IPB
ENSEIRB
MATMECA
BORDEAUX

Rapport de projet PR311

Intitulé : Réalisation d'une plateforme de géolocalisation basé sur le moteur et le format Open Street Map (OSM)

Réalisé par :

Amel Raboudi

Reda Sabir

Zakaria Hili

Encadré par :

M.Antoine Rollet

Table des matières

Table des figures

Introduction

Afin de compléter notre formation en réseaux et systèmes répartis, nous avons réalisé ce projet de développement système sous l'encadrement de monsieur Antoine Rollet. Il consiste à développer une plateforme de géolocalisation basé sur Open Street Map et qui met en pratique nos connaissances réseaux et systèmes (répartis/distribués).

Nous nous sommes partagés en équipe de 3/4 personnes en vue de mettre en place pour chacun, des serveurs pour la gestion des zones géographiques distinctes et des clients se déplaçant sur ces zones. L'objectif est alors de mettre en place des clients et des serveurs interopérables entre elles indépendamment de la technologie et des implémentations utilisées par chaque équipe. Pour cela, toutes les équipes ont participé à la définition du protocole de communications unifié entre toutes les entités communicantes.

Dans ce rapport, nous allons présenter notre projet que nous avons l'appelé Sekai (le monde en japonais). Dans un premier temps nous présenterons les spécifications que nous avons établi pour notre projet. Dans un second temps, nous détaillerons la conception et la mise en œuvre de notre solution. Et dans un dernier temps, nous expliquerons les choix techniques de notre projet et le travail réalisé.

Chapitre 1

Spécifications des besoins

Cette partie du rapport décrit les descriptions fonctionnelles et non fonctionnelles à respecter dans notre démarche.

1.1 Spécification des besoins fonctionnels

L'application OSM doit satisfaire un certain nombre de conditions fonctionnelles comme suit :

- Afficher les véhicules dans l'écran du serveur en fonction de leurs coordonnées GPS.
- Permettre à chaque serveur d'afficher la zone de la carte OSM dont il est responsable.
- Visualisation instantanée des déplacements des véhicules dans les serveurs.
- La gestion de la communication entre différents clients et différents serveurs en se basant sur le protocole établit préalablement : acceptation des communications, redirection,...
- Chaque serveur doit garder un fichier de configuration pour définir la zone dont il est responsable et afin de connaître les serveurs responsables de ces zones voisines.
- Traçage du trajet d'une véhicule spécifique, pour la journée, ou pour la veille.
- Simulation du trajet d'un véhicule par envoie des coordonnées GPS chaque 10 secondes. Ce trajet doit être le plus réel possible.

1.2 Spécification des besoins non fonctionnels

Les besoins non fonctionnelles de ce projet sont :

— L'interopérabilité : le serveur doit pouvoir détecter les autres clients qui sont dans sa zone et les afficher.
— La réactivité : Le serveur doit pouvoir mettre à jour sa carte pour suivre le trafic instantanément.
— La fiabilité : Les valeurs enregistrées et affichées sur la carte doivent être les plus précis possibles.
— L'identification : Le serveur doit pouvoir différencier entre les clients identifiés
— La maintenance : L'application développée doit être facilement configurable après son déploiement.

Chapitre 2

Conception de notre solution : Sekai

Dans cette partie nous allons présenter les diagrammes de conceptions de notre application.

2.1 Diagrammes des cas d'utilisation

La solution que nous proposons pour ce projet contient trois acteurs essentiels : Un client

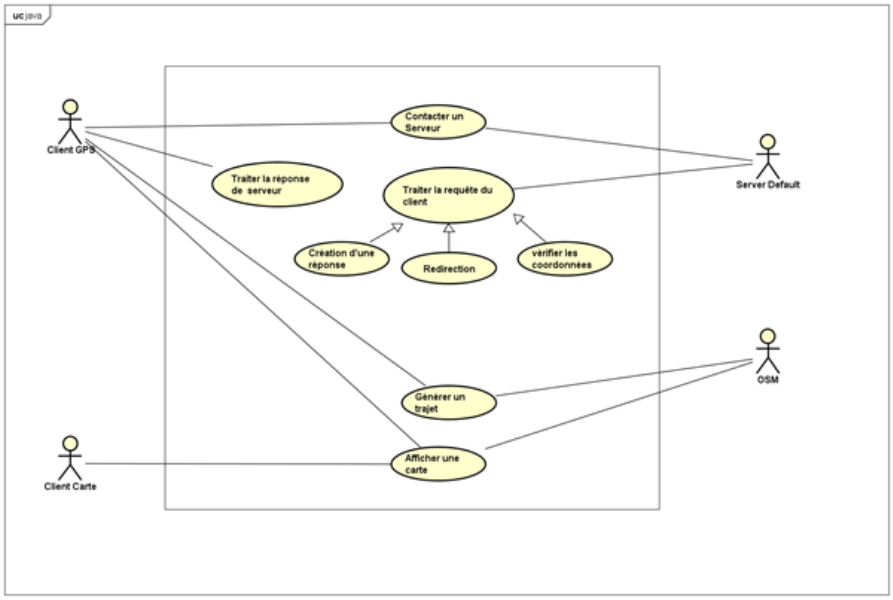

FIGURE 2.1 – Diagramme des cas d'utilisation

GPS : c'est l'acteur simulant le déplacement d'un véhicule sur le réseau,il génère un trajet en se basant sur la carte OSM et il l'envoie à son serveur par défaut en attendant sa réponse pour la traiter.

Un client Carte : qui affiche la carte du serveur selon une configuration donnée ainsi que les véhicules détectés sur cette carte.

Un serveur : c'est l'acteur principale dans le système, il traite les requêtes des clients et redérige le trafic vers le serveur à contacter.

2.2 Diagramme de séquence

Le diagramme de séquence suivant montre les scénarios prévus et attendus lors de la conception.

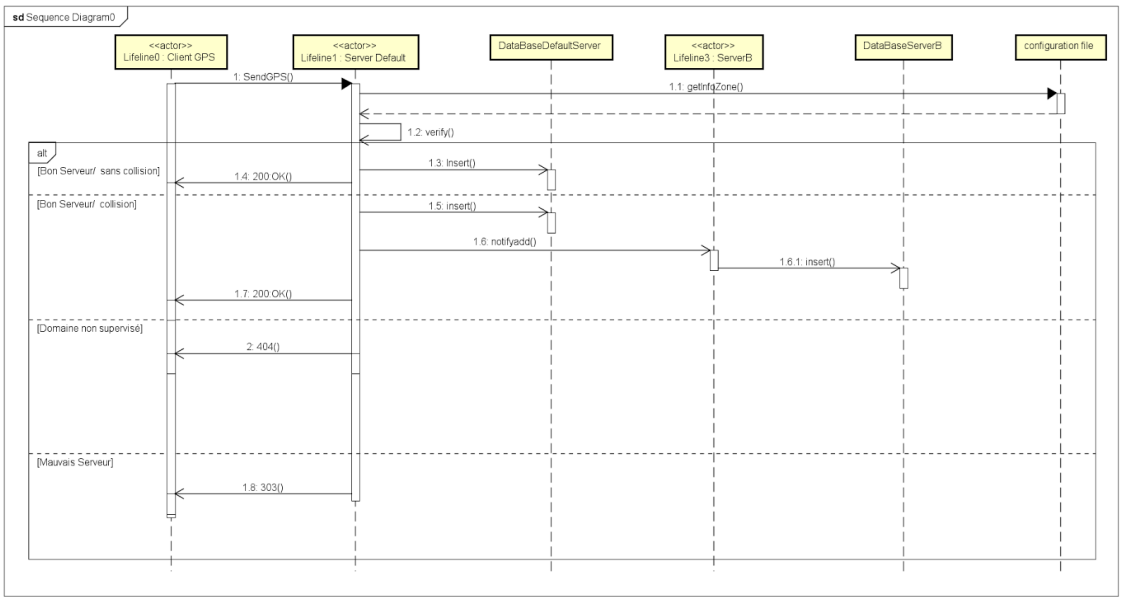

FIGURE 2.2 – Diagramme de séquence

1. Le client envoie sa requête à son serveur par défaut par invocation de la méthode sendGPS.
2. Le serveur contacté vérifie ses informations de zone et renvoie une réponse personnalisée selon les cas de figure ci après :
   (a) Le serveur contacté est le seul bon serveur , i. e. les coordonnées GPS envoyés sont dans la zone de la carte géré par ce serveur et dans aucune autre zone d'un autre serveur.
   (b) Le serveur contacté est le bon serveur mais il y a d'autre serveurs qui gèrent la même partie de la carte.
   (c) Le serveur contacté n'est pas le bon serveur mais il connaît le serveur à contacter.

(d) Les coordonnées GPS envoyés ne sont dans aucune zone gérée par un serveur.

Dans le premier cas, le serveur ajoute le client dans sa base de donnée en vue d'affichage sur la map. Dans le second, il demande aux autres serveurs concernés par ce client de l'ajouter aussi dans leurs bases de donnée. La réponse au client dans les deux cas est 200 OK. Pour la troisième cas, il envoie au client une réponse de redirection 303 en indiquant le bon serveur à contacter. Et pour le dernier cas, il renvoie un 404 au serveur pour lui dire que sa zone n'est couvert par aucun serveur.

2.3 Architecture globale

L'application suit le modèle trois tiers comme suit :

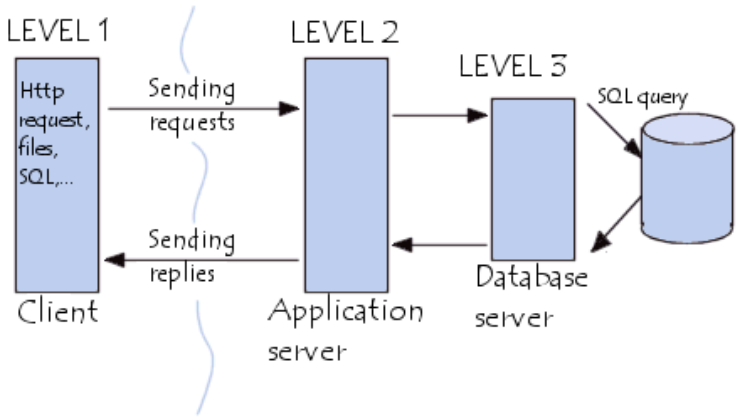

FIGURE 2.3 – Architecture trois-tiers de toute la solution

— Une application cliente qui contacte le serveur distant par des requêtes SQL.
— Un serveur d'application gérant les processus de l'application et exécutant ses fonctionnalités.
— Un serveur de base de donnée qui gère l'accès au données en écriture ou en lecture.

2.4 Architecture du serveur

Nous détaillons dans cette partie l'architecture conceptuelle de notre serveur et ses principaux fonctionnalités ainsi que la conception de la base de donnée locale.

2.4.1 Diagramme de paquetages

L'application serveur suit le modèle MVC comme le montre le diagramme de paquetages suivant :

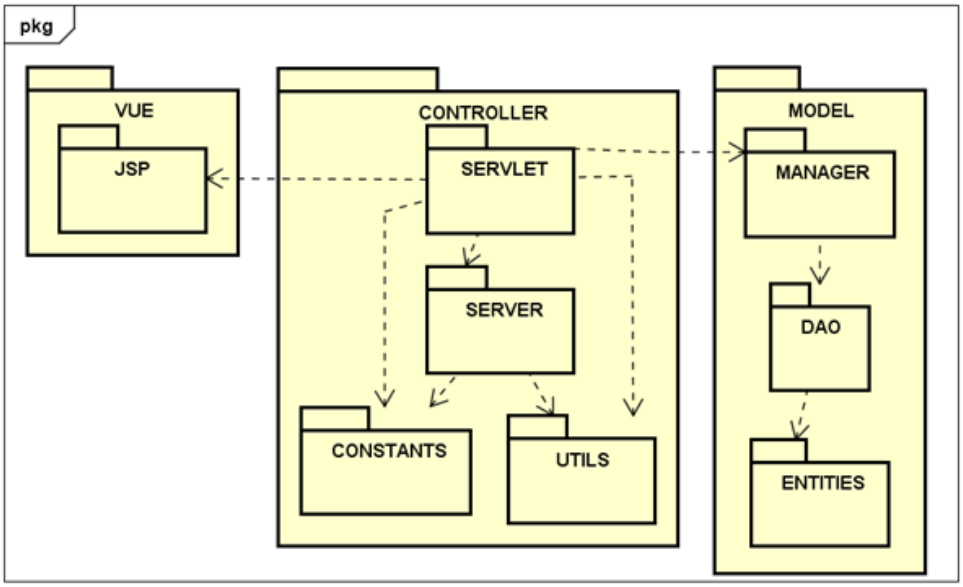

FIGURE 2.4 – Diagramme de paquetage de l'application serveur

-Le package "MODEL" est celui qui représente le modèle, il est responsable de la gestion de la base de donnée et représente la couche d'accés aux données fournis par le serveur de la base de donnée. Il est lui même répartis en paquetages.

le MANAGER responsable de la communication avec la couche applicative et le serveur de Base de donnée : la connexion à la base, l'exécution des requêtes, l'envoie des résultats bien formatés à la couche supérieure.

le DAO responsable de l'accés à la base : la modification de la liste des utilisateurs, des leurs positions et la formulation des requêtes.

les ENTITIES qui représente les entités correspondantes aux tables de la base de donnée.

-Le paquetage "CONTROLLER" C'est la où les traitements sont centralisés. Il contient trois groupes de paquetages essentiels

Le SERVLET responsable de la communication avec les autres couches du modèle MVC : il reçoit les requêtes des clients et renvoie la réponse adaptée, il demande la liste ac-

tualisée et récente des clients de la base de donnée pour pouvoir les afficher dans la couche vue.

Le SERVER c'est le paquetage métier qui représente le coeur de notre application : il est responsable du traitement des requêtes clientes et de la génération de la réponse adéquate.

Les CONSTANTS et UTILS des paquetages pour les fonctionnalités d'aide au développement.

-Le paquetage "VUE" C'est la où se trouve les traitement d'affichage de la carte OSM en se basant sur des fonctions javascripts et du code jsp.

2.4.2 Modèle conceptuel des données

Nous avons conçu notre base de donnée de façon à être facilement évolutive et préparée à l'intégration d'autres fonctionnalités.

Le modèle relationnel de la base de donnée conçue pour ce projet englobe deux entités essentielles :

— Les Utilisateurs du serveurs : Représenté par l'entité "user" ayant trois attributs :

Le ID : clé primaire auto-incrémentable servant au référencement dans la base de donnée.

Le NOM : sert éventuellement à attribuer des noms localement au utilisateurs du serveur

L'identifiant : stocke la valeur d'identification envoyée par le client au serveur.

— Les noeuds de la carte déjà visités : Représentés par l'entité "Node" ayant trois attributs :

Le ID : clé primaire auto-incrémentable servant au référencement dans la base de donnée.

Les coordonnées gps représentés par : La LONGITUDE et La LATITUDE.

Ces entités ont comme relation un lien N-N : many-to-many traduisant qu'un utilisateur peut être associé à plusieurs noeuds et un noeud peut être associé à plusieurs utilisateurs, cette relation est caractérisée par la date de son ajout dans la base.

Ci-après le modèle entité-association relatif à cette description.

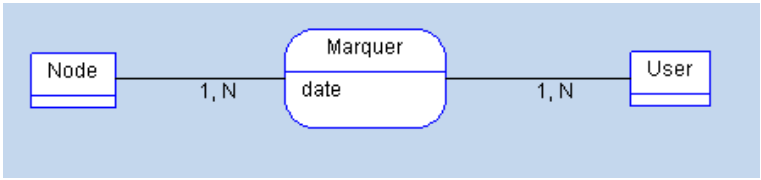

FIGURE 2.5 – Modèle entité-association de la base de donnée

2.5 Architecture du client

Dans ce qui suit une présentation en détail de l'application cliente que nous avons développé pour interroger nos serveurs de géolocalisation.

2.5.1 Architecture générale

Nous avons adopté l'architecture MVCS. Ce paradigme regroupe les fonctions nécessaires en quatre catégories :

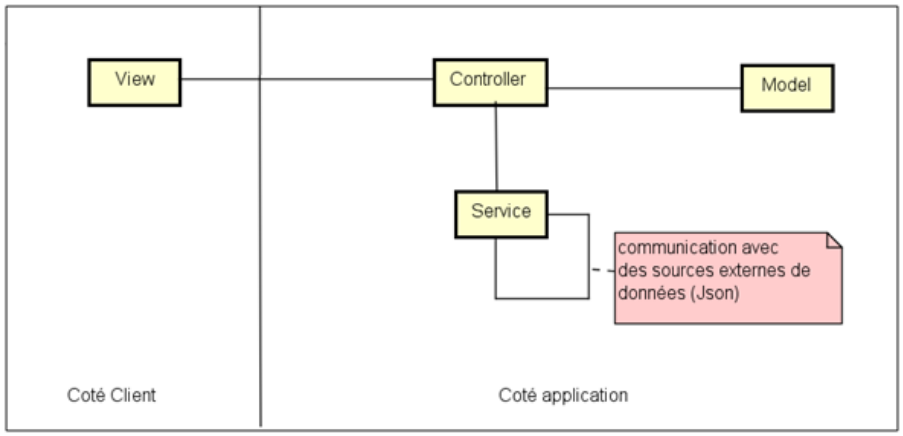

FIGURE 2.6 – Architecture globale de l'application cliente

— un modèle M (modèle de données) : responsable de la gestion de la base de donnée locale utilisé pour le stockage des données de géolocalisation et des information sur les serveurs contactés
— une vue V (présentation, interface utilisateur) : pour permettre aux utilisateurs de simuler l'envoie des coordonnées GPS.
— un contrôleur C (logique de contrôle, gestion des événements, synchronisation) : qui orchestre tout les entités du projet et qui est responsable du code métier.

— un service S (interface avec des API et d'autres sources externes) : qui est responsable de la gestion des interactions avec la carte OSM et d'autres ressources extérieurs.

2.5.2 Diagramme de paquetage

Le diagramme de paquetage offre une vue plus en détail de notre architecture de l'application cliente.

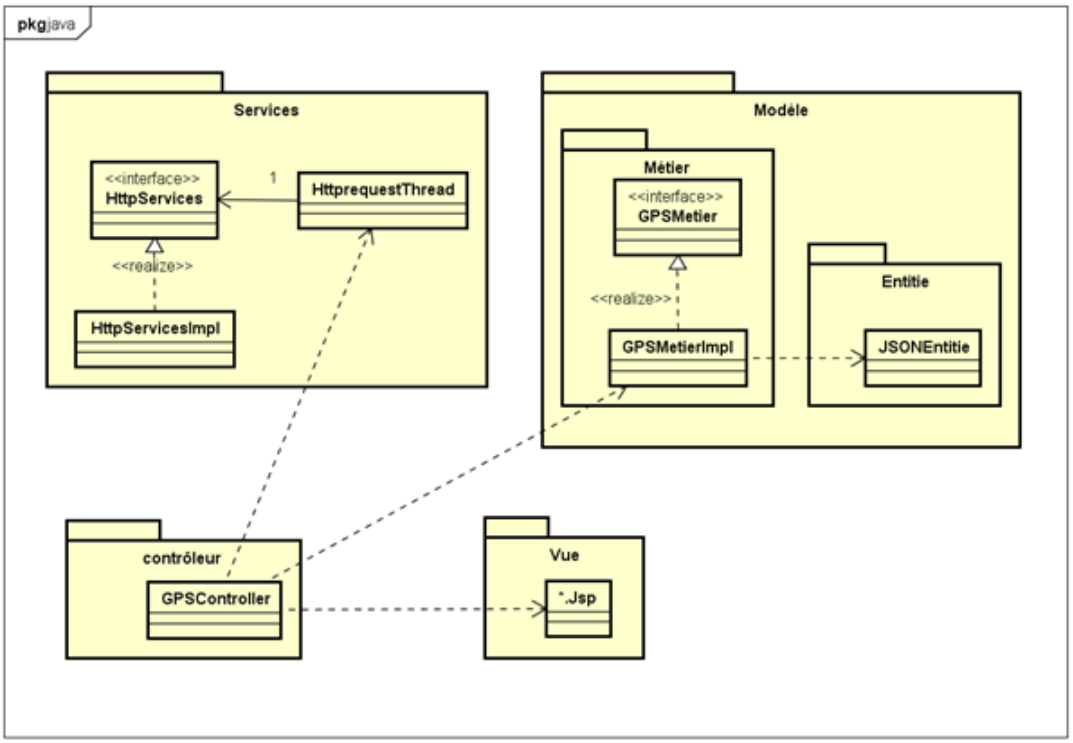

FIGURE 2.7 – Diagramme de paquetage de l'application cliente

Dans le paquetage service, nous trouvons l'interface "HttpService" responsable de la communication avec la l'API Mapquest et le serveur de géolocalisation.

Le paquetage modèle prend en compte les données relatives aux coordonnées GPS et les sauvegardent via le paquetage "Entitie".

Chapitre 3

Réalisation

Dans cette section, nous allons présenter les différents outils utilisés par notre application et le manuel d'utilisation de l'application.

3.1 Outils de travail

Avant que nous puissions faire la spécification et le cahier des charges de Sekai, nous avions commencé par discuter et débattre sur le langage qu'on utilisera majoritairement dans le projet. Grâce à nos expériences professionnelles et académiques, notre choix s'est porté sur le langage Java et son plateforme JEE. Ce choix est dû à la simplicité qu'offre java dans l'organisationnel, dans la gestion d'erreur et aussi à la disponibilité de plusieurs Api et Framework pour ce langage.

A la fin de la spécification du cahier des charges et de la construction des cas d'utilisation, nous avons décidé que nous allons utiliser Html et Css comme langage pour l'implémentation de l'interface graphique. Cette décision se justifie par trois principales raisons :

— Le protocole utilisé pour la communication entre le serveurs et entre le client est le Http. Ainsi il est plus facile de gérer la communication entre le serveur en utilisant du JavaScript et un Framework comme le Jquery.

— Il est plus simple de programmer une interface graphique en utilisant la combinaison Html/CSS/JS.

— Le framework proposé pour le javascript et par OSM (Leaflet) est très simplifié et plus complet.

Plusieurs autres outils ont été adoptés suites aux choix cités précédemment. Ainsi, nous

avons choisi **Apache Tomcat** comme conteneur web libre de servlets et JSP Java EE. Cette décision était une évidence pour nous car nous avions tous déjà programmé de gros projet avec ce conteneur open source. nous avons ensuite utiliser **Leaflet** pour sa simplicité et la taille de sa communauté.

Cela étant fait, il nous restait plus que deux décisions à prendre : l'outil qu'on utilisera pour la sauvegarde des données et l'outil du calcul des itinéraires. A la suite de quelque recherche fait, nous avons adopté **Mapquest** comme Api pour le calcul des itinéraires. Mapquest offre plusieurs services lié au développement d'une application utilisant une map ou à la géolocalisation. Nous trouvons ainsi parmi ces services le calcul d'itinéraires gratuitement si nous ne dépassons pas 15000 requêtes par mois. Au-delà de ce seuil, notre application ne reçoit plus de réponse d'itinéraires. Vu que Sekai est un système distribué de géolocalisation et que le calcul d'itinéraire n'est qu'un outil de tests, nous avons ainsi choisit d'accepter ce contrainte.

Par ailleurs, notre choix pour la sauvegarde s'est porté sur l'utilisation d'une base de données SQL. Ce choix s'est formé suite à notre souhait de rajouter la fonctionnalité historique. Ainsi pour pouvoir faire des sélections compliquées, nous avons du se tourner vers une base de donnée SQL sans pour autant rendre plus complexe notre projet. Ainsi, on avait choisi au début **Sqlite** vu qu'il demande qu'une librairie et un fichier à ajouter. Cependant après l'implémentation de notre serveur et les tests réalisés, nous avons remarqué que notre base de données ne pouvait pas accepter plus de deux clients par seconde sans lancer des exceptions (dû à l'écriture et la lecture du fichier). Ainsi, on a du changer de base de donnée par une autre plus performante et c'est ainsi qu'on a choisi **Mysql**. Pouvant géré jusqu'à 79000 requêtes par seconde, nous avons trouvé Mysql comme le bon compromis pour notre application.

3.2 Mode d'emploi

Cette section explique comment utiliser les sources de notre solution de géolocalisation : Sekai. Nous nous trouvons notamment le manuel d'installation ainsi celui de l'utilisation.

3.2.1 Manuel d'installation

Il existe deux façons pour utiliser Sekai. nous pouvons soit utiliser les machines virtuelles toutes prêtes ou installer les dépendances du projet manuellement. Il faut noter que l'utilisation de la machine virtuelle est très intéressante pour avoir plusieurs serveurs.

L'utilisation de la machine virtuelle

Dans les sources du rendu jointes à ce rapport, vous trouvez le fichier Sekai.ova à importer dans VirtualBox comme le montre la figure 3.1.

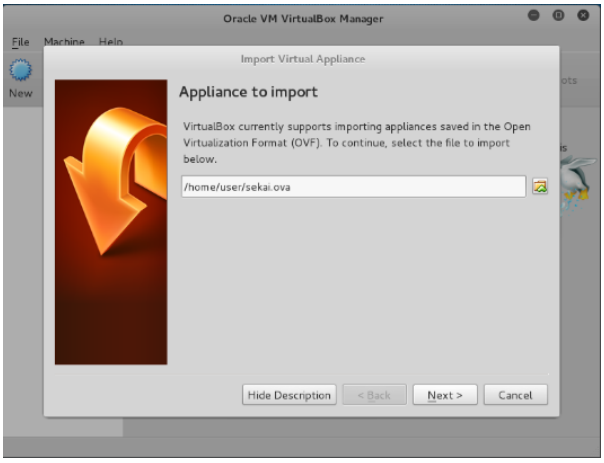

FIGURE 3.1 – Importation de la machine serveur dans VirtualBox

Une fois la machine importée et lancée, il faut configurer l'interface réseau pour pouvoir y accéder depuis l'extérieur comme le montre la figure ci-après.

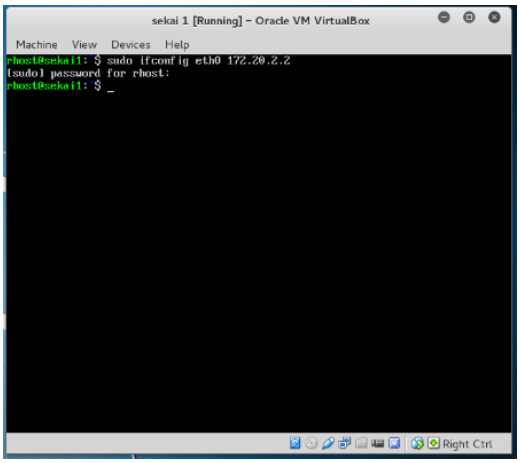

FIGURE 3.2 – Configuration de l'interface communiquant avec l'extérieur

L'installation manuelle

Dans ce qui suit une explication de la façon à déployer le serveur avec des lignes de commandes sur une distribution linux. Il faut avoir le ROOT.war à la racine du dossier actuel et exécuter les commandes suivantes comme administrateur.

```
>& sudo su
    ># apt-get update
    ># apt-get install default-jre
    ># apt-get install tomcat7
    ># apt-get install mysql-server
    ># cp ROOT.war /var/lib/tomcat7/webapps
    ># service tomcat7 start
    ># cp /var/lib/tomcat7/webapps/ROOT/WEB-INF/lib/mysql..in.jar
    /usr/share/tomcat7/lib/
    >#service tomcat7 restart
```

3.2.2 Manuel d'utilisation

Dans cette partie, nous allons présenter le mode d'utilisation de Sekai décrivant l'utilisation des différentes fonctionnalités.

Serveur

Pour lancer le serveur, vous devez lancer les deux services correspondants aux MySQL server et Tomcat server :

```
sudo service mysql start
sudo service tomcat7 start
```

Vous pouvez ainsi tester si le serveur marche en accédant dans un navigateur à l'adresse suivante : « Ip_Serveur :8080/GetMap ». Le résultat qui doit s'afficher est comme dans la figure 3.3.

Pour que la carte s'affiche correctement, vous devez activer le javascript pour cette page. Ainsi dans cette page, on pourra voir tous les clients qui sont présents dans le domaine contrôlé par le serveur comme le montre la figure 3.4.

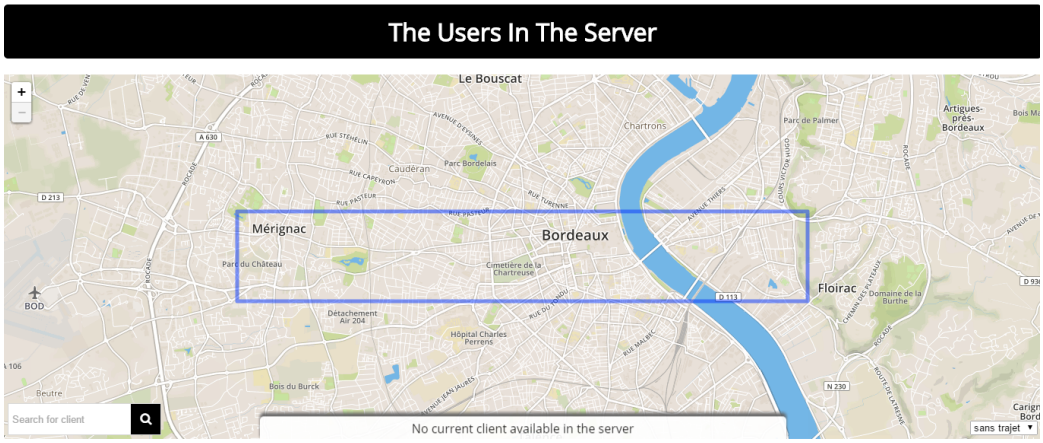

FIGURE 3.3 – Interface de la Map sans clients

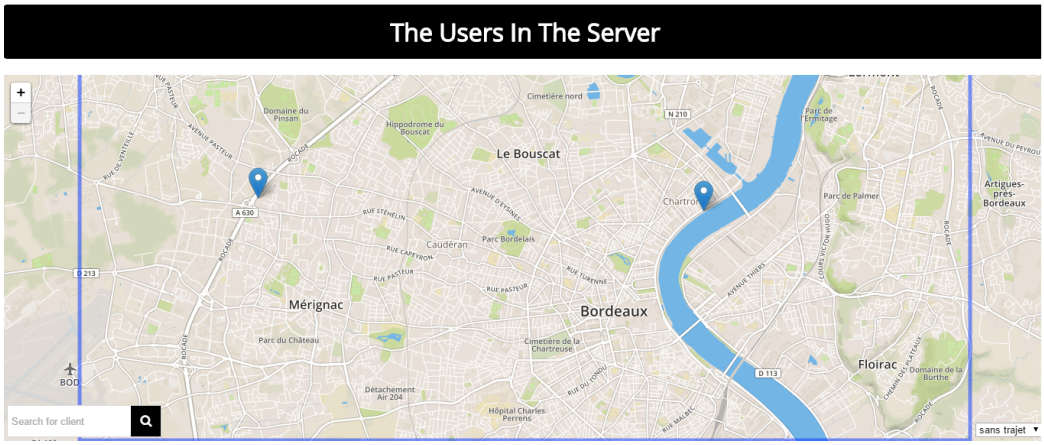

FIGURE 3.4 – Interface de la Map avec clients

Pour pouvoir sélectionner un client précis, il suffit d'appuyer sur la barre de recherche et de composer l'id du client. La fonctionnalité auto completion peut vous aider dans votre saisie. Si l'auto-completion ne vous renvoie aucune suggestion, cela voudrait dire que le client n'est pas dans la base du serveur.

Après avoir sélectionné un client, nous pouvons utiliser ensuite la fonction historique avec le bouton présent en bas à gauche. Ce bouton nous permettra de voir le trajet du client dans le serveur au cours de la journée, ou la veille. Ainsi, on pourrait avoir l'image suivante :

Client

Pour utiliser le client, il faut lancer tomcat sur une autre machine. Ensuite il faut ouvrir un navigateur et accéder à "Ip_Client :9999" tout en activant le javascript. Ainsi, l'interface qui

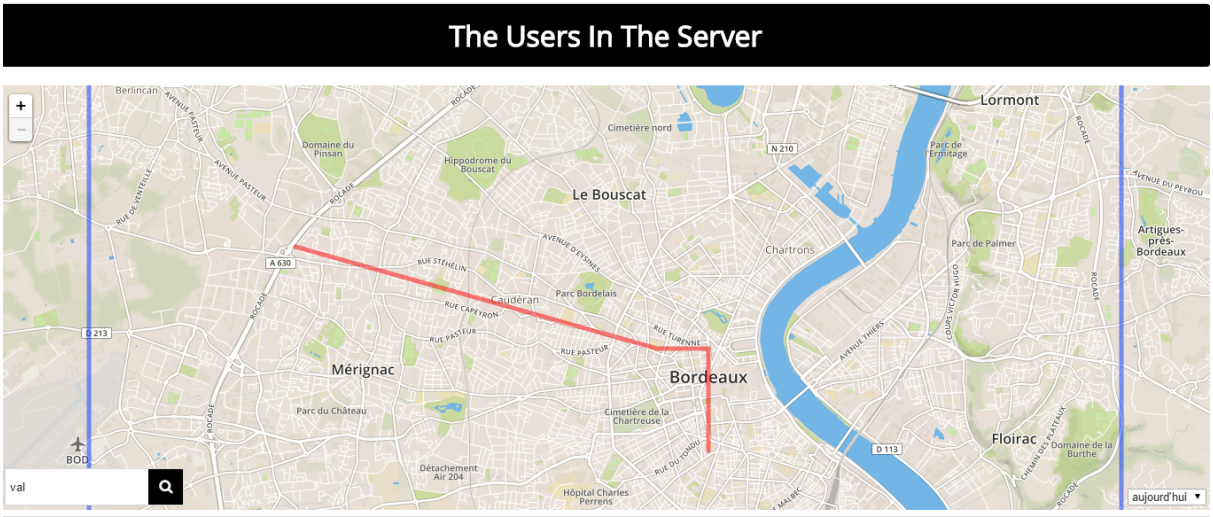

FIGURE 3.5 – Interface affichant l'historique du trajet de l'utilisateur

s'affiche devrait ressembler à celle-ci :

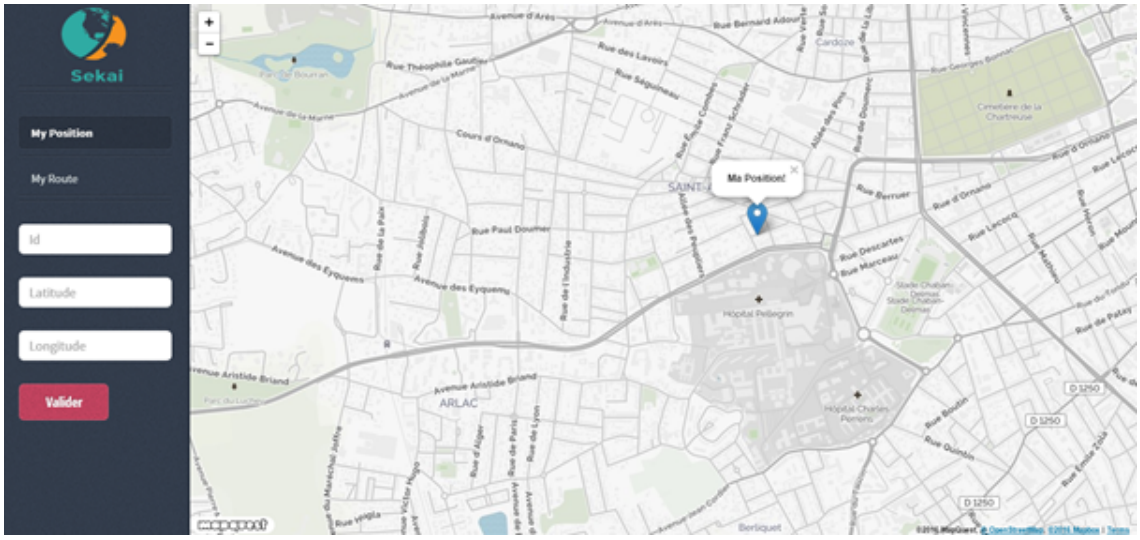

FIGURE 3.6 – Interface de l'application cliente

Il existe deux modes d'utilisations pour le client, le mode normal et le mode simulation de trajet. Pour utiliser le mode normal, il suffit de appuyer sur le bouton **My position** et de remplir le formulaire et appuyer sur valider.

Pour pouvoir utiliser la simulation du trafic, il faut appuyer sur le bouton **My route** et remplir le formulaire avant de appuyer sur valider.

Conclusion

Ce projet s'est révélé très enrichissant dans la mesure où il a consisté en une approche concrète du métier d'ingénieur. En effet, on a dû suivre tous les étapes q'un ingénieur en génie logiciel doit respecter : l'analyse du sujet, la rédaction du cahier des charges, la spécification fonctionnelle, la conception UML, le choix des outils, le développement logiciel et le test du logiciel. De plus, il nous a permis d'appliquer nos connaissances et de les améliorer tout en apprenant à utiliser de nouveaux outils.

Nous sommes aussi satisfaits du résultat car nous avons réussi à développer toutes les fonctionnalités qu'on s'est fixé, notamment : La gestion de plusieurs serveurs ayant des domaines en communs, l'historique, le développement d'interface d'administration et le déploiement.

Néanmoins, Sekai manque aussi plusieurs fonctionnalités nécessaires pour qu'il soie déployé au grand public. nous pouvons citer notamment :

— L'ajout de l'authentification pour l'accès à l'interface d'administration
— Le développement d'application mobile pouvant communiquer avec Sekai
— La gestion de sécurité des communications entre les serveurs