



Institut Supérieur d'Informatique
de Modélisation et de leurs
Applications

1 rue de la Chebarde
TSA 60125
CS 60026
63 178 Aubière cedex

Rapport d'ingénieur

Projet de 3^e année

Filière : Génie Logiciel et Systèmes Informatiques

WatchDogZZ

Présenté par : Benjamin BARBESANGE & Benoît GARÇON

Tuteur : Pierre COLOMB
Référent : Eva HASSINGER

Soutenu le 21 Mars 2017
Projet de 120 heures

Remerciements

Avant de débuter notre étude, nous tenons à remercier M. Pierre Colomb, tuteur de ce projet, pour l'accompagnement et les réponses qu'il a su apporter à nos questions. Nous remercions également Eva Hassinger, référente ISIMA au cours de ce projet.

Résumé – Abstract

Résumé

Le travail présenté dans ce rapport concerne l'élaboration d'une solution visant à faciliter l'**orientation** des usagers au sein d'un bâtiment. Pour ceci, les utilisateurs disposent d'une **carte interactive** sur **plateforme mobile** se mettant à jour par des requêtes sur un **service web**, affichant temps réel la position des autres usagers ou de points d'intérêt. Ce projet s'inspire de la carte du **maraudeur** de l'univers Harry Potter et se découpe donc en deux parties distinctes : une partie service web, ainsi qu'une partie application mobile **Android**.

La partie service web est réalisée en utilisant le module **Express** ajouté au framework de base **NodeJS**, permettant de réaliser simplement un serveur web. D'autres modules complémentaires viennent s'ajouter pour disposer de plus de fonctionnalités. Ce service permet de traiter des requêtes soumises par les clients mobiles, ainsi que de stocker des données relatives au bon fonctionnement de l'application.

La seconde partie concernant l'application Android permet à un utilisateur d'afficher la carte interactive et de se connecter au service web. Cette application effectue des requêtes de mise à jour de la position de l'utilisateur sur le service web. Ainsi l'application des autres utilisateurs est capable de récupérer ces **données ouvertes** et de les placer au sein sa propre carte interactive.

Les tests de cette solution ont pu montrer qu'il est possible d'afficher la position des utilisateurs sur une carte modélisée de l'ISIMA. Cependant, des erreurs de positionnement se retrouvent dans la position des utilisateurs, du fait de la mauvaise réception GPS par le mobile. Cette solution mise en place pourra faire l'objet d'améliorations futures telles que l'ajout d'itinéraires entre deux utilisateurs ou vers un point d'intérêt. L'aspect modulaire de cette solution offre de multiples possibilités d'améliorations dans le futur.

Mots clés : Orient, interactive map, mobile platform, web service, marauder, Android, Express, NodeJS, open data.

Abstract

The work introduced in this report is about a solution to make it easier for people in buildings to **orient** themselves. For that, users will be able to consult an **interactive map** displayed on a **mobile platform**. This map is constantly updating itself by making requests on a **web service**, allowing to view in real time other users' positions or interest points. This project is inspired by Harry Potter's **marauder's map** and is splitted into two parts : a web service part, and an **Android** mobile application.

The web service part is realized using the **Express** module added to the **NodeJS** framework, allowing to easily create a web server. Other additional modules will also be added to get more features. This service allows to process requests send by mobile clients as log as storing data relative for proper application running and updating.

The second part concerns the Android application and allows for a user to visualize the interactive map by connecting on the web service. This app makes requests to update the user's position on the web service. Thus, other users are able to recover these **open data** and place them on their own map modeling the building.

The tests on the solution demonstrated that it is possible to print the users' positions on a modeled map of ISIMA. However positioning errors are still present when locating users, due to bad GPS reception in the building. The build solution could be enhanced by adding a feature to visualize route between two user or to an interest point. The modular aspect of this solution offers several enhancement opportunities in the future.

Keywords: Orientation, carte interactive, plateforme mobile, service web, marauder, Android, Express, NodeJS, données ouvertes.

Table des matières

Remerciements	i
Résumé – Abstract	ii
Table des matières	iv
Liste des figures, tableaux, algorithmes et extraits de code	v
Glossaire	vi
Introduction	1
1 Etudes préalables	2
1.1 Présentation du projet	2
1.2 Analyse de l'existant	3
1.2.1 Les services open data	5
1.2.2 Les applications similaires	6
1.3 Spécifications du projet	7
1.3.1 Architecture	8
1.3.2 Partie serveur	9
1.3.3 Partie Android	10
1.3.4 Intégration continue	11
1.4 Organisation du travail	11
2 Conception de la solution	13
2.1 Conception du couple client/serveur	13
2.1.1 Web Service	13
2.1.2 Application Android	14
2.2 Technologies utilisées	19
2.2.1 Service Web	19
2.2.2 Android	20
2.2.3 Intégration continue	23
2.3 Fonctionnalités introduites par la solution	24
2.3.1 Concepts introduits	24
2.3.2 Utilisation du service	25
2.4 Méthodes de développement	30
3 Résultats	33
3.1 La solution apportée	33
3.2 Améliorations possibles	38
Conclusion	40
Références webographiques	viii

Liste des figures et extraits de code

Liste des figures

1.1	Morceau de la carte du Maraudeur tirée du film Harry Potter	2
1.2	Service Safety Check de Facebook	4
1.3	Suivi des déplacements sur Google Maps	5
1.4	Localisation en temps réel des bus de la ville de Rennes	6
1.5	Application Find My Friends de Family Safety Production	7
1.6	Architecture générale de la solution	8
1.7	Diagramme de Gantt théorique	12
2.1	Architecture du Service Web	13
2.2	Patron de conception MVC	15
2.3	Diagramme du modèle	16
2.4	Diagramme du contrôleur	16
2.5	Diagramme de la vue	17
2.6	Utilisation du token Google	18
2.7	Fonctionnement de la 3D Android	19
2.8	Fonctionnement d'un requête Volley	21
2.9	Environnement de développement intégré officiel d'Android	22
2.10	Diagramme de séquence des requêtes sur le service	29
2.11	Liste d'issues (en haut) suivie de la milestone de l'itération 1 (en bas)	31
2.12	Diagramme de Gantt réel	32
3.1	Architecture finale simplifiée	33
3.2	Logo et page de connexion de WatchDogZZ	34
3.3	Vue principale de la carte et menu glissant	35
3.4	Liste détaillée des utilisateurs	35
3.5	Ajout d'un point d'intérêt	36
3.6	Liste détaillée des points d'intérêt	37

Liste des extraits de code

2.1	Exemple de fichier de configuration Travis (service web)	23
2.2	Script de déploiement du service	24
2.3	Corps général de la réponse serveur	26
2.4	Corps de la requête POST /login	26
2.5	Corps de la réponse GET /who	26
2.6	Corps de la réponse GET /where	26
2.7	Corps de la requête POST /where	27
3.1	Hashage du nom en couleur	36

Glossaire

Activité : En Android, une activité est une entité importante représentée par une vue visible sur l'écran d'un terminal.

API : Une API (Application Programming Interface) est une interface d'un logiciel proposant des services au travers d'une bibliothèque ou d'un framework.

DNS : Domain Name System, permet d'attribuer une URL textuelle à une adresse ip. Ce type de service est principalement utilisé pour accéder à des sites web plus facilement.

DynDNS : Dynamic Domain Name System, méthode de mise à jour automatique d'un service DNS. Permet de mettre à jour une ip dynamique correspondant à un DNS.

Fragment : En Android les fragments sont des morceaux d'activité qui permettent de réutiliser des sous parties d'une activité sans avoir à instancier plusieurs activités.

Framework : Un framework est un ensemble cohérent de bibliothèques.

Hashage : On appelle fonction de hashage, une fonction qui à partir d'un élément donne son empreinte permettant de l'identifier rapidement.

HTTPS : HyperText Transfer Protocol Secure, combine le protocole http avec une couche de sécurité TLS. Ce protocole permet de vérifier l'identité du site web auquel un utilisateur accède par l'intermédiaire d'un certificat d'authentification émis par une autorité de confiance. Ce protocole garantit également la fiabilité et la confidentialité des données envoyées.

IDE : Un Integrated Development Environment ou Environnement de Développement Intégré (EDI) en français est un outil de génie logiciel permettant de produire du code source assisté par tout une suite d'outils (éditeur de texte, refactoring, autocomplétion, builder, versionning, etc.).

Intent : En Android, les intents sont des descriptions d'une action à effectuer et pouvant comporter des données nommées. Les intents sont produits par les applications pour être dirigées par le système Android vers l'application la plus capable d'effectuer l'action et en cas de choix multiple, la décision est donnée à l'utilisateur.

JSON : JavaScript Object Notation, format de représentation textuel des objets en Javascript.

Milestone : Une milestone est une étape clé d'un projet.

OpenGL : C'est un framework « normalisé » permettant la conception de programme 3D sur de nombreuses plateformes.

Pathfinding : C'est la dénomination donnée aux algorithmes permettant de trouver un chemin entre deux points.

SDK : Un SDK (Software Development Kit) est un ensemble d'outils utiles dans le cadre d'un développement logiciel.

TLS : Transport Layer Security, protocole de sécurisation des échanges sur le web.

URL : Uniform Resource Locator, permet de localiser une ressource sur le web (page, image, article).

User Story : Une user story ou récit utilisateur est l'explication simple d'une fonctionnalité à développer et comportant les réponses aux questions : qui ? quoi ? pourquoi ?

WYSIWYG : Le « what you see is what you get » ou « ce que vous voyez est ce que vous obtenez » est une interface graphique permettant de créer visuellement d'autres interfaces sans l'utilisation de code.

Introduction

Ce rapport a été rédigé dans le cadre du projet de troisième année du cycle ingénieur à l'Institut Supérieur d'Informatique de Modélisation et de leurs Applications (ISIMA) réalisé sur une durée de 120 heures par personne. Nous avons proposé de notre propre initiative le sujet de ce projet : la conception d'une carte interactive d'un établissement. Cette idée se base sur un constat très simple : il est parfois difficile de s'orienter dans un bâtiment de grande taille et de trouver une personne en mouvement en son sein.

Ce projet s'inspire en grande partie de la carte du maraudeur de l'univers Harry Potter, carte sur laquelle il est possible de suivre en temps réel le déplacement de toutes les personnes dans l'enceinte de Poudlard, l'école des sorciers. L'objectif est donc de proposer et mettre en place une solution évolutive, innovante et pratique pour les utilisateurs afin de s'orienter dans les bâtiments de l'ISIMA. Nous pouvons penser que ce type de solution peut s'étendre à tout type de bâtiment au sein duquel il est autorisé et possible d'être localisé. Cette solution peut avoir des applications dans le domaine du secourisme, ce qui peut permettre aux sapeurs-pompiers de localiser des personnes facilement dans un bâtiment enfumé, permettre à des entreprises hébergeant des données sensibles de localiser ses visiteurs ou collaborateurs, ou encore d'optimiser les déplacements de personnes dans des bâtiments de grande taille comme une aide à l'orientation de médecins dans un hôpital ou de techniciens dans une usine.

Nous verrons que la solution mise en place s'organise en deux principaux composants. Le premier composant consistera en un service web capable de répondre à des requêtes utilisateur de type HTTP. Ces requêtes permettront aux utilisateurs d'envoyer leur position afin de la stocker sur le serveur et de l'envoyer aux autres utilisateurs qui en font la demande. Le second composant consistera en la création d'un client du service web qui affichera à la fois les données sur les lieux mais permettra aussi le suivi et l'interaction avec ses usagers. La plateforme cible choisie pour le développement du client est une plateforme mobile afin qu'il puisse être utilisé en tout lieu et à tout moment. Ces deux parties, quoi que centrales, s'articulent au sein d'un ensemble plus complet d'outils de génie logiciel donnant à la solution une identité unique et que nous détaillerons par la suite.

L'étude débutera par une partie d'études préalables plus précises sur le sujet tant au niveau du travail à fournir pour la réalisation de la solution que de l'état de l'art en la matière. Ensuite, nous détaillerons dans une seconde partie la conception de cette solution en détaillant nos méthodes et outils, pour enfin terminer ce rapport par les résultats finaux de notre travail et discuter du potentiel de notre application.

1 Etudes préalables

Avant de commencer à parler de tout l'aspect technique et conception nous allons résigner notre projet en définissant ses objectifs, son plan de route et ses inspirations.

1.1 Présentation du projet

Ce projet a été mis en place de notre propre initiative et nous avons donc défini les objectifs à atteindre ainsi que les fonctionnalités de nous-même, avec l'aide du tuteur de projet.

L'idée originelle que nous avions proposée était inspirée de la carte du Maraudeur de l'univers Harry Potter visible sur la figure 1.1. Cette carte permet grâce à la magie de suivre en temps réel toutes les personnes présentes à Poudlard, l'école de sorciers, puisque leurs noms apparaissent dessus avec leur nom. Cet objet, bien que magique, nous semblait aujourd'hui pouvoir être réalisé grâce aux avancées liées au domaine des objets connectés.



Figure 1.1 – Morceau de la carte du Maraudeur tirée du film Harry Potter

Nous avons donc décidé de proposer une manière simple pour tout le monde de pouvoir s'orienter et retrouver des personnes dans les locaux de l'ISIMA au travers d'une application. Cette solution devra donc récolter un ensemble des données sur les usagers de l'ISIMA et les reporter sur une carte qui sera bien sûr électronique. Nous voulons ainsi que chacun puisse sortir notre carte de sa poche pour retrouver un ami ou un collègue instantanément.

De manière générale, nous devons être capable de visualiser une carte de l'ISIMA, mais également de pouvoir trouver facilement une personne, un bureau ou une salle de cours. De plus, il serait intéressant de pouvoir faire ceci en temps réel. L'actualisation des données doit donc être assez rapide pour que la localisation des utilisateurs soit la plus précise possible.

Une fois cette solution éprouvée avec les bâtiments de l'ISIMA, nous pouvons penser l'étendre à n'importe quel autre bâtiment dont nous pouvons avoir les plans. L'intérêt ici étant de développer une preuve de fonctionnement du suivi de personnes dans un bâtiment qui plus est considéré comme étant très surchargé de signaux. Comme nous l'évoquions plus tôt ceci pourrait trouver différentes utilisations dans différents contextes comme par exemple le fonctionnement d'un hôpital, la surveillance d'une prison, l'évacuation d'un bâtiment. Avec l'avènement de l'open data, il y a aussi de nombreuses questions éthiques à résoudre vis-à-vis des informations récoltées et diffusées. La CNIL au travers de la loi Informatique et Libertés réglemente l'utilisation de la géolocalisation d'employés dans une entreprise [1] en mettant en avant les droits des employés : il faut que cette géolocalisation est une finalité claire et prédéterminée, que les données ne soient pas conservées plus longtemps que prévu, l'employé doit avoir connaissance du dispositif et doit pouvoir l'interrompre. Toutefois, la CNIL n'est pas très précise quant à la géolocalisation de particuliers. Il existe aussi une directive européenne sur les traceurs [2] qui demande à ce que les internautes donnent leur consentement avant l'introduction d'un traceur, par exemple un cookie. Cette demande de consentement existe déjà plus ou moins sur les applications Android puisque lors de l'installation d'une application, celle-ci fait part du besoin d'accéder au gps, ce que l'utilisateur est libre d'autoriser ou non. A ceci on peut rajouter les lois sur la protection de la vie privée et des données personnelles.

Il faut aussi se demander comment notre application va se positionner par rapport aux applications existantes. Pour ce faire nous allons maintenant aborder l'analyse de l'existant.

1.2 Analyse de l'existant

Nous ne nous attarderons pas dans cette partie sur la carte magique imaginée par J. K. Rowling mais nous allons plutôt voir les différentes applications existant autour de la localisation d'usagers.

Le premier exemple est celui des réseaux sociaux. En effet, avec leur avènement est arrivé aussi un moyen pour des grands groupes de récolter une masse importante d'information sur leurs utilisateurs : ce que l'on appelle le Big Data. Dans cette masse de données, il y a bien évidemment des données de géolocalisation. Celles-ci sont de plusieurs types : soit données, soit mesurées.

Les premières, dites données, sont les informations que l'utilisateur va donner de lui-même aux réseaux sociaux comme par exemple poster une photo d'une après-midi entre amis en spécifiant le lieu de la prise de vue. Cette information peut être légitimement utilisée puisque donnée par l'utilisateur mais cela ne représente pas une source fiable ni même précise. Même si cette information n'est pas fiable, les réseaux sociaux ont trouvé depuis 2014 une toute nouvelle utilisation à ce type d'information : l'aide aux personnes en danger. En effet, Facebook a développé un service nommé Safety Check [3] qui repère les personnes présentes à proximité d'une zone de danger que ce soit une catastrophe naturelle ou bien encore une action terroriste et leur propose de se signaler en sécurité auprès de leurs proches. Le Safety Check visible à la figure 1.2 est un bon exemple de localisation de personnes mais demande à ce que les personnes recherchées effectuent une action.

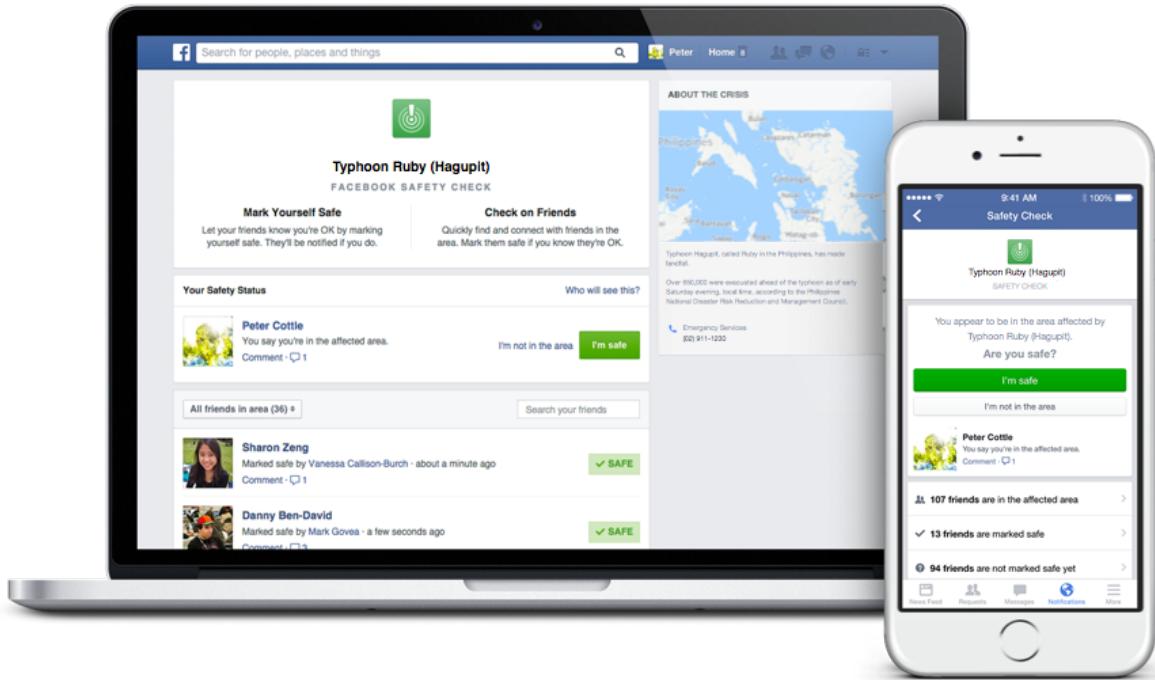


Figure 1.2 – Service Safety Check de Facebook

Le second type dit données mesurées représente toutes les informations que les réseaux sociaux récupèrent sur les appareils des utilisateurs. Cette récupération est rendue possible tout d'abord par l'existence de ces données. En effet depuis quelques années avec l'arrivée des smartphones et des objets connectés, la majeure partie des appareils électroniques possèdent des fonctionnalités de géolocalisation. Il est alors facile pour une application d'accéder à la position de son utilisateur. En général, l'appareil en question est un smartphone et se trouve très souvent au même endroit que son propriétaire, voire sur lui-même. On peut clairement constater la récupération de ces informations lorsque la localisation d'un utilisateur est associée directement au message qu'il poste ou bien encore sur les historiques. En effet, Google propose à ses utilisateurs d'accéder à tout l'historique de leurs déplacements [4] comme on peut le voir sur la figure 1.3. Google récupère en permanence la position de utilisateurs afin de pouvoir fournir à la communauté des statistiques sur l'affluence de certains lieux, de demander des informations et photos sur les lieux en cours de visite ou bien proposer des services à proximité. Google arrive ici à localiser avec précision dans quel établissement une personne se trouve mais ne partage pas les données individuelles au reste de la communauté.

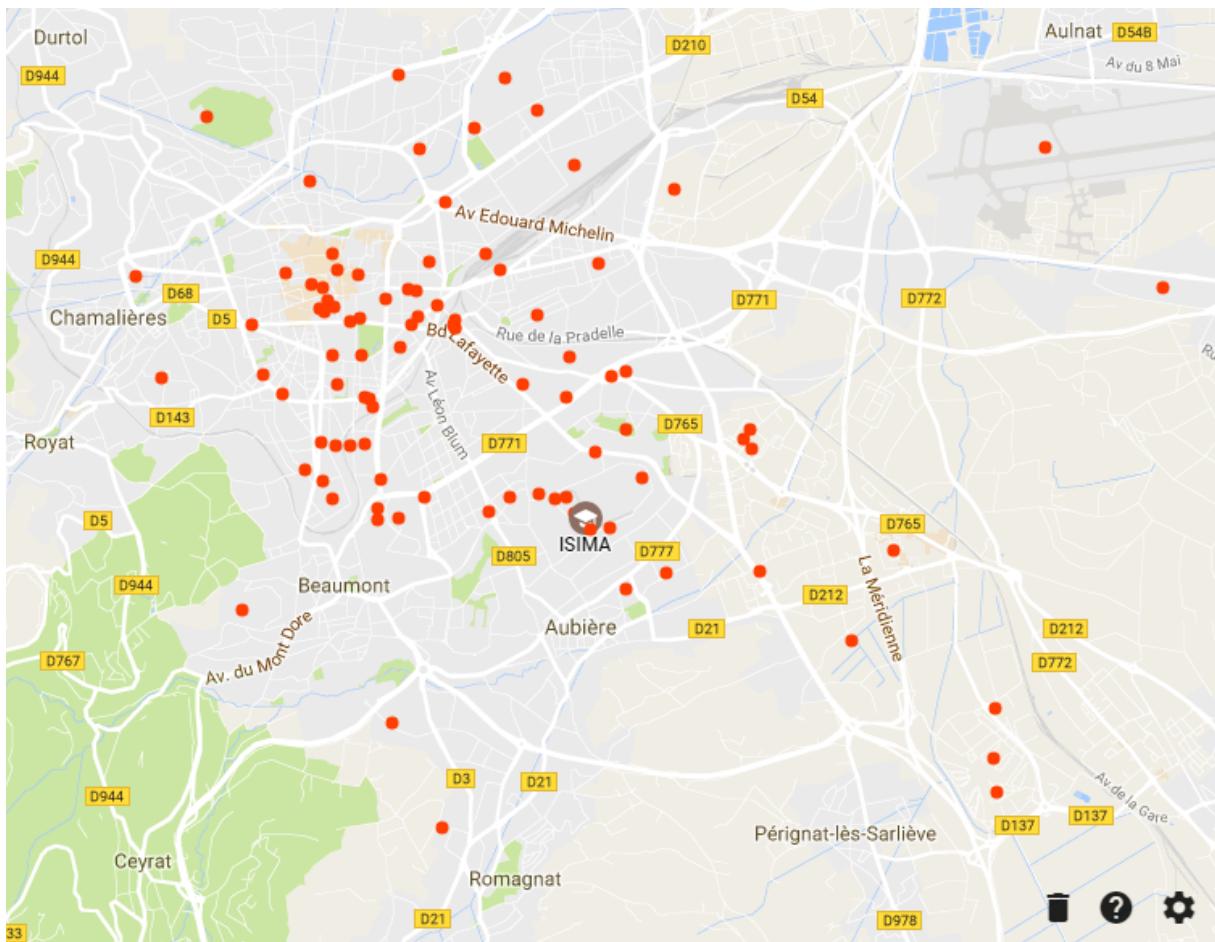


Figure 1.3 – Suivi des déplacements sur Google Maps

Les réseaux sociaux sont donc un acteur de premier plan dans la géolocalisation de personnes puisqu'ils disposent de nombreuses données sur le sujet mais ne diffusent qu'une fraction de celles-ci. Nous allons maintenant voir qu'il existe des services plus ouverts.

1.2.1 Les services open data

Dans les communautés de développeurs le phénomène open source qui consiste à partager librement les sources de ses propres programmes connaît beaucoup de succès. Sur cette base a émergé un nouveau concept : celui de l'open data. Comme son nom l'indique l'open data consiste à partager des données utiles récoltées.

Ces données sont le plus souvent fournies sous la forme d'un service web ou bien d'un fichier facilement découpable et analysable comme un fichier xml ou csv. L'intérêt est que ces données soient réutiliser par d'autres personnes en vue de concevoir un service plus consistant que des données brutes. Ainsi on donne aux développeur le moyen de concevoir des applications se basant sur de vraies données ce qui a pour effet de motiver l'innovation et la concurrence dans un domaine.

Un des exemples les plus connus en France et qui rejoint notre projet est celui de la société Keolis qui gère le service de transport en commun de la ville de Rennes. Cette société a décidé depuis sept ans déjà d'offrir au grand public des données utiles sur son activité de transport dans la ville de Rennes. En outre, Keolis a équipé ses bus de balises gps et partage en temps réel la position des bus des diffé-

rentes lignes [5]. Il y a deux avantages à cette ouverture pour Keolis : la communauté des développeurs peut concevoir d'innombrables applications reposant sur des données sans cesse renouvelées et ainsi disposer sans efforts des meilleurs services mobiles en matière de transport en France. En effet, en comparaison d'une ville comme Clermont-Ferrand où les applications de transports reposent sur les horaires fixes décidés en début d'année, la ville de Rennes propose une application souple qui s'adapte même aux retards individuels de chaque véhicule.

Comme on peut le voir sur la figure 1.4, la flotte rennaise peut être localisée à tout instant.

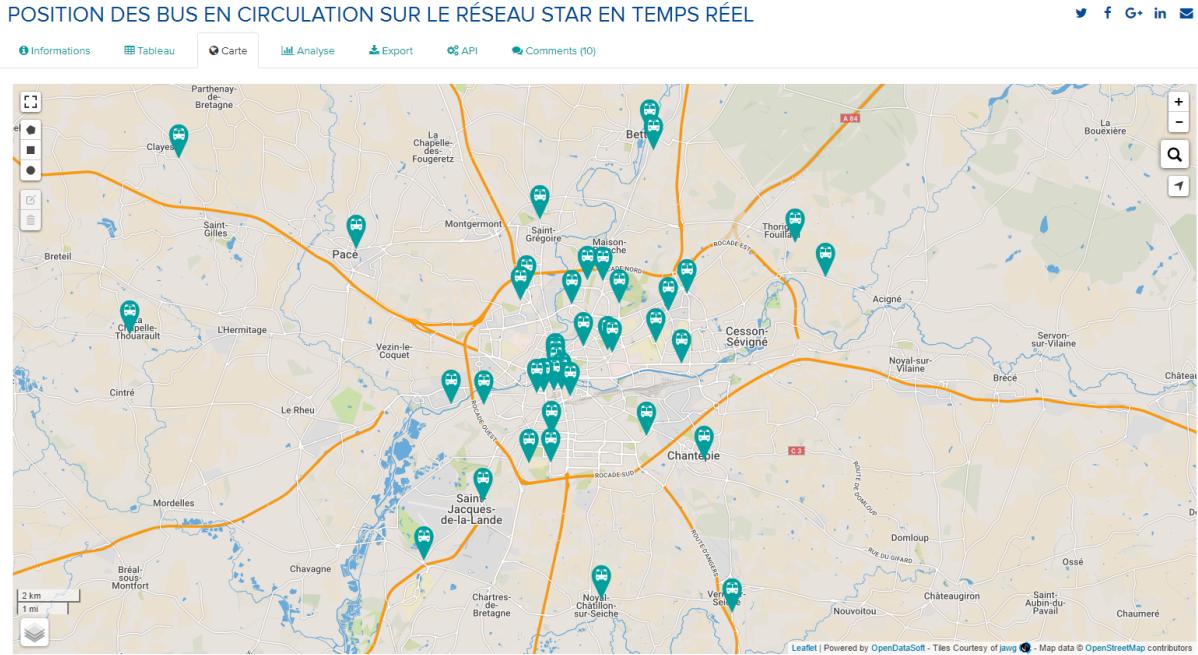


Figure 1.4 – Localisation en temps réel des bus de la ville de Rennes

Dans ce cas, il est vrai que l'on ne parle pas de localiser directement des personnes mais des véhicules. Cependant des problèmes similaires se posent et se sont posés à commencer par la protection de l'identité. En effet, le service web proposait initialement d'obtenir pour chaque bus son numéro d'identification mais ceci permettait de suivre un véhicule en permanence. On peut imaginer beaucoup de choses sur l'utilisation d'une telle information. A chaque véhicule étant associé un chauffeur, n'importe qui pouvait donc localiser un chauffeur en temps réel dans le cadre de l'exercice de ses fonctions mais cette fonctionnalité a été retirée puisqu'elle n'apportait rien de plus aux usagers.

Il y a donc possibilité de produire des applications qui donnent accès à des positions en temps réel au grand public mais celles-ci posent vite des problèmes d'éthique et légaux. Nous allons donc voir les applications réelles qui existent sur la localisation de personnes.

1.2.2 Les applications similaires

Dans la plupart des recherches effectuées, on trouve des dispositifs électroniques permettant de suivre une personne dépendante comme un enfant, une personne âgée ou handicapée.

Par exemple la société Geotek propose à ses clients des boîtiers GSM permettant de localiser une personne [6]. Ils proposent leur utilisation pour les personnes dépendantes et même pour l'optimisation des déplacements d'employés dans une entreprise. Cette offre se rapproche un peu plus de notre

projet mais s'en éloigne un peu dans le sens où elle nécessite l'utilisation de matériel supplémentaire et concerne la localisation de seulement quelques personnes auprès d'une seule personne.

En cherchant sur le Google Play qui est la plateforme officielle de téléchargement des applications Android on trouve quelques applications de localisation. Ces applications prennent bien souvent la forme de réseaux sociaux pour les soucis de confidentialité évoqués plus tôt.

Dans cette catégorie on peut parler de Find My Friends [7] produite par Family Safety Production (figure 1.5). Cette application propose des fonctionnalités intéressantes puisqu'elle permet de localiser ses amis qui possèdent l'application. Mais ce n'est pas tout, ils proposent aussi de pouvoir localiser les personnes qui n'ont pas l'application en envoyant juste un SMS à cette personne et si elle répond « oui » sa position s'affiche chez le demandeur. Cette fonctionnalité est très intéressante car elle permet de s'abstraire de l'application chez une partie des personnes.

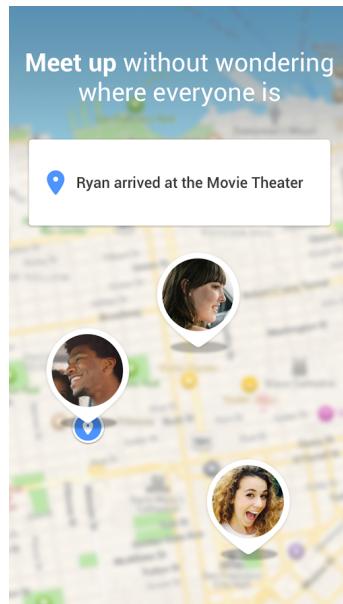


Figure 1.5 – Application Find My Friends de Family Safety Production

Dans le même esprit on retrouve aussi l'application Foursquare. Cette application permet aux utilisateurs de se géolocaliser auprès des autres dans lieux comme des magasins. A ceci est associé un système de gamification, c'est-à-dire que des passages répétés dans un même endroit permet de gagner des badges en récompense. Foursquare proposait aussi de trouver le domicile des utilisateurs et les contacts à proximité. La fonctionnalité signalement du domicile a cependant été modifiée par soucis de confidentialité.

Le point étant fait sur les applications existantes nous allons maintenant définir plus précisément ce que nous souhaitons proposer au travers de notre application.

1.3 Spécifications du projet

Etant donné que peu de solutions sont disponibles sur le marché, nous avons choisi de partir de zéro et créer notre solution. Cela nous permet également d'avoir un contrôle total sur les données du projet, les diverses implémentations de fonctionnalités ainsi que la manière dont nous souhaitons utiliser notre solution.

La solution la plus évidente en termes de support de visualisation pour les utilisateurs est de leur proposer une application qu'ils pourront installer sur leur smartphone, PC, montre connectée ou encore tablette.

Afin de rendre cette application dynamique et de proposer un suivi de position d'utilisateurs en temps réel, il est convenu d'utiliser un service web. Ce service devra également pouvoir stocker des informations utiles aux utilisateurs, ce qui permettra également d'alléger le volume de données stockées sur leurs terminaux.

Nous allons donc détailler dans la suite les principaux éléments que nous voulons produire pour donner une vision de l'objectif final que nous souhaitons atteindre.

1.3.1 Architecture

Tout d'abord nous souhaitons fournir un service à un groupe de personnes plutôt important : nous avons donc besoin d'une architecture qui puisse gérer plusieurs utilisateurs sur une même application. L'architecture choisie pour organiser notre solution en est une de type client-serveur comme elle peut être décrite sur la figure 1.6.

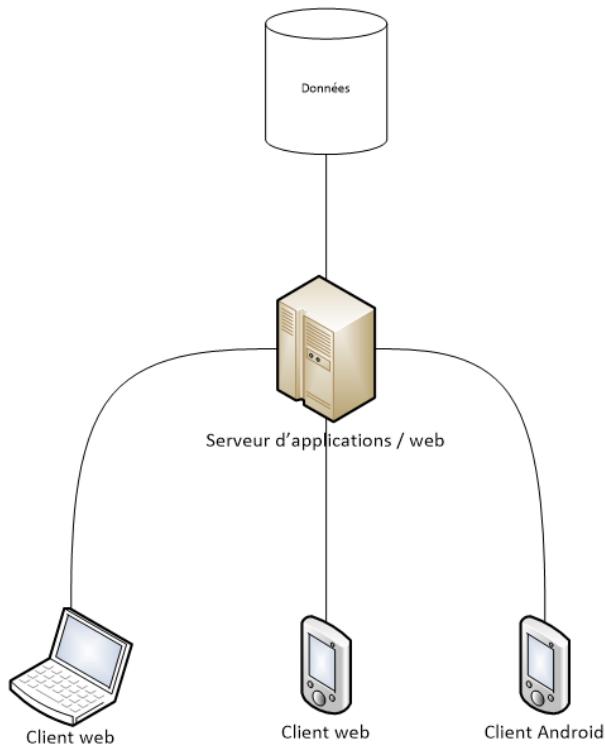


Figure 1.6 – Architecture générale de la solution

Nous pouvons donc observer sur cette figure un schéma tout à fait classique de système client-serveur qui nous permettra de dissocier le développement du service du développement des différentes interfaces utilisateur. Un intérêt majeur est de pouvoir faire évoluer un des clients sans que cela n'impacte le cours de vie des autres projets reposant sur le service ou le service lui-même.

Pour ce faire nous allons donc développer un service web de type REST [8] pour que différents clients, illustrés dans notre projet par un client Android, puissent l'utiliser avec un minimum de contraintes. Les évolutions du serveur et du client n'auront donc pas d'impact l'un sur l'autre sauf bien sûr sur les

modifications de l'interface. Si jamais un problème se déclarait dans la conception d'une partie nous comptons sur notre système d'intégration continue pour jouer le rôle de garde-fou en contrôlant le code versionné.

Nous allons maintenant aborder en détails les objectifs fixés pour chacun de ces éléments.

1.3.2 Partie serveur

Le serveur est le point reliant tous les clients et possèdent différents services mais la partie centrale est celle du service web. Le serveur doit être capable de répondre aux différentes connexions et requêtes des utilisateurs se connectant depuis l'application Android ou depuis tout autre client. Un service web n'est autre qu'un ensemble de services distants disponibles sur internet, ils peuvent permettre de récupérer des données ou bien d'effectuer des traitements spécifiques. L'intérêt est que le mode de communication avec un service web se veut standard ce qui permet à des utilisateurs de topologie différentes de le consommer.

Le serveur sera un service web de type **REST** c'est-à-dire qu'il sera capable de répondre à des requêtes http dans un format standardisé, le JSON. Ce type de service est de plus en plus répandu car il est supposé **sans état** du client sur le serveur, effectue des requêtes **auto-descriptives** (informations sur son contenu, sa date de création, son expiration, ...) sur des URL **identifiant** des ressources de manière **précise**. Ceci en fait donc une base idéale pour les architectures basées sur des services web.

Le serveur disposera des fonctionnalités minimales suivantes :

- un système d'authentification afin de valider l'identité des utilisateurs ;
- un service de collection des positions des utilisateurs ;
- un service d'obtention de la liste des personnes connectées ;
- un service de récupération de la position actuelle de l'utilisateur connecté ;
- un système de connexion/déconnexion des utilisateurs.

Ces fonctionnalités seront essentielles pour le fonctionnement de base de l'ensemble de la solution. Par la suite, des fonctionnalités supplémentaires pourront être ajoutées dans l'application, comme par exemple :

- un historique des positions ;
- un service de calcul d'itinéraire entre deux positions dans le bâtiment ;
- un service de partage de position (par sms, mail, etc.).

Pour stocker les diverses informations utilisateur (login, position), nous avons convenu d'utiliser une base de données qui s'interfacerai directement avec le service web.

L'ensemble devra donc être disponible sur une machine atteignable sur internet et devra avoir une haute disponibilité. C'est pourquoi il n'est pas concevable de le déployer sur un ordinateur personnel. Deux solutions s'offrent donc à nous :

- l'utilisation d'une machine dédiée autogérée ;
- la location d'un IaaS ou CaaS (Infrastructure/Container as a Service).

La première solution consiste à utiliser une machine personnelle connectée en permanence à internet depuis notre domicile et la seconde à louer un service chez un fournisseur comme Amazon ou Google qui pourrait soit être une machine virtuelle Linux soit un conteneur d'applications. Le format aaS est à

privilégier pour des raisons de performances et de disponibilité mais nous verrons que des soucis de budget nous ont conduit à opter pour la première solution.

Nous avons fait le tour de ce à quoi devra ressembler le serveur, abordons maintenant la question du client.

1.3.3 Partie Android

La seconde partie de ce projet concerne le client qui sera une application Android mais qui pourrait tout aussi bien être déclinée pour iOS et Windows Universel. Cette application sera donc une interface entre l'utilisateur et le service et devra lui permettre de profiter des fonctionnalités minimales attendues par rapport à la carte du maraudeur de Harry Potter.

Les fonctionnalités majeures seront donc :

- l'inscription ;
- la connexion-déconnexion ;
- l'envoi de sa position gps au service web ;
- la réception des positions gps d'autres utilisateurs connectés ;
- la visualisation en temps réel sur une carte des positions.

Ces fonctionnalités forment le noyau dur indispensable au niveau minimal de qualité de notre application. Des améliorations sont envisageables pour augmenter le niveau de service, ainsi l'application pourra évoluer et proposer :

- une carte en version 3D ;
- une carte en version réalité virtuelle ;
- le partage de position ;
- l'ajout d'informations sur la carte (lieu / point de rendez-vous).

La mise à jour des informations sur le client sera initiée par l'application qui effectuera une requête sur le serveur. Le serveur renverra les positions des utilisateurs connectés qui permet sa mise à jour chez les utilisateurs.

Le choix d'une application Android se justifie par plusieurs facteurs. Tout d'abord il existe une communauté très active et de nombreux documents autour de l'Android. Ceci est dû en grande partie que le marché est majoritairement composé de terminaux Android, il y a donc plus de références et de documentation. Enfin le langage Android se basant sur du Java il nous est plus accessible de partir sur cette plateforme plutôt que sur du iOS ou même du Windows Universel, de plus nous avons quelques notions en Android.

Les versions du framework Android qui seront supportées seront les version 11 et ultérieures pour fonctionner sur un maximum de terminaux. Toutefois la version de compilation sera la 25 afin de pouvoir profiter des dernières mises à jour du framework. Ceci est rendu possible grâce aux ressources de post-compatibilité Android (appcompat) qui apportent aux terminaux plus anciens certaines des améliorations non-existantes dans leur version.

Nous aurions pu faire le choix de développer dans une technologie cross-platform se basant sur des technologies web comme Ionic mais nos besoins en accès matériel de part le gps et OpenGL nous semblait être en désaccord avec un tel choix.

Nous allons maintenant aborder un aspect important dans le développement de ce projet : son intégration continue.

1.3.4 Intégration continue

Un des objectifs de ce projet est de mettre en place une intégration continue et un déploiement automatique.

Dans les milieux industriels, le développement de solutions comporte souvent 3 phases :

- L'analyse ;
- Le développement ;
- Les tests et l'intégration.

Cette dernière étape est l'une des plus importantes, puisque c'est au cours de cette phase que la solution va être validée à l'aide de tests de non-régrégation, puis mise en production afin d'être livrée à ses clients. C'est ce que l'on appelle la phase d'intégration continue, qui permet d'assurer la qualité du code tout au long de l'évolution de la solution.

Cette phase est généralement automatisée afin de libérer du temps aux développeurs, éviter des tâches répétitives, mais surtout permettre l'élimination d'erreurs humaines et assurer la reproductibilité des builds et des tests. Ces derniers critères sont assurés par la présence de fichiers de configuration de l'intégration continue qui sont versionnés au même titre que le code de la solution. Ainsi, ces fichiers évoluent en même temps que la solution elle-même. Des outils sont donc disponibles sur le marché pour effectuer ces tests de non-régrégation et effectuer l'opération menant à l'établissement d'une version de production pour la solution visée.

Nous avons donc vu à quoi notre solution vise à ressembler, nous allons maintenant développer notre stratégie de planification du travail.

1.4 Organisation du travail

L'organisation temporelle théorique du travail est décrite dans le diagramme de Gantt de la figure 1.7. Elle se veut assez simple et découpée en quatre phases :

- la phase de lancement qui s'étend jusqu'à la rentrée des vacances de la Toussaint et comporte toutes les travaux d'analyse et de mise en place du projet ;
- la phase de développement basique qui s'étend jusqu'à début 2017 et concerne le développement des applications minimales du projet.
- la phase d'amélioration durant laquelle l'ajout d'améliorations supplémentaires sera fait à l'application de base ;
- la phase de finalisation comportant la préparation de la documentation autour du projet.

La première phase est celle qui se prête le plus à un travail commun, puis les phases suivantes ont été pensées de façon modulaire afin de pouvoir travailler indépendamment de l'autre sur chaque tâche. Nous verrons plus tard comment le déroulement du projet a eu lieu dans les faits et quelles modifications de planning ont été apportées.

Nous avons discuté des différents points caractérisant notre projet et les objectifs que nous souhaitions atteindre, nous allons maintenant expliquer comment nous nous y sommes pris pour la conception dans la section suivante.

Diagramme de Gantt initial du projet WatchDogZZ

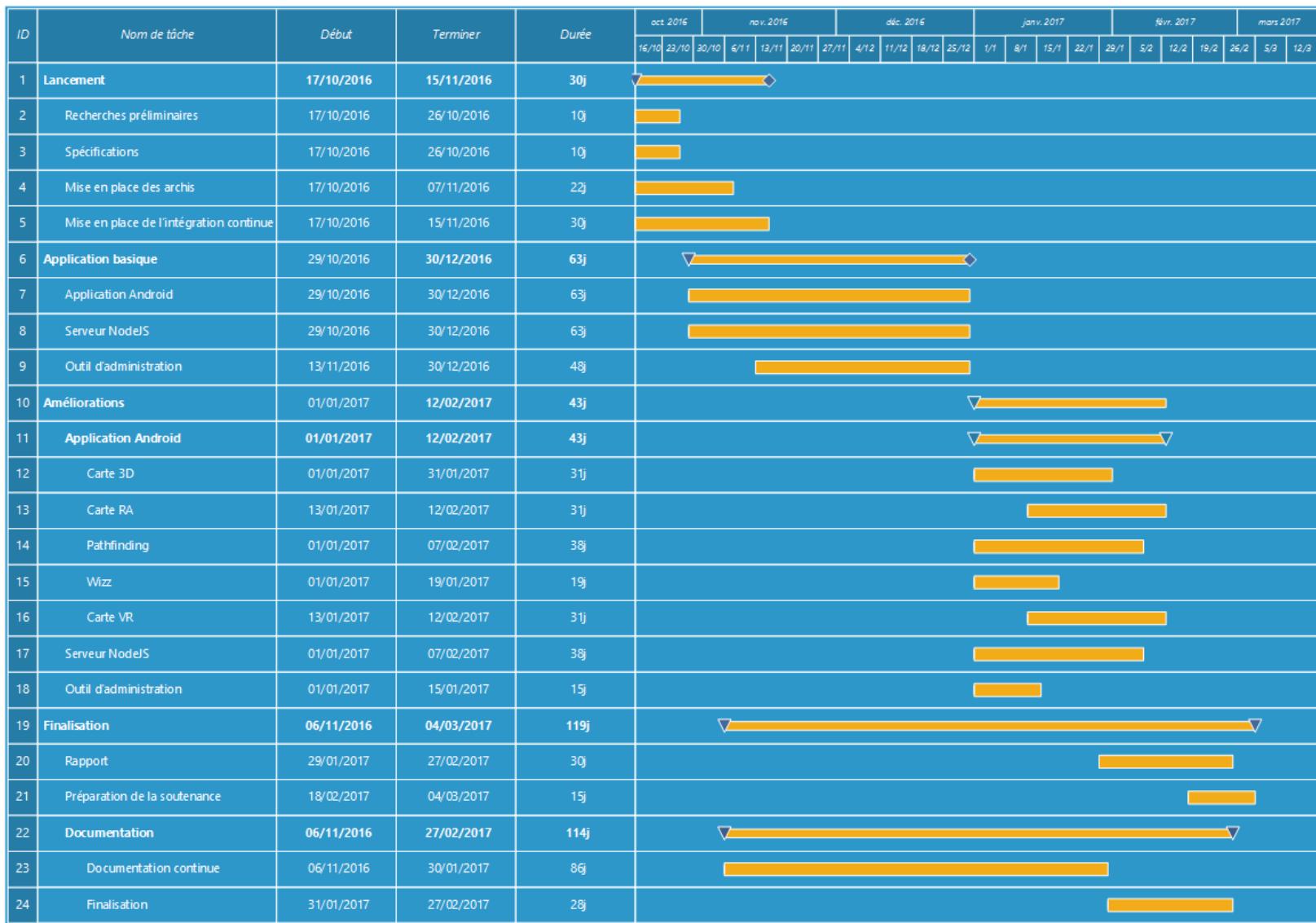


Figure 1.7 – Diagramme de Gantt théorique

2 Conception de la solution

Dans cette section nous allons aborder plus précisément le déroulement de la conception de notre solution en détaillant nos travaux outils et méthodes.

2.1 Conception du couple client/serveur

Notre solution se décompose en effet en deux parties majeures et quasiment indépendante si l'on ne prend pas en compte l'interfaçage entre les deux. Nous allons donc voir en quoi celles-ci consistent exactement.

2.1.1 Web Service

Du coté serveur, il faudra gérer plusieurs parties. D'un coté, il faut s'occuper des requêtes utilisateur, et de l'autre, il faut stocker certaines données que l'on renverra aux utilisateurs.

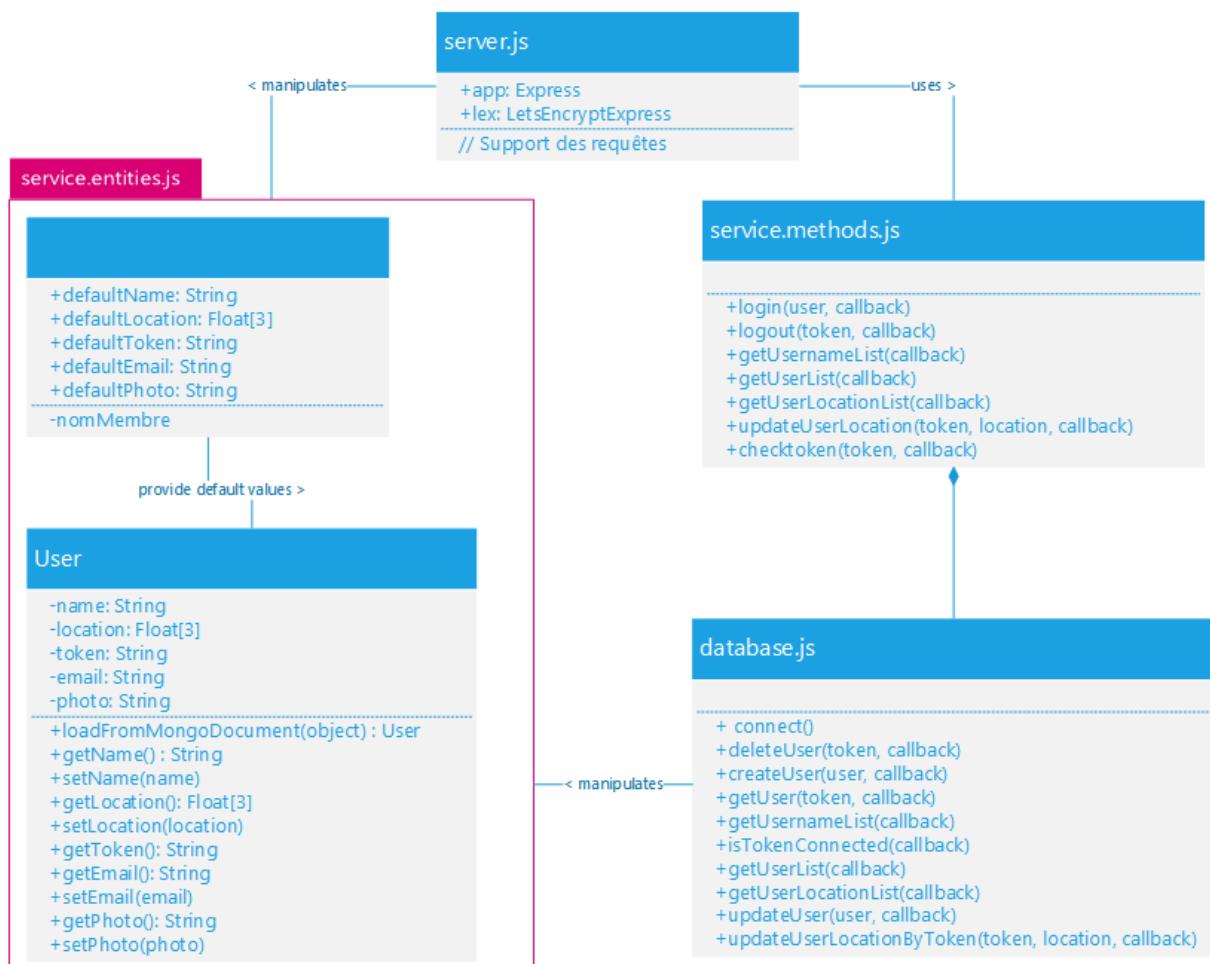


Figure 2.1 – Architecture du Service Web

Comme nous pouvons l'observer dans la figure 2.1, nous distinguons plusieurs parties.

La première partie concerne les entités que l'on va manipuler. Ces entités se trouvent dans un fichier **service.entities.js**. Nous allons retrouver une classe **User**, permettant de stocker et manipuler les utilisateurs connectés sur le service. Dans ce fichier se trouvent également des données par défaut à utiliser pour les utilisateurs qui ne renseigneraient pas leurs informations. Ceci permettra par la suite de disposer par exemple d'une photo à afficher par défaut pour un utilisateur qui ne la partage pas ou n'en a pas.

Une autre partie importante va concerner le stockage des données. Ceci s'effectue par l'intermédiaire de la base de données. Avant de choisir une technologie, nous souhaitions qu'il soit possible de facilement modifier le gestionnaire de base de données (Oracle, MySQL, MongoDB, Microsoft Access).

Pour que ceci se fasse plus facilement, il a été décidé de mettre en place un patron de conception "bridge" (à quelques exceptions, puisque Javascript ne dispose pas du concept d'interfaces), que nous détaillerons plus tard.

Nous remarquons donc dans la figure 2.1 que notre fichier **database.js** dispose de méthodes manipulant les entités du service pour un certain type de base de données. Ce fichier et ensuite inclus dans le fichier **service.methods.js**, ce qui va permettre de les utiliser. Dans l'application, ce seront finalement les méthodes du fichier **service.methods.js** qui seront utilisées, permettant de masquer quel gestionnaire de base de données nous utilisons.

Pour le gestionnaire de données, le patron de conception "bridge" est utilisé. Cela signifie que nous allons spécifier des méthodes que nous souhaitons utiliser dans un fichier (généralement en utilisant un interface, mais ce n'est pas possible avec Javascript). Ici, ces méthodes seront écrites dans le fichier **service.methods.js**. Lorsque l'on souhaite ajouter un gestionnaire de base de données, il faut lui créer un autre fichier associé, et re-créer dans celui-ci les méthodes définies plus tôt. Ensuite, pour utiliser ce gestionnaire, il faut le référencer dans le fichier **service.methods.js**, qui lui-même se chargera de "wrapper" (masquer) les appels des méthodes du gestionnaire. Ainsi, dans l'application, nous utiliserons nos méthodes définies dans le fichier **service.methods.js**, ce qui masque le gestionnaire de base de données utilisé.

Pour changer de gestionnaire de base de données, il suffit de créer un autre fichier, par exemple **database-sql.js**, d'y recréer les mêmes fonctions que dans le fichier **database.js** en modifiant la logique de celles-ci. Ensuite, il suffira simplement de modifier le fichier inclus (**database.js**) dans **service.methods.js** par **database-sql.js**.

2.1.2 Application Android

Le développement du client mobile a suivi le schéma classique de conception d'une application Android. Pour rappel les objectifs initiaux majeurs de ce client étaient de pouvoir récupérer des données sur un web service, les exploiter, les afficher à l'utilisateur et enfin retourner des données au service en question. De façon général la solution Android s'organise en deux projets directeurs : les tests et l'implémentation de l'application.

Il n'a pas été choisi ici de faire du développement dirigé par les tests car la technologie Android était dans le cadre de ce projet une découverte et il aurait été hasardeux de définir des tests Java sur les concepts Android. Les tests ont donc ici vocation à valider les mécaniques métiers a posteriori ainsi que le bon fonctionnement et l'intégrité de l'application au fur et à mesure de l'ajout de fonctionnalités. La partie relative aux tests se subdivise en deux autres : les tests unitaires et les tests instrumentés. Les premiers sont plutôt classiques et permettent de tester les mécaniques métiers et de vérifier tout ce qui

est mockable, autrement dit simulable. Toutefois, il y a certains aspects dans un programme Android qu'il n'est pas possible de mocker sans enlever l'intérêt du test. Nous parlons ici de fonctionnalités s'appuyant intrinsèquement sur le système Android comme les appels réseaux, le GPS, l'écran, etc. Il est nécessaire dès lors que l'on veut simuler une fonctionnalité Android même basique, de simuler tout un système Android. D'où l'intérêt de la deuxième catégorie de tests : les tests instrumentés. Ceux-ci vont être exécutés sur un émulateur Android directement afin de pouvoir tester dans notre cas les appels réseaux et l'utilisation du GPS.

Le projet de tests est donc un projet annexe venant en soutien au projet principal : celui du développement de l'application Android. L'ensemble doit gérer des données et les afficher, c'est pourquoi un modèle MVC semblait adapté. Le modèle MVC se compose de trois parties en interaction comme c'est visible figure 2.2.

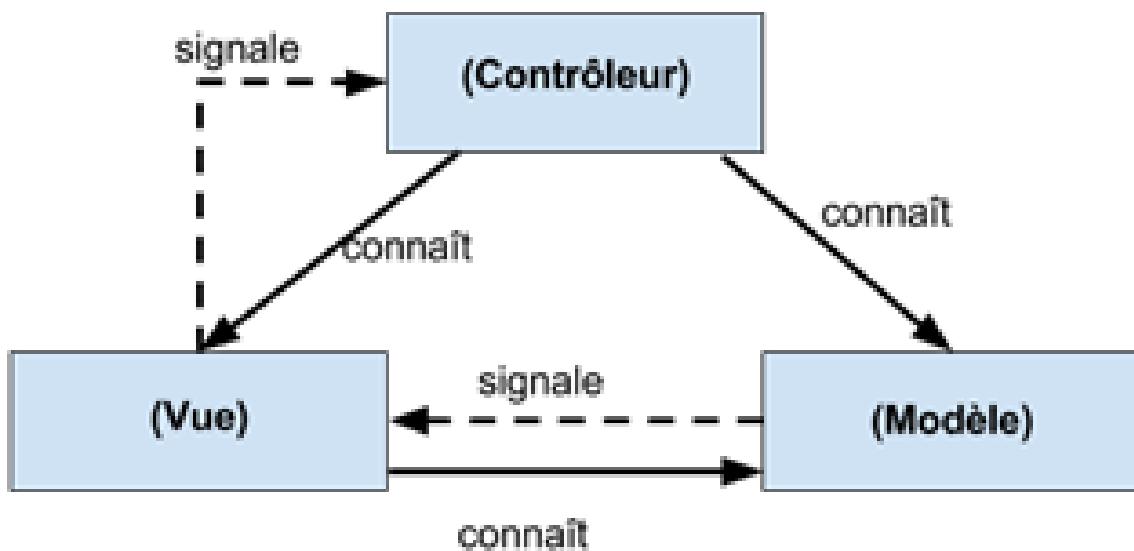


Figure 2.2 – Patron de conception MVC

Le modèle (M) représente les données traitées, dans le cas de l'application ce sont principalement les utilisateurs et leur position GPS. Les deux autres composants n'interagissent pas directement avec les données brutes mais ont accès à l'API du modèle symbolisée par le gestionnaire d'utilisateur (User-Manager) comme sur la figure 2.3. Le modèle gère donc tous les petits traitements bas niveau sur les données et les sert au reste de l'architecture selon les besoins.

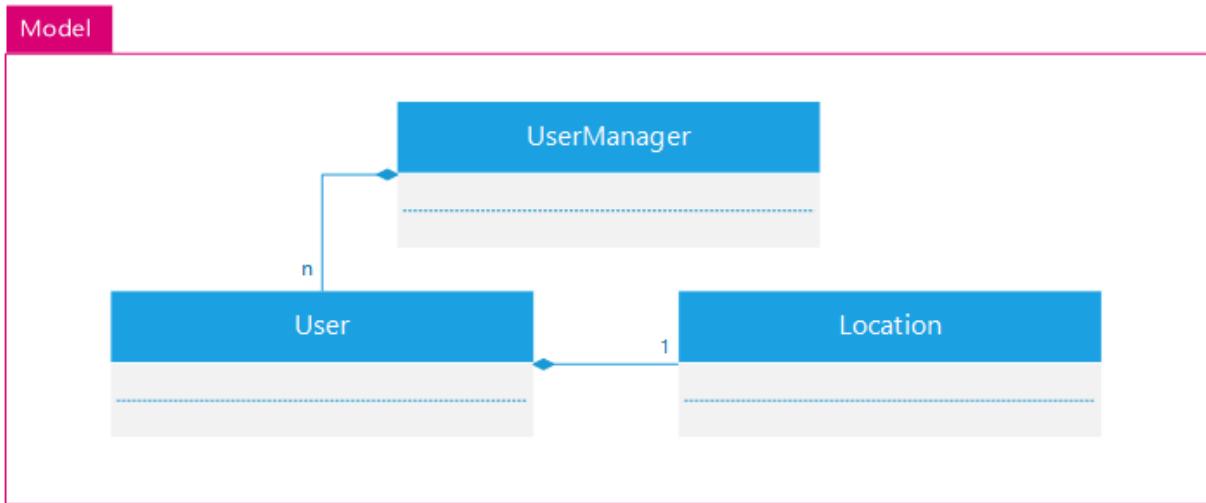


Figure 2.3 – Diagramme du modèle

Le contrôleur (C) s'occupe des interactions avec l'utilisateur, il doit pouvoir transmettre les commandes émanant de l'utilisateur aux autres composants. Dans le cadre de l'application Android, le contrôleur est symbolisé par les activités : ce sont les activités Android qui proposent les interfaces de contrôle utilisateur. Elles permettent de recevoir les évènements utilisateur comme un clic ou encore les messages du système comme l'arrivée d'un SMS ou encore l'actualisation de la position GPS. Les informations reçues peuvent ensuite être utilisées pour effectuer une mise à jour du modèle ou un rafraîchissement de l'affichage. Les différents types d'activités sont visible sur la figure 2.4.

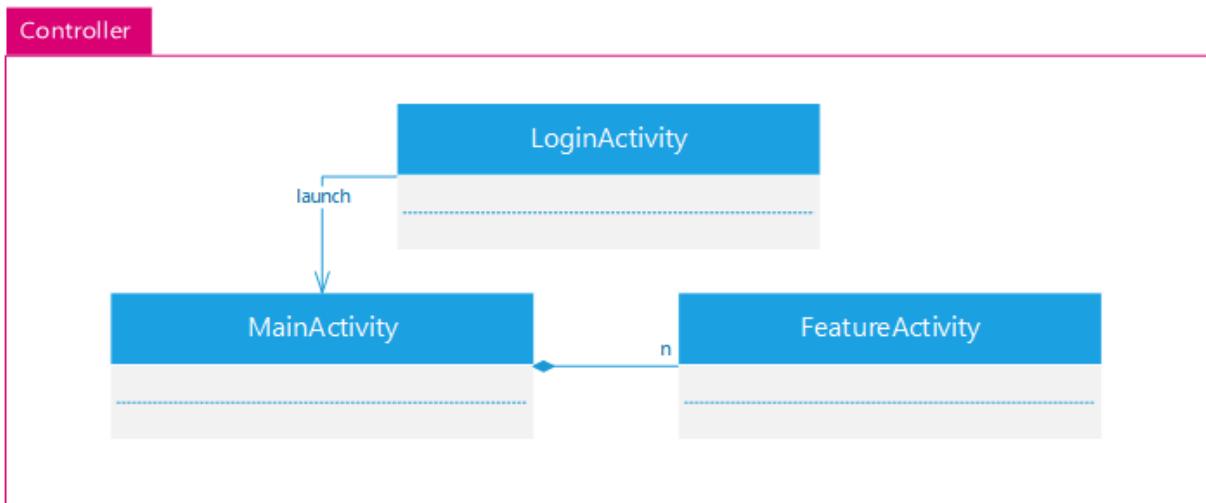


Figure 2.4 – Diagramme du contrôleur

La vue (V) est l'ensemble des éléments constituant la façon dont vont être présentées les données. La réalisation concrète dans l'application Android de la vue est faite par les fichiers XML de layout et autres ressources. Dans la vue entre aussi un pan important du concept de l'application : la gestion de l'affichage 3D. En effet, pour la plupart des éléments d'interface, tout peut être géré par des fichiers de métadonnées mais pour générer la carte en trois dimensions d'un lieu il est nécessaire de décrire dans le code la manière d'utiliser les données pour les afficher avec de véritables classes Java. La vue est donc un ensemble complexe constitué de diverses ressources comme on peut le voir sur la figure 2.5.

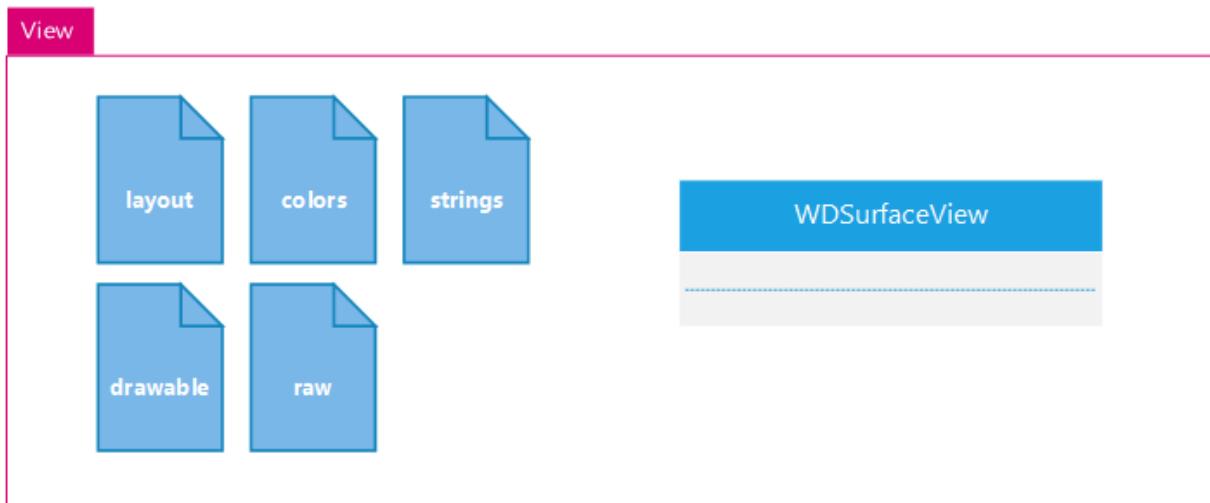


Figure 2.5 – Diagramme de la vue

Le tout s'interconnecte et interagit donc pour faire fonctionner l'ensemble. Mais pour alimenter l'application en données exploitables, des services ont dû être mis en place. Il en existe deux qui servent de sources de données au programme. Le premier est le service GPS, il permet à l'application de récupérer la position du dispositif Android. Dans un premier temps, ce service était basé uniquement sur les données matérielles de l'appareil mais afin d'améliorer la précision, l'utilisation des Google Services a été choisie. Les Google Services utilisent à la fois les données GPS pures mais aussi leur historique ainsi que les données du gyroscope et de l'accéléromètre pour déterminer la position avec une plus grande précision. Ceci s'est avéré indispensable pour une localisation correcte en intérieur.

Le deuxième service est celui responsable de la communication avec le serveur et qui permet de le requêter. Ainsi ce service permet à la fois d'envoyer sa propre position au serveur mais aussi de récupérer la liste des utilisateurs connectés et leurs informations. Ce service exécute en parallèle du thread principal des requêtes http sur le web service WatchDogZZ. Pour ce faire, il utilise la bibliothèque Volley qui permet d'effectuer simplement des communications sur le réseau.

Le patron de conception majeur dans ce système applicatif est celui observer-observable : il permet à une des entités d'être prévenue par une autre après inscription qu'un traitement a été effectué. Ceci permet le parallélisme des services et de ne pas bloquer l'interface graphique lors d'un traitement long (communication réseau).

L'ensemble est sécurisé par le système d'authentification Google. Ce choix a été motivé par la simplicité puisque le client étant sur un système Android, il possède forcément un compte Google associé. Il suffit alors d'effectuer une requête sur les serveurs de Google avec les informations d'authentification de l'appareil pour récupérer un token validant l'identité de l'utilisateur. Ce token est ensuite transmis lors de chaque requête au serveur WatchDogZZ en vue de vérifier l'identité du demandeur (voir la figure 2.6).

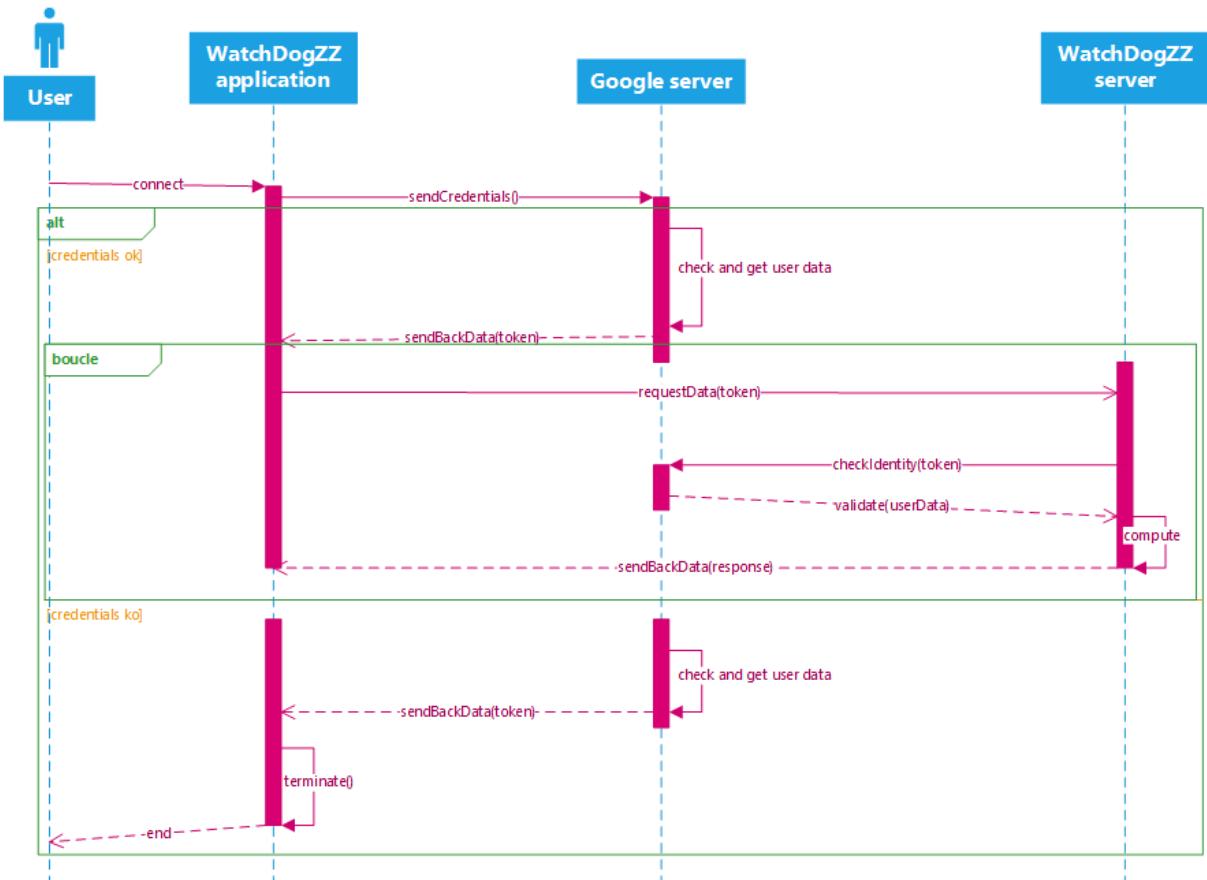


Figure 2.6 – Utilisation du token Google

La carte du maraudeur est une vue qui a été créée spécialement pour l'application. Elle se base sur une SurfaceView reposant sur de l'OpenGL ES 2. L'intérêt était à la fois de pouvoir dessiner en deux mais aussi trois dimensions. Un Renderer spécial a été implémenté ainsi que des managers de ressources 3D. Il est ainsi possible de gérer cette vue comme un observer du UserManager. La vue pourra par la suite afficher les scènes 3D avec les différents éléments à chaque notification. Les différentes classes entrant en jeu sont visibles sur la figure 2.7 ainsi que leur rôle dans le MVC.

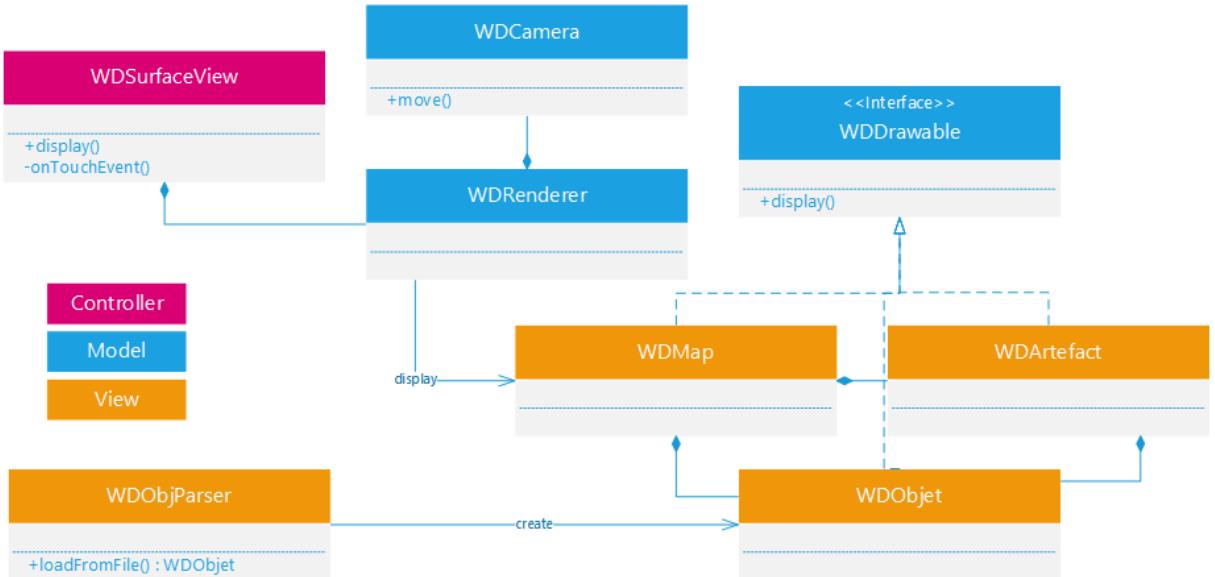


Figure 2.7 – Fonctionnement de la 3D Android

La conception de l'application Android est très simple mais fait intervenir de nombreux éléments et services qui touchent énormément d'aspects de la programmation Android. Il existe sur les versions les plus récentes du Framework des fonctionnalités plus intéressantes et performantes toutefois le choix a été fait d'essayer de faire l'application la plus diffusable possible et donc de supporter un maximum d'appareil. Au début du développement, la version choisie était la 9 mais suite à des contraintes inévitables de développement nous avons dû monter à la version 12. Ceci reste correct d'autant plus que cela représente toujours plus de 99% du marché Android.

2.2 Technologies utilisées

Cette partie détaille les technologies ainsi que les outils de génie logiciel utilisés dans le développement de l'application.

2.2.1 Service Web

Pour la conception du Service Web, nous avions besoin d'une technologie disposant des caractéristiques suivantes :

- Facile à utiliser ;
- Disposant de nombreuses fonctionnalités ;
- Rapide à l'exécution ;
- Exécution légère sur serveur ;
- Configuration rapide.

Toutes ces caractéristiques se retrouvent avec le framework NodeJS [9]. C'est un framework Javascript qui dispose de nombreuses librairies installables à l'aide du gestionnaire de modules NPM [10]. De plus, étant donné que l'un d'entre-nous avait déjà utilisé une telle technologie, cela nous permettait de démarrer plus rapidement.

Le gestionnaire de modules NPM permet d'effectuer plusieurs choses. Premièrement c'est lui qui va permettre l'installation des modules nécessaires au bon fonctionnement du Service. Ensuite, il va se

charger de résoudre les dépendances entre modules. C'est à dire que si un module à besoin d'un autre module pour fonctionner, alors celui-ci sera installé automatiquement. Enfin, il est possible de disposer de plusieurs listes de modules à installer :

- **Production** : comporte les modules nécessaires au lancement du Service en mode production, donc sans les outils de debug ;
- **Dev** : comporte les modules installés en production ainsi que des modules complémentaires utilisés lors de la conception du Service ou à des fins de debuggage.

Afin de mettre en place notre Service Web, nous avons utilisés plusieurs modules :

- **Body-parser** : parser le contenu JSON des requêtes ;
- **Express** : créer un serveur Http ouHttps ;
- **Jasmine** : effectuer des tests de spécifications ;
- **Letsencrypt-express** : gérer les certificats Https du serveur ;
- **Mongodb** : système de gestion de base de données ;
- **Request** : effectuer des requêtes http ou https ;
- **Winston** : faire des logs sur plusieurs niveaux (info, error, warning, debug).

Comme décrit dans la partie d'étude de ce projet, nous avons également besoin d'une base de données pour stocker certaines informations utilisateur. Pour ce faire, nous avons décidé d'utiliser une base **MongoDB**. Ce type de base de données est très simple à mettre en place (et l'utilisation avec NodeJS se fait à l'aide d'un paquet) et se base sur un format de stockage texte BSON (JSON binaire). De ce fait, les opérations de stockage et écriture sont rapides et peut gourmandes en espace. Ce stockage JSON permet également de manipuler facilement les données et résultats de requêtes en Javascript.

MongoDB permet de stocker des informations dans des **collections**. Ces collections sont en quelque sorte l'équivalent des tables **SQL**. La différence ici est que les données que nous allons stocker ne nécessitent pas de suivre un format défini (avec des champs spécifiques). Cela signifie qu'il est possible d'ajouter à la volée (sans devoir reconfigurer la base) certains champs dans nos objets stockés, ce qui constitue un gain de temps en terme de développement. Pour maîtriser plus facilement cela, il est possible de créer des classes d'Objets Javascript, permettant de maîtriser les champs à stocker et surtout d'être conscient du type de chaque champ (chaîne de caractère, entier, flottant).

2.2.2 Android

Le développement d'une application Android s'effectue généralement en utilisant le Java ainsi qu'Android Studio [11], permettant d'inclure les bibliothèques Android. Ici, nous avons donc utilisé ces technologies pour permettre une intégration parfaite sur les systèmes Android.

Android est un système d'exploitation à destination des plateformes mobiles supporté par Google et basé sur un noyau Linux. Il équipe aujourd'hui plus de 80% des terminaux mobiles mais s'adapte aussi sur une large gamme de produits : smartphones, tablettes, montres, téléviseurs, etc. Les systèmes Android sont capables d'exécuter des programmes dérivés du Java grâce à leur environnement d'exécution : une machine virtuelle Dalvik ou l'Android Runtime selon la version. Il est donc possible de produire des exécutables de type apk à base de Java en s'appuyant sur le SDK Android, à la rédaction de ce rapport l'API la plus récente est la 25.

La programmation Android est donc très proche du développement d'applications Java modernes mais comporte tout de même des concepts spécifiques apportées par le SDK. On peut citer par exemple le concept d'activité qui représente une vue dans une application ou encore un intent qui est une demande

de service qu'une application peut envoyer au système. L'API propose aussi une pléthore de fonctionnalités utilisant les composants du terminal mobile : gps, accéléromètre, appels réseaux, contacts, sms, etc.

En plus du SDK Android, notre projet nécessite l'utilisation de trois autres bibliothèques : les Google Services, Volley et OpenGL ES.

Le premier, les Google Services, est un ensemble de fonctionnalités avancées fournies par Google. Les Google Services permettent à tous ceux utilisant un terminal Android d'accéder aux dernières fonctionnalités sans pour autant avoir la dernière version de l'OS. La subtilité derrière ces services est qu'officiellement, Android est censé être un système open source, toutefois pour garder la maîtrise du produit Google a décidé de freiner sa participation au cœur d'Android et propose ses meilleurs services et évolutions au travers des Google Services ce qui leur permet de rendre Android presque totalement dépendant de Google. Le géant américain participe donc de façon fermée à un projet open source en empêchant toute concurrence. Dans le cadre de ce projet c'est le système de géolocalisation amélioré qui nous intéresse dans ces services. En effet, la version Android fournit juste la position donnée par le capteur alors que les Google Services prennent en compte plusieurs paramètres comme l'historique de position, les mouvements, les déplacements, etc.

La bibliothèque Volley permet d'effectuer des requêtes réseaux avec une plus grande facilité. Comme on peut le voir sur la figure 2.8, Volley est capable d'exécuter des appels asynchrones et de cacher les réponses. Les gains en performances et en expérience utilisateur sont donc tout à fait de la partie.

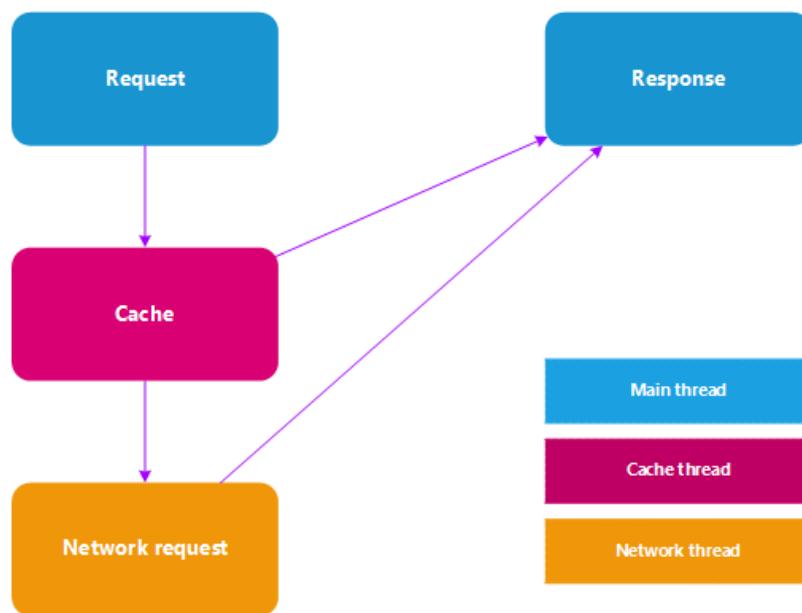


Figure 2.8 – Fonctionnement d'un requête Volley

Cette bibliothèque apporte donc des fonctionnalités d'un peu plus haut niveau que le basic HttpURLConnection de java.net si l'on considère les appels implicites à d'autres threads et des performances améliorées grâce au caching (pour le chargement d'images par exemple).

Enfin le dernier framework utilisé est OpenGL ES ou Embedded System. Ce n'est autre que la version portable du célèbre framework graphique qui a été conçu spécialement pour la programmation embarquée. Comme son ainée, cette version dispose d'une API de bas niveau mais permet au moins de réaliser énormément de choses. En effet il aurait été possible d'utiliser le framework Unity (haut niveau)

comme beaucoup d'applications 3D le font, mais cela implique d'utiliser des ressources propriétaires payantes alors que les vues 3D à développer dans notre cas ne sont pas très évoluées.

La dernière technologie à présenter en lien direct avec Android est bien sûr l'IDE utilisé. Nous avons opté pour Android Studio, visible sur la figure 2.9, qui est l'environnement de développement officiel d'Android. Android Studio s'inspire grandement des autres outils de la société JetBrains en intégrant beaucoup d'utilitaires spécifiques au développement Android. Il existe d'autres solutions comme des versions ou même des plugins Eclipse mais depuis l'arrivée en 2014 d'Android Studio, il n'est vraiment pas recommandé d'utiliser encore Eclipse. Un autre, Processing permet d'effectuer du développement Android mais on est quand même loin de l'outil de JetBrains au niveau des fonctionnalités.

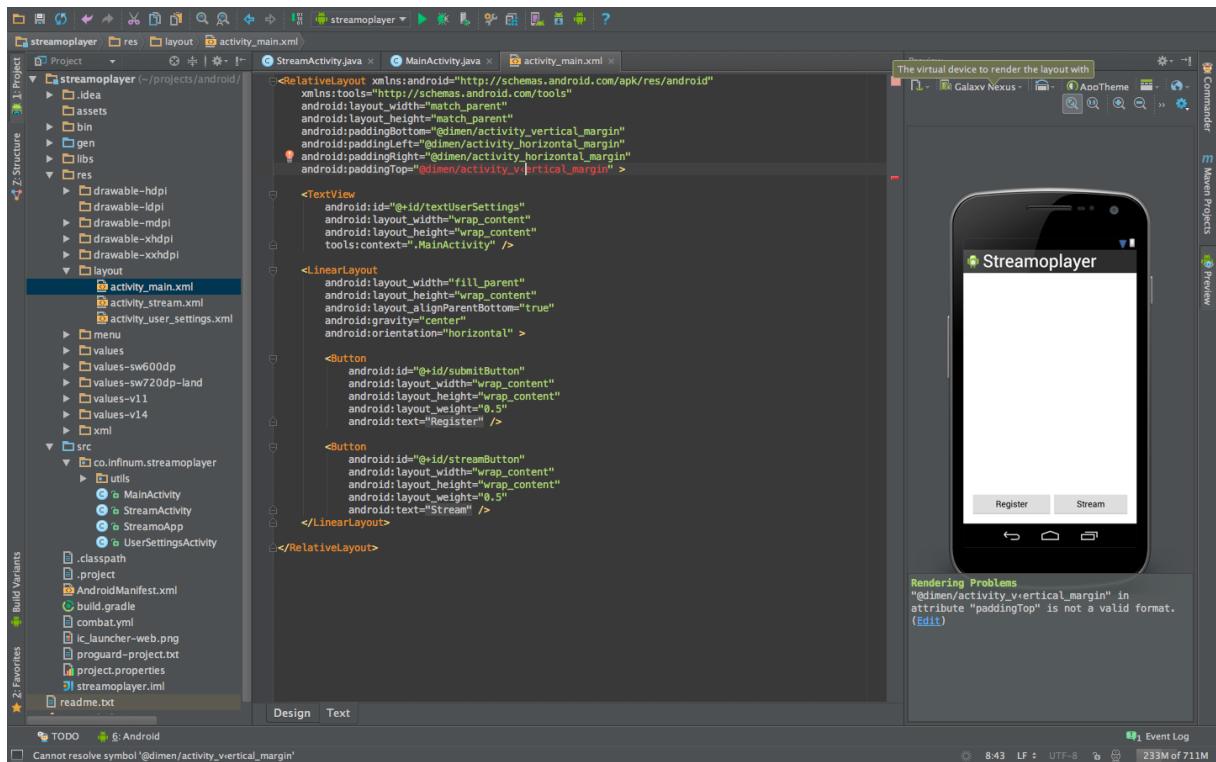


Figure 2.9 – Environnement de développement intégré officiel d'Android

Nous n'allons pas passer en revue ici toutes les fonctionnalités d'Android Studio qui comporte bien sûr toutes les fonctionnalités que l'on peut attendre d'un environnement professionnel mais il faut remarquer qu'il dispose d'outils remarquables.

Le premier est son gestionnaire de SDK : celui-ci permet de maintenir à jour et de manipuler toutes les versions de SDK disponibles. C'est plutôt utile pour un développement où chaque client est différent même si cela reste de l'Android.

Le second est la mise à disposition d'émulateurs Android. Il permet de simuler des terminaux Android de toutes sortes sans besoin de disposer du matériel en réalité. De plus le moniteur intégré permet de suivre en détail l'activité du simili-système ce qui est un plus lors de l'analyse d'exécution d'une application.

Enfin, Android Studio permet de développer les interfaces graphiques soit en mode texte soit en mode WYSIWYG ce qui peut avoir l'avantage d'avoir une prévisualisation d'une vue sans avoir à exécuter toute l'application.

Nous allons maintenant aborder les technologies tierces utilisées au cours de ce projet.

2.2.3 Intégration continue

Lors de l'ajout de la phase d'intégration continue pour notre solution, nous avions plusieurs choix :

- Travis CI ;
- Circle CI ;
- Amazon AWS CodePipeline ;
- Jenkins.

Les outils Jenkins ou CodePipeline sont des outils largement utilisés dans les industries et permettent une customization très fine des opérations de test et de déploiement. Cependant, ceci nécessite une configuration avancée des outils et parfois un serveur sur lequel placer l'outil (par exemple pour Jenkins). Du fait que nous n'avions pas besoin d'une configuration particulière pour tester et produire notre solution, nous nous sommes orienté vers l'outil **Travis CI**, qui permet de gérer aussi bien **Android** que **NodeJS**.

Cet outil s'utilise très facilement avec la plateforme GitHub [12] que nous avons utilisé pour gérer le code source des différentes parties de la solution. Il faut simplement ajouter le service d'intégration pour chaque dépôt contenant le code à tester et déployer.

La configuration de ce service s'effectue à l'aide d'un fichier **.travis.yml**. Ce fichier est versionné au même titre que le code source de la solution. Ainsi, à chaque évolution de la solution, ce fichier peut également évoluer et permettre les tests et le déploiement du code concerné. Un exemple de fichier de configuration est donné dans le code 2.1.

```
1 language: node_js
2 node_js:
3   - "node"
4
5 before_install: # Install the dev dependencies
6   - sudo apt-get -qq update
7   - sudo apt-get install -y mongodb-org
8
9 install:
10  - npm install -d
11
12 before_script: # Run the server in background
13  - npm start &
14  - ./start_database.sh &
15
16 script:
17  - npm test
18
19 after_script: # Stop the server launched and the database
20  - kill %1
21  - kill %2
```

Code 2.1 – Exemple de fichier de configuration Travis (service web)

Nous pouvons ainsi indiquer quelle version d'un framework utiliser, ajouter des paquets, effectuer des actions supplémentaires (par exemple lancer la base de données dans le cas du service web).

Le déploiement dans le cadre de l'application Android ne se fera pas sur le Google Play mais sur GitHub Releases pour des questions de budget. Un **.apk**, qui est le format des applications sous Android, sera déposé automatiquement par le serveur d'intégration sur **GitHub Releases**, et accessible depuis la page principal du dépôt Android.

En ce qui concerne le déploiement du service web, celui-ci est effectué sur une machine personnelle, exécutant **Ubuntu Server 16.10**. Cette machine dispose également d'une installation de MongoDB. Pour le déploiement (ou mise à jour) du service, un script (voir code 2.2) se charge d'effectuer les opérations suivantes :

1. Arrêt de l'ancienne instance ;
2. Mise à jour du code du service ;
3. Mise à jour des dépendances ;
4. Relance du service.

Les opérations du script sont lancées en ssh à partir d'une machine dont la clé ssh est copiée sur le serveur. Pour plus de sécurité, l'authentification par mot de passe en ssh a été désactivée. Ce mode de mise à jour du service implique une période d'indisponibilité entre les étapes 2 et 3 décrite au-dessus.

```
1 ssh -i ${KEY_FILE} ${USERNAME}@${IP} <<'ENDSSH'
2   cd server
3   npm stop
4   git pull
5   npm install
6   npm start &>/dev/null &
7   exit
8 ENDSSH
```

Code 2.2 – Script de déploiement du service

2.3 Fonctionnalités introduites par la solution

Dans cette partie, nous allons traiter des différentes fonctionnalités présentées par la solution. Nous aborderons tant le point de vue des applications utilisatrices de ce service, que le point de vue de la gestion du service.

2.3.1 Concepts introduits

Avant de présenter les diverses fonctionnalités, certains mécanismes visant à la qualité du service vont être présentés.

Pour commencer, un service de DNS est utilisé. Ce type de service permet d'attribuer une URL textuelle à un adresse IP (l'adresse du serveur web). Ainsi, les utilisateurs retiendront plus facilement cette URL plutôt que l'adresse IP, et cela permet de configurer les applications clientes plus facilement (en renseignant simplement l'URL textuelle).

De plus, il faut noter que dans certains cas, l'adresse IP d'un serveur n'est pas attribuée de manière statique. Pour remédier à cela, nous avons ajouté un service DynDNS. Ce service va mettre en place un mécanisme mettant à jour l'adresse IP à associer à l'URL textuelle définie avec le DNS. De ce fait, il

n'y a plus besoin de s'occuper des changements dynamiques d'IP du serveur web. Grâce à cela, nous pourrons toujours accéder au service à l'aide de l'URL <https://watchdogzz.ddns.net>.

Comme nous pouvons le constater dans l'URL fournie précédemment, nous utilisons le protocole [https](https://). Comme l'indique son dernier 's', celui est sécurisé par une couche de chiffrement [TLS](#). Cette couche de chiffrement va s'occuper de chiffrer les données fournies dans la requête de l'utilisateur, qui ne seront donc pas visibles si une personne parvient à intercepter la requête. Pour introduire ce concept, le paquet **Letsencrypt-express** est utilisé et vient compléter le serveur **Express**. Ce paquet va en fait utiliser le service en ligne Letsencrypt [13], qui va se charger de gérer automatiquement les certificats relatifs à l'authentification du serveur ainsi que l'envoie des informations d'authentification et de chiffrement au client. Ce service permet donc de disposer d'une autorité de certification valide, et donc d'assurer aux utilisateurs que ce service est bien celui qu'ils demandent, et non un service malicieux.

Un dernier mécanisme introduit dans le service est celui du **logger**. Ce mécanisme consiste à enregistrer dans des fichiers (généralement dits "de log") certaines actions qui s'effectuent sur le service. Nous pouvons par exemple décider d'enregistrer des informations de contrôle relative au démarrage du service, des erreurs système, ou encore des informations de [debug](#) lors de l'ajout de nouvelles fonctionnalités.

Ce qui va nous intéresser principalement avec le logger, c'est la possibilité de pouvoir analyser les requêtes renvoyant des erreurs aux utilisateurs. De ce fait, si nous sauvegardons suffisamment d'informations lorsqu'une requête échoue, nous serons capable de déterminer la cause de l'erreur, et ainsi éventuellement corriger un comportement menant à des erreurs involontaires.

Le logger est mis en place à l'aide du module **Winston**, et va permettre de gérer les fichiers de log suivants :

- **server-error.log** : erreurs critiques du serveur, de la base de données erreurs de requêtes ;
- **server-info.log** : informations de contrôle sur le serveur ;
- **server-debug.log** : informations utilisées lors du développement de nouvelles fonctionnalités.

Il faut noter que lorsque l'on souhaite logger une information (normale ou une erreur), il faut indiquer son niveau de严重性. Avec le module ici utilisé, les niveaux sont les suivants (un niveau plus bas implique une importance plus grande) :

1. **Error**
2. **Warn**
3. **Info**
4. **Verbose**
5. **Debug**
6. **Silly**

Les fichiers de log décrits précédemment correspondent chacun à un niveau de严重性. Ainsi, nous aurons dans le fichier **server-error.log** uniquement les erreurs, ce qui permet un traitement facile de celles-ci ; dans le fichier **server-info.log**, nous disposerons d'informations de contrôle et des erreurs ; et enfin dans le fichier **server-debug.log**, nous disposerons des erreurs, informations de contrôle et informations de debug.

2.3.2 Utilisation du service

La principale fonctionnalité du Service Web est de répondre à des requêtes. Des URL sont mises à disposition par le service et permettent d'effectuer certaines tâches. Le passage de paramètres pour

ces URL se fait directement dans le corps de la requête sous forme de JSON. Les réponses du service sont également sous forme d'objet JSON dans le corps de la réponse. Le paquet **Body-parser** utilisé dans le serveur permet d'effectuer le parsing automatique du corps de la requête pour le transformer en JSON, utilisable dans le code.

En cas d'erreurs, les réponses contiennent les champs suivants :

```

1  {
2      'status': 'ok' / 'fail', // L'état de la requête
3      'error': 'description' // Une description de l'erreur s'il y en a une, sinon ce
4          champ est absent
5  }

```

Code 2.3 – Corps général de la réponse serveur

La plupart des requêtes décrites nécessitent que l'utilisateur soit authentifié. Pour ce faire, il doit effectuer une première requête sur l'URL **/login** en utilisant la méthode **POST**. Les paramètres suivants sont attendus par le service :

```

1  {
2      'name': 'username', // Le nom de l'utilisateur à connecter
3      'token': 'azertyuiop12345', // Le token de connexion fourni par Google
4      'location': [1.0, 2.0, 3.0], // La position courante de l'utilisateur
5      'photo': 'http://photo/user1', // L'URL vers la photo de l'utilisateur
6      'email': 'mail@example.com' // L'adresse mail de l'utilisateur
7  }

```

Code 2.4 – Corps de la requête POST /login

Notons que les champs **photo** et **email** sont optionnels. S'ils ne sont pas renseignés lors de la connexion, des données par défaut seront utilisées. Ces données pourraient par exemple venir à manquer si l'utilisateur ne souhaite pas donner son adresse mail, ou si par exemple il ne possède pas de photo (auquel cas une photo par défaut doit effectivement être utilisée par le client).

Le service est capable de retourner la liste des personnes connectées sur le Service en effectuant une requête de type **GET** sur l'URL **/who**. Le résultat est transmis sous la forme suivante :

```

1  {
2      'list': [
3          'userName1',
4          'userName2',
5          'userName3'
6      ]
7  }

```

Code 2.5 – Corps de la réponse GET /who

Si l'on souhaite disposer d'informations supplémentaires en plus du nom des personnes connectées (url de photo, position, adresse mail), il est possible d'effectuer une requête **GET** sur l'URL **/where**.

```

1  {
2      'list': [
3          {
4              'name': 'username1',
5              'location': [1.0, 2.0, 3.0],
6              'photo': 'http://photo/user1',
7              'mail': 'mail@example.com'
8          }
9      ]
10 }

```

```

6     'photo': 'http://photo/user1',
7     'email': 'mail1@example.com'
8   },
9   {
10    'name': 'username2',
11    'location': [4.0, 5.0, 6.0],
12    'photo': 'http://photo/user2',
13    'email': 'mail2@example.com'
14  },
15  {
16    'name': 'username3',
17    'location': [7.0, 8.0, 9.0],
18    'photo': 'http://photo/user2',
19    'email': 'mail3@example.com'
20  }
21 ]
22 }
```

Code 2.6 – Corps de la réponse GET /where

Pour mettre à jour sa position sur le Service, l'utilisateur effectue une requête sur cette même URL **/where**, mais cette fois avec la méthode **POST**, et en spécifiant les champs suivants dans le corps de sa requête :

```

1  {
2   'token': '1A2Z3E4R5T6Y7U8I9OOP',
3   'location': [1.0, 2.0, 3.0]
4 }
```

Code 2.7 – Corps de la requête POST /where

Cette requête peut également permettre de connecter un utilisateur directement si celui-ci ne l'est pas déjà. En effet, puisque son token de connexion est fourni (et peut permettre de l'identifier de manière unique), il est possible de vérifier s'il est connecté (auquel cas, sa position est mise à jour simplement), sinon, nous allons pouvoir le connecter en utilisant les informations qu'in va nous fournir. Il est possible de fournir les mêmes informations que dans le code 2.4. Ainsi l'utilisateur sera créé avec les informations qu'il a soumises et sa position initiale est enregistrée. Il peut ensuite effectuer de simples requêtes POST sur l'URL /where pour mettre à jour sa position.

Ces requêtes, lorsque effectuées correctement sur le service, ne retournent pas d'erreurs. Cependant, il est possible que dans certains cas, elles n'aient pas fonctionnées sur le serveur, pour plusieurs raisons :

- Le service connaît une erreur interne (configuration, crash) ;
- Le service ne parvient pas à communiquer avec la base de données ;
- La requête utilisateur ne fourni pas les informations suffisantes au traitement de sa requête.

Pour indiquer cela à l'utilisateur, le code de retour de la requête est choisi parmi les suivants :

- Pas d'erreur : code 200 ;
- Erreur interne ou paramètres insuffisants pour la requête : erreur 400 ;
- Erreur de communication avec la base de données : erreur 503.

Ces erreurs peuvent par la suite être gérées par le client. Il peut ainsi choisir soit de réitérer sa requête (dans le cas d'une erreur 503 par exemple), soit d'informer l'utilisateur que le service est indisponible ou que sa requête n'est pas valable (cas d'une erreur 400).

Le diagramme suivant (figure 2.10) permet de mieux apprécier l'enchaînement des requêtes ainsi que leur cheminement sur le serveur. Pour des raisons de simplification, nous représenterons l'utilisateur comme une application capable d'utiliser correctement le service ainsi que toutes ses fonctionnalités.

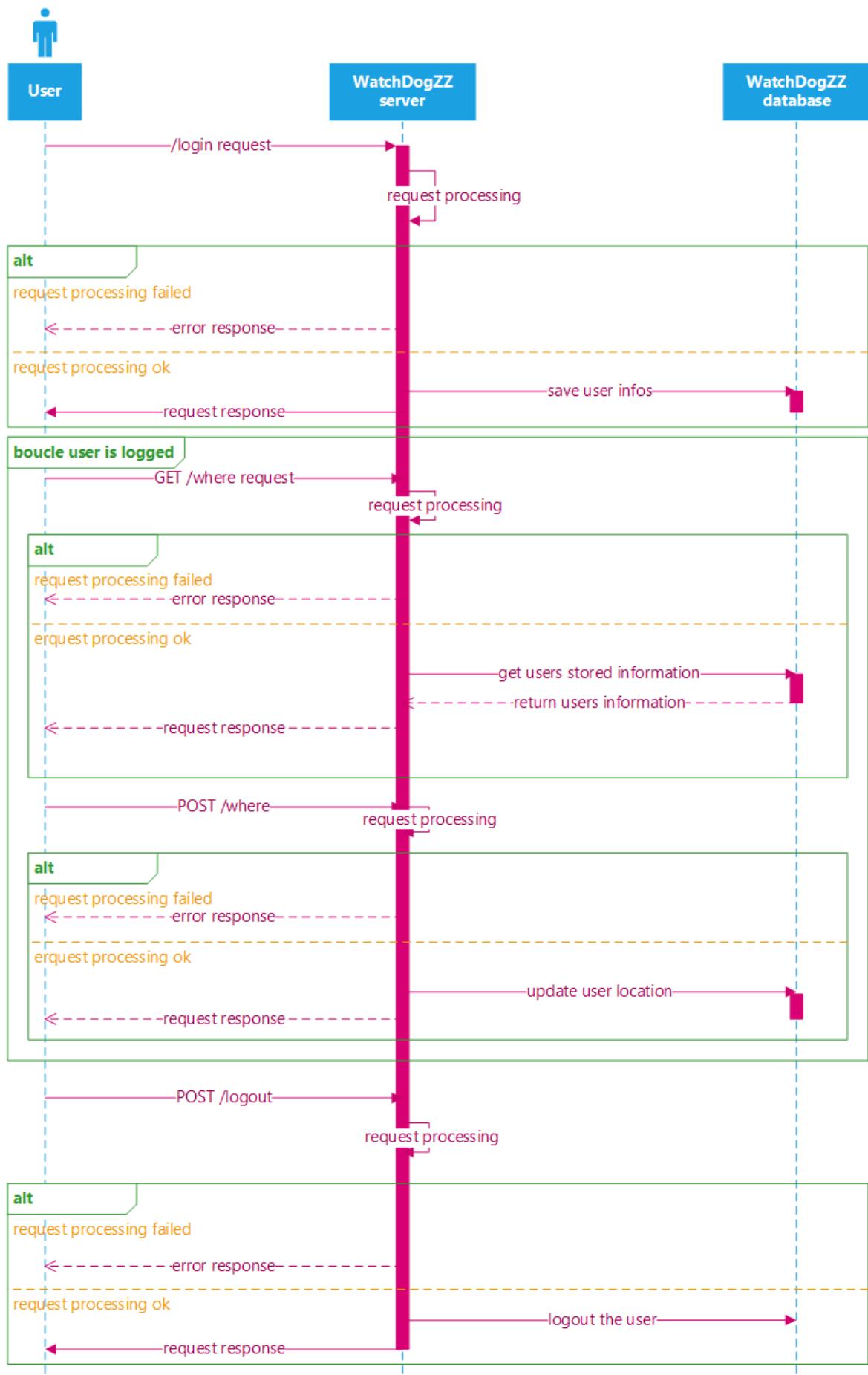


Figure 2.10 – Diagramme de séquence des requêtes sur le service

2.4 Méthodes de développement

La méthode adoptée pour la conception de cette solution devait pouvoir convenir à un projet hétérogène et à une équipe de taille faible, un binôme. De façon générale le projet se décomposait en trois sous-projets indépendants : la partie serveur, la partie client et la partie documentation.

La partie documentation est la seule qui a réellement fait l'objet d'un travail commun avec concertation, échange de points de vue et vérification du travail de l'autre puisqu'elle a consisté en la mise au point des spécifications et en la rédaction de ce rapport. Durant la phase d'analyse et devant l'état des lieux de tout ce qui devait être fait, il a paru équitable et logique de détaché une personne sur le projet Android et une autre sur le projet serveur. Ainsi la répartition des tâches et l'expertise sur les différents projets étaient très contrôlé.

Une fois que les besoins de l'application ont été analysés et que les spécifications ont été posées, nous avons transformé ses documents en un kanban regroupant les user stories principales. Ces user stories forment l'ensemble minimal des tâches à effectuer pour avoir une application fonctionnelle répondant aux demandes fondamentales du cahier des charges. Dès lors que cette sélection a été faite, le développement des projets primitifs du client et du serveur a commencé. Le kanban utilisé est celui proposé par GitHub, toutefois nous avons vite abandonné son utilisation. Des échanges réguliers sur les outils de travail d'équipe de GitHub, que nous détaillons plus loin, ont permis de suivre l'évolution du développement linéaire de chaque projet et de rendre compte du travail effectué. L'objectif de cette période était donc de livrer une version fonctionnelle de l'application pour la mi-janvier.

Au terme de cette première période nous avons pu livrer une application répondant aux critères minimaux et permettant de suivre des personnes dans l'ISIMA. En partant de cette base fonctionnelle nous avons commencé le développement de fonctionnalités plus poussées avec une méthode un peu différente : une méthode plus agile.

Nous avons listé tous les bugs connus, les améliorations et les fonctionnalités que nous souhaitions ajouter. Ensuite nous avons commencé à fonctionner en itérations agiles d'une durée de deux semaines. En début d'itérations nous sélectionnions un ensemble d'items qui étaient des « issues » sur GitHub et nous en faisions une milestone, voir figure 2.11. Nous travaillions ensuite sur ces issues et en fin de cycle nous faisions le point sur l'itération terminée et nous rajoutions des issues en fonction de la situation. Le développement agile a permis de rajouter des fonctionnalités avancées sur deux ou trois itérations.

2 – Conception de la solution

The figure consists of two screenshots of a Jira interface. The top screenshot shows a general view of issues with filters set to 'is:issue is:open'. It displays 6 open issues and 13 closed issues. The bottom screenshot shows the 'Iteration 1' milestone, which has 0 open issues and 10 closed issues. Both screenshots include standard Jira navigation and filtering tools.

Iteration 1

⚠ Past due by 25 days 100% complete

Issue Type	Summary	Labels	Assignee	Comments
bug	Listing des utilisateurs	feature		
bug	Listing	feature		
bug	unit tests	enhancement		
bug	Vue OpenGL : scrolling	bug enhancement		
bug	Mise en pause du thread GPS	bug		
bug	Vue OpenGL : Différenciation des marqueurs	enhancement		

Figure 2.11 – Liste d’issues (en haut) suivie de la milestone de l’itération 1 (en bas)

L’ensemble de ces éléments fait que nous avons pu développer étape par étape une application qui semblait très complexe à mettre en place. Sans cette méthode nous aurions peut-être été perdu devant tout le travail mais dans les faits, malgré quelques difficultés nous avions toujours une version fonctionnelle qui répondait aux critères minimaux du cahier des charges.

Finalement, le planning final ressemble sans entrer dans les détails au planning initial mais il comporte quand même des différences assez remarquables. Tout d’abord, la phase de développement d’une base a été un peu plus longue que prévue. Ensuite la phase d’ajout de fonctionnalités avancées a pris la forme d’un développement agile mais le temps d’installer ce processus nous n’avons eu le temps de faire que deux itérations. L’ensemble est représenté sur la figure 2.12.

Nous avons maintenant explorer tous les aspects de la conception de notre solution et nous allons enfin pouvoir passer à la présentation de nos résultats.

Diagramme de Gantt final du projet WatchDogZZ

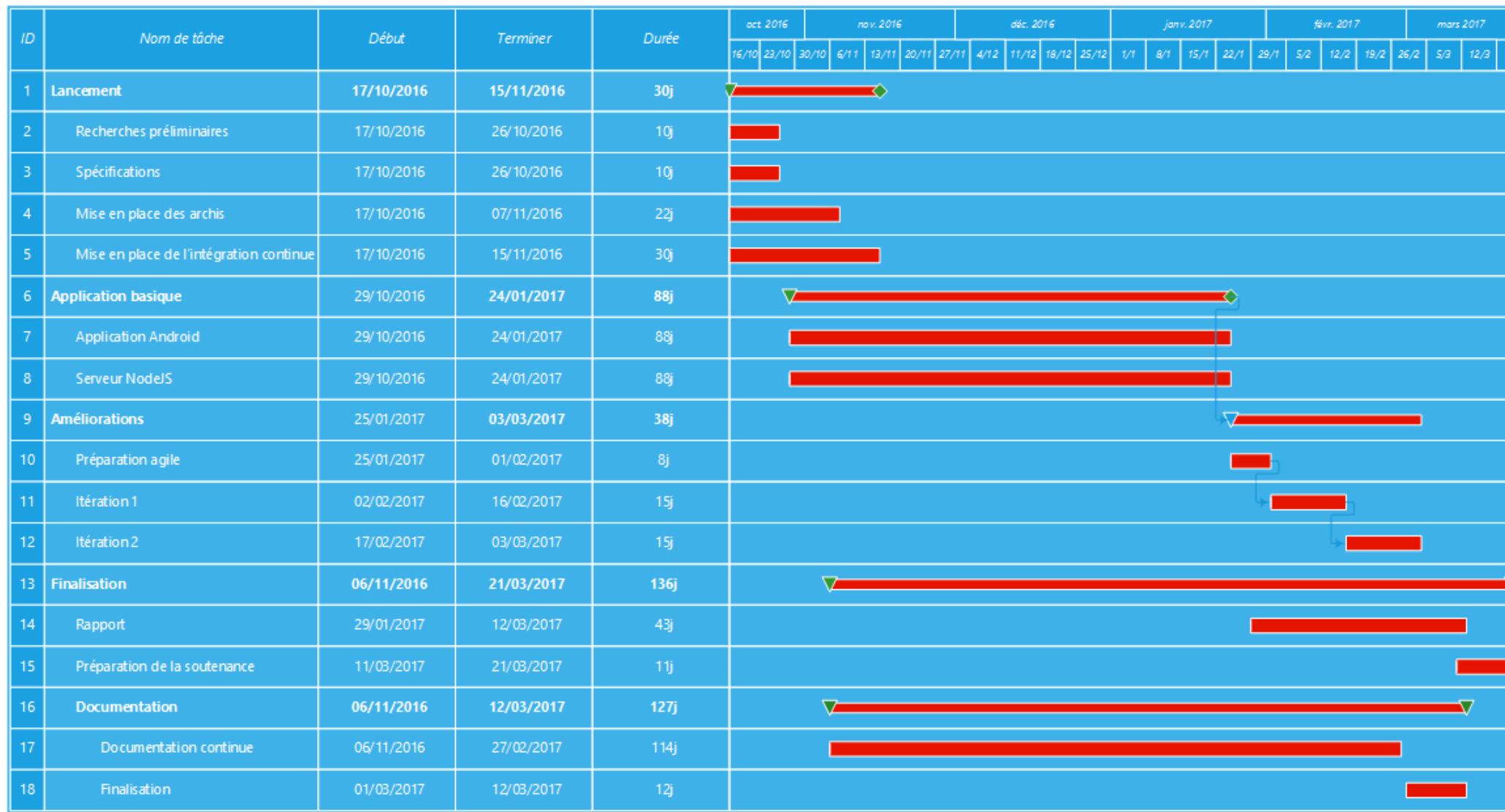


Figure 2.12 – Diagramme de Gantt réel

3 Résultats

Dans cette dernière section nous allons effectivement aborder le bilan de notre projet en présentant le résultat final obtenu et les diverses améliorations qui pourraient encore être faites sur ce sujet.

3.1 La solution apportée

Nous allons donc reprendre tour à tour les différents points présentés dans les parties précédentes et détailler point à point les fonctionnalités réalisées ainsi que la structure finale.

Nous avons donc réussi à atteindre nos objectifs minimaux durant ce projet et même à réaliser certains de nos objectifs supplémentaires. Nous avons réalisé un service web et une application Android qui sont tous les deux versionnés sur GitHub et intégrés en continu grâce à Travis CI qui nous permet de valider l'intégrité de notre code et de déployer de façon automatique nos solutions. Les différentes versions majeures validées par l'intégration continue sont disponibles sur le site GitHub Releases et le service web est déployé sur une machine privée accessible depuis internet. Notre service web étant de type REST il pourrait être utilisé par différents clients en plus de l'application Android, mais nous n'avons développé que celui-ci par manque de temps c'est pourquoi il n'y a pas d'interface web comme on peut le voir à la figure 3.1. Notre service est toutefois pleinement accessible depuis n'importe quel navigateur ou programme pouvant effectuer des requêtes http.

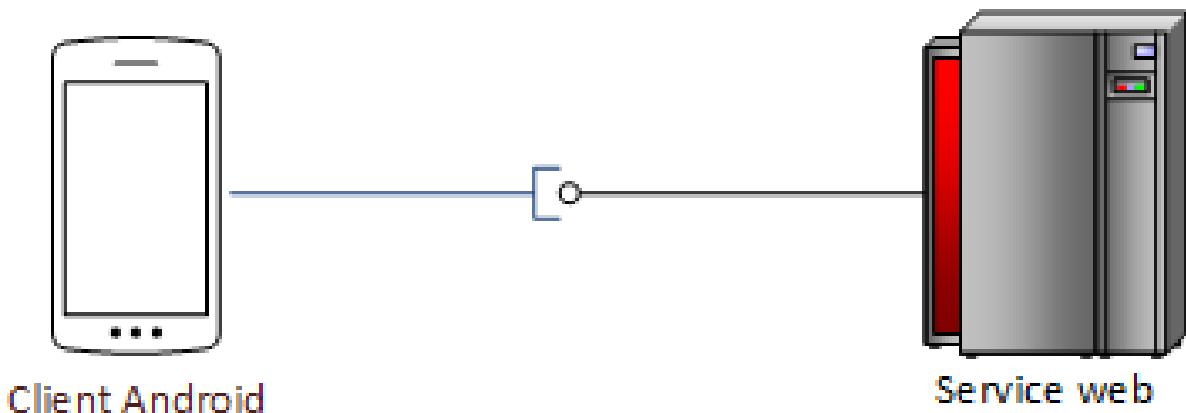


Figure 3.1 – Architecture finale simplifiée

L'architecture est donc assez simple et représente très bien le caractère binaire de notre projet : à savoir une partie serveur universel d'un côté et une partie client lourd de l'autre.

La dernière version de notre solution offre donc les fonctionnalités suivantes. Comme nous pouvons le voir sur la figure 3.2, nous avons mis en place un logo qui sert d'icône et d'image à notre produit, nous avons aussi une première activité ou vue, permettant la connexion des utilisateurs par le biais de leur compte Google. Comme expliqué dans la section précédente cette activité contacte dans un premier temps les services de Google pour authentifier l'utilisateur courant, pour se faire il faut envoyer l'adresse email, le mot de passe et la clé d'application de WatchDogZZ à Google qui va ensuite renvoyer un jeton en cas de succès. Ce jeton sera alors envoyé pour toutes les communications avec notre service.

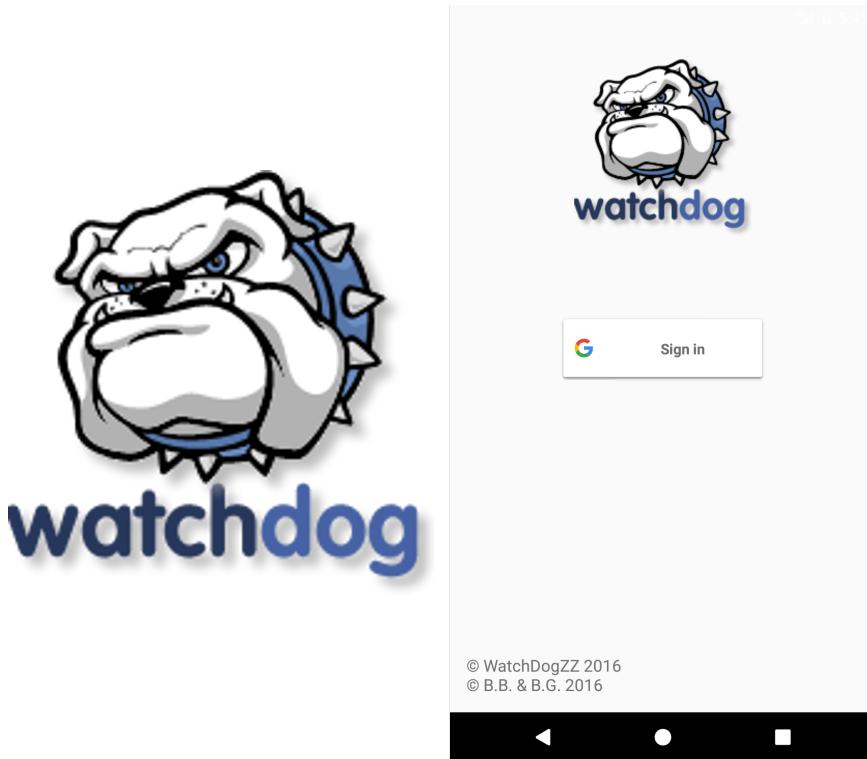


Figure 3.2 – Logo et page de connexion de WatchDogZZ

Cette première activité est un classique des applications connectées Android moderne et apporte un niveau souvent bien plus fiable de sécurité qu'un système d'authentification géré par le développeur où les failles peuvent vite devenir nombreuses (mots de passe non chiffrés en base, etc.).

Une fois cette phase d'authentification passée, l'utilisateur arrive donc sur l'activité principale de l'application : la carte de l'ISIMA. Cette carte en trois dimensions est une simple vue OpenGL où l'arbre de scène du Renderer a été réimplémenté afin d'obtenir les fonctionnalités de navigation et d'affichage voulues de la carte.

Comme on peut le voir sur la figure 3.3, les différents utilisateurs sont visibles sur la carte, symbolisés par de petites sphères colorées. Leur position se met à jour automatiquement toutes les secondes. Un bouton dans le coin inférieur droit de l'écran permet aussi de positionner des points d'intérêt sur la carte.

L'application dispose d'un menu glissant utilisable en tirant le menu depuis le bord gauche de l'écran. Ce menu donne accès aux différentes fonctionnalités ajoutées comme :

- la carte normale ;
- la liste des utilisateurs ;
- la liste des points d'intérêt ;
- le partage de position.

Les autres items n'ont pas encore été implémentés, ils n'ont pas été jugés prioritaires en regard des autres fonctionnalités.

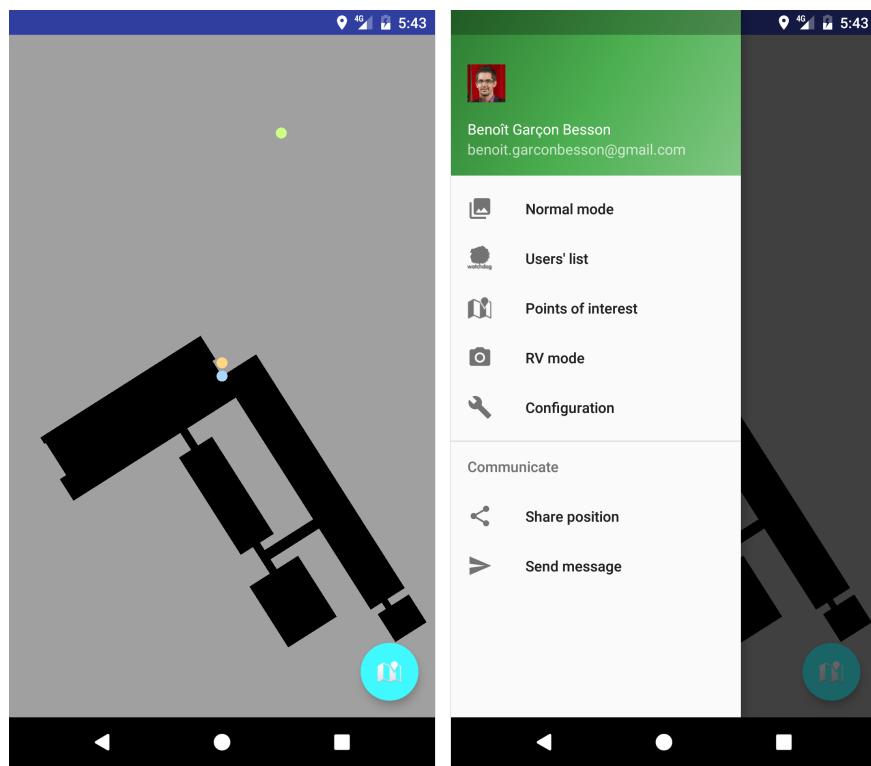


Figure 3.3 – Vue principale de la carte et menu glissant

La liste des utilisateurs permet donc d'identifier plus aisément chaque utilisateur. On dispose comme sur la figure 3.4 de la liste des utilisateurs avec leur photo et leur nom.

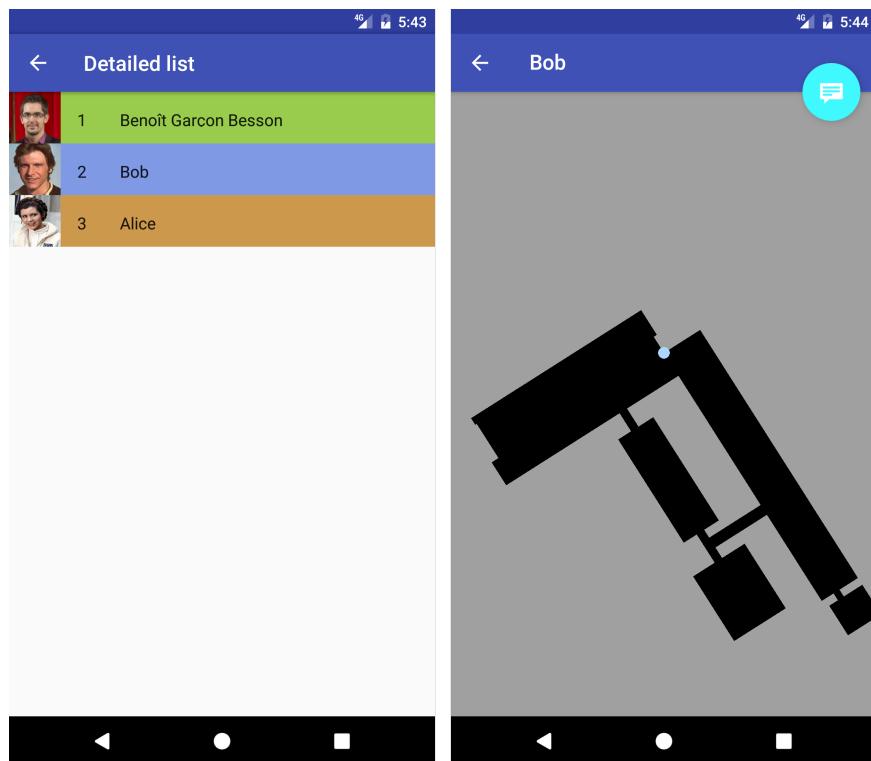


Figure 3.4 – Liste détaillée des utilisateurs

La couleur associée à chaque utilisateur est issue du hashage de leur nom : ainsi leur couleur dans la

liste est la même que celui de leur item sur la carte ce qui nous permet de les identifier. L'algorithme de hashage du nom est très simple et peut se traduire comme sur le code suivant.

```

1 int hash = label.hashCode();
2 r = abs((hash % 10)*255/10f/255f); // calcul de la composante rouge
3 hash/=10;
4 g = abs((hash % 10)*255/10f/255f); // calcul de la composante verte
5 hash/=10;
6 b = abs((hash % 10)*255/10f/255f); // calcul de la composante bleue

```

Code 3.1 – Hashage du nom en couleur

De plus en sélectionnant un utilisateur dans la liste, on peut obtenir une carte simplifiée avec seulement le suivi de cette personne pour un suivi plus clair.

Pour en revenir aux points d'intérêt, leur ajout se fait comme dit précédemment par l'utilisation du bouton flottant au bas de l'écran. Cette action invite l'utilisateur à positionner le point vert à l'emplacement désiré et à valider ce placement avec bouton d'ajout. Cette nouvelle action lance un nouveau fragment permettant de nommer ce point (figure 3.5).

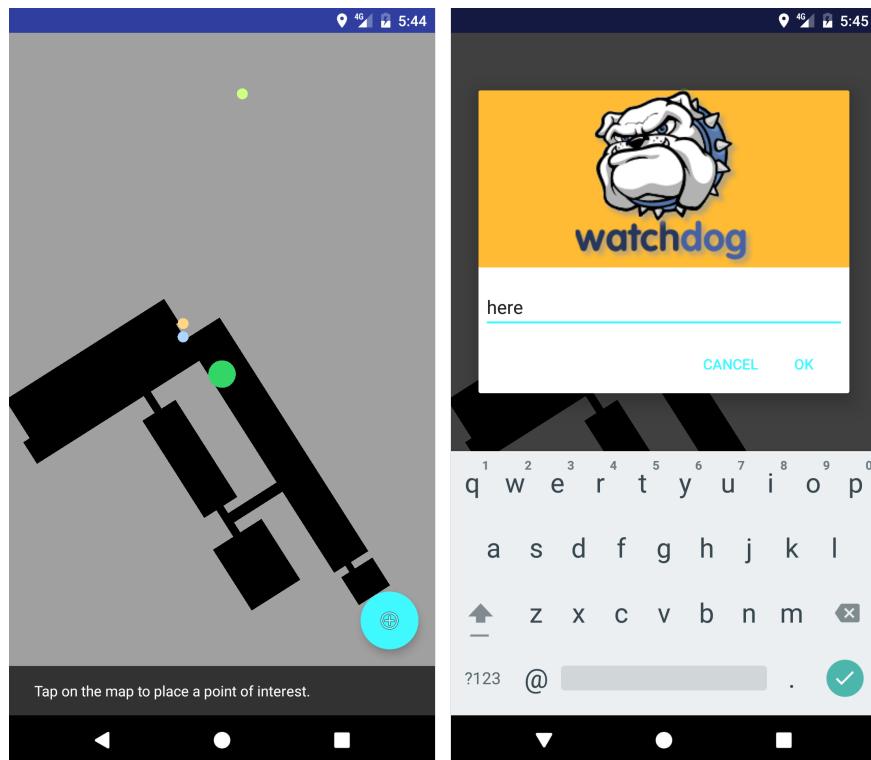


Figure 3.5 – Ajout d'un point d'intérêt

Ce point est ensuite disponible dans la liste des points d'intérêt qui se base sur les mêmes principes que la liste d'utilisateurs. Comme on peut le voir sur la figure 3.6, une couleur est associée associée à chaque point par rapport à son nom et il est possible d'afficher le point sur la carte en le sélectionnant dans la liste.

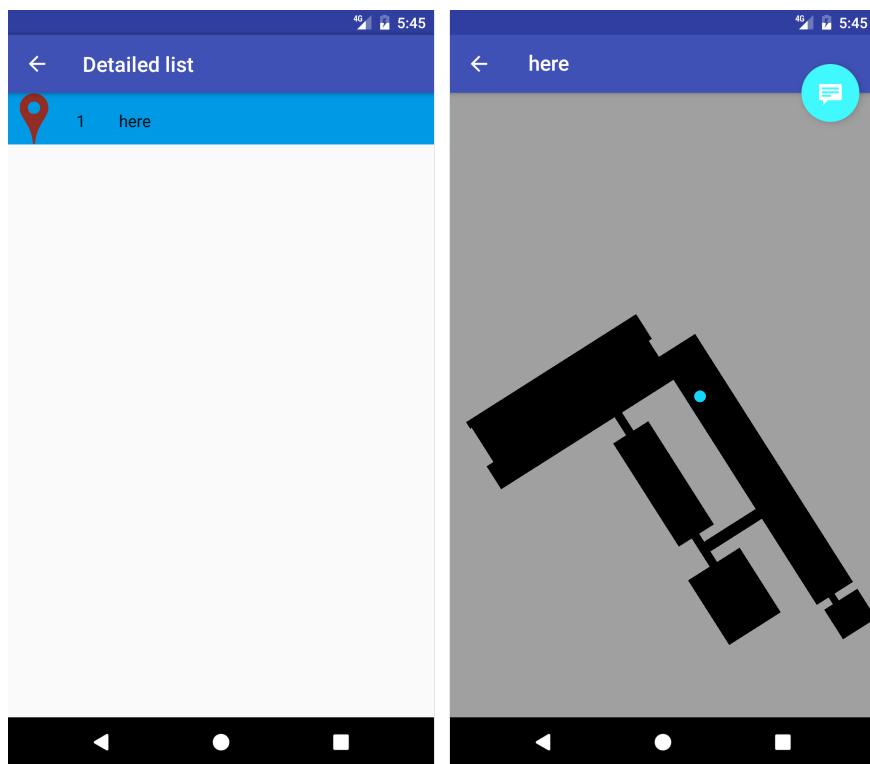


Figure 3.6 – Liste détaillée des points d'intérêt

Toutes ces fonctionnalités nécessitent l'utilisation du serveur, mais la dernière, le partage de position est une fonctionnalité qui permet de s'affranchir du service web puisqu'il permet d'envoyer sa position à n'importe qui par n'importe quel moyen de communication. Ceci est rendu possible par les intents Android. Les intents sont des descriptions d'une action à effectuer et pouvant comporter des données nommées. Les intents sont produits par les applications pour être dirigées par le système Android vers l'application la plus capable d'effectuer l'action et en cas de choix multiple, la décision est donnée à l'utilisateur. Ainsi en produisant un intent devant envoyer le texte comportant notre position n'importe quel processus de type messagerie SMS, email ou autre peut l'envoyer : c'est au choix de l'utilisateur. On peut donc utiliser l'application même en cas de problème de communication avec le serveur.

Enfin concernant les communications avec le serveur, celles-ci sont sécurisées puisqu'elles utilisent le protocole https. Il a fallu pour ce faire créer un certificat pour le serveur enregistré auprès d'un service de certification et ensuite ajouter ce service de certification auprès des services de confiance de l'application. Au final ceci permet d'échanger avec le serveur toutes les informations personnelles nécessaires en garantissant la sécurité minimale.

Concernant les performances du service, elles semblent plutôt bonnes pour un nombre limité d'utilisateur. La position du terminal courant peut être récupérée à une fréquence au moins égale à la fréquence d'affichage sur le service gps et celles des autres ne nécessitent que quelques dizaines de millisecondes ce qui est plutôt convenable pour des appels réseaux répétés. Toutefois nous n'avons pas d'information sur le comportement de notre service à une échelle plus réaliste c'est-à-dire avec un nombre d'utilisateur représentatif d'une situation réelle. Nous aurions pu utiliser un outil comme Gatling pour effectuer des tests de performances et simuler l'utilisation de notre service par un grands nombre d'utilisateurs mais le temps nous manquait.

Nous avons vu toutes les fonctionnalités disponibles sur notre solution mais il en existe de nombreuses autres que nous n'avons pas pu développer et certains points pourraient aussi être améliorés.

3.2 Améliorations possibles

Bien que l'application WatchDogZZ soit à l'heure actuelle une application complète et répondant à nos attentes, ce projet reste un sujet très vaste et très ouvert sur lequel il est possible de faire énormément.

La première chose qui pourrait apporter un plus à WatchDogZZ serait une interface d'administration. Initialement nous avions prévu d'en développer une, mais devant la quantité importante de travail nous avons préféré resté concentré sur la partie client-serveur pure. En effet deux des objectifs de ce projet était de monter en compétences en Android et en développement de service web en NodeJS, le développement web d'un frontend ne nous aurait pas forcément apporté quelque chose dans ce cadre.

De plus il y a des détails de fonctionnement qui pourraient être améliorés comme par exemple l'utilisation du jeton d'authentification Google. Il est utilisé pour valider l'identité de l'utilisateur et ensuite transmis à chaque échange mais ce token n'est pas vérifier à chaque fois par le service auprès de Google. Ceci peut entraîner une faille de sécurité importante et devrait être corrigé avant une distribution de l'application. Concernant l'application Android elle aussi dispose d'une faille de sécurité puisque pour le moment elle est signée à l'aide d'une clé de debug au lieu d'une vraie clé de signature pour des raisons de simplicité de développement.

Ensuite, il y a quelques fonctionnalités que nous voulions ajouter mais elles n'ont pas été sélectionnées au cours des itérations. On peut citer par exemple :

- la gestion des étages : actuellement, même si la carte est en trois dimensions, la composante altitude est uniformisée par soucis de calibrage ;
- l'utilisation d'une carte en réalité virtuelle avec la technologie Google Cardboard qui permet d'utiliser un smartphone comme un casque de réalité virtuelle en produisant deux images sur l'écran et en positionnant le smartphone à quelques centimètres des yeux de l'utilisateur ;
- le calcul d'itinéraire : l'application possède dans sa dernière version toutes les briques nécessaires pour faire un calcul d'itinéraire entre deux points (utilisateur ou intérêt) et l'afficher mais il reste à développer un algorithme de pathfinding manipulant nos données ;
- l'amélioration de l'interface utilisateur : l'interface est assez austère et n'est pas un produit commercialisable en l'état, il manquerait aussi à finaliser des micro-fonctionnalités générales comme le menu de configurations et des petits détails que l'on retrouve dans bon nombre d'application modernes, par exemple l'application a un système d'internationalisation qui n'a que deux langues, le français et l'anglais ;
- l'interception des SMS : nous aurions aimé pouvoir intercepter les SMS comportant un schéma spécial afin d'intégrer des informations provenant de sources multiples dans l'application. Par exemple ; lorsqu'un utilisateur partage sa position par SMS, si le destinataire possède l'application il pourrait être en mesure de l'afficher directement dans l'application sans savoir qu'il a reçu un sms avec des coordonnées gps brutes ;
- la communication entre usagers : il aurait été intéressant d'intégrer un service de messagerie à l'application pour pouvoir communiquer avec les autres usagers même si on ne dispose pas de ses coordonnées téléphoniques ;
- la gestion des logs du service : afin de pouvoir surveiller au mieux l'activité sur le serveur et pouvoir requêter des informations concernant une date, un utilisateur ou tout autre information utile ;
- un système de backup pour la base de données du serveur : en effet à l'heure actuelle aucune mesure de sauvegarde ni de restauration des données n'existe, ce qui ne serait pas tolérable sur un serveur professionnel.

3 – Résultats

Ce projet qui peut être perçu comme sans fin dispose d'énormément de possibilités d'évolutions et d'améliorations limitées seulement par notre imagination et notre budget.

Conclusion

Ce projet a donc été l'occasion de concrétiser au travers de l'application WatchDogZZ une idée personnelle. Nous avons pu développer une application complète reprenant les principes fondamentaux de la carte du maraudeur à savoir la géolocalisation d'usagers dans un établissement. A ceci de nombreuses fonctionnalités ont pu être ajoutées comme le partage de position, la gestion de points d'intérêts, etc. Ceci est rendu possible par le développement combiné d'une partie serveur et d'un client. Le client est une application Android compatible avec tout dispositif (smartphone, tablette, télévision, etc.) disposant d'un système en version 12 ou supérieur. Le web service est basé sur le framework NodeJS couplé à une base de données NoSQL : MongoDB ; permettant la communication de données sécurisées par le protocole HTTPS. Ce sont donc des technologies émergentes qui sont utilisées dans cette solution.

Au terme de ce projet tous les objectifs initiaux ont été atteints et de nombreuses fonctionnalités supplémentaires et de concepts originaux restent à développer. La taille du projet, ses spécifications et sa pluralité technologique en font un projet très complexe et demanderait beaucoup plus de temps pour être complet. C'est pourquoi, en début de projet, il a été décidé de se concentrer sur les fonctionnalités de base du service que nous souhaitions implémenter, afin de disposer d'une base solide et évolutive. Ceci a ensuite permis d'implémenter des fonctionnalités plus complexes dans des itérations de type "agile" en assurant qu'au terme du projet nous aurions une application fonctionnelle et répondant aux critères initiaux. C'est ce cheminement qui a permis la création de la solution WatchDogZZ, fonctionnelle, et disposant de certaines fonctionnalités supplémentaires.

Ce projet nous a permis d'atteindre quatre objectifs que nous nous étions fixés. Les deux premiers étaient des objectifs techniques à savoir progresser dans la maîtrise d'Android et NodeJS afin d'augmenter nos atouts techniques sur le marché du travail. Le troisième était de produire une application dans la lignée des applications modernes c'est-à-dire interagissant avec des services web et proposant des interactions sociales. Enfin le dernier objectif qui fut proposé par notre tuteur et qui consistait à progresser dans nos méthodes de génie logiciel en utilisant de l'intégration continu et du développement agile. L'atteinte de ces objectifs relève d'une plus grande satisfaction que la production de la solution elle-même.

De nombreux axes d'amélioration et d'évolution restent encore ouverts pour notre projet. Tout d'abord, toutes les fonctionnalités auxquelles nous avons pensé n'ont pas toutes été implémentées comme par exemple la vue en réalité virtuelle de l'établissement ou encore le calcul d'itinéraire. Celles-ci restent facilement intégrables dans l'application qui possèdent tous les prérequis à leur intégration. De plus le fait d'avoir un projet open source accessible sur GitHub permet à des contributeurs de pouvoir améliorer ou poursuivre notre projet, comme par exemple dans le cadre d'un projet ISIMA l'année prochaine.

Du point de vue de la diffusion de l'application, deux points majeurs peuvent être améliorés. Le premier concerne la portabilité du client : en effet il n'est actuellement disponible que sur les appareils Android, un portage iOS et Windows Phone permettrait de cibler la quasi-totalité du marché mobile. Le second point est le passage à l'échelle du web service : les tests exécutés montrent que pour un nombre très faible d'utilisateurs les performances du service sont parfaites, cependant pour un nombre d'usagers plus important le comportement de notre solution nous est encore inconnu. Ces améliorations potentielles pourraient faire de WatchDogZZ une application complète et sérieuse pouvant satisfaire les cas réels présentés dans ce rapport.

Références webographiques

- [1] COMPANEO. *La CNIL, la loi et la géolocalisation*. Récupéré sur : <http://www.companeo.com/geolocalisation-de-vehicules/guide/cnil-loi-et-geolocalisation>. 2016 (cf. p. 3).
- [2] CNIL. *Site web, cookies et autres traceurs*. Récupéré sur : <https://www.cnil.fr/fr/site-web-cookies-et-autres-traceurs>. 2017 (cf. p. 3).
- [3] FACEBOOK. *Safety Check*. Récupéré sur : <https://www.facebook.com/about/safetycheck/>. 2014 (cf. p. 3).
- [4] GOOGLE INC. *Vos trajets*. Récupéré sur : <https://www.google.fr/maps/timeline>. 2017 (cf. p. 4).
- [5] STAR DATA EXPLORE. *Position des bus en circulation sur le réseau star en temps réel*. Récupéré sur : <https://data.explore.star.fr/explore/dataset/tco-bus-vehicules-position-tr/>. Novembre 2014 (cf. p. 6).
- [6] GEOTEK. *Suivi de personnes*. Récupéré sur : <https://www.geotek.fr/geolocalisation/besoins/suivi-personnes.html>. 2017 (cf. p. 6).
- [7] FAMILY SAFETY PRODUCTION. *Find My Friends*. Récupéré sur : <https://play.google.com/store/apps/details?id=com.fsp.android.friendlocator>. Février 2017 (cf. p. 7).
- [8] WIKIPÉDIA. *Representational state transfer — Wikipédia, l'encyclopédie libre*. Récupéré sur : https://fr.wikipedia.org/wiki/Representational_state_transfer. Décembre 2016 (cf. p. 8).
- [9] NODE.JS FOUNDATION. *Node.js*. Récupéré sur : <https://nodejs.org/en/>. Octobre 2016 - Février 2017 (cf. p. 19).
- [10] NPM, INC. *Build amazing things*. Récupéré sur : <http://www.npmjs.com>. Octobre 2016 - Février 2017 (cf. p. 19).
- [11] GOOGLE INC. *Download Android Studio and SDK Tools*. Récupéré sur : <https://developer.android.com/studio/index.html>. Octobre 2016 - Février 2017 (cf. p. 20).
- [12] GITHUB, INC. *GitHub*. Récupéré sur : <https://github.com/WatchDogZZ>. Octobre 2016 - Février 2017 (cf. p. 23).
- [13] INTERNET SECURITY RESEARCH GROUP (ISRG). *Let's Encrypt*. Récupéré sur : <https://letsencrypt.org/>. Février 2017 - Février 2017 (cf. p. 25).
- [14] TRAVIS CI, GMBH. *Travis CI User Documentation*. Récupéré sur : <https://docs.travis-ci.com/>. Octobre 2016 - Février 2017.
- [15] TRAVIS CI, GMBH. *Travis CI - Test and Deploy Your Code with Confidence*. Récupéré sur : <https://travis-ci.org/>. Octobre 2016 - Février 2017.
- [16] Michael KATZ. *How To Write A Simple Node.js/MongoDB Web Service for an iOS App*. Récupéré sur : <https://www.raywenderlich.com/61078/write-simple-node-jsmongodb-web-service-ios-app>. Octobre 2016 - Février 2017.
- [17] MONGODB, INC. *MongoDB for GIANT Ideas / MongoDB*. Récupéré sur : <https://www.mongodb.com/>. Octobre 2016 - Février 2017.