ISIMA
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 3e année

Filière architecture et génie logiciel

WatchDogZZ

Étudiants :
Benjamin BARBESANGE,
Benoît GARÇON

Tuteur :
Pierre COLOMB
Tuteur ISIMA :
Eva HASSINGER

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.

Résumé – Abstract

Résumé

La travail présenté dans ce rapport concerne l'élaboration d'une solution visant à **orienter** facilement des utilisateur au sein d'un bâtiment. Pour ceci, les utilisateurs disposent d'une **carte interactive** sur **mobile** se mettant à jour en effectuant des requêtes sur un **Service Web**. Ce travail se découpe donc en 2 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 va permettre 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 de se connecter au Service Web. Cette application effectuera des requêtes visant à mettre à jour la position de l'utilisateur sur le Service Web. Ainsi les applications d'autres utilisateurs seront capables de récupérer ces positions et les placer au sein d'une carte modélisant un bâtiment.

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 2 utilisateurs ou vers un point d'intérêt.

Mots clés : Carte interactive, Express, NodeJS, Android, Service Web, mobile.

Abstract

English abstract

Keywords: NodeJS, Android, Web Service, mobile

Table des matières

Liste des figures, tableaux, algorithmes et extraits de code

Liste des figures

Liste des tableaux

Liste des algorithmes

Liste des extraits de code

Glossaire

Word : Definition

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

1.1 Présentation du projet

Ce projet à été mis en place selon 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.

Le but de ce projet est de proposer une manière simple pour tout le monde de pouvoir s'orienter dans le locaux de l'ISIMA. Une fois cette solution éprouvée avec les bâtiments de l'ISIMA, nous pouvons penser l'étendre à n'importe quel autre bâtiment donc nous pouvons avoir les plans. Cette idée est inspirée d'un film de Harry Potter, avec la fameuse carte du Maraudeur qui lui permet de suivre les déplacements de tout le monde dans Poudlard.

De manière générale, nous devons être capable de visualiser une carte de l'ISIMA, mais également de pouvoir trouver facilement un bureau ou une salle de cours. De plus, il serait intéréssant de pouvoir également localiser n'importe quelle autre personne visualisant la carte en même temps. Cette visualisation doit s'actualiser assez rapidement pour que la location de personnes soit la plus précise possible pour les utilisateurs.

1.2 Analyse de l'existant

Suivi dans les bâtiments...

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.

La solution la plus évidente en terme de support de visualisation pour les utilisateurs et 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 permettra également de stocker des informations utiles aux utilisateurs, ce qui permettra également d'alléger le volume de données stockées sur leurs terminaux.

1.3.1 Architecture

L'architecture choisie pour organiser notre solution est la suivante.

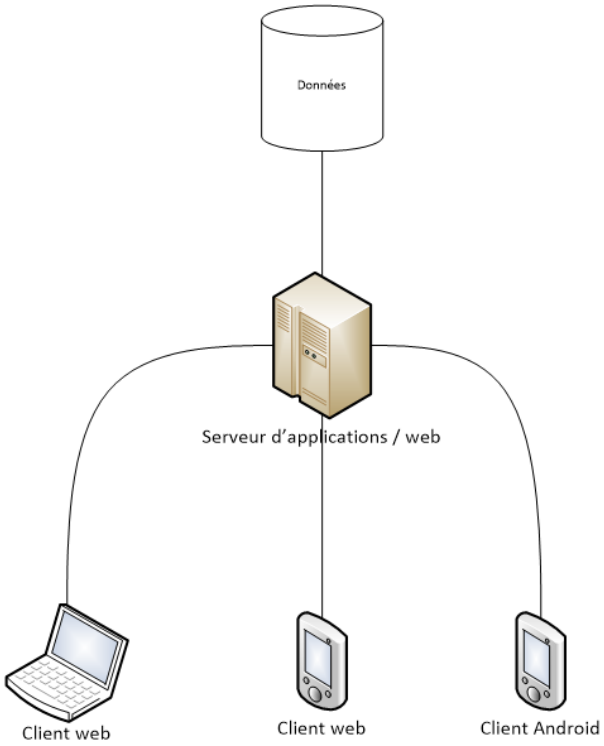

Figure 1.1 – Architecture

Nous pouvons observer dans la figure 1.1 que celle-ci ressemble à une architecture client/serveur, ce qui est le but recherché.

1.3.2 Partie serveur

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

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

Le serveur pourra aussi en option faire :

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

Le serveur comportera une base de données dans laquelle les informations des utilisateurs seront stockées (login / mot de passe). Les dernières positions des utilisateurs seront également stockées. Il y aura une table permettant d'identifier quels utilisateurs sont actuellement actifs. Il a été convenu d'utiliser NodeJS du coté serveur pour plusieurs raisons :

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

Base de données La base de données utilisée est une base MongoDB. Ce type de base de données est trés simple à mettre en place et se base sur un format JSON binaire pour stocker les différentes informations que nous souhaitons. Ceci va faire qu'il est facile d'utiliser les résultats des requêtes directement en Javascript. Requetes sur le web service 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 sous forme d'objet JSON dans le corps de la réponse. Les réponses contiennent les champs suivants :

Code 1.1 – Corps de la réponse serveur

```
{
    'status': 'ok' / 'fail', // L'état de la requête
    'error': 'description' // Une description de l'erreur s'il y en a une
}
```

La plupart des requêtes décrites nécéssitent que l'utilisateur soit authentifié. /login La première requête à effectuer sur le service doit s'effectuer avec la méthode POST sur cette URL. Pour se connecter, l'utilisateur doit envoyer les paramètres suivants :

Code 1.2 – Corps de la requête login

```
{
    'name': 'username', // Le nom de l'utilisateur àconnecter
    'location': [1.0, 2.0, 3.0] // La position courrante de l'utilisateur
}
```

/who Cette URL permet de récupérer en méthode GET une liste de noms de personnes actuellement en ligne. Le retour est sous la forme suivante :

Code 1.3 – Corps de la requête who

```
{
    'list': [
        'userName1',
        'userName2',
        'userName3'
    ]
}
```

/where En utilisant la méthode GET, cette URL retourne la liste des utilisateurs ainsi que leur position. En utilisant la méthode POST et en passant les paramètres adéquats, l'utilisateur peut mettre à jour sa position. En méthode GET, le service renvoie une liste des utilisateurs connectés avec leurs positions :

Code 1.4 – Corps de la requête where GET

```
{
    'list': [
        {
            'name': 'username1',
            'location': [1.0, 2.0, 3.0]
        },
        {
            'name': 'username2',
```

```
9             'location': [1.0, 2.0, 3.0]
10        },
11        {
12            'name': 'username3',
13            'location': [1.0, 2.0, 3.0]
14        }
15    ]
16 }
```

En utilisant la méthode POST, l'utilisateur met à jour sa position. Les paramètres à envoyer sont les suivants :

Code 1.5 – Corps de la requête where POST

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

Si l'utilisateur n'est pas connecté au moment de mettre à jour sa position, le serveur va tenter de l'authentifier et sauvegarder sa position. Partie administration Les fonctionnalités minimales pour l'application d'administration sont :

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

Ensuite les fonctionnalités avancées pourront être :

- gestion des utilisateurs

1.3.3 Partie Android

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

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

L'application pourra évoluer et proposer :

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

Le choix d'une application Android se justifie par :

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

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

1.3.4 Intégration continue

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

— un serveur d'application
— un serveur d'intégration

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

1.4 Organisation du travail

L'organisation temporelle théorique du travail est décrite dans le diagramme de Gantt ci-dessous.

Diagramme de Gantt initial du projet WatchDogZZ

ID	Nom de tâche	Début	Terminer	Durée
1	**Lancement**	**17/10/2016**	**15/11/2016**	**30j**
2	Recherches préliminaires	17/10/2016	26/10/2016	10j
3	Spécifications	17/10/2016	26/10/2016	10j
4	Mise en place des archis	17/10/2016	07/11/2016	22j
5	Mise en place de l'intégration continue	17/10/2016	15/11/2016	30j
6	**Application basique**	29/10/2016	**30/12/2016**	**63j**
7	Application Android	29/10/2016	30/12/2016	63j
8	Serveur NodeJS	29/10/2016	30/12/2016	63j
9	Outil d'administration	13/11/2016	30/12/2016	48j
10	**Améliorations**	01/01/2017	**12/02/2017**	**43j**
11	**Application Android**	**01/01/2017**	**12/02/2017**	**43j**
12	Carte 3D	01/01/2017	31/01/2017	31j
13	Carte RA	13/01/2017	12/02/2017	31j
14	Pathfinding	01/01/2017	07/02/2017	38j
15	Wizz	01/01/2017	19/01/2017	19j
16	Carte VR	13/01/2017	12/02/2017	31j
17	Serveur NodeJS	01/01/2017	07/02/2017	38j
18	Outil d'administration	01/01/2017	15/01/2017	15j
19	**Finalisation**	**06/11/2016**	**04/03/2017**	**119j**
20	Rapport	29/01/2017	27/02/2017	30j
21	Préparation de la soutenance	18/02/2017	04/03/2017	15j
22	**Documentation**	**06/11/2016**	**27/02/2017**	**114j**
23	Documentation continue	06/11/2016	30/01/2017	86j
24	Finalisation	31/01/2017	27/02/2017	28j

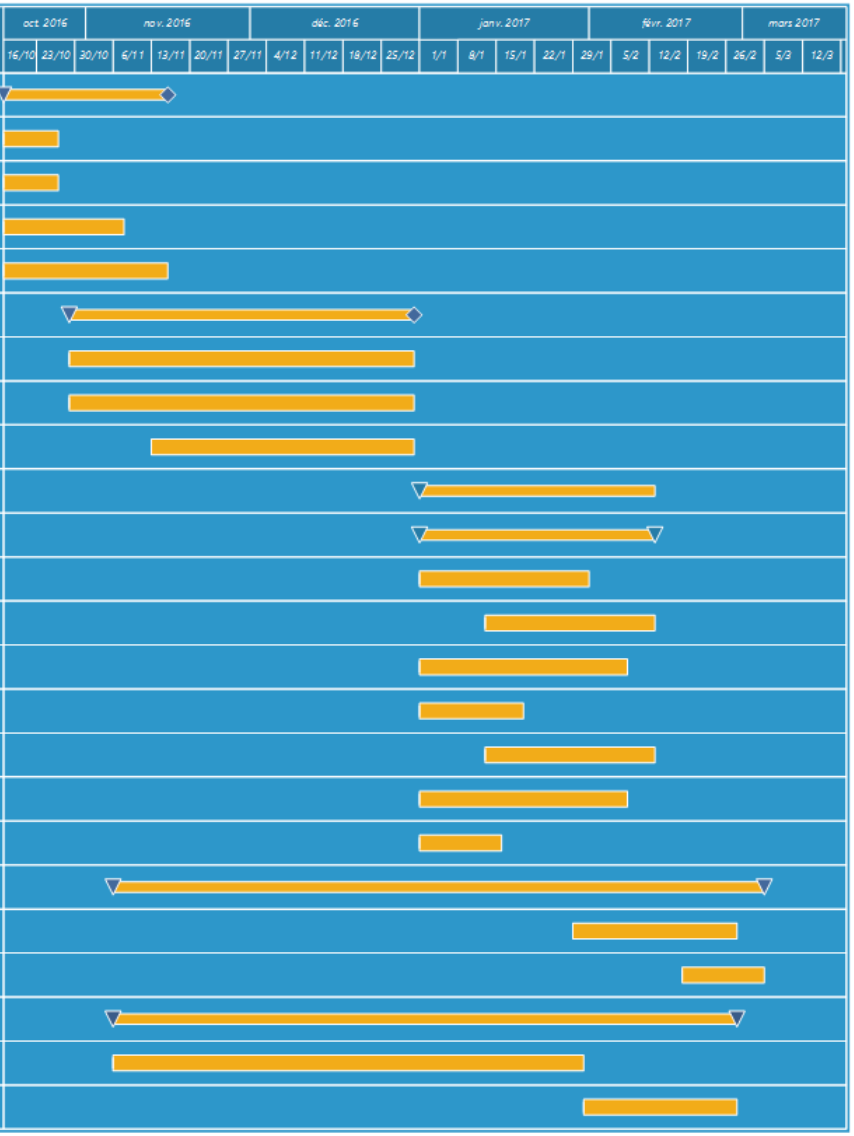

Figure 1.2 – Diagramme de Gantt théorique

suite...

2 Conception de la solution

2.1 Technologies utilisées

2.1.1 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 [1]. C'est un framework Javascript qui dispose de nombreuses librairies installables à l'aide du gestionnaire de modules NPM [2]. 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 nécéssaires 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 nécéssaires 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 ou Https ;

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).

2.1.2 Android

Le développement d'une application Android s'effectue généralement en utilisant le JAVA ainsi qu'Android Studio [3], permettant d'inclure les bibliothèques Android. Ici, nous avons donc utilisé ces technologies pour permettre une intégration parfaite sur le système Android.

2.2 Architecture de la solution

2.2.1 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. Pour ce faire, l'architecture est la suivante.

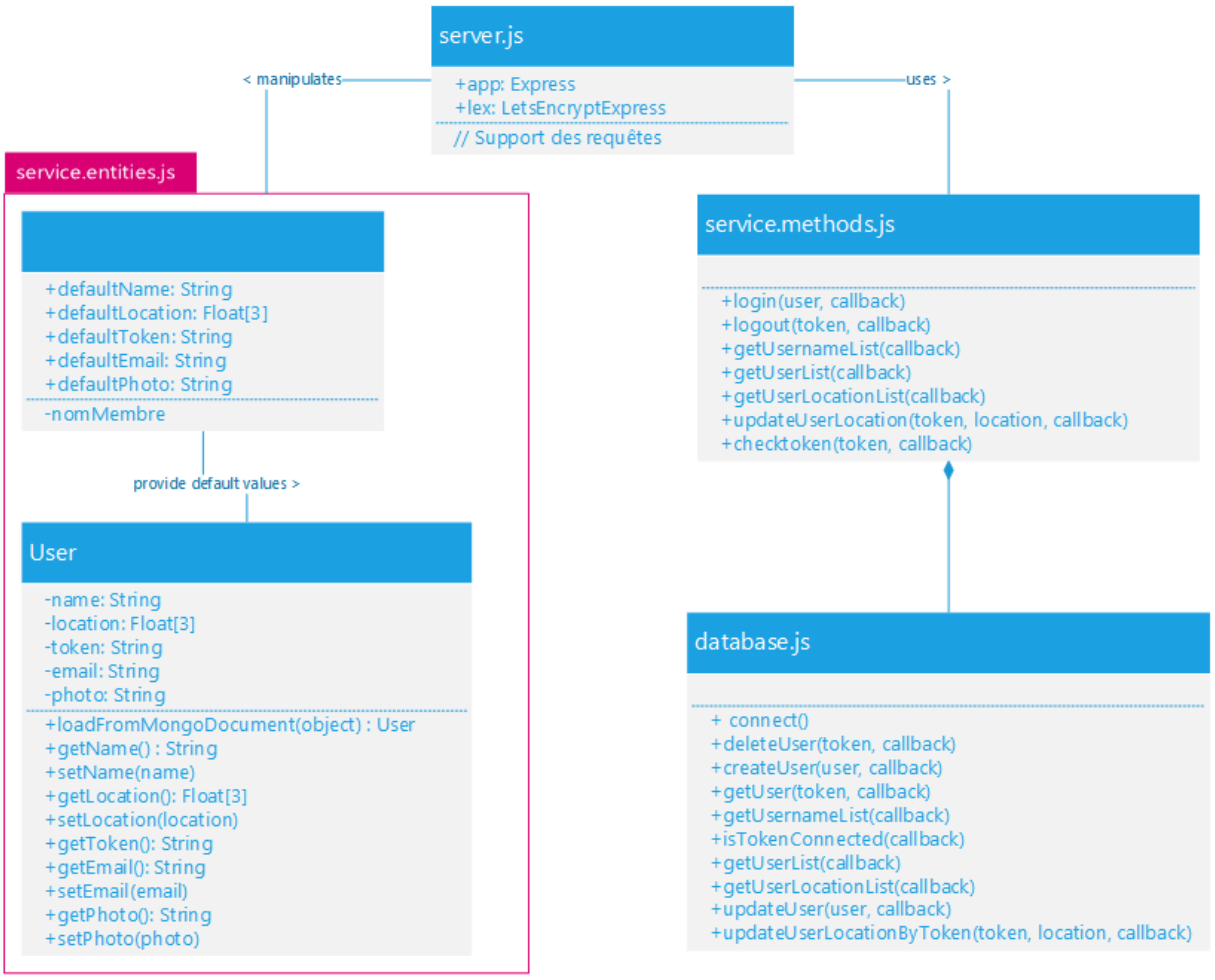

Figure 2.1 – Architecture du Service Web

Comme nous pouvons l'observer dans la figure 2.1,

Le serveur + Mongodb. Pattern bridge pour la bdd. Des entités.

2.2.2 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 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 était 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 2.2.

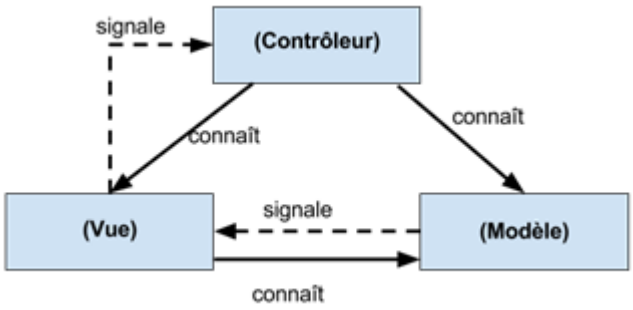

Figure 2.2 – Patron de conception MVC

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 on accès à l'API du modèle symbolisée par le gestionnaire d'utilisateur (UserManager). Le modèle gère donc tous les petits traitements bas niveau sur les données et les sert au reste de l'architecture selon les besoins.

Le contrôleur (C) s'occupe des interactions avec l'utilisateur, il doit pouvoir transmettre les commandes émanant de l'utilisateur aux autres composants.

2.3 Fonctionnalités introduites

3 Résultats

3.1 Test de la solution

3.2 Améliorations possibles

Conclusion

Ce projet a donc été l'occasion de concrétiser au travers de l'application WatchDogZZ notre 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 intégral des parties client et serveur. 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 la technologie nodeJS couplé à une base de données NoSQL MongoDB permettant la communication de données sécurisées par le protocole HTTP.

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. Devant ce constat le choix judicieux a été fait en début de projet de se concentrer dans un premier temps à produire une solution primaire, élémentaire et avec seulement les fonctionnalités de bases afin d'avoir un livrable fonctionnel. Ceci a ensuite permis d'implémenter des fonctionnalités plus complexes dans des itérations agiles en assurant qu'au terme du projet nous aurions une application fonctionnelle et répondant aux critères initiaux. Ainsi nous avons pu produire WatchDogZZ avec certaines fonctionnalités supplémentaires.

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. Cependant celles-ci restent facilement intégrables dans l'application qui possèdent tous les prérequis à leur intégration. D'un point de vue plus diffusion de l'application, deux points majeurs peuvent être améliorés. Le premier est la portabilité du client : en effet il n'est actuellement disponible que sur les appareils Android, le porter sur iOS et Windows Phone permettrait de pouvoir toucher la quasi-totalité du marché cible. Le deuxième 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 mais concernant 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

Références

[1] NODE.JS FOUNDATION. *Node.js*. Récupéré sur : https://nodejs.org/en/. Octobre 2016 - Février 2017 (cf. p. 8).

[2] NPM, INC. *Build amazing things*. Récupéré sur : http://www.npmjs.com. Octobre 2016 - Février 2017 (cf. p. 8).

[3] GOOGLE INC. *Download Android Studio and SDK Tools*. Récupéré sur : https://developer.android.com/studio/index.html. Octobre 2016 - Février 2017 (cf. p. 8).

[4] TRAVIS CI, GMBH. *Travis CI User Documentation*. Récupéré sur : https://docs.travis-ci.com/. Octobre 2016 - Février 2017.

[5] TRAVIS CI, GMBH. *Travis CI - Test and Deploy Your Code with Confidence*. Récupéré sur : https://travis-ci.org/. Octobre 2016 - Février 2017.

[6] GITHUB, INC. *GitHub*. Récupéré sur : https://github.com/WatchDogZZ. Octobre 2016 - Février 2017.

[7] 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.

[8] MONGODB, INC. *MongoDB for GIANT Ideas | MongoDB*. Récupéré sur : https://www.mongodb.com/. Octobre 2016 - Février 2017.