Geoloc Indoor

Projet RICM4 - Polytech Grenoble

Gambro Antoine - Cochinho Louis

2016-2017

SOMMAIRE

Présentation du projet **3**

A. Contexte **3**
B. Fonctionnement de la solution **3**

Conception **3**

A. Architecture générale **3**
B. Diagramme de classe de l'application android **7**

Réalisation **8**

A. Choix techniques **9**
B. Rendu **10**
C. Améliorations **10**

I. Présentation du projet

A. Contexte

Aujourd'hui on utilise la géolocalisation pour se guider d'un point A à un point B en extérieur. Néanmoins la technologie "classique" de localisation est rendue beaucoup moins précise lorsqu'il s'agit de l'utiliser en intérieur. En effet, les signaux GPS sont plus difficiles à capter lorsqu'ils traversent les bâtiments.

Le projet Geoloc Indoor tente de pallier à ce problème en proposant une solution précise de localisation d'objets en intérieur.

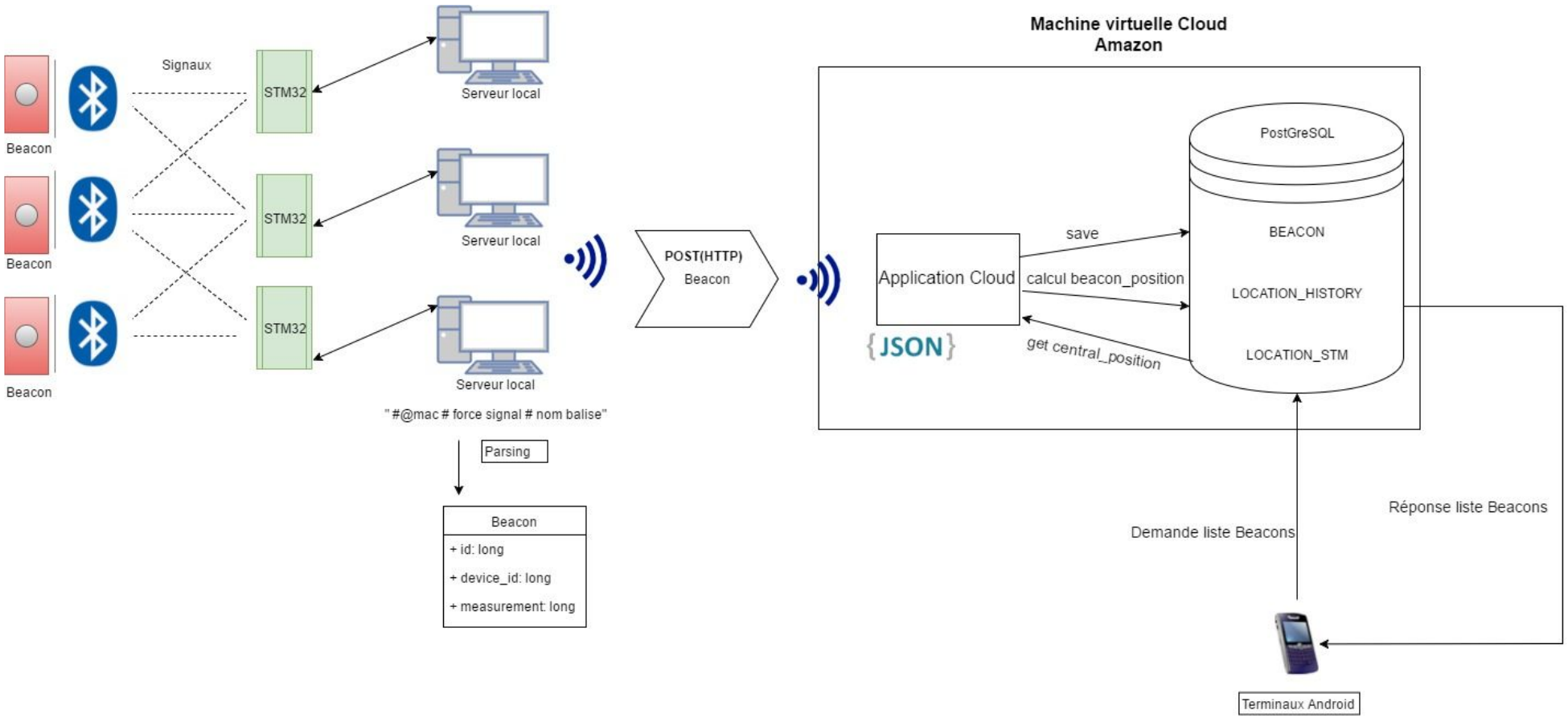
B. Fonctionnement de la solution

Basé sur la technologie Bluetooth Low Energy, la solution Geoloc Indoor repère les objets grâce à des micro contrôleurs placés dans un bâtiment. Ces derniers détectent les signaux BLE émis par les objets, font remonter les informations concernant les objets jusqu'à un serveur qui va les traiter. Les objets vont pouvoir être ensuite visualisés sur une carte via une application mobile.

II. Conception

A. Architecture générale

Ci-dessous un schéma représentant l'architecture "hardware" de la solution Geoloc Indoor.


Beacon
Beacon
Beacon
Signaux
STM32
STM32
STM32
Serveur local
Serveur local
Serveur local
"#@mac # force signal # nom balise"
Parsing
Beacon
+ id: long
+ device_id: long
+ measurement: long
POST(HTTP)
Beacon
Machine virtuelle Cloud
Amazon
Application Cloud
{JSON}
save
calcul beacon_position
get central_position
PostGreSQL
BEACON
LOCATION_HISTORY
LOCATION_STM
Demande liste Beacons
Réponse liste Beacons
Terminaux Android

Les objets à localiser sont représentés en rouge à gauche. Ce sont des beacons. Un beacon est un petit capteur sans fil à basse consommation d'énergie. Lorsqu'il est allumé, il diffuse en continu un signal Bluetooth contenant son identifiant.

Des cartes électroniques de type STM32 équipé d'un adaptateur Bluetooth (Shield BLE) sont capables de détecter ces signaux bluetooth et de mesurer la puissance du signal reçu. Les cartes sont représentées en vert sur le schéma.

Chacune des cartes électroniques font remonter les informations vers un serveur local auquel elle est connectée en filaire. Le rôle de ce serveur local est de transformer(parser) les informations reçues sous la forme d'une chaîne de caractère, en un objet instance de la classe Beacon. Cette chaîne contient :

- L'adresse MAC de la carte électronique
- La puissance du signal
- L'identifiant du beacon

Bien sûr, on retrouvera ces informations dans l'objet Beacon.

Ensuite, l'application locale envoie l'objet Beacon à un autre serveur en ligne. Ce serveur est une machine virtuelle cloud Amazon contenant une application et une base de données. Cette application dans le cloud va recevoir l'objet Beacon et le sauvegarder dans la base de données.

La base de données est composée de 3 tables :

- **Beacon** : Dans cette table sont sauvegardées les beacons envoyés par les serveurs locaux.
- **Location_history** : C'est dans cette table que seront sauvegardées les positions des beacons après calcul de cette position par l'application.
- **Location_stm** : Contient les positions des cartes STM32.

Remarques diverses :

- Quand on parle de position, on parle de couple latitude/longitude.
- Les positions des cartes STM32 sont inscrites dans la base de données par un administrateur système.

Le calcul de la position des beacons est effectué par l'application cloud et utilise l'algorithme de calcul de centre de masse. Pour cela, l'application à besoin de récupérer depuis la base de données :

- La position des cartes STM32
- La puissance des signaux reçus.

Concrètement, le calcul de la position est effectué de la manière suivante :

$$X_{obj} = \frac{\sum_{i=1}^{n} (signalStrength_i * X_{Stmi})}{\sum_{i=1}^{n} signalStrength_i} \qquad Y_{obj} = \frac{\sum_{i=1}^{n} (signalStrength_i * Y_{Stmi})}{\sum_{i=1}^{n} signalStrength_i}$$

- Le couple (Xob,YObj) représente la position de l'objet.
- n est le nombre de carte STM utilisée pour le calcul.
- signalStrength(i) est la puissance du signal reçue par la ième carte STM32.
- Le couple (Xstmi,Ystmi) représente la position de la ième carte STM32.

Lorsqu'on veut localiser précisément un objet, il faut prendre aussi en compte l'étage dans lequel il se trouve. Pour ce faire, le même principe de calcul est appliqué.

Important : Pour que le calcul de la position soit efficace, 3 cartes STM32 minimum sont nécessaires(triangularisation).

Enfin, les beacons sont visualisables sur une carte via une application android. Lorsque l'application est lancée, une requête est envoyée à la base de données pour récupérer une liste de beacons. L'utilisateur va pouvoir choisir quels beacons il veut voir être placés sur la carte avec un rendu de l'intérieur des bâtiments.

L'utilisateur aura aussi la possibilité de se géolocaliser sur la carte.

B. Diagramme de classe de l'application Android

L'application Android est structurée en suivant le modèle Model - View Model - Model(MVVM). Ce patron de conception est dérivé du patron Model - View - Controller. Dans le modèle MVC :

- **Le Modèle :** définit la structure des données et communique avec le serveur.
- **La Vue** : affiche les informations du Modèle et reçoit les actions de l'utilisateur
- **Le Contrôleur :** gère les évènements et la mise à jour de la Vue et du Modèle

L'inconvénient de ce modèle c'est que si le modèle de données est amené à changer, nous devons obligatoirement modifier l'ensemble de nos entités. Le modèle MVVM rajoute une couche ViewModel à notre architecture. Cette couche s'occupe de :

- Présenter les données du Model à la Vue,
- Recevoir les changements de données de la Vue

L'avantage de ce modèle est qu'il nous permet de découpler les différentes parties de l'application en étant capable de la faire évoluer de manière modulaire.

Dans notre diagramme de classe :

- La couche **Services :** contient l'ensemble des méthodes qui vont permettre de communiquer avec la base de données, ainsi que les méthodes de gestion de la connectivité et de la géolocalisation
- La couche **Model :** contient les classes telles qu'elles sont présentes en base de données : Par exemple, la classe BeaconModel contient exactement les mêmes attributs que dans la table Beacon de la base de données. Les méthodes d'obtention de beacon de la couche services renverront un beaconModel.
- La couche **Controller :** Fait appel aux services en contrôlant les données qui y transitent mais aussi pour faire la transformation d'un objet qui arrive de la base de données en un objet présentable pour la vue.
- La couche **ViewModel :** Contient les mêmes classes que la vue Model mais adaptée pour la vue. Par exemple la classe BeaconViewModel va reprendre quelqu'uns des attributs(ceux utiles à afficher)de la classe BeaconModel correspondante avec en plus un objet Geopoint représentant ses coordonnées et un objet Marker que la carte va afficher
- La couche **Vue :** Contient l'activité principale de l'application ainsi la classe MyMap représentant la carte.

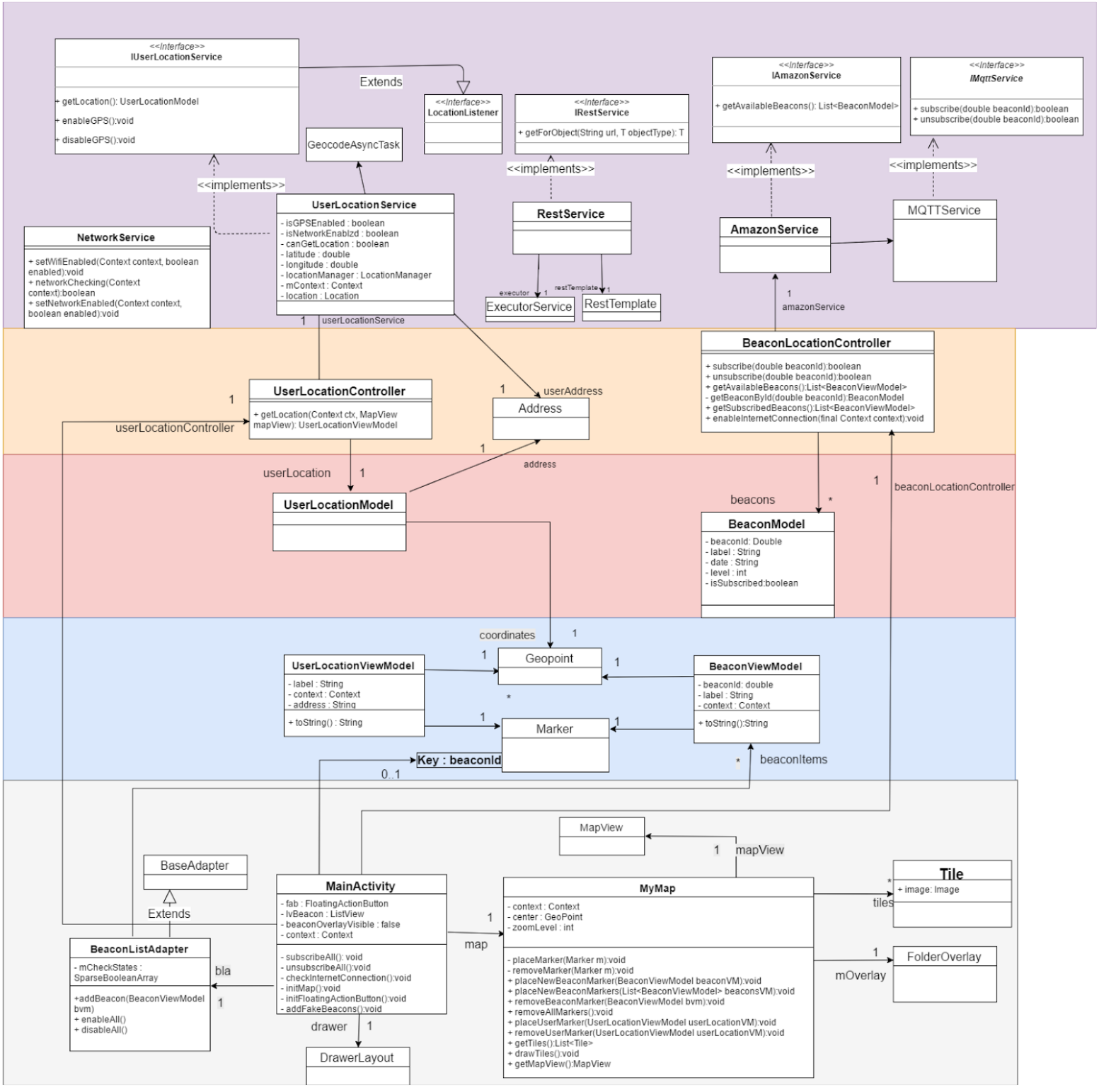
SERVICES

CONTROLLER

MODEL

VIEW MODEL

VIEW



III. Réalisation

A. Choix techniques

Les STM32 utilisés peuvent fonctionner de deux façons possibles :

- Soit il scanne lui même les périphériques(beacons dans notre cas) en émettant une requête de paquet.
- Soit il attend que des paquets de périphériques lui parviennent et réagit à ceux-ci (mode passif).

Pour des raisons de consommation d'énergie, c'est le mode qui est utilisé. Les périphériques émettent donc des paquets bluetooth (paquets d'advertising) et les STM32 réagissent à la détection de ces paquets en les transmettant au serveur local auquel il est connecté en filaire.

Les cartes électroniques Genuino 101 possède un module Bluetooth intégré. Cependant, elles ne supportent pas encore le mode passif. En conséquence nous n'utilisons que des cartes STM32.

L'application qui représente le serveur du projet est faite en Java, en utilisant le framework Spring MVC. Comme dit précédemment, son objectif est de calculer la position du dispositif, de la sauvegarder dans une base de données et de les envoyer aux clients Android.

C'est une machine virtuelle Amazon qui déploie l'application chargée de faire le calcul de la position des beacons et de les sauvegarder en base. La base de données est une base PostGreSQL et est elle-même déployée sur cette machine virtuelle.

Pour que l'application locale détecte les données arrivant sur l'un de ses ports(modélisés par un objet de type SerialPort), un EventListener se charge d'écouter le port auquel chaque carte sera connectée.

L'envoi des objets de type Beacon entre l'application locale et l'application cloud se fait via un Post HTTP d'un objet RestTemplate englobant cet objet beacon.

La requête HTTP de demande de calcul de position est une requête de type GET et renvoie au format JSON le calcul final de la position qui sera sauvegardée dans la table location_history.

Pour configurer plus facilement l'application Java Spring et écrire automatiquement les requêtes SQL, le framework SpringBoot est utilisé.

Pour la connexion à la base de données, c'est JPA et Hibernate qui interviennent.

Maven est utilisé pour pour gérer et automatiser plus facilement l'ensemble des processus de production logiciel en permettant un ajout aisé de toutes les dépendances nécessaires pour le projet dans le fichier pom.xml.

Pour gérer les abonnements du client android aux différents dispositifs, le mode d'envoi d'information choisi est MQTT. MQTT est un protocole de messagerie basé sur le modèle Publish/Subscribe. Ce protocole a pour avantage d'être très léger et de ne consommer que très peu de bande passante. Dans la solution Geoloc Indoor, si les clients Android souhaitent connaître la dernière position mesurée d'un beacon, alors il s'abonne via MQTT à cette dernière.

Enfin, la carte utilisé pour le rendu est une carte OpenStreetMap. La bibliothèque de rendu utilisée est OSMDroid.

B. Rendu

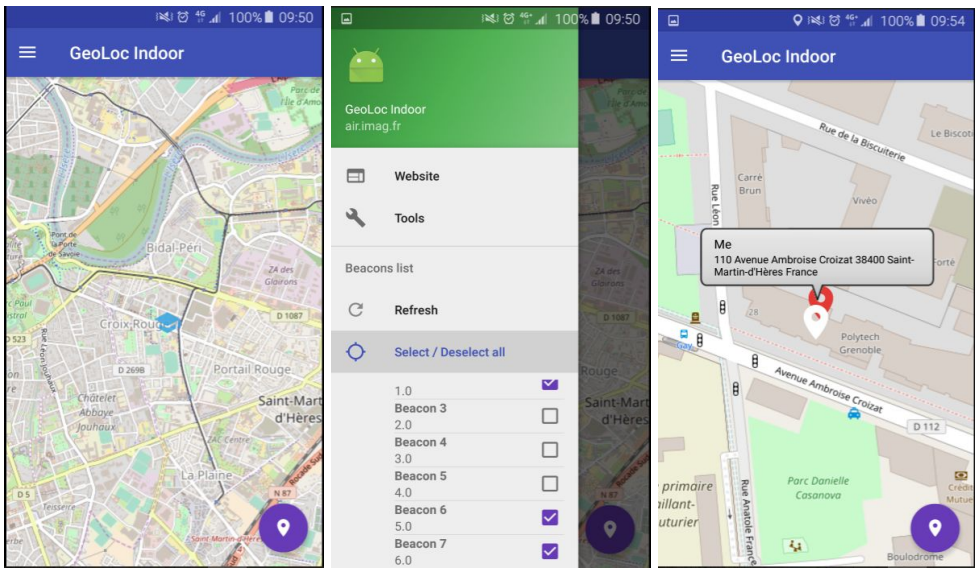
Le client Android a la possibilité de choisir quelle beacon visualiser via une liste de beacon à cocher.

Un bouton flottant permet à l'utilisateur de se géolocaliser.

Un clic sur un marqueur permet de voir les informations sur le beacon associé ainsi que sa position. Pareil pour le marqueur associé à l'utilisateur(une fois qu'il est localisé).

Si la récupération des beacons en base échoue, des faux beacons sont ajoutés sur la carte en guise de test.

Le rendu sur une carte d'intérieur n'est pas encore fonctionnel. Des tuiles représentant les plans des étages de Polytech existent mais elles n'ont pas encore pu être intégrées à la carte.



C. Améliorations

Une première amélioration serait d'intégrer les tuiles à la carte OSM et d'offrir à l'utilisateur la possibilité de visualiser l'étage de Polytech qu'il souhaite (et donc de changer de tuiles en conséquence).

Un premier système de reverse geocoding (trouver une adresse à partir d'une coordonnée) à été mise en place pour l'utilisateur. Une amélioration possible serait de mettre en place ce système aussi pour les beacons en indiquant par exemple le numéro de salle et l'étage.

Il est possible de géolocaliser l'utilisateur (en intérieur) non plus en utilisant le système GPS mais en utilisant la technologie bluetooth de son terminal pour au final de se comporter comme un beacon et être géolocalisé très précisément dans une salle d'un bâtiment comme polytech.

Le système d'abonnement MQTT n'a pas encore été mis en place. Pour implémenter ce système, la librairie Java PAHO est envisagée.