Team 21

iBeacons for personalized Content

Luca Meneguzzi 167932
luca.meneguzzi@hotmail.com
Alberto Stani
stanich91@gmail.com

Abstract

This project is devoted to build a system made by a mobile application (client) and a server, which has the role to send personalized multimedia content to the client accordingly to his position. Basically, the position of the mobile client is estimated using these beacons, that are not more than simple bluetooth antennas that periodically send some information about themselves (uuid). The server is able to receive the client's query, containing the beacon's information, and it has to decide smartly which is the most appropriate content to be delivered to the mobile application.
In order to do so, we properly implement a matching algorithm able to choose between all the contents, that are stored in a proper database, deliberately built for this system. This project has been thought to be applied in a specific contest, that in our case for example was a shop (e.g. SmartShop), and all the multimedia contents have been chosen in order to help the customer. In particular, we decided to send a welcome/goodbye message as soon as he/she enters/leaves the shop, and if we detect that the customer is on the middle of the shop, we send some information that will help him in finding what he's looking for, like for example a detailed map of the shop (in general case it could be quite big). Another case could be when the customer is near a particular zone, and we decided to send information about products and even some special offers reserved to him; this of course could motivate the usage of the app (SmartShopApp) and even increase the visits in the shop!
Of course, this system could be quickly adopted to any kind of environment, either changing the database and/or the matching algorithm (specifying new rules).

1. Objectives

The main objectives of this project were basically two: the implementation of the mobile application and the implementation of the server one.
In particular, regarding the mobile application, we had to properly detect the signals sent by the beacons, and to extract the information we needed. Secondly, the mobile application has to send a query (http) to the server, containing the detected information, and it has to be able to receive and display the multimedia contents in the proper way. In our case, the multimedia contents are images, audios and videos, and we decided to send also combination of images and audios.
With respect to the implementation of the server application, our work has been mainly focused on 3 points:

- ***a)*** the building and research in the database.
- ***b)*** the selection of the proper contents .
- ***c)*** the delivery of the contents to the end user in proximity of the beacons

2. Achievements

We managed to make the all system works, which means that we are able to detect the beacons' s presence with the mobile app, to send the query to the server and to display/ play the received contents on the mobile phone. With respect to the server application, it's able to handle the client request, to make a query to the database containing the content and to convert them in the proper format for the mobile application.
Always in relation to the server, we can handle properly the contents stored in the database adding or removing easily some of them, thanks to the web page that we built, which allows us to upload the multimedia file using the browser.

We decided to implement all the request using HTTP protocol, because we retained it the simplest and best choice for the transmission of any kind of message over the network (at least regarding the messages that we needed to send). HTTP is supported by client & server in any language, it's used by the browser (really helpful in the test & debug phase, which allowed us to analyze in detail all the messages sent by the server). HTTP allows you to use a standard way of referencing resources, parsers already exist for every language and their semantics are well understood, which was really helpful for uploading the multimedia contents using browser. In addition, it could be possible to improve the security, using HTTPS.

The server has been built using nodejs, a great platform for running server applications in javascript. Even for this choice, there are many motivations; javascript is very easy to understand and really practical, it can easily manage JSON data and HTTP request , and this helped us a lot in preparing the proper response to the mobile app. In addition to this, node js has a proper packet manager (npm) which does a great job at specifying and installing the project dependencies. Moreover, we wanted to upload our server application on the web, and nodejs allows as to do this exploiting heroku, which is a platform as a service that enables developers to build and run applications entirely in the cloud. Last but not least, it handles the request to object database like Mongo, which we had to use.

In fact, we built our database using *MongoDB*, which is data structured, flexible and highly scalable, therefore it's really easy to expand or modify our database, in order to adapt the hole system to different environments.

With respect to the mobile application, we build it using Titanium as development environment, because we have been suggested that it's easier for what regards the graphical interface together with the beacons detection. For this last objective, we used a module called *com.liferay.beacons*, that has only to be included in the titanium project folder.

We used 2 beacons of *Kontakt-io*, which basically transmit the information of uuid associated to a specific region, that could be the same for all the beacons in a small area (e.g. the Smart Shop), together with others numbers called major /minor. We decided to use the uuid and major information in order to distinguish between them.

The following diagram shows how the system works, in a high level view, and it will be better explained in the next chapte.

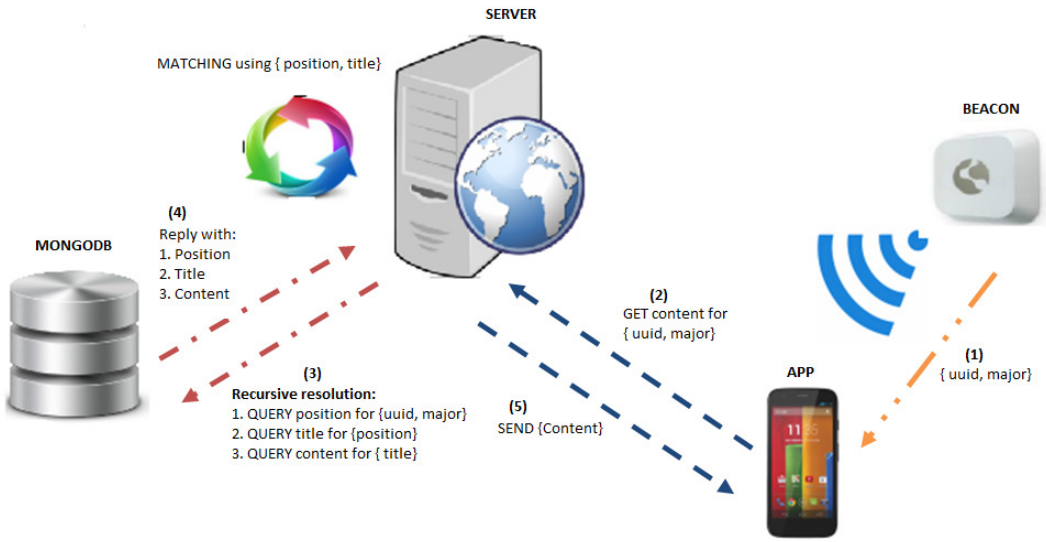


Figure 1*: High level view of the system*

Basically, when the mobile application detects the beacon' s signal, it extracts the uuid and major information, and accordingly to the received signal strength index (*rssi*), it's able to estimate the distance from the beacons. For our purpose, we designed 5 different positions that the mobile application can detect: enter in range of the SmartShop beacons, exit it, being in the middle between 2 beacons, or being close to one of them. Depending on this different situations, the mobile application makes different query to the server, communicating it the received information by the beacons.

Then, everything is handled by the server, which is designated to analyze the client's request and to understand its the position given the information (for example the same server could handle different SmartShop, that means receiving different uuid/major information).
Once discovered the position associated to that uuid/major info, the server makes different iterative queries to the database, and eventually it selects one (or even many) content, that must be delivered back to the mobile application.

At this point, the client has only to understand which kind of object it has received (image /audio/video) and it has to properly display it in the case of images and videos, or to playback the audio. Together with the multimedia file, we decided to send also some messages to the customer, like a welcome title and some particular sentences (e.g. “Take a look to our products!” or “Hey, this is a special offer reserved only for you!”).

For the way in which we structured the system and the database, it's really simple to extend it to many different environment and applications, it's just needed to change few things in the mobile and server application in order to track different beacons (even more than 2) and everything can be adopted building an ad hoc database.

A particular attention must be paid for the beacons installation in the environment, because we noticed that different beacons use different power (probably due to the battery level), and this could lead to some issues in detecting their presence by the mobile application. In fact, it's not so easy to understand where is the range in which the customer is actually able to receive the wanted contents, therefore a precise test on the field is needed.

In order to run the server, it' s just needed to be placed in the folder containing the software and run the command: *node index.js* (NodeJs required with express, body-parser and multer).
We put an online database on *mongolab*, so it's not even needed to have a local database running.

It's also possible to access to the online server through this link (*heroku*):

http://ibeaconsteam21.herokuapp.com/

Both the local and online servers can be managed through browser (Chrome is recommended in order to see all the contents, with particular attention to the audio ones), accessing to the web page that we created.

The mobile application can be installed using the *apk (ibeacons.apk)*, that has been built for *android 4.4.4* (also works for 4.4.2), and we tested it on a *Motorola moto G.*

3. Project Implementation

As previously mentioned this project is divided in two parts:

a) a mobile application, through which the end-user can go around in the shop listening audio and visualizing offers and products on his smartphone;

b) a server application that handles the request (*http get),* made by the users, sends a query to the mongo db and chooses the proper content to send back to the client, given a specific position.

Everyone entering in a SmartShop with his smartphone, gifted with SmartShopApp, can have an augmented shop experience! The phone starts scanning for the beacon's signal (using Bluetooth 4.0), and it's able to estimate the relative position of the user in the shop, accordingly to the beacons that it's able to see (distinguished using uuid and major) and to the received signal strength index (*rssi*). The combination of those information gives the possibility to define different type of regions inside the shop:

- ENTRANCE REGION: first beacon found → Welcome in SmartShop
- MACRO REGION: many distant beacons → You are in a wide area (ex "Center")
- MICRO REGION: one near beacon (uuid+major) → You are infront of a specific shelf
- EXIT REGION: last beacon found → Goodbye and thanks

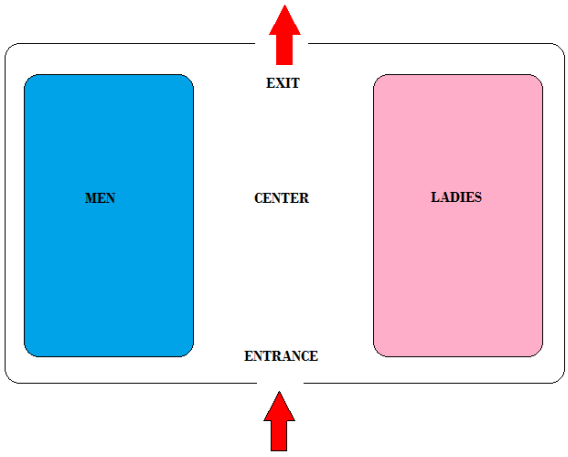


Figure 2*: Example of a SmartShop and its related regions*

Mobile Application: Beacons Detection

When SmartShopApp finds one or more beacons, it makes a request (http get) to the server passing it information about the specific region where it is (the information are contained in the url, e.g. "*/position/entered=region A*"). We decided to use a get request (instead of post) because in this case, the parameter to be passed it's just a string, thus a get is easier to be implemented, and in addition in this way it's possible to test the system using a simple browser.
In order not to stress the user to much, our application keeps in memory the positions for which the content has already been received and shown, otherwise every time it detected one regions, it would ask the related content to the server, even if it has already been required few seconds before.
In the standard case, our application is monitoring for both the beacons, in order to detect the case (more complex) in which the client is in the middle between both of them; we also implemented a special function which allows to detect only the proximity to the beacon, without considering the other one.
This function can be activated in the app with a check box, and basically sends a query only when the customer is quite near to one of the tracked beacons.

Server Application

Once received the request, the server processes everything (the matching algorithm is explained later) and sends as response an '*application/JSON*' containing some information about the multimedia content to be shown by the mobile application: {content type, title, message to the customer, link to the real content}. The content type and the link are used by the app to make the second and last query: the download of the specific content and it's correct reproduction on the base of the type previously passed. In this way, the smartphone can visualize/reproduce the content in a specific window correctly sized and with all the control for audio and video. SmartShopApp has also the possibility to handle at the same time two different content like showing an image while playing a music track in background. The server is waiting for a request from the mobile app, and when it arrives it takes the parameters contained in the url and uses them to make a queries on Mongo.

Mongodb

Mongo is a no sequel database in which is possible to insert some records in json document format, to make query in order to find a particular record and to group the similar documents in a so called collection. In a first time we created, populated and used a Mongo db on mongolab: a fully managed cloud database service that hosts MongoDb databases. In addition, the host service mongolab provides the possibility to visualize and edit dbs, collections and records from browser in a very simple way.

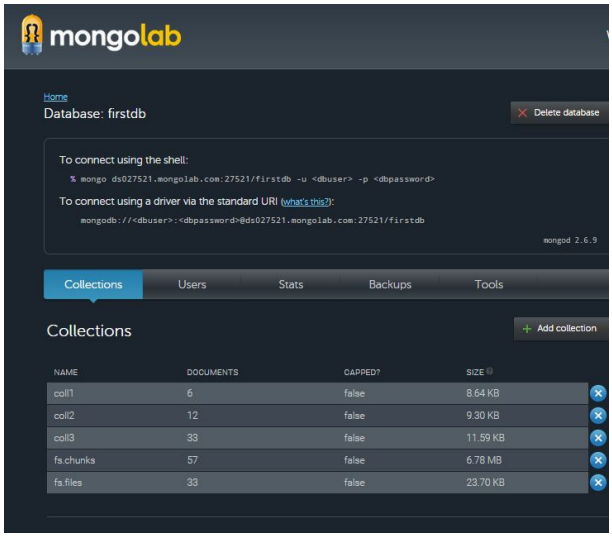


Figure 3: *Our database on Mongolab*

Afterwards we have transferred the database in a local VPS created by MongoDb on the same machine in which runs our server application.
In this way is possible test the entire system without any internet connection, with more reliability and in real-time (the connection to a online mongolab db introduces an high delay).
For this project we have created a db called Fisrtdb in which we have inserted 4 collections:

- coll1: MATCHING UUID-POSITION
- coll2: MATCHING POSITION-TITLES
- coll3: MATCHING TITLE-CONTENTS
- fs: divided in fs.chunks and fs.files containing the MULTIMEDIA CONTENTS file in bson format divided in chunks of a fixed length; in this way is possible to load big files in many smaller ones

MongoDB manages data as documents, with a maximum size of 16MB. GridFS is a specification that manages large files and their associated metadata as a group of small files. When you query for a large file, GridFS automatically reassembles the smaller files into the original large file, like explained in following scheme (***Figure 4***):

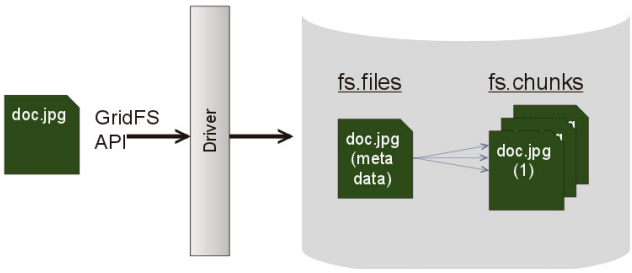


Figure 4: *Storing multimedia files into mogoDB using GridFS*

For a given detected beacon, the resolution to the specific content follow these steps, in particular each step is a query to the database:

1. MATCHING UUID-POSITION: given the region found by the mobile app we make a query on Firstdb "coll1", it return a record containing the searched region and the associated position.

   Example: Entered the shop --> GET "*position/entered=region A";*
   Query to coll1: find{ *entered=region A* } gives {*entrance}*;

2. MATCHING POSITION-TITLES: given the position, the server makes a query on the "coll2" that returns an array of records with all the titles associated to the position. *The title is the message typology that we want to send to the user* (for example enter –> good morning , good evening,..)

   Example:
   Given *entrance,* query to coll2: find{*entrance*} gives [{*good morning},{ good afternoon},…*]

   * there are different criteria used in order to choose the proper title between the ones associated to the same position, and they depend on the particular position in analysis:

   ✓ if the customer has entered/exited the shop, we choose the title in respect to the current time, for example if it's morning, the good morning title will be selected.

   ✓ if the customer is in a micro region, we generally pick up the products title, that will contain all the information about the products in the nearby of that region. We decided to select instead the offers title only if that particular query is the fifth (this number can be set changing the constant value), that basically means that every 5 customers who received some contents about the products, the system will select a special offer, available only for that lucky customer!

3. MATCHING TITLE-CONTENTS: given a title the server sends a query on the "coll3" , and obtains the list of contents associated, where we can have many different type of content.

   Example: Given {*good morning*}, query to coll3 gives [{*image1*},{*image2*},{*audio1*}..]

   * if the list contains only one type of content we choose randomly between them, otherwise, if both audios and images are available, we send both simultaneously. In the case in which there are all the 3 types we make a random choice between video and image + audio.

Content Reception

In this way we have obtained the selected content which can be finally sent to the SmartShopApp in a proper json object, that will contain all the information needed in order to download and play/display the file. Actually, this object has the link that points directly back to the server, and contains the content' id; this is needed to make the query to mongo, and after having recombined all the chunks it is possible to send back the final file to the smartphone.
Therefore, in relation with the previous examples, it could be that when the customer is in the entrance, it's smartphone receives *image1+audio1*, with a message that says: "good morning dead customer, welcome in our shop!"

Figure 5 describes a detailed example of an hole query made by the client, with all the steps used for the resolution:

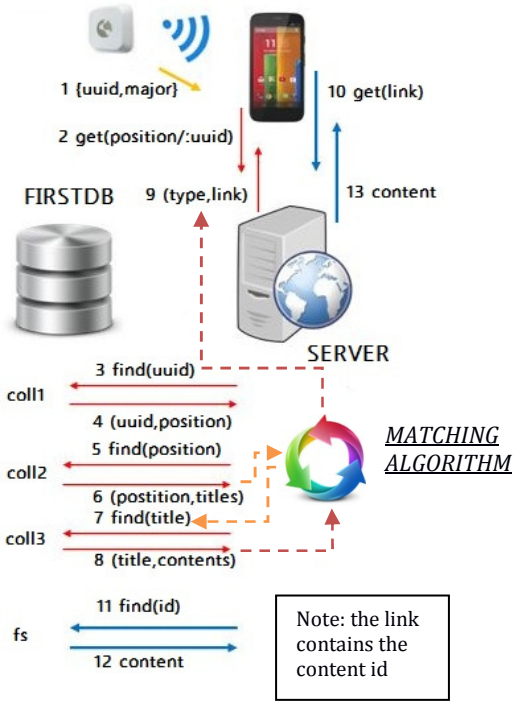


Figure 5 *Detailed example of the total resolution of a client' s query*

Admin Functions

Other different functions have been implemented on the server, and are useful to handle the records; For all this functions we have a web page which helps the SmartShop admin to manage the server and the database with few simple commands , without the necessity to known the code structure and many specific instructions. Almost all these funtions use an *http post* in order to send the wanted parameters!

POSITION/ALL: visualizes (in text format) all the uuid-position matching present in the database.

POSITION/ADD: adds a new specific uuid-position record in the "coll1"

POSITION/REMOVE: removes the specified uuid-position record from the "coll1"

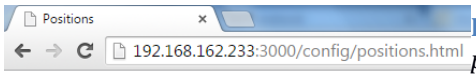
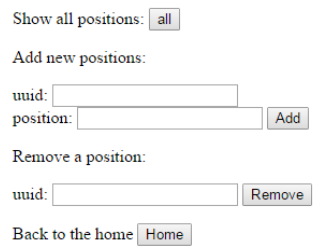


Figure 6 *Page for configuration of positions*



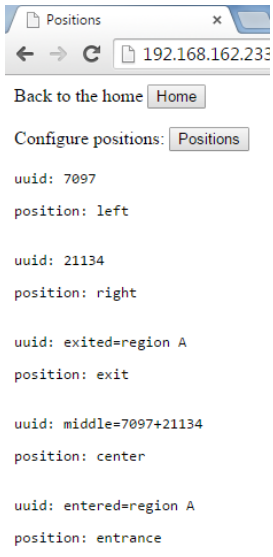


Figure 7: *Page that shows all the positions*

CONTENT/ADD: adds a new content in the "fs" collection, insert it's id and type (automatically extracted) and the specified title in a new record in the "coll3"; if it is a new title for the specified position, add a new position-titles matching in the "coll2".
Note that the uploaded file is checked by the server (Figure 9)

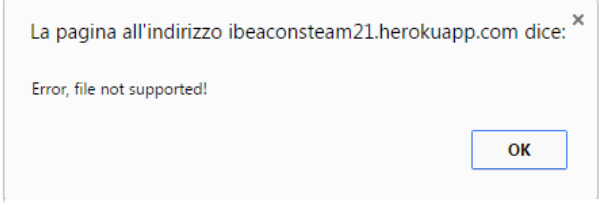


Figure 9: *control of the uploaded file*

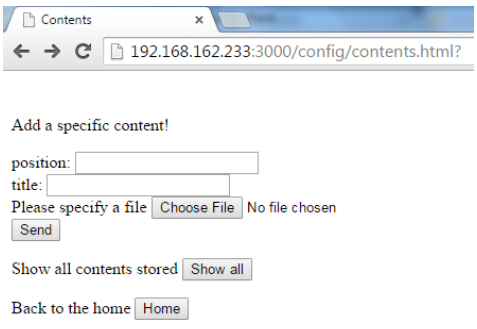


Figure 8: *Page for configuration of contents*

CONTENT/REMOVE: removes content by id, it will be deleted from the "fs" collection and if it is the last content for its title, this one is also removed from "coll2".

CONTENT/ALL: allows the visualization of all the images, audios and videos divided by position and title.

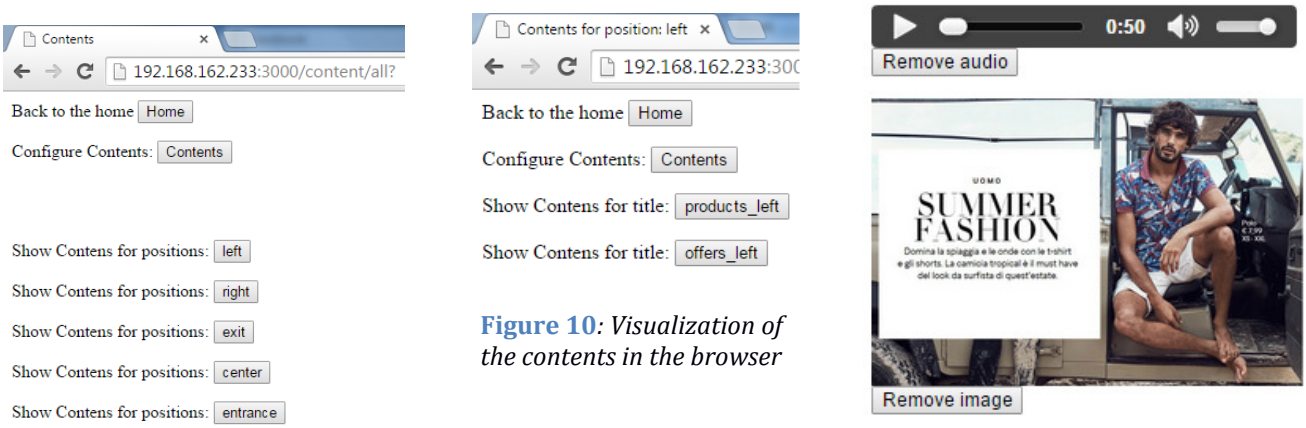


Figure 10: *Visualization of the contents in the browser*

SETTINGS: gives the possibility to set the server ip and port, used to send the proper link to the application (ex: "http//ip:port/contentimage/id_image");

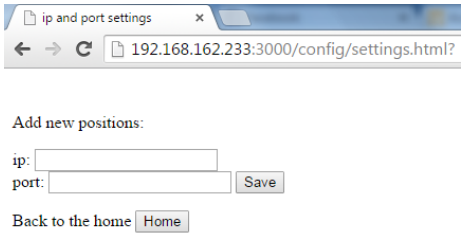


Figure 11: *Page fot setting the ip and port (used to generate the link)*

In Figure 12 there's an example of the content showed in the mobile application, when the customer enters the shop:

Figure 12: *screenshot of the mobile application*

4. Conclusions

SmartShopApp is only at the beginning! It could be extended in many directions..!
There are many real situations in which our application can be useful: in a shop like we represented, in a museum, in a park or maybe in a city to help the visitors to understand what they have around. In a specific scenario, with the possibility to use many i-beacons, this app gives the base to create an improved content transmission system; it's sufficient to have an ad-hoc created database filled with selected contents.

Of course this system could be improved, for example it would be possible to add a login service for a better personalization of the information, or to increase the system responsivity using a better host service, for example the payment version of mongoDB.

Another improvement could regard the beacons scanning, in fact, now the mobile application is scanning for a specific beacon uuid, set during the building; it would be possible to add a specific function capable of downloading from the server the hole list of the uuid to monitor. This of course would make the system even more scalable.

We also noticed that using the online server, the response time is quite big, it means that in order to display an image while playing an audio, it could takes also 5 seconds after having detected the beacon. Of course it's related to the internet connection, we tested it in our apartment where the bandwidth is very limited: less than 3 Mbit/s in download and just few kbit/s for the upload! Even using the database and the server in the local lan, it could takes more than 2 seconds to download an image + audio, if their dimensions are bigger then 200kHz. In order to solve this problem we downsampled all the images of the database, which doesn't lead to an hard degradation of the quality, since the size of the display is very small.

Also the interactivity with the users could be increased using precise information about the scene where the customer is, sending to him objects description and even interactive contents, rendering his experience a fast, complete, specific, smart, pretty and personalized total immersion in the surrounding environment !!

5. References

https://github.com/jamesfalkner/liferay-android-beacons/blob/master/documentation/index.md

https://www.heroku.com/

https://nodejs.org/

https://www.mongodb.org/

https://mongolab.com/

http://docs.appcelerator.com/titanium/3.0/

http://developer.android.com/tools/help/adb.html

http://developer.android.com/tools/help/sdk-manager.html

http://docs.mongodb.org/manual/core/gridfs/