que la position des utilisateurs soit captée de manière erronée et si nous faisons le calcul du barycentre avec ces positions alors ce dernier n'arrêtera pas de changer et cela peut causer de sérieux problèmes. Pour pallier ce problème, nous avons appliqué un ensemble de règles sur le barycentre pour qu'il soit modifié :

- Distance entre l'ancien et le nouveau barycentre doit être supérieur à 500m.
- Le délai entre une mise à jour et la dernière effectué doit être supérieur à 5min : Cela afin d'assurer que les utilisateurs aient le même barycentre.

Et on demande la mise à jour du meeting chaque **3min** à partir de l'Android. La figure suivante résume l'ensemble

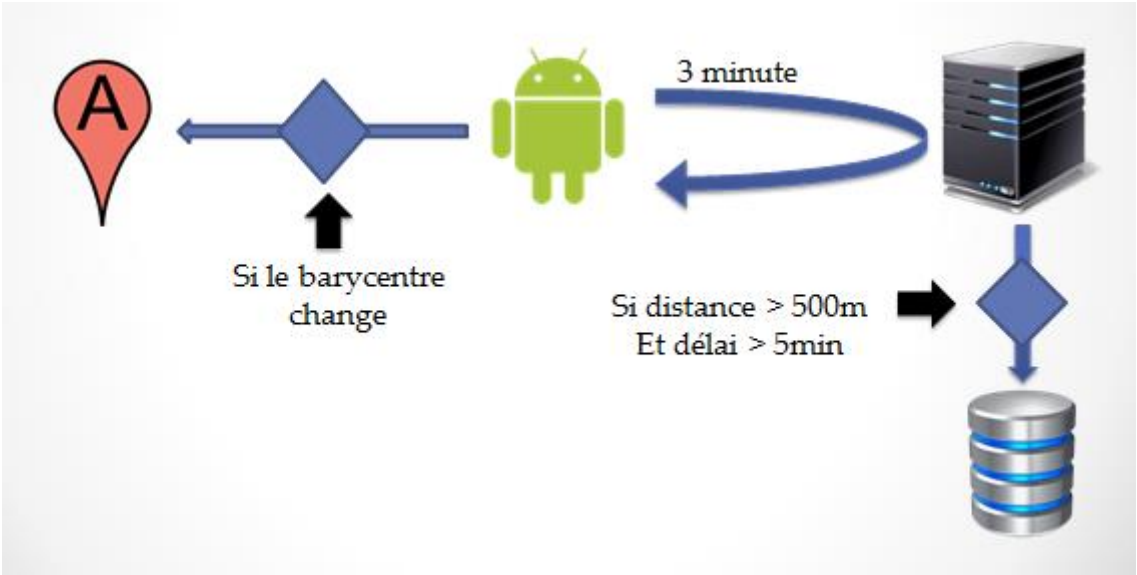



Figure 37: Récapitulatif de la mise à jour d'un barycentre

## 3- La position des abonnés à un meeting :

L'un des points important qu'il fallait rafraichir est la position des utilisateurs dans un meeting. Le même raisonnement que précédemment a été suivi pour le franchissement et cela se fait avec un délai de 1 minute et 45 secondes. Nous avons choisi cette période afin d'éviter l'appel en même temps du serveur pour à la fois la mise à jour du meeting et la mise à jour des abonnés.

La solution qu'on avait conçue au départ rafraichissait toute la carte pour mettre à jour la position des « Markers » ce qui était couteux en temps, en performance et en ergonomie et pour remédier à ce problème nous avons déclaré un tableau de « Markers » dont l'identifiant est celui de l'utilisateur que nous pointons sur la carte et lorsqu'on reçoit les nouvelles positions alors il suffit de récupérer les « Markers » par leurs positions et de les mettre à jour et s'il y a des utilisateurs qui n'existaient pas au départ alors on crée de nouveaux « Markers » et on les affiche.

Nous avons également veillé à ce qu'on ne change pas la position de l'utilisateur principal qui est en mode guidage.

```
function updateMarkersOnMap(jsonUsers, jsonMeeting, idUser) {
    var users = JSON.parse(jsonUsers);
    var meeting = JSON.parse(jsonMeeting);

    var users = JSON.parse(jsonUsers);
    var meeting = JSON.parse(jsonMeeting);
    users.result.forEach(function(route) {
        var val = getMarker(markers, route.idUser);
        if(val != -1) {
            markers[val].setPosition(new google.maps.LatLng(route.XCoord,route.YCoord));
        } else {
            if(markers[val].id != idUser) {
                var marker = new google.maps.Marker({
                        id: route.idUser,
                        map: map,
                        position: new google.maps.LatLng(route.XCoord,route.YCoord),
                        title: route.email,
                        zIndex: i
                    });

                var contentHtml = "<div style='width:200px;height:100px'><h3>"+route.nom + " " +route.prenom+"</h3>"+route.email + " " + route.phone+"</div>";
                var infowindow = new google.maps.InfoWindow({
                    content: contentHtml
                });

                google.maps.event.addListener(marker, 'click', function() {
                    infowindow.open(map,marker);
                });

                marker.locid = route.idUser;
                marker.infowindow = infowindow;

                markers.push(marker);
            }
        }
```

Récupération de l'identifiant du Marker

Si le Marker exixte déjà alors on modifie la position

Si le Marker n'existe pas et est différent du Marker de l'utilisateur principal alors on en crée un

Figure 38: Principe de mise à jour des positions des abonnés à un meeting

Avec cette solution nous avons pu dynamiser la solution assurant un rafraîchissement optimal de données sans avoir à recharger toutes la carte Google Map.

## 3 Sécurité

La sécurité est un point primordial auquel il fallait absolument penser afin d'éviter les attaques serveurs et de ne pas permettre aux personnes n'utilisant pas l'application de faire des appels. En effet, il nous a été demandé de mettre en place un système permettant d'identifier de manière unique chaque téléphone et cette identification servira à générer une clé également identique qu'on testera continument avant chaque appel Web Service. Nous allons ici parlant des deux clés générées ainsi que leur principe de fonctionnement.

### 1- *Clé d'identification*

Cette première clé avait pour but d'identifier le téléphone pour une première fois surtout pour la création d'un utilisateur. La bonne manière d'identifier un Android de façon unique est d'utiliser l' « Android_ID », cette propriété renvoie une chaine de caractères contenant un nombre 64-bit propre à chaque Android. Il nous a été demandé de nous assurer que la clé envoyé pour la première fois au serveur provienne bien d'un Android pour éviter que quelqu'un envoie des requêtes SOAP au serveur en générant des clés de même taille que l' « Android_ID », utiliser directement cet identifiant n'était donc pas une bonne solution. La solution a été d'ajouter 3 caractères à la fin de l' « Android_ID » dont la somme vaut 10 et une fois sur le côté serveur on s'assure que les 3 derniers caractères valent bien 10 pour permettre la fonction de création sinon l'appel ne pourra pas être fait.

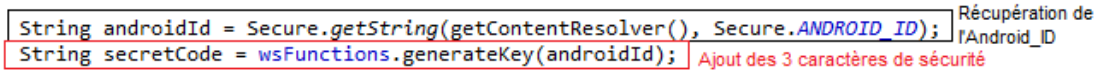



Figure : Génération de la clé d'identification

```
public String generateKey(String key) {
    int min = 0;
    int max = 9;

    int range1 = min + (int)(Math.random() * ((max - min) + 1));
    if(range1==0) ++range1;
    max = 10-range1;
    int range2 = min + (int)(Math.random() * ((max - min) + 1));
    int range3 = 10 - range2 - range1;
    return key+range1+range2+range3;
}
```

Figure 39: Code permettant d'ajouter les 3 caractères sécurité

Cette clé est stockée sur l'Android dans un fichier XML et une fois sur le côté serveur, il suffit de vérifier le code reçu pour utiliser l'application comme présenté en figure suivante :

```
@WebMethod
public User createUser(@WebParam(name="name") String name,
                       @WebParam(name="prenom") String prenom,
                       @WebParam(name="phone") String phone,
                       @WebParam(name="email") String email,
                       @WebParam(name="password") String password,
                       @WebParam(name="secretCode") String secretCode) {   Code envoyé par l'Android

    User userResult = new User();
    boolean bool = true;
    boolean boolCode = verifySecretCode(secretCode);   Vérification du code

    if(boolCode) {   Contrôle
        //vérification existence user par mail + phone
        if(userDao.findUserByMail(email).isEmpty()) userResult.setEmail("false");
        if(userDao.findUserByPhone(phone).isEmpty()) userResult.setPhone("false");

        if(userResult.getEmail().equalsIgnoreCase("false") && userResult.getPhone().equalsIgnoreCase("false")) {
            if (!checkEmail(email)) {
                userResult.setEmail("invalid");
                bool = false;
            }
```

Figure 40: Vérification sur le côté serveur

Une fois le premier appel fait avec le web service et une fois que ce dernier s'ait assuré que la clé reçu provient bien d'un web service, le serveur la stocke dans la base de données et elle servira comme contrôle dans chaque méthode du web service. On donne comme titre exemple la méthode « *getMeetingById* » qui permet de renvoyer les informations d'un « rendez-vous »

```
@WebMethod
public List<Meeting> getMeetingById(@WebParam(name="idMeeting") String id,
                                    @WebParam(name="email") String email,
                                    @WebParam(name="secretCode") String secretCode) {

    if(!userDao.findUserByMail(email).isEmpty()) {
        User u = userDao.findUserByMail(email).get(0);
        if(u.getSecretCode().equals(secretCode)) {
            return meetingDao.findMeetingById(id);
        }
    }

    return null;
}
```

Envoie de l'identifiant

Contrôle du code secret avant de renvoyer les informations

Figure 41: contrôle du code secret avant chaque appel du web service.

## 2- *Clé d'activation*

Cette deuxième clé permet d'activer un compte utilisateur, en effet lorsque celui-ci est créé il est en mode « inactif » et ne peut être utilisé tel quel. Nous générons une clé sur le serveur que nous envoyons par mail à l'utilisateur. Une fois sur l'interface de vérification, l'utilisateur devra renseigner ce code pour activer son compte.

# III. Bug et amélioration

Nous présenterons dans cette partie les bugs connus et nous ferons des propositions pour améliorer l'application.

## 1. Bug connu

- **<u>Zoom sur la carte Google Map lorsqu'il y a un guidage :</u>**

Lorsque nous n'affichons que des « Markers » sur la carte Google Map, nous utilisons la propriété « bounds » qui permet d'ajuster le zoom et fait en sorte que tous les « Makers » soient visible sur la carte, comme le montre la figure suivante :

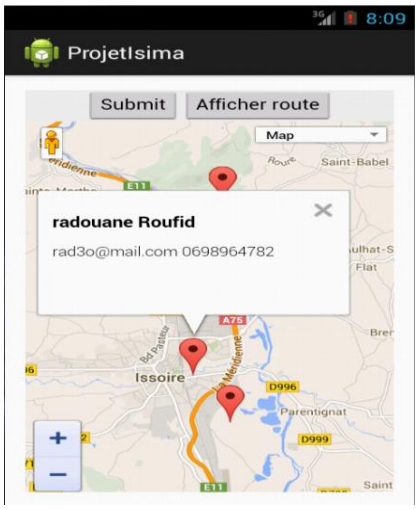



Figure 42: Ajustement zoom et cadrage avec la propriété « Bounds »

Mais lorsqu'il s'agit d'un guidage utilisateur la propriété « bounds » ne fonctionne plus et le cadrage se fait automatiquement sur le guidage comme le montre la figure suivante :

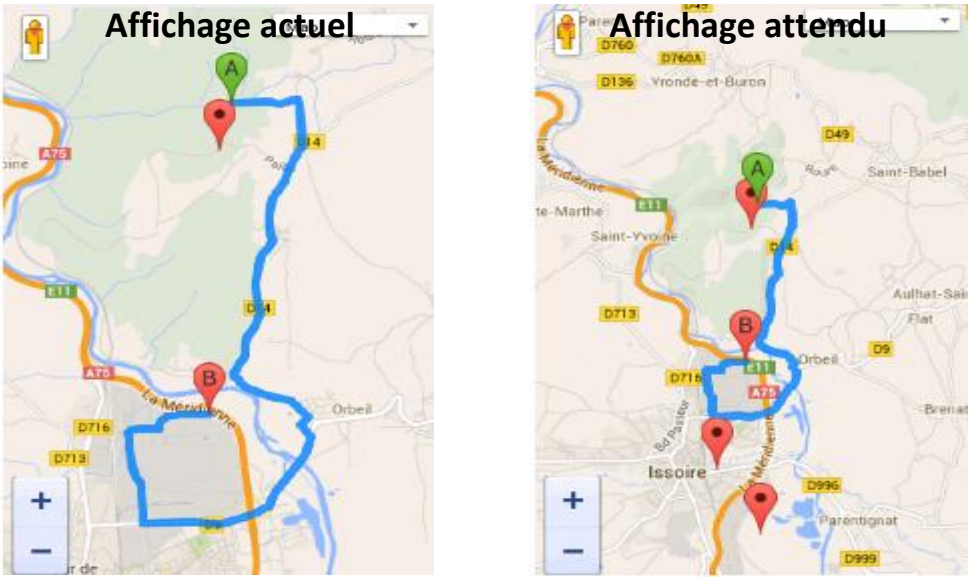



Figure 43: Bug de cadrage lors d'un guidage

Note : Le terme bug ici est mal choisi puisque Google Map trouve judicieux que le cadrage soit effectué sur le guidage de l'utilisateur principal qui doit bien voir la route qu'il doit suivre plutôt que sur l'ensemble de ses amis. L'utilisateur peut dezoomer un peu pour voir ses amis.

## 2. Améliorations pouvant être apportées à l'intranet

- **Messagerie instantanée :**

L'une des améliorations que nous trouvons intéressante est d'ajouter un système de messagerie instantanée permettant aux utilisateurs de rester en contact en s'envoyant des messages. Cette fonctionnalité permettra de rendre l'application plus interactive.

- **Tchat intelligent :**

La bonne structure de l'application actuelle permet d'ajouter une fonctionnalité très intéressante celle d'ajouter un système de tchat permettant à tous les utilisateurs de l'application de communiquer entre eux. Par exemple, lors du lancement de l'application l'utilisateur est redirigé avec la carte de sa ville où sont affichés tous les utilisateurs de l'application dans une zone de 100km par exemple. L'utilisateur peut interagir avec les autres connectés en leur envoyant des messages, les ajoutant comme contact et même organiser des rendez-vous avec eux.

# Conclusion

Le développement du GPS communautaire a globalement suivit le planning prévu, bien que nous avons trouvé quelques difficultés techniques qui ont ralenti un peu la progression mais au final la plupart des fonctionnalités établies au cahier de charges ont été implémentées. Nous avons veillé à ce que l'application ait une bonne ergonomie, que l'application soit robustesse et qu'elle soit utilisable pour toute personne. Les objectifs ont donc bien été atteints.

Ce projet de fin d'étude nous a permis de nous ouvrir sur de nouvelles technologies qui sont d'ailleurs de plus en plus utilisées dans le monde de l'entreprise. En effet, nous avons vu les principaux Frameworks JEE associés à Java Android. Ce projet nous également appris à respecter les plannings, la fiabilité du code source et de veiller à respecter le cahier de charge. Nous avons pu nous convaincre, grâce à cette expérience, que nous sommes parfaitement capables de nous adapter facilement à un milieu technique et à un domaine en très forte croissance tel que le développement mobile.

Pour conclure, ce projet de fin d'étude à l'ISIMA s'est avéré très intéressant et nous a donné un grand souffle nous assurant une bonne insertion dans le monde professionnel.

# Glossaire

## **A**PI

***Application Programming Interface,*** ensemble normalisé de classes, des méthodes ou des fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

## **D**ao

***Data Access Object,*** Modèle regroupant les accès aux données persistantes. Permet de manipuler les objets du modèle avec la base de données.

## **E**JB

***Enterprise JavaBeans,*** est une architecture de composants logiciels côté serveur pour la plateforme de développement Java EE

## **F**ramework

Kit de composants logiciels structurels, qui définissent les fondations ainsi que les grandes lignes de l'organisation de tout ou partie d'un logiciel.

## **G**oogle Map

Une API de Google offrant des fonctionnalités de gestion sur les cartes graphiques.

## **G**lassFish

Un serveur d'applications compatible Java EE

## **G**ps

***Global Positioning System*** (Système de localisation mondiale) est un système de géolocalisation fonctionnant au niveau mondial.

## **J**2EE

Edition du "Java Framework" destinée à un usage professionnel, avec la mise en œuvre de serveurs.

## **J**avaScript

Langage de programmation de scripts utilisés pour les pages web interactives.

## **J**PA

***Java Persistence AP,*** Interface permettant le mapping entre une base de données et des classes Java en utilisant des annotations introduites dans le code.

## **J**son

***JavaScript Object Notation,*** est un format de données textuelles, générique, dérivé de la notation des objets du langage ECMAScript.

## **M**arker

Objet Google Map permettant de pointer sur une position.

## **M**VC

***Model View Controller,*** Type d'architecture qui cherche à séparer nettement les couches de présentation, métier et accès aux données.

## **N**etBeans

Un environnement de développement intégré (EDI), placé en open source par Sun

## **S**oap

***Simple Object Access Protocol***, un protocole permettant la transmission de messages d'un objet client à un serveur. Le transfert se fait souvent à l'aide du protocole HTTP.

## **U**ML

***Unified Modeling Language***, est un langage de modélisation de données.

## **W**ebServices

Programme informatique permettant la communication et l'échange de données entre applications dans un environnement distribué.

## **W**ebView

Widget android permettant de faire un affichage

## **X**ML

***eXtensible Markup Language***, Langage à balises extensible permettant de mettre en forme des documents grâce à des balises.

# Webographie

[Documentation - Google Map Api]
https://developers.google.com/maps/documentation/javascript/
[Dernière consultation le 20 Février 2014]

[Documentation - Android]
https://developer.android.com/develop/index.html
[Dernière consultation le 10 Décembre 2013]

[Documentation - JPA]
http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html/
[Dernière consultation le 24 Janvier 2014]

[Documentation - JEE]
http://docs.oracle.com/javaee/
[Dernière consultation le 05 Janvier 2014]

[Création des web services]
http://docs.oracle.com/javaee/6/tutorial/doc/gijti.html
[Dernière consultation le 15 Novembre 2013]

[Parsing de données XML et Json]
http://cynober.developpez.com/tutoriel/java/xml/jdom/
http://json.org/java/
[Dernière consultation le 20 Janvier 2014]

[Création d'un EJB]
http://www.isima.fr/~lacomme/JEE/part1/cours_JEE_1.pdf
[Dernière consultation le 10 Novembre 2013]

[Présentation LIMOS]
http://limos.isima.fr/

# Bibliographie

Les web services : Concevoir et utiliser des applications 2.0 C#, Java, PHP, API Javascript, Android SDK, iOS SDK
[Par : Philippe LACOMME, Libo REN, Jonathan FONTANEL]