USiBeacon

Ciani Andrea, D'Avico Simone, Gili Daniele

July 2, 2015

Contents

1 iBeacon Technology

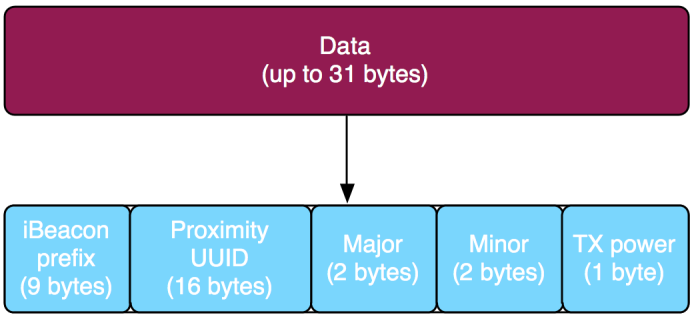
The project is heavily based on the iBeacon technology, a protocol standardised by Apple and introduced in 2013. The protocol is usually exploited in the manufacturing of a class of Bluetooth low energy devices typically called *beacons*.

Beacons use Bluetooth low energy to transmit a universally unique identifier that can be used to determine the beacon's physical location, in order to trigger a location-based action (such as approximating indoor position). The technology is already being successfully used for applications such as mobile commerce (offering customers special deals though mobile marketing) and can enable mobile payments through point of sale systems.

iBeacon differs from some other location-based technologies as the broadcasting device is only a 1-way transmitter to the receiving smartphone or receiving device, and necessitates a specific app installed on the device to interact with the beacons. This ensures that beacons cannot track users against their will as they passively walk around them.

1.1 Reading the iBeacon signal

At the most simple form, an iBeacon emits advertisements following a strict format, that being an Apple defined iBeacon **prefix**, followed by a variable **UUID**, and a **major**, **minor** pair. The last field of the advertisement is called **TX Power**.



- The **UUID** usually identifies the owner of the beacons (be it a company, or a store);
- The **major** number is used to group a related set of beacons. In our implementation, each beacon in the same classroom has the same major number;
- The **minor** number is used to identify individual beacons;
- The **TX Power** field is the strength of the signal measured at 1 meter from the device, and can be used to determine the distance from the beacon.

iOS provides built in capabilities to read the fields of an iBeacon advertisement through its CoreLocationFramework.

2 Indoor Localisation

Wikipedia provides a very good introduction about what the Indoor Localisation is and related problem:

"An indoor positioning system (IPS) is a solution to locate objects or people inside a building using radio waves, magnetic fields, acoustic signals, or other sensory information collected by mobile devices.There is currently no de facto standard for an IPS systems design. Nevertheless, there are several commercial systems on the market.

Instead of using satellites, IPS solutions rely on different technologies, including distance measurement to nearby anchor nodes (nodes with known positions, e.g., WiFi access points), magnetic positioning, dead reckoning. They either actively locate mobile devices and tags or provide ambient location or environmental context for devices to get sensed. The localized nature of an IPS has resulted in design fragmentation, with systems making use of various optical,radio, or even acoustic technologies.

System designs must take into account that at least three independent measurements are needed to unambiguously find a location (see trilateration). For smoothing to compensate for stochastic (unpredictable) errors there must be a sound method for reducing the error budget significantly. The system might include information from other systems to cope for physical ambiguity and to enable error compensation."

Developing the indoor localisation for our system we faced exactly the problems described in Wikipedia.

2.1 Our first implementation

We started developing our personal solution using only one beacon, positioned in the center of the room. Then we measured the power loss depending to the distance to the beacon (in terms of db, decibels). When we found the average signal quality a device has inside and outside a specific room we could "guess" where it was. If, for instance, a dice has a signal strength between 5 and 15 db when it is inside the room, then if the signals is less the 5 the device *should* be outside.

The problem about this first implementation was the hight imprecision on the edge of the room. It was quite impossible to distinguish about a device that is close to the door with lots if people inside the room, respect to one that is outside the room but with an empty lecture room.

2.2 Our second implementation

The second implementation we tried was using multiple beacons. In that case we had more information about the position and we could better guess about the position of the student respect to the lecture room.

In this implementation we tried to use less beacon possible for a triangulation, that means: three. The idea behind this solution is quite simple and intuitive. We position three beacons inside the room and by calculating the distance the device has respect to each of them we can try to figure out where it is inside the room.

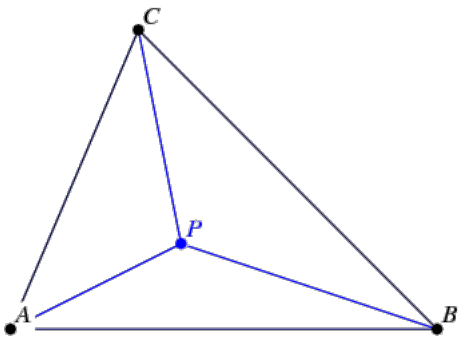


Figure 1: The triangulation idea.

Following the algorithm from this article, and this StackOverflow discussion the solution could be implemented in a few lines of code. Unfortunately, even with this solution we had lots of troubles about the precision of the localisation. It wasn't easy to determine with enough precision the position of a device close to the entering/exit doors and how much a user is close to each wall (maybe someone is trying to "cheat" sticking the device close to a wall). From this idea we thought that, since every room has four walls, the best idea was to triangulate the position using not three beacons but four of them.

2.3 Our third implementation

Using four beacons (one for each wall of the room) increased a lot the localisation precision with this last implementation. By positioning a beacon in the center of each wall we could triangulate better the final position.

Even if we improved a lot the quality of the tracking there still were mainly two issues:

- In non-controlled environments, where you can find metals, and other objects that affect the signal, the received signal strength of the beacons changes so often that it seems impossible to get error range below 5 meters.
- Depending on the way that the user is handling the receiver device, the readings can change a lot as well. If the user puts his/her hand over the bluetooth antenna, then the algorithm will have low signals as input, and thus the beacons will supposed to be very far from the device. See this image to see the precise location of the Bluetooth antenna.

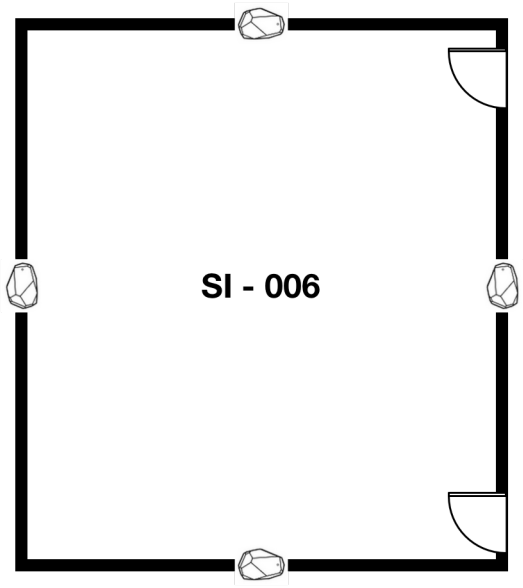


Figure 2: The final result with a triangulation done using three beacons.

2.4 Our final implementation

Since we moved closer and closer to the implementation adopted by Estimote we concluded that the best solution was to move to that implementation using the Estimote Frameworks (see 3.1), available for free in the Estimote website. It uses the same idea we had in the third implementation but adds more then a year of refinement about error correction, wireless noise detection and reduction, as well as the possibility to map the position of each beacon respect to the others by counting the distance between them using an ad-hoc app developed by the Estimote. We thought about implementing the same solution ourselves,

but the error correcting algorithm and heuristics were nontrivial (and also, not everything is publicly disclosed).

3 Estimote Framework and Hardware

Estimote is a polish company that develops small wireless sensor implementing the iBeacon **??** technology providing along the sensors an effective and reliable framework for mobile devices like iOS and Android[1].

Estimote Beacons are so small and lightweight that you can attach them to any location or object. They broadcast tiny radio signals which your smartphone can receive and interpret, unlocking micro-location and contextual awareness.

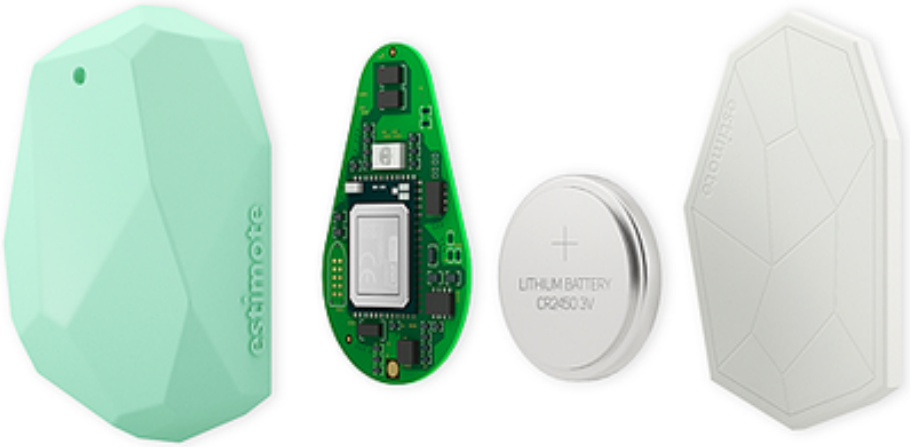


Figure 3: An Estimote sticker cutaway

With the Estimote SDK, apps on the smartphone are able to understand their proximity to nearby locations and objects, recognising their type, ownership, approximate location, temperature and motion.

We used these capabilities in USiBeacon to develop a solution that localise a user in an indoor environment and recognise when he/she is inside or outside a specific region (the lecture's room).

[1] At this time the Android framework is not been released yet

3.1 The framework

The Estimote framework is build on the Core Location framework made by Apple and updated on 2014 to support the iBeacon **??** technology.

Estimote develop this framework to wrap the Apple standard indoor functionality and extend the power of localisation by adding the support to the Estimote stickers and their functionality (such as the temperature, the signal power, the identifiers and the connection to the Estimote Cloud).

One of the main feature the Estimote engineers focused on is the accuracy of the indoor localisation. The frameworks provides additional support for developers that want to implement their custom localisation in the software, helping to avoid the problems that come from signals noise, peoples in the room, nearby wireless waves etc... (see 2). Even if there aren't much information about *how* Estimote is able to improve the quality of the localisation, it seems that they choose an higher frequency for the signal broadcast (from 1Hz specified by Apple in the iBeacon protocol to 5Hz) and a different refresh rate for their devices (less that 1s, maximum limit imposed by the iBeacon). For these reason a proprietary framework between the application and the Estimote devices is quite mandatory.

4 The iPhone Application

The two main important aspect about our project was **how well the system works** and **how much easy it is to use**. For this second reason we chose to develop an application that was easy to use.

In order to create a mobile application that is easy to use for everyone it must focus on the final purpose, that in our case is to permit the users to check in in the lectures. That's why we create as many views as possible, letting the application do the rest.
We can split the behaviours manly in two set:

1. The single login operation (stateless)
2. The set of actions the user can do when he/she is *not* inside a lecture room (**browse operations**)
3. The set of action the user can do the he/she is inside the lecture room (**checkin operations**)

4.1 The technical operation

Of course since the application is meant to be linked to a single user is mandatory provide an authentication mechanism. Since USiBeacon is also meant to work closely with the iCorsi platform we chose to use the same user account of iCorsi also for the USiBeacon platform.

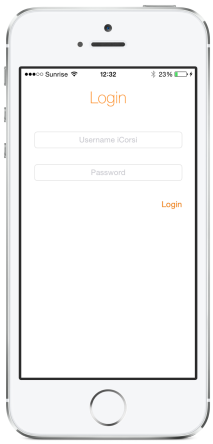


Figure 4: The login view of USiBeacon

From then, the first technical step the user need to do is to **login** to the platform. We tried to keep the step as little as possible, saving the credential locally in the device once the login is successful.

4.2 Outside behaviours

The operations permitted to the user when he/she is outside the lecture room are the following:

- Login (stateless)
- Search for the upcoming lectures
- Choose the lectures he/she is going to attend
- Start the position tracking
- Chose a different upcoming lecture

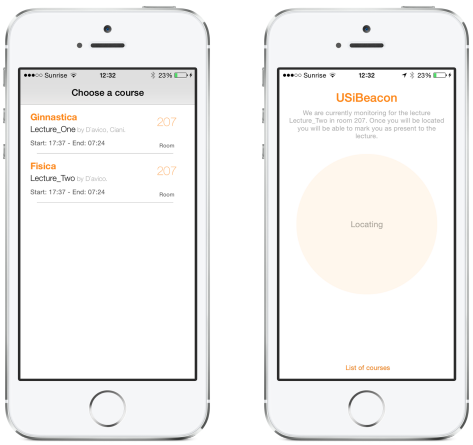


Figure 5: The course selection (left) and the searching view (right)

We chose to implement all these operation in a way that was as fast as possible: when the users open the application (and the are logged in) it automatically retrieves the list of the upcoming lectures that user should attend **within a time window of 15 minutes**. If there is only one result the application will automatically go in "tracking mode", starting to search for the room associated to that lecture, if instead the user has some overlaps (so there are multiple upcoming lectures) the application shows the course selection view

(image 5 on the left) that permits the user to choose for which lecture start the tracking.

The last point (chose a different lecture) is achieved by a little button on the button of the searching view (image 5 on the right). This button permits to open again the course selection and choose a new lecture to track.

4.3 Inside operation

When the user move inside the room the application changes the behaviours permitting the user to **check in** to the lecture.

The application modifies the tracking view showing the checkin button as shown in figure 6:

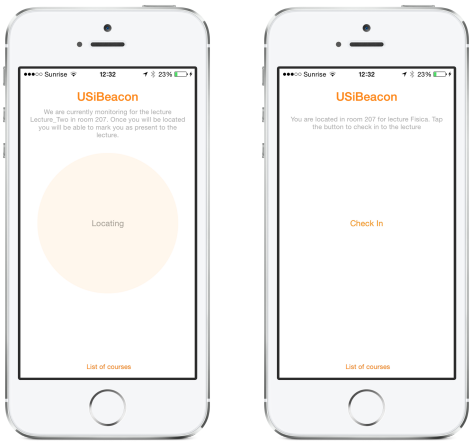


Figure 6: The tracking view outside the room (left) and inside the tracked room (right)

Once the user tap on the *check in* button the application replies with the result of the checkin process that could be one of the following:

- **Successful:** The checkin process is concluded and the user is correctly signed to the lecture.
- **Already Signed:** If the user tries to sign again to the same lecture a warning is visualised.
- **Credential Error:** If the user is trying to checkin but his/her token is expired or the iCorsi password is changed an error is visualised and the user is *not* signed to the lecture. In this case the login view (image 4) is visualised again to refresh the status.

5 Backend

5.1 Server

As always, for a good application, a good backend is needed.
In our case, we decided to build it over the Nodejs platform, which we had used many times before, together with the Expressjs framework, which allowed us to build what we needed rapidly and easily.
Since we are working with sensitive data, we built the server in order to work with the HTTPs protocol and made sure to create authentication and authorisation middleware based on tokens in order to ensure the safety of this data. All the requests to the database are also made through our server only, which made the DB secure enough for the demo we had to build.

5.2 Database

The choice for the Database went to MongoDB, as it is easy to implement with Nodejs and, again, we had already used it before for other works. Through the module of Mongoosejs, creating all the Schema we needed was just a matter of minutes, where the biggest problem was just to decide which fields we needed for each document in the DB.
Unfortunately, we couldn't use any of the data present in the Moodle database, as it would have violated the privacy of the users registered there. To overcome the problem, we asked for help to the Icorsi developers here at USI, which gladly helped us by providing a Backend route to call in order to obtain a small subset of the data of a defined (and authenticated) user.
Given this, we could actually build the rest of the Database Documents that best reflected our needs, ending up with the following:

- User - Schema representing either a student, a TA or a Teacher
- Course - Schema representing a single course
- Event - Schema representing a lecture to show in the calendar
- Presence - Schema representing the presence of a user to a lecture
- Region - Schema representing the settings for the beacons related to a defined Room
- Room - Schema representing a room of the Campus

5.3 Authentication

Even if we couldn't use Moodle database, we could still ask the service to authenticate the users of our app.

At login, the user is asked for his username and password, which are sent, through a serie of HTTPs requests, to the Moodle platform.
What we have in response, in case of successful authentication, is a set of cookies containing different session data, including the Moodle token necessary to make sure that the actual user exists in the Moodle database, which is the only thing we need at this point.
Once obtained this token, we could use the Backend route that the Icorsi developers built for us in order ti retrieve a set of the user data from Moodle, such as Id, Username, Email, First name, Last name and others. In this way, we could define a precise relation between our database and the Moodle database through the Id of the user and use this to be sure that our app users are actually registered to the university platform. Every time the user goes through our login, all this steps are done once again in order to make sure that there aren't any important changes in the Moodle database that should be reflected in our own, such as change in the user email, username or others.

5.4 Authorisation

Once logged in and authenticated, we know that the actual user is a real, existing user already registered to the Icorsi Platform. This, however, is not enough to provide a strong security level for our application. The app requests are still sent to **our** server, and we needed some form of authorisation in order to acknowledge the valid requests and refuse the others.
For this reason, we implemented an authorisation level based on JWT tokens:
On a successful login, we generate a token - different form the Moodle token we obtained, which we use only for authentication - that is sent to our mobile app together with the user data. This token is saved on the user phone and is sent with every request the app does to our server. Since the same token is saved on our database, we are able to verify that the token in the request exists, belong to that user and it is not expired yet. In case one of this checks fails, the request is immediately refused and a new login is requested to the user.

6 The Dashboard

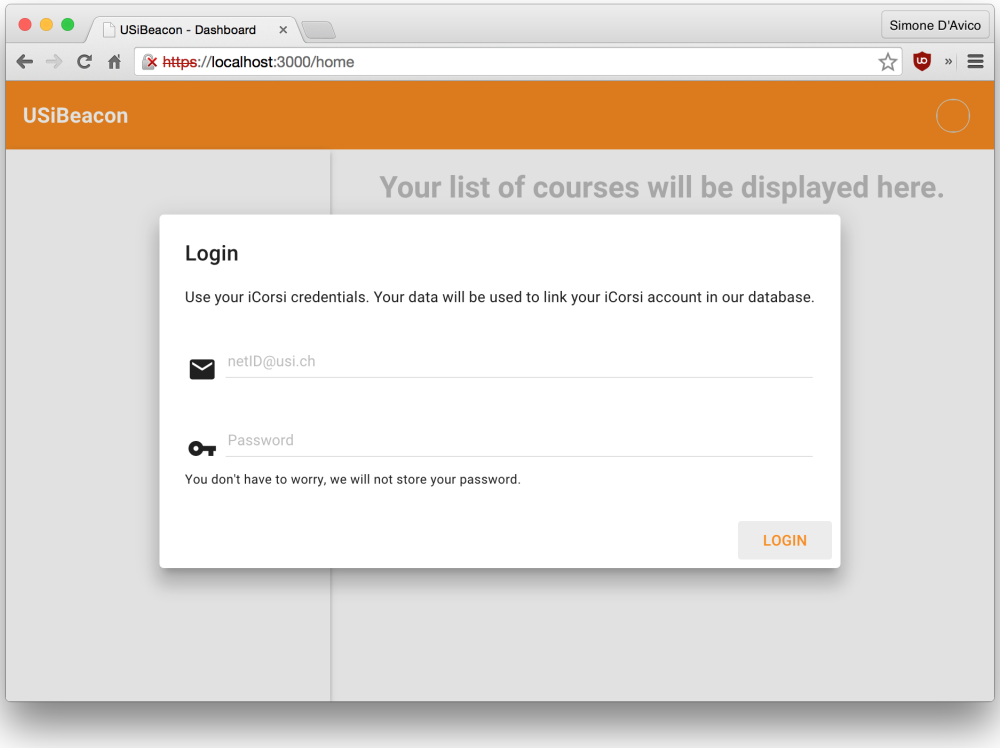
The last thing we wanted to implement for the project was a small dashboard were students would have been able to track the lessons they attended for each course, together with some overview statistics. Originally, we would have liked to implement it by integrating it with iCorsi on the Moodle platform. Unfortunately, after consulting the eLab we concluded that it was not possible, due to several problems:

- Moodle adopts its own plugin systems for additions to the platform; the modules use their own paradigm, and require knowledge of PHP (which we never tried);
- Even if we wanted to try to write a Moodle plugin in PHP, the eLab wouldn't allow us to develop the plugin directly on the iCorsi platform; they could have set up a test instance of Moodle for us, but it would have taken some time;
- They would have to give us a test instance of the databases used for storing the student's informations, and we would have to understand its structure and complement it with new tables/fields required to store our own informations.

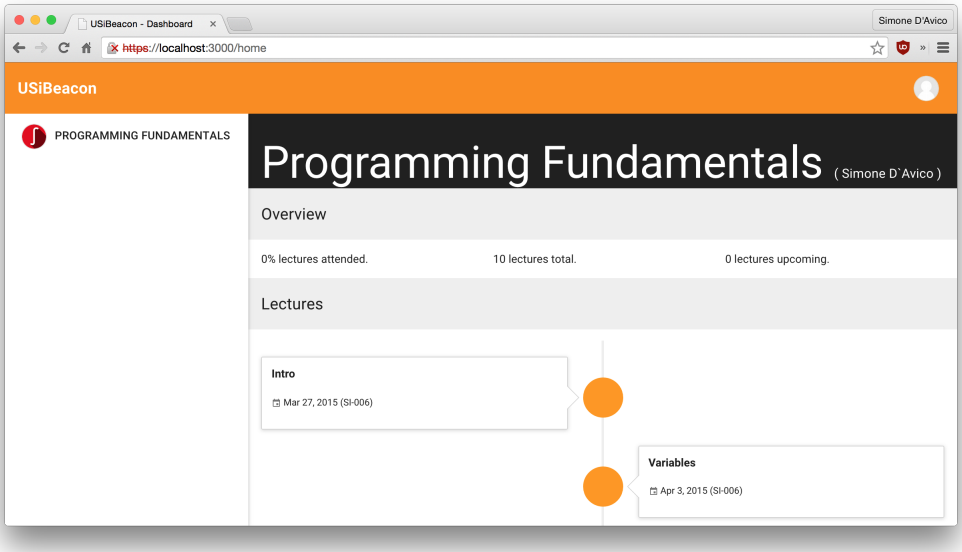
These issues made the Moodle integration not viable, so we had to settle with our own backend (discussed in the previous chapter) and provide the dashboard as a separate application.

To implement it, we chose to adopt state of the art web technologies: the application is implemented using **AngularJS**, and we secured it by implementing the protocol for **JSW** (Javascript Web Token) authentication.

When the application is started, it will require the student to log in using his/her iCorsi credentials:



On login, the application will store the JWT token received by the backend, which will be valid for 7 days from its creation. After the expiration, the student will have to login again. This is what the student will see:



In the overview for each lecture, the student will be able to see the percentage of lessons attended for a given course, the total number of lectures, and the number of upcoming lectures. He/she will also be able to look at a timeline of all the lectures of the course, where each lessons he/she attended will be ticked.