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

Campus des Cézeaux
1 rue de la Chébarde
TSA 60125
CS 60026
63 178 Aubière CEDEX

# RAPPORT D'INGENIEUR

# PROJET DE 3EME ANNEE

# FILIERE : GENIE LOGICIEL ET SYSTEMES INFORMATIQUES

# PROJET WATCHDOGZZ

## DEVELOPPEMENT D'UNE APPLICATION DE CARTE DU MARAUDEUR

*Présenté par :* **Benjamin BARBESANGE & Benoît GARÇON**

Responsable ISIMA : **Pierre Colomb**

**Année 2016 – 2017**
**Projet de 120 heures**

# TABLE DES FIGURES ET ILLUSTRATIONS

# TABLE DES MATIERES

# INTRODUCTION

Dans le cadre de notre projet de troisième année, nous souhaiterions développer une application Android s'inspirant de la carte du maraudeur de Harry Potter. L'objectif serait de pouvoir parcourir en mode réalité augmentée ou virtuelle l'ISIMA en ayant la position et les déplacements des utilisateurs. Concernant la partie réalité virtuelle nous pouvons nous procurer un casque dans un second temps pour augmenter l'immersion.

On peut à tout cela ajouter des objectifs secondaires comme un pathfinding vers d'autres utilisateurs, ou bien encore des invitations à des points de rendez-vous.

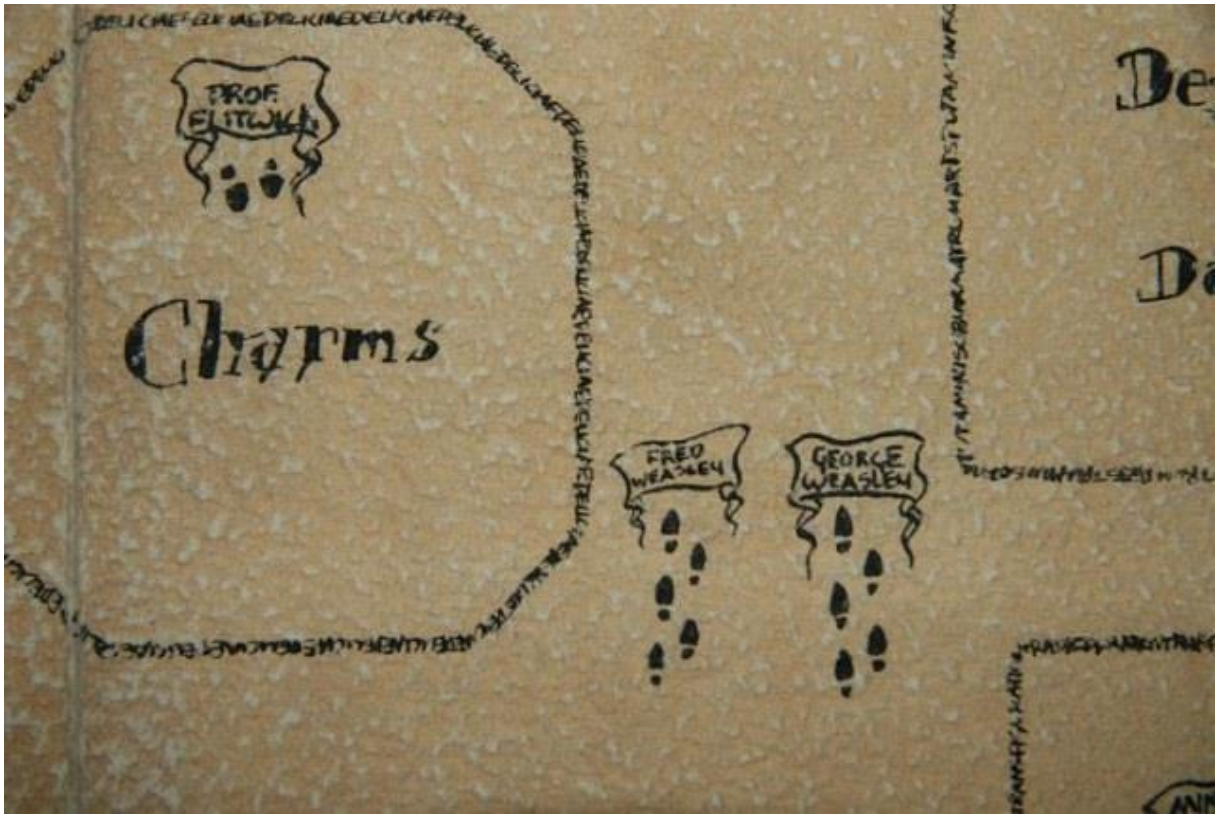


**Figure 1 - Exemple de carte du maraudeur tiré du film Harry Potter**

# Chapitre 1 : SPECIFICATION GENERALES DU PROJET

## 1.1 Architecture

Nous avons choisi de traiter l'architecture du projet en trois parties. La première est la partie serveur, celle-ci sera faite en nodeJS soutenu par une base de données MongoDB. La seconde L'application android sera développé en Java et compatible pour les frameworks Android version 3.0 et supérieurs. L'interface d'administration du serveur quant à elle sera développée en AngularJS. Le serveur d'applications sera un serveur d'application compatible NodeJS sur AWS.

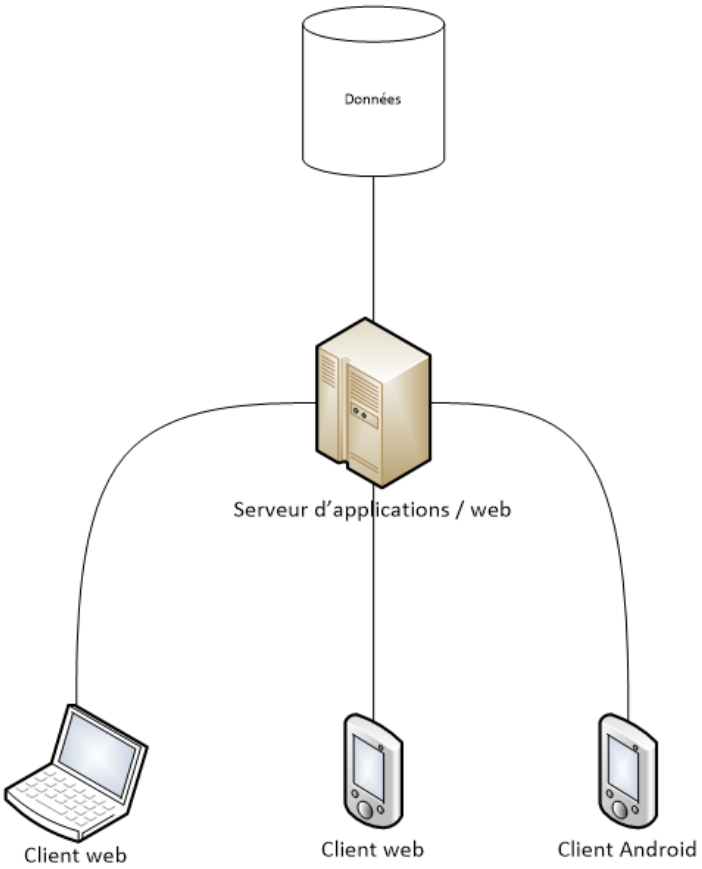


**Figure 2 - Architecture générale**

## 1.2 Partie serveur

La partie centrale est celle du web service. Le serveur doit être capable de répondre aux différentes connexions et requêtes des utilisateurs (se connectant sur l'application Android ou tout autre...). Le serveur sera un service Web permettant (au minimum) :

- de s'authentifier
- d'obtenir les positions d'autres personnes connectées sur l'application
- obtenir la liste des personnes connectées
- envoyer la position actuelle de l'utilisateur connecté
- proposer l'APK

Le serveur pourra aussi en option faire :

- historique des positions
- calcul d'itinéraire
- partage de position

Le serveur comportera une base de données dans laquelle les informations des utilisateurs seront stockées (login / mot de passe). Les dernières positions des utilisateurs seront également stockées. Il y aura une table permettant d'identifier quels utilisateurs sont actuellement actifs.

Il a été convenu d'utiliser NodeJS du coté serveur pour plusieurs raisons :

- rapidité de mise en place
- faible nombre d'utilisateurs
- l'un de nous a des connaissances dessus

### 1.2.1 Requêtes sur le web service

Les requêtes sont soumises directement sur l'URL du serveur avec les différents paramètres requis. Les retours de requêtes seront :

- Pour les positions, une liste de JSON (ou un seul élément): `[{"user":<name>, "position":[<long>,<lat>]}, ...]`
- Pour la liste des utilisateurs : `[<user1>, <user2>, ...]`

Les requêtes possibles sont les suivantes :

- GET
  - /who : retourne la liste des noms des personnes connectées
  - /where : retourne les positions des personnes connectées
  - /where/<user> : retourne la position de l'utilisateur
- POST
  - /me:<long>:<lat> : envoie la position courante de l'utilisateur

L'utilisateur doit être authentifié pour avoir accès aux positions des utilisateurs ou envoyer sa propre position. Cette authentification sera rendue possible par la génération d'un token unique à chaque connexion d'un utilisateur. Ce token sera par la suite utilisé dans chacun de ses requêtes.

## 1.3 Partie administration

Les fonctionnalités minimales pour l'application d'administration sont :

- visualisation des logs du serveur
- visualisation du contenu de la base de donnée

Ensuite les fonctionnalités avancées pourront être :

- gestion des utilisateurs

## 1.4 Partie Android

Une seconde partie comporte une application Android (qui pourrait être déclinée pour iOS et Windows Phone). Cette application permettra :

- de s'inscrire
- de se connecter
- envoyer sa position gps au service web
- recevoir les positions gps d'autres utilisateurs connectés
- visualiser en temps réel sur une carte les positions

L'application pourra évoluer et proposer :

- la carte en version 3D
- la carte en version VR
- l'ajout d'informations sur la carte (lieu / point de rdv)

Le choix d'une application Android se justifie par :

- la communauté Android est active
- la quantité de terminaux
- l'un de nous a des connaissances en Android

La mise à jour des positions est soit faite par l'application qui effectue une requête sur le serveur, soit c'est le serveur qui renvoie les positions des utilisateurs ayant bougé d'un delta suffisant qui permet sa mise à jour chez les utilisateurs. Les frameworks 3.X+ seront supportés pour fonctionner sur un maximum de terminaux.

## 1.5 Intégration continue

Un des objectifs de ce projet est de mettre en place une intégration continue et un déploiement automatique. Pour ce faire il est nécessaire d'avoir deux serveurs :

- un serveur d'application
- un serveur d'intégration

Le premier sera certainement un CaaS Amazon pour NodeJS. Le serveur d'intégration sera un serveur Travis CI puisqu'il gère à la fois le NodeJS et l'Android (nouvelle fonctionnalité). Cela permettra contrairement à un serveur Jenkins de déployer facilement sans avoir à gérer un serveur mais juste en utilisant un service.

# Chapitre 2 : AVANCEMENT

## 2.1 Partie serveur

Du codé du serveur nous disposons à l'heure actuelle de deux parties qui s'interfacent.

La première partie est les Service Web permettant de gérer les requêtes des utilisateurs. Ces requêtes sont prises en compte par un serveur d'application Express utilisé avec NodeJS. Un routing est mis en place et permet les opérations suivantes en fonction des URLs et des méthodes :

- Login : l'utilisateur envoie son nom et sa position initiale avec une méthode POST pour être placé dans la base de données ;
- Where : en utilisant une méthode POST, l'utilisateur est capable d'envoyer son nom et sa position, ce qui va avoir pour effet de mettre à jour sa position dans la base de données ;
- Where : cette fois en utilisant une méthode GET, n'importe quel utilisateur est capable de récupérer une liste des utilisateurs qui sont connectés ainsi que de connaitre sa position ;
- Where/username : une requête GET effectuée sur la route *where* en ajoutant un nom d'utilisateur va permettre d'obtenir uniquement la position de l'utilisateur souhaité ;
- Users : une requete GET permet ici de disposer d'une liste contenant uniquement les noms d'utilisateurs des personnes connectées ;
- Logout : l'utilisateur peut effectuer une requête POST, ce qui va permettre de le déconnecter (en pratique il est supprimé de la base de données).

Afin d'effectuer des manipulations faciles dans le service, des entités User ont été créés. Des instances d'utilisateurs seront créées à partir des informations contenues en base de données lorsque l'on souhaitera effectuer un post-traitement, d'autres instances pourront être créées lors de l'ajout d'un nouvel utilisateur afin d'effectuer un prétraitement particulier si nécessaire.

Une autre partie concerne la base de données. Ici, c'est une base de données MongoDB qui est utilisée du fait de sa configuration aisée et surtout de sont interfaçage facile avec le Service Web en NodeJS, puisque les données transitent en JSON, qui est un formalisme de base pour les données en Javascript.

Cette base de données vient s'inscrire dans le Service Web en utilisant un pattern *bridge*. Ceci permettra par la suite de modifier facilement le type de base de données que l'on utilise pour passer vers SQL, Oracle ou autre.

La base de données est capable d'effectuer les opérations élémentaires sur nos entités Users du Service Web (création, mise à jour, consultation, suppression).

Des tests sous forme de spécifications sont également présent pour tester le bon fonctionnement de ces méthodes. Ce sont ces tests qui sont lancés lors de l'intégration (avant le déploiement) afin de s'assurer du bon fonctionnement du Service et ainsi poursuivre l'étape de déploiement.

En ce qui concerne le déploiement, la plateforme choisie est Amazon opsWorks. L'application est pour l'instant capable de se déployer automatiquement sur une instance d'opsWorks, mais pas encore de lancer le Service Web (des conflits de version NodeJS apparaissent).

## 2.2 Partie client Android

Deux activités ont été produite : une première permettant la connexion et une seconde affichant la carte de l'ISIMA. Cette carte est une vue 3D des bâtiments produite en OpenGL ES. Cette vue 3D est navigable et propose même un menu.

Du data binding a été mis en place pour intégrer les utilisateurs récupérés depuis le service dans l'application, nous avons donc une liaison entre le modèle (liste d'utilisateurs) et la vue (la carte). Ainsi des utilisateurs peuvent être visualisés sur la carte.

Nous avons avancé sur la partie connexion Google, nous avons créé chez Google Dev un projet et inscrit une application Android et un web service. Du coup l'application Android récupère les données du compte Google sur le téléphone, contacte Google avec une clé fourni par Google Dev écrite en dur dans un fichier de configuration et Google renvoie gentiment tout ce qu'on veut (le token surtout). Il ne reste plus qu'au serveur à récupérer ce token et à faire une requête de vérification auprès de Google avec les clés définies dans le projet Google. L'application Android s'authentifie chez Google et récupère les données utilisateurs pour les utiliser dans l'application.

Une clé de debug est utilisée pour signer l'APK et ce n'est pas très sécurisé. Nous avons fait une clé de release avec passphrase mais du coup on ne peut plus passer par Android studio pour build l'application et l'utiliser sur la VM.

La position GPS de l'utilisateur s'affiche sur la carte aux alentours de l'ISIMA et peut-être partagée via un service de messagerie.

L'application dispose d'un service capable de consommer un web service (pour l'instant https://randomuser.me/api/).

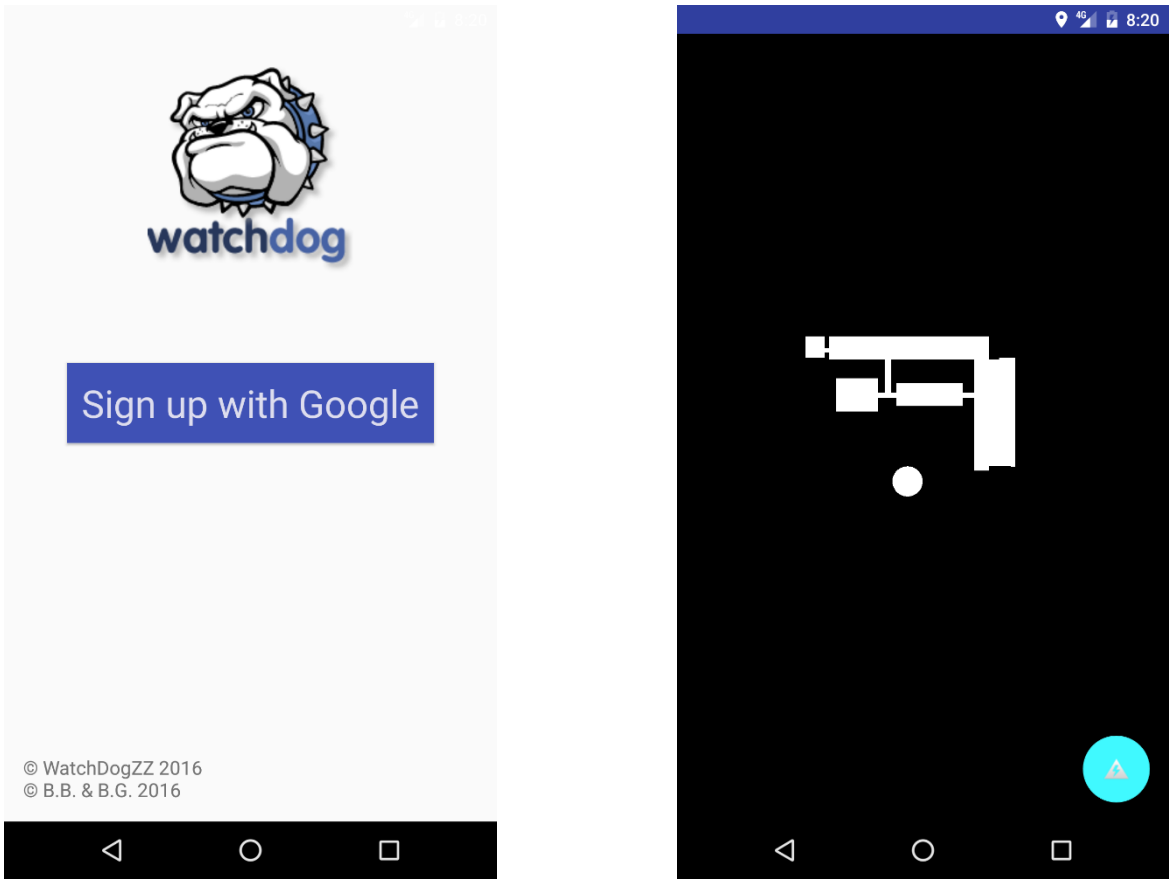


**Figure 3 - Aperçu de l'application Android**

## 2.3 Intégration continue

L'application Android et le serveur sont développés en intégration continue grâce à travis-ci.org.

Lors de chaque nouvel ajout sur le dépôt du projet, les tests de déploiement sont automatiquement lancés et permettent de définir si l'application pourra être déployée (et donc remplacer l'ancienne version) ou non.