Indoor Navigation

by

Lennart (XXXXXXX)
Jan (XXXXXX)
Eridy (XXXXXXX)
Andreas (XXXXXXX)

A project documentation submitted to

Technische Universität Berlin
School IV - Electrical Engineering and Computer Science
Department of Telecommunication Systems
Service-centric Networking

Project Documentation

February 17, 2016

Supervised by:
Sebastian Zickau and Mathias Slawik

Abstract

The well known Global Positioning System (GPS) works nice outdoors but indoors it gets much worse and not really precise. So GPS does not work well indoors. The goal of this project is to research possibilities for indoor navigation to find other participants indoors. We had to implement three parts to achieve our purpose. An iOS and Android application as well as a backend to handle communication between our clients and to persist project data. One main technology provided by the SNET department at Technical University Berlin (TU Berlin) were Estimote Bluetooth beacons which were a very nice opportunity to operate and test the technology of Bluetooth Low Energy (BLE) devices. To manage and authenticate users in our system we used an implementation of Keycloak which allowed us to have a single-sign-on and good security solution for our project. The implementation is the CYCLONE Federation Provider which is developed by the SNET department. At the end we tested our project in the main canteen of TU Berlin.

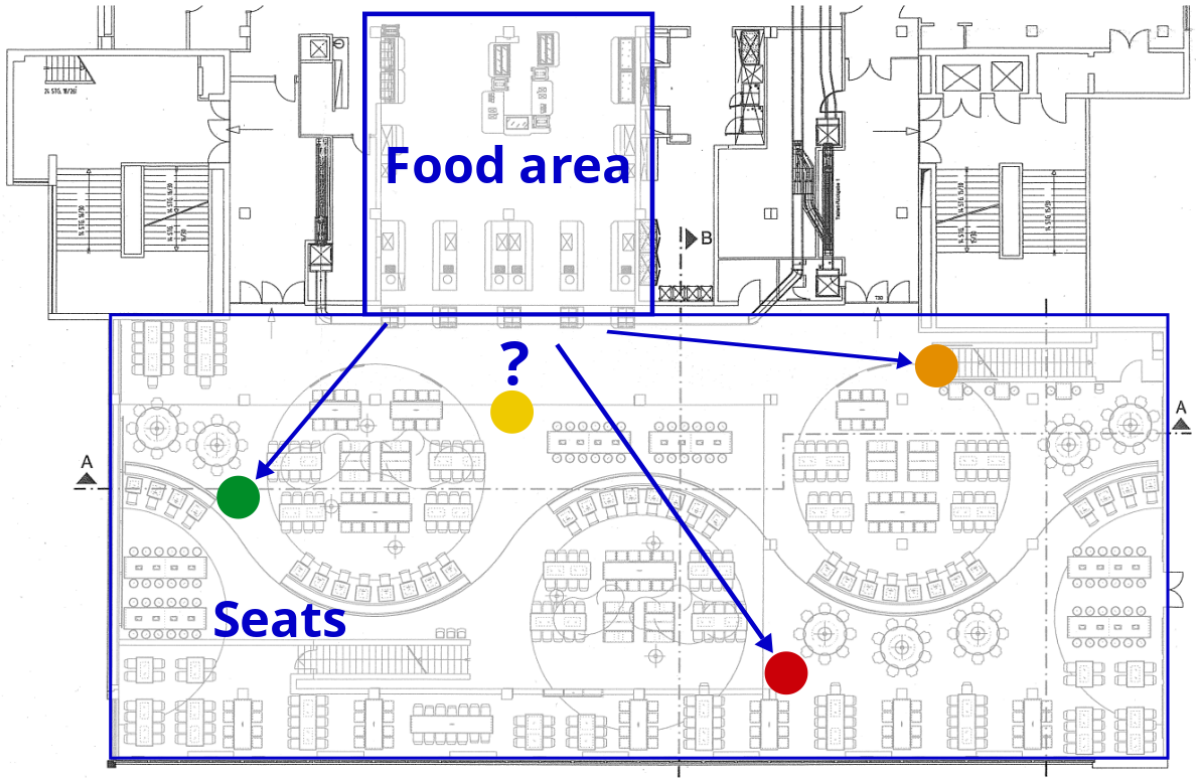
Contents

1 Introduction

1.1 Motivation

The mensa in Hardenbergstraße at TU Berlin.

Probably every person who has ever been to a mensa around lunch time will know the problem our project was about. You are hungry and your colleagues are too. Therefore in a group of maybe four people you enter the mensa and immediately start to spread out. Everyone likes a different dish, also some queues take longer than others and cash desks work after their own rhythm anyway. The result is: Finally done with paying for your food and in joyful anticipation of your deserved lunch you realize that at this point you don't have any clue anymore, where everyone went. Your colleagues all were faster than you but still managed to get away from the cash desks at different times and all started to look out for a place to sit with four people. This could look something like what the following graphic illustrates:



Yellow person looking for her/his friends in crowded mensa.

The same situation can happen in other big, crowded places where people try to meet as a group. As such, an example could be an university library in examination weeks where maybe a few people want to meet in order to study for their exams or even simply later want to have lunch in the mensa and experience the problem a second time. Either situation, exactly this problem of finding your peers in a user- yet privacy-friendly manner with modern hardware is the problem our project was focused to solve on.

As just mentioned, our project tackled this problem from a technological perspective. After all, we were part of the "Internet of Services Lab" (IoSL)[1], a project by the SNET department at TU Berlin. At the beginning of the semester we were brainstorming while preparing slides for the Milestone Workshop and we came up with the following vision of how our to-be-developed system should look like to an user:

[1] https://www.snet.tu-berlin.de/menue/teaching/winter_term_2015_2016/internet_of_services_lab_wt20152016/

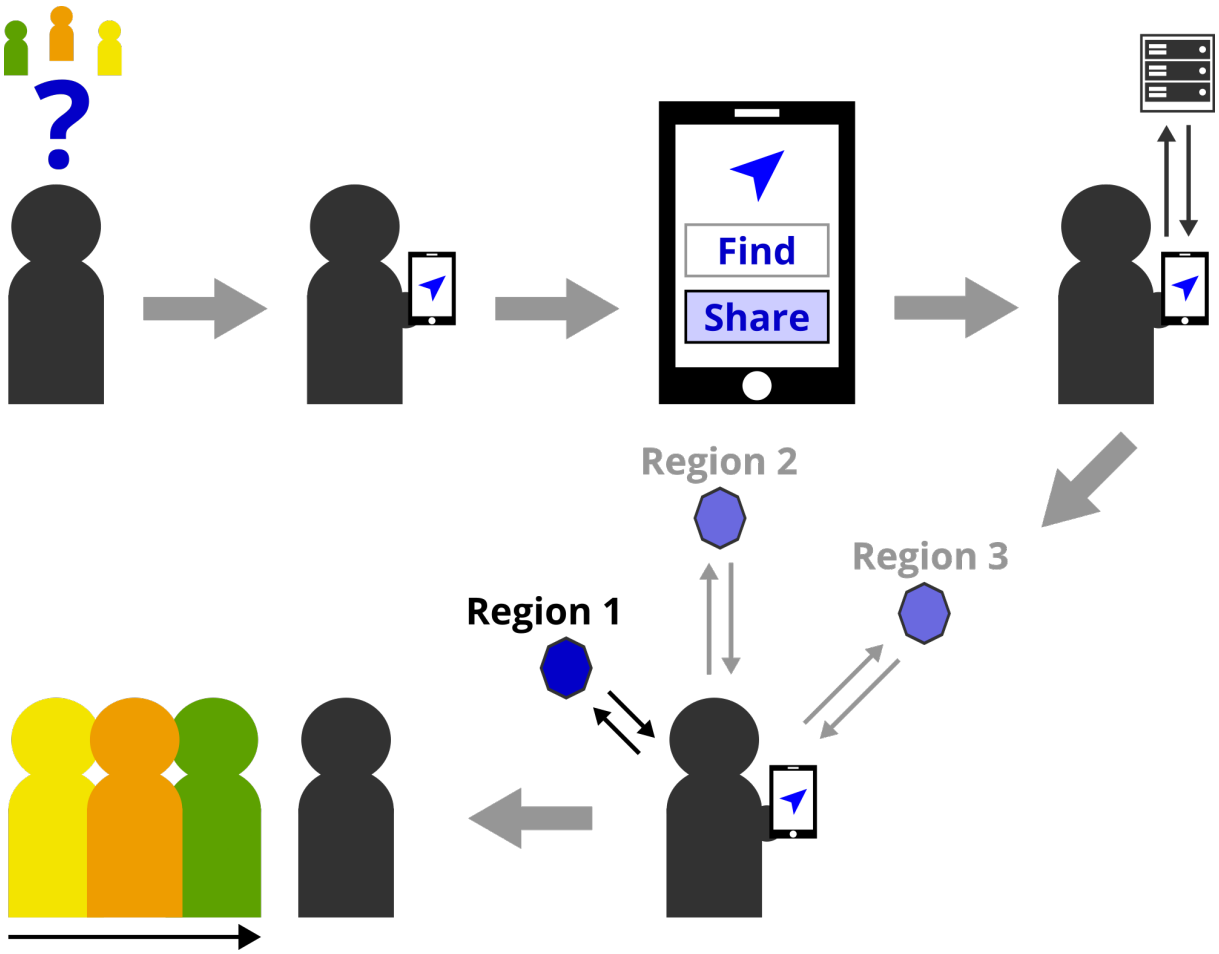
Imagined use of our app on a mensa tray.

The use of a smartphone was quite clear and of course, to interconnect users, a backend would be needed. What we had to figure out in the beginning though, was which localization approaches we could use and which would work in such densely crowded spaces as the ones described above. Therefore and to generally plan our project, we decided on the following semester's structure.

The first part would be a research phase in which we would evaluate which technologies yielded which advantages and which disadvantages and we would decide on which of those to use in our architecture. That first phase resulted in chapter 2, our reasoning about "Localization Technologies". After that we would put our gained knowledge together and also incorporate resources from the department to lay out the envisioned system we where trying to build. We wrote about these concept ideas in chapter 3, "Concept and Design". Somewhat parallel we already started to set up our system and began the implementation. This would shortly after become the main and most important part of the project, of course. The result you can read about in chapter 4, "Implementation". Of course, realized projects need to be evaluated and tested, also in preparation for the approaching Final Workshop. What we achieved and not achieved, you will be able to read about in chapter 5, "Evaluation". And after that we summarized what this project was about, how far we got and what future work aspects we recommend. Read about it in chapter 6, "Conclusion".

Over the course of the semester the mentioned parts were structured as two week sprints where at the end of each sprint we would meet with our supervisors, Sebastian Zickau and Mathias Slawik, for a milestone meeting. During these meetings each of the project's sub teams would report on the progress in the respective area and on experienced issues. On demand, special discussion would follow and at the end goals to achieve for the next milestone were defined. The mentioned sub teams inside our project were the obvious split up to focus on one part of the technology stack involved in the architecture. Jan took care of the Android client development, Eridy worked on the iOS client and Andreas and Lennart developed the backend part.

With that set, let's take a look at the final flow a user experiences by using our service. The following graphic shows the way from the initial problem of not knowing where a user's friends are, over the sharing of the user's region based location (further information on this in the following chapter) to the final success of finding back to the user's friends with the help of our service.



Steps on user's way to her/his friends.

1.2 Constraints

The idea of this project described above already introduced some implicit constraints in order to make it work. To have a complete overview of what we expect to have available to use our service, we defined the following constraints:

- **Everyone has a smartphone**: Obvious point. As this project involves developing clients for Android and iOS and relying on these clients to interact in an user friendly manner with the backend, a smartphone is needed.
- **Participants have a TUB account (edugain account)**: The service we provide needs an authentication prior to being able to modify or delete objects on backend side. Therefore some kind of user management was needed. As we will explain more deeply later in this document, for user authentication and session management we rely on a department service called CYCLONE Federation Provider[2]. Via this service users with an edugain (and therefore a TUB) account are able to authenticate against our backend.
- **Minimal interaction**: As described in the previous section our intended use case is placed in situations where you probably carry a tray full of food and will not be having your hands free to interact with complex user interfaces. Therefore a very important constraint for our project was to provide applications that do not need a lot of intention in order to work as intended while at the same time strictly comply with the user's set preferences.
- **Easy to use**: This constraint correlates with the previous one. The setting our applications are intended for do not allow for complex user interfaces. Actions needed to be taken while using the services need to be easy to perform.
- **Low impact on device battery**: The possible use of Bluetooth introduced the topic of battery usage. We therefore defined low battery usage as an important constraint for our project.

1.3 Functional Requirements

After motivation and constraints were set, we defined our functional requirements. These were the functions we had to implement to follow along our imagined flow throughout the application. Therefore a user has to be able to...

- **...login via university account**: As mentioned in our constraints, login should happen via the university account. We are using the CYCLONE Federation Provider and therefore things work a little different than in a self-developed login mechanism. For us this meant to integrate the provided OpenID Connect authentication flow in a user friendly way into the mobile applications. The applications in turn would then be authenticated against the backend.
- **...add friends via university mail**: All connected edugain members each have their own unique domain name and internally only contain unique users. It was the logical step to provide the functionality to add friends by their mail address as the unique identifier.
- **...share her/his location**: The most important part of our system is the modifiability

[2] http://www.cyclone-project.eu/

of the logged in user only. This includes updating or deleting the user's location and defining in which conditions and with which friends the location should be shared.

- **...see shared locations of friends**: Friends' locations were supposed to be visualized on a map. To realise this a functionality to retrieve locations of friends directly in the user's area needed to be developed.

2 Localization Technologies

2.1 Related Work

Before we started our project we did research on related work regarding applied indoor navigation and positioning techniques applied on mobile devices. Most of them rely on calculating Received Signal Strength Indicator (RSSI) triangulation and/or Bluetooth BLE Beacon techniques or Wi-Fi Based Fingerprinting. The latter is a core idea of an indoor navigation system developed by [3]. They combine two methods in order to calculate a precise position. At first they use matching of prerecorded received signal strength from nearby access points. This method is called "fingerprint matching". The data is combined with a distance-based trilateration approach with at least three known access point coordinates which are also detected on the device. By this combination of both methods they received a high accuracy of the user position in an indoor environment.

The group in [6] implemented an indoor Bluetooth-based positioning system on Bluetooth-capable devices as a PDA, which is comparable to the smartphones of today. They calculated range estimations based on approximations of relations between RSSI and distance between sender and receiver. The location estimation was also calculated via triangulation method.

Apple Inc. presented on WWDC2015 new features of there well known Core Location Frameworks providing Indoor positions and Floor information [16]. The closed beta is restricted by access to developers that are registered on Apples Indoor Program Maps Connect. Although Apple does not publish its algorithms, as stated on WWDC2015 by Apple Software Engineer Vitali Lovich the Core Location Frameworks also takes advantage of indoor WiFi signals in combination with motion and altitude sensors in order to provide the users with accurate position information as latitude and longitude.

In 2013 Apple also developed the iBeacon protocol which is a Bluetooth Low Energy (BLE) communication standard supported by iOS and Android devices [12]. The iBeacon protocol opened the door for new Beacon technologies in order to improve indoor navigation on mobile devices. The standard is incorporated by Estimote Inc. with Estimote Beacons. This indoor technique has successfully been installed in museums such as the Metropolitan Museum of Art project called: MediaLab Beacon Art Walk [4]. However, this project aimed to provide additional location information to the visitors rather than providing indoor positions. Next to the battery issues also faced in the MediaLab Project, Beacons having difficulties in cases of many people covering the clear line of sight to a device, makes actual retrieving of Latitude/Longitude coordinates in a room more difficult. Next to this, installed beacons made temporarily maintaining of Beacons mandatory, since one missing Beacon could prevent successful indoor

navigation in a calibrated system.

The Cisco MSE determines the location of any wireless device in a specific building. This approach moves the calculation of a position away from a mobile device [5]. Correctly installed Cisco MSE determines the location of any wireless device in a building. The access points listen for Wi-Fi signals of mobile devices and estimate its position also via trilateration. This solution has been deployed successfully at a few locations as mobile application for visitors, like the Explorer App at the American Museum of Natural History [17].

2.2 Short Range Localization Techniques

Different localization techniques are presented and compared in book [2]. For purpose of this project, we were interested mostly in short range positioning technologies. Short range positioning technologies cover small areas and are frequently employed for positioning in indoor environments. The most popular technologies are WiFi and Bluetooth, which are most commonly used to determine the proximity location of a mobile device connected to a network. Other available technologies are Radio Frequency Identification (RDIF), Ultra Wide Band Positioning, ultrasonic positioning, infrared positioning, camera-assisted and sensor-assisted positioning. We focused our research on Wi-Fi and Bluetooth as those were the technologies provided for us by our supervisors.

2.2.1 WiFi-Based Positioning

The basic principle of using WiFi for indoor positioning is measuring strength of signals received at two or more access points. The signals used for positioning are called beacon frames and are primarily intended for announcing the presence of a wireless LAN. There are two ways of transmitting these signals for positioning, up-link, when beacon frames are generated by mobile devices and down-link, where beacon frames are generated by access points.

Actual positioning can be done in three ways. The simplest method is determining the position of a mobile device from the position of the access point with the strongest signal, which is the closest to the mobile device. In this case a data linking access points and their locations is needed. The second method uses signal strength data from multiple access points to calculate the position of a mobile device. This method provides more accurate position data. The third method uses a fingerprinting approach, in which the position is determined by matching signal data on the mobile device against a set of precollected data on signal strengths across the area. This approach offers the best results, but it requires initial measurements of the area before position tracking can take place.

WiFi based positioning only works in areas with good wireless network coverage. It is most commonly used in closed spaces such as offices, airports, cafes, shopping markets, hotels etc. It can also be used for outdoor positioning, usually in dense populated urban areas.

In general, WiFi positioning provides a proximity location instead of more specific coordinates. For example, it may provide information, in which building, floor and room the mobile device is located. When using WiFi positioning there is no need for additional infrastructure, except for an existing WiFi network and a positioning server with a database of all available access points along with their positions.

2.2.2 Bluetooth-Based Positioning

Various solutions have been developed using Bluetooth for short range positioning. Bluetooth enabled devices are able to transmit signals containing information such as device identity and profile. When a mobile device is in short range of a device transmitting its location it can pick up such signals and use them for positioning. Signals can also be processed to determine the position of a device, especially in case when the mobile device can receive multiple signals from different devices. Position data can be exchanged between mobile devices using ad-hoc Bluetooth networks or with a location server in the network.

The signal strength decreases logarithmically with distance. This property can be used for calculating positions. One method of positioning a device is to triangulate data from different devices. This method offers higher accuracy than just detecting the closest device in range.

Bluetooth positioning can be deployed rapidly with easy maintenance and low cost. It can be used in applications where approximate positioning is sufficient.

Different implementations of WiFi- and Bluetooth-based localization technologies are described in more detail in following sections.

2.3 Bluetooth Low Energy

Multiple solutions are based on the already mentioned Bluetooth Low Energy (BLE) standard. To understand the reasons for this standard in comparison to classic Bluetooth better, we had "Bluetooth Low Energy - The Developer's Handbook" by Robin Heydon, published with Prentice Hall in 2012, available [9]. In the following we try to point out some specifics of BLE taken from the just mentioned book.

Heydon starts by stating that although BLE has obvious connections to its parents, the classic Bluetooth standards, "Bluetooth low energy should be considered a different technology, [...]." (ibid., p. 3). This is a consequence derived from the different design goals the developers had when they set out to standardize what would become Bluetooth Low Energy. The new standard is not a performance upgrade but by design focused on totally different obstacles. "When the low energy work started, the goal was to create the lowest-power short-range wireless technology possible." (ibid., p. 7). This resulted in following goals: worldwide operation, low cost, robust, short range, low power (ibid., p. 7).

Bluetooth Low Energy does not try to be yet another bandwidth upgrade for Bluetooth but rather a step towards extremely low power consumption as well as being very cheap. This latter directive makes it possible for Bluetooth Low Energy to be deployed in high volumes. In order to achieve this, Bluetooth Low Energy makes use of following three design points:

- **Operation on 2.4 GHz ISM band**. It's an overused band and has bad characteristics as it is heavily absorbed by water which is the biggest part of the human body. But also no license fees are taken to operate on this band and therefore, "choosing to use the ISM band lowers the cost" (ibid., p. 5).
- **Intellectual property (IP) license for usage**. Basically, the Bluetooth Special Interest Group (SIG) is claimed to be cheaper than other SIGs. Licenses are given under a FRAND policy which stands for "Fair, Reasonable, and Non-Discriminatory" (ibid., p. 5).

- **Low power consumption**. Aiming at low power consumption overall reduces the costs of accompanying services and material costs.

This results in transmission speeds that are way below the ones of the classic Bluetooth versions, even lower than the speed of some very early standard versions of Bluetooth. Following table reviews the speeds of the existing Bluetooth variants (ibid., table 1-1, p. 4):

Bluetooth standard	Maximum bandwidth
v1.1	1 MBps
v2.0	3 MBps
v3.0	54 MBps
v4.0 (BLE)	0.3 MBps

Therefore BLE is not tailored on the use cases classic Bluetooth tackles, "[...] because single mode Bluetooth low energy does not support audio for headsets and stereo music or high data rates for file transfers." (ibid., p. 6). It is to be used in situations where extremely low power consumption is needed while at the same time the amount of data transferred is kept low.

To survive in the congested frequency band it operates on, Bluetooth Low Energy uses adaptive frequency hopping. A technique that detects sources of interference, helps to avoid them in the future and quickly recovers from dropped packages. "It is this robustness that is absolutely key to the success of any wireless technology in the most congested radio spectrum available." (ibid., p. 8).

Another consideration is the distance BLE is able to cover and Heydon concludes that "[s]hort range means that Bluetooth low energy should be a **person area network**" (ibid., p. 8).

2.4 Estimote Beacons

Estimote Beacons and Stickers are wireless sensors that can be attached to any location or object, embodying a wireless sensor network. The Beacons consists of a 32-bit ARM Cortex CPUs, equipped with an accelerator, temperature sensor and a 2.4 Ghz radio. Using Smart Bluetooth 4.0 (Bluetooth Low Energy), the beacons send out signals with a range of up to 70 meters (Beacons) and 15 meters (Stickers). Though the signals are often distracted under real world conditions, a range of about 40-50 meters (Beacons) can be expected. As stated in [14], the battery is able to last more than 3 years on default settings on a single CR2477 battery.

Estimote beacons are working with the Apple iBeacon protocol as well as the Eddystone open beacon format introduced by Google. These protocols working on top of BLE are implemented in all smartphone devices enabling them to support new technologies like the Apple Watch or fitness trackers.

Using the Estimote SDK [14] mobile applications are enabled to receive and understand BLE Estimote signals in order to calculate the proximity of nearby locations and objects. The beacons specifics provide information about their type, ownership and approximate locations, temperature or motions.

By detecting a beacon signal a phone can estimate the distance by measuring the received signal strength [14]. Since Bluetooth Low Energy does not need any pairing process between

sender and receiver, the phone can constantly process new signals. This opens the door for new technological opportunities such as indoor positioning.

2.4.1 iBeacon Protocol

The iBeacon protocol is a Bluetooth Low Energy communication protocol developed by Apple Inc. in 2013 and was introduced in iOS7 for indoor navigation. The protocol is supported by iOS7 devices as well as Android from version 4.3 up. The signal sent by a beacon is called advertisement.

These advertisements provide a so called iBeacon identifier, that is 20 bytes long and divided into three sections:

- UUID (16 bytes)
- major number (2 bytes)
- minor number (2 bytes)

These values provided by the advertisement can be modified according to own wishes. The hierarchical configuration of these values provides identifying information about the beacon. While the UUID can be distinguished to a corporation, major and minor values can be used to distinguish between regions and sub-regions of a corporation.

2.4.2 Region Monitoring

Region monitoring triggers actions on the device on entering or exiting a beacon defined region's range. This works in depending the devices capabilities while an app is in foreground, background or suspended mode. An app is limited to 20 regions being monitored. However, by using a single UUID in multiple locations, a device can monitor many physical locations simultaneously [12].

2.4.3 Ranging

Ranging however triggers actions on the device based on the proximity to a beacon. The iBeacon protocol applies filters to the accuracy of an advertisement of one beacon. The filtered estimation regarding the proximity to a beacon is indicated using one of four proximity states.

- **Immediate**
  The Immediate proximity state represents a high level of confidence that the device is physically very close to the beacon. This is for example the case when holding the smartphone directly onto a beacon.
- **Near**
  The Near proximity state indicates a proximity of about 1-3 meters, if there are no obstructions between the device and the beacon which might cause distractions.
- **Far**
  The Far proximity state indicates a detected beacon without much confidence in the accuracy that is too low to determine whether it is Near or Immediate. The Far proximity

state relies on the accuracy property to determine the potential proximity to a beacon.

- **Unknown**
  The Unknown proximity state indicates a state where beacons can not be determined. This might happen if the ranging has just begun or if the accuracy level is insufficient for measurements to determine a state that is either Far, Near or Immediate.

2.4.4 Estimote Beacon Drawbacks

For the implementation of Estimote Indoor Navigation for the project we decided to use monitoring on the devices in order to detect the beacons, specifying a certain region. However, the devices still take at least about 30 seconds to recognize the fact that a beacon is out of range. This is a built-in and non-adjustable delay in order to prevent "false" exit events [14]. This is a major drawback regarding an use case of the project, where an user might pass multiple beacons of a location and constantly update new locations without the need to stop at each beacon.

This is solved by an additional applied ranging of all beacons that are monitored in order to process the "nearest" beacon as an user's location. The devices report the beacons in an order that is the best guess of their proximity regarding issues of signal attenuation. This order however may still not be correct [12].

2.5 Apple Core Location Framework

As firstly presented on WWDC2015 [16] Apple presented new functionalities to the already existing Core Location Framework which is the major framework on iOS devices for location service [11]. This framework takes the user in charge of whether the app can use locations services on the device or not.

The Core Location Framework uses cellular data to provide a proximity in which area in a city a user is. Additionally it uses GPS based on satellite signals to improve the position of the user as well as surrounding WiFi signals.

As soon as the user enters an indoor venue, the iOS system turns down GPS and cellular sensors and enlightens motion and WiFi sensors. These sensors are used in combination with the remaining GPS signals, coming through the windows to locate the user indoors. The motion sensor hereby gives information to the system that a person is moving and how fast the user is moving, while WiFi signals are fed to the CLLocation Framework to calculate the exact position.

Apple also added the altitude and floor attributes to the CLLocation Framework in order to provide the user with the correct floor attributes [11].

For the project the new CLLocation Framework was used to show the user's position in the mensa. The first indoor tests in the mensa revealed precise positioning of an user indoors. However the framework also revealed large susceptibility in areas where the WiFi signal was apparently weak. I.e. in the upper left corner of the mensa which is surrounded by walls that are affecting the router's WiFi signals.

The framework started in an early beta state when this project started and is constantly im-

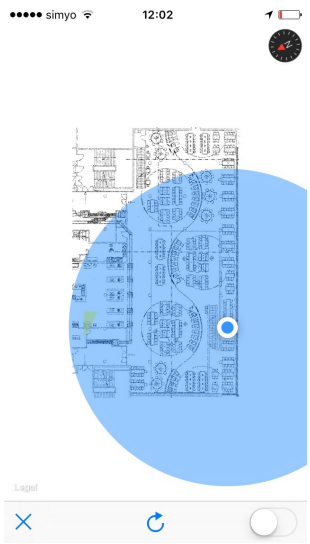


Figure 2.1: Indoor Position on start

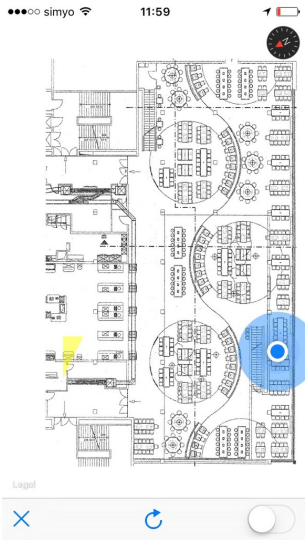


Figure 2.2: Accuracy drastically improved after a couple of seconds

proved. To enable the full abilities of CLLocation Framework indoors, the venue needs to be registered and enabled by Apple, using the Maps Connect program [13] in order to unlock CLLocation Framework's full potential.

Regarding this project we tested the indoor navigation framework to see if it's accuracy is already sufficient for our use case in the mensa. As shown in 2.1 and 2.2 the user's location is shown precisely on the indoor floor plan. The accuracy as shown in 2.1 started with a large circle but constantly improved while moving through the building. We also accounted situations in which the framework lost precision dramatically as we moved to the upper left corner of the mensa which is surrounded by walls. However the provided coordinates of the shared position of the user to the system were still accurate enough in order to find a friend.

We decided to use this technology, since it is the only solution for this project in order to show the user position on a map without any additional installations in the environment of the mensa, out of the box and without any additional user interaction.

2.6 CISCO MSE API Wrapper Tests

In order to determine whether the CISCO MSE API wrapper provided by tubIT would be sufficient for the project's requirements we were asked to perform tests on it. Especially it was asked for details on how the API worked where, so what values could be retrieved via the API wrapper in which locations on campus and how precise the values would turn out to be.

Concerning use cases our project was focused on the Mensa and the library, therefore we initially planned to be conducting tests in only these two locations. As the provided wrapper around the CISCO MSE API we had access to delivered back one short, simple XML line we decided to invest some time into developing a small tool which routinely queried the API for it's current status and saved the result into an easily readable JSON file for later investigation. The source code of the developed tool you can find in appendix 2.

```
<Info changedOn="2015-10-30T15:32:27.185+0100"
confidenceFactor="40.0" building="MAR" floor="Erdgeschoss"
WLAN-Status="EDUROAM" username="[redacted]"
lon="13.323857725896488" lat="52.516801767504241"/>
```

What the CISCO MSE API wrapper response looks like.

It was planned to be conducting the tests on one day in the Mensa and on another day in the library. On the first day we started around noon and ran the test tool on one of our notebooks connected to university WiFi, "eduroam". We started in the south-western corner near the windows, walked towards the south-eastern corner, went to the stairs in the northern part and upstairs and again at the window front to the south-western corner on the first floor. As it turned out, the results we got back were definitely not what we had hoped for, most importantly because longitude and latitude of the requesting user were missing completely. The following listing shows the first ten results logged in two second periods from the API wrapper:

```
1  {
2      "signal": [
3          {"timestamp": 1445344391, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "Mensa", "floor": "
               Mensa 1. OG"},
4          {"timestamp": 1445344393, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "Mensa", "floor": "
               Mensa 1. OG"},
5          {"timestamp": 1445344395, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "Mensa", "floor": "
               Mensa 1. OG"},
6          {"timestamp": 1445344397, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "Mensa", "floor": "
               Mensa 1. OG"},
7          {"timestamp": 1445344399, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "Mensa", "floor": "
               Mensa 1. OG"},
8          {"timestamp": 1445344401, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "Mensa", "floor": "
               Mensa 1. OG"},
9          {"timestamp": 1445344404, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "Mensa", "floor": "
               Mensa 1. OG"},
10         {"timestamp": 1445344406, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "Mensa", "floor": "
               Mensa 1. OG"},
11         {"timestamp": 1445344408, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "Mensa", "floor": "
               Mensa 1. OG"},
12         {"timestamp": 1445344410, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "Mensa", "floor": "
               Mensa 1. OG"},
13         ...
```

```
14      ]
15  }
```

Clearly it can be observed that the longitude and latitude values are unusable. Another take away was that the floor change during our test did not reflect into our captured results. Therefore we decided to directly test the library for comparable results. Inside the library, we started on ground floor and went upstairs in "circles" through the different levels. From the fourth floor we went back down straight forward. During that second test ten of the first fifteen responses from the API looked like:

```
1  {
2      "signal": [
3          {"timestamp": 1445345843, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "BIB", "floor": "
               Erdgeschoss"},
4          {"timestamp": 1445345845, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "BIB", "floor": "
               Erdgeschoss"},
5          {"timestamp": 1445345848, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "BIB", "floor": "
               Erdgeschoss"},
6          {"timestamp": 1445345850, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "BIB", "floor": "
               Erdgeschoss"},
7          {"timestamp": 1445345852, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "BIB", "floor": "
               Erdgeschoss"},
8          {"timestamp": 1445345854, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "BIB", "floor": "
               Erdgeschoss"},
9          ...
10         {"timestamp": 1445345870, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "BIB", "floor": "1.
                Obergeschoss"},
11         {"timestamp": 1445345872, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "BIB", "floor": "1.
                Obergeschoss"},
12         {"timestamp": 1445345874, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "BIB", "floor": "1.
                Obergeschoss"},
13         {"timestamp": 1445345876, "latitude": "0.000000000000000", "
               longitude": "0.000000000000000", "building": "BIB", "floor": "1.
                Obergeschoss"},
14         ...
15     ]
16 }
```

First, longitude and latitude were again unusable. This time though the floor information worked quite reliably and indicated very fast on which floor we currently measured. After that we were wondering whether eventually we would get back longitude and latitude values anywhere on campus and decided to give it one last try by taking one more measurement in the MAR building (Marchstraße).

One more measurement turned into three as during the first two attempts we got sudden disconnects and therefore unusable results. We walked the whole foyer from north to south side and somewhere near the entrance we suspect the wireless signal got bad and our notebook conducting the tests disconnected. In the third try though we finally were able to get back usable results, in which chosen ten results logged looked like this:


```
{
    "signal": [
        {"timestamp": 1445350550, "latitude": "52.516903688639005", "
           longitude": "13.323958376544699", "building": "MAR", "floor": "
           Erdgeschoss"},
        {"timestamp": 1445350552, "latitude": "52.516903688639005", "
           longitude": "13.323958376544699", "building": "MAR", "floor": "
           Erdgeschoss"},
        {"timestamp": 1445350554, "latitude": "52.516903688639005", "
           longitude": "13.323958376544699", "building": "MAR", "floor": "
           Erdgeschoss"},
        {"timestamp": 1445350557, "latitude": "52.516903688639005", "
           longitude": "13.323958376544699", "building": "MAR", "floor": "
           Erdgeschoss"},
        ...
        {"timestamp": 1445350686, "latitude": "52.516864921942748", "
           longitude": "13.323953890659670", "building": "MAR", "floor": "
           Erdgeschoss"},
        {"timestamp": 1445350688, "latitude": "52.516864921942748", "
           longitude": "13.323953890659670", "building": "MAR", "floor": "
           Erdgeschoss"},
        {"timestamp": 1445350690, "latitude": "52.516864921942748", "
           longitude": "13.323953890659670", "building": "MAR", "floor": "
           Erdgeschoss"},
        ...
        {"timestamp": 1445350845, "latitude": "52.516402095317481", "
           longitude": "13.323531401046099", "building": "MAR", "floor": "
           Erdgeschoss"},
        {"timestamp": 1445350847, "latitude": "52.516402095317481", "
           longitude": "13.323531401046099", "building": "MAR", "floor": "
           Erdgeschoss"},
        {"timestamp": 1445350849, "latitude": "52.516402095317481", "
           longitude": "13.323531401046099", "building": "MAR", "floor": "
           Erdgeschoss"},
        ...
    ]
}
```


Finally we received some longitude and latitude values. Unfortunately the three different pairs of longitude and latitude above were the only ones we could observe during the whole walk from north end to south end of the foyer, thus still rather unusable values.

In the end, our conclusion at that point in the project progress was to use the CISCO MSE API wrapper provided by the tubIT in order to retrieve the rough position of a user. This means the building and floor the request originated from. We recommended back to our supervisors not to use this API for receiving longitude and latitude as these values were either quite imprecise

or not available at all. The full result files of all our three measurements can be found at [18].

2.7 Evaluation of Available Technologies

As mentioned in 2.2, WiFi and Bluetooth based positioning technologies are the most suitable and most widely used for indoor positioning. With technology development such as Bluetooth Low Energy and different wrapper implementations such as CISCO MSE API those two technologies became even more important players in indoor navigation. Review of related work confirms this statement, as majority of related work use those two technologies.

This research and related work presented the basics for our decision on which technologies to use. Our decision was also biased by the infrastructure and equipment we had at our disposal. TubIT provides the wrapper for CISCO MSE API in the university campus and because WiFi based localization techniques are widely used for indoor positioning, we decided to give it a try. It turned out to be quite ineffective at retrieving exact coordinates of users. However, since it provides quality information about building name and floor all over the university campus, we decided to include it in our project as least accurate positioning technique. We also had 3 Estimote Beacons and 6 Stickers at our disposal. Since Bluetooth positioning techniques are widely used for indoor positioning, we considered Estimote Beacons and Stickers as possible localization technique for our project. Their usage of Bluetooth Low Energy which provides low cost, robustness and low power usage represented big plus in their favor. The biggest plus was their support for iBeacon protocol which makes them compatible with all modern mobile devices. All those factors supported our decision to use Estimote Beacons and Stickers in our project.

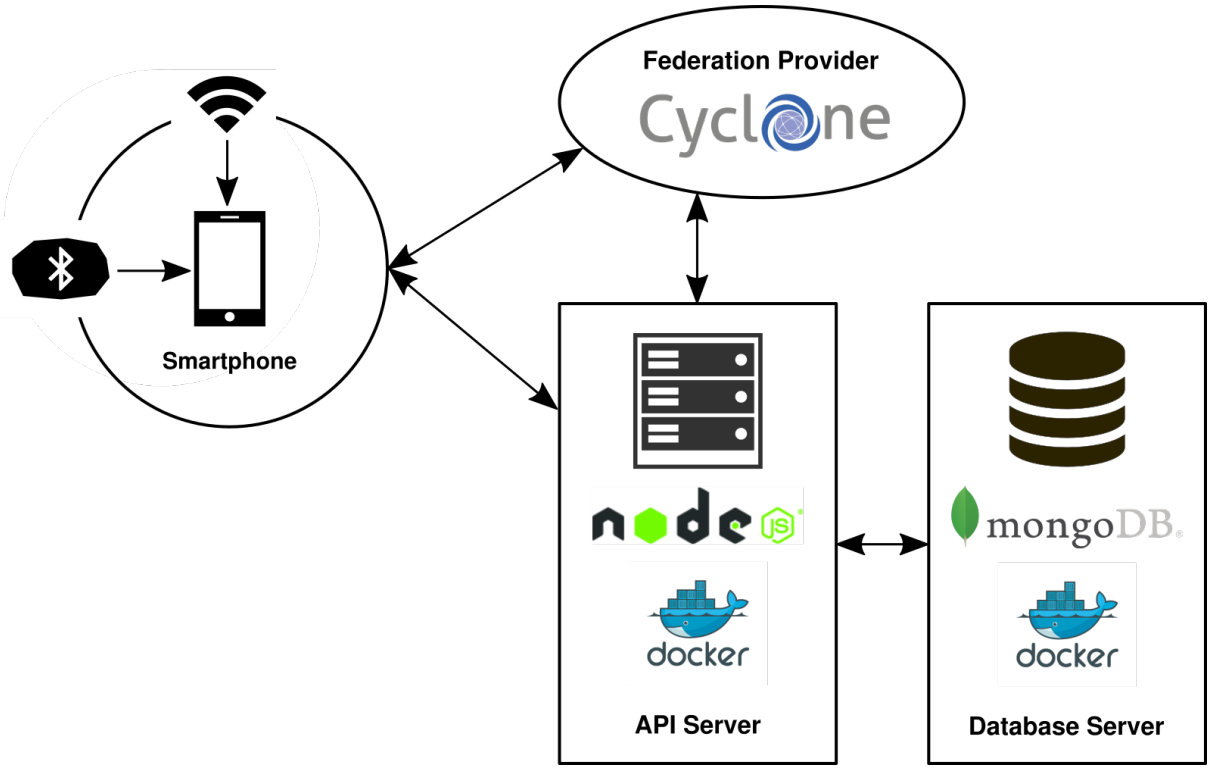
Since we planned to implement both Android and iOS client, we decided to use Apple Core Location Framework on iOS devices. Decision was supported by the fact that this new technology provides quality indoor location information using wildly deployed and available technology WiFi and that it may play an important role as indoor navigation technology in future releases of iOS operating system.

3 Concept and Design

3.1 Big Picture

As already mentioned our system consisted of two parts, the mobile clients side and the backend side which interconnectedly exchanged data. The backend part itself was split up again in three parts of which one was an "external" (means: SNET) resource, the CYCLONE Federation Provider. This entity provided us with user management and session handling functionalities so that we were able to out source these tedious and error-prone tasks to them. On the other side CYCLONE profited from our experiences with its rather young service. Concerning our part of the backend side, the task was to serve two machines independently, the API server and the database server. This was expected to be achieved with Docker.

To gain an understanding of what system we were trying to build, the following image should be at help:



All components of our envisioned system working together.

To sum it up, the idea was to gather position information aligned to the user's preferences locally on the smartphone, preform some processing steps on it, contact our backend for which authentication via the CYCLONE Federation Provider is needed and after that create, read, update or delete information (CRUD principle[1]) in the backend and therefore in the database.

3.2 API Considerations

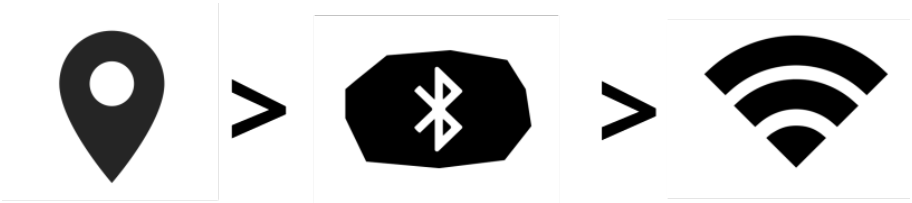
We had to decide on which paradigm our API should be based on. With the client-server and stateless nature of our envisioned system setup in combination with the just mentioned approach of relying on HTTP verbs such as POST, GET, PUT and DELETE for the CRUD operations, the decision to go for a RESTful API was quite clear ([7], chapter 5).

Furthermore a widely supported and light-weight format for interchanging data between clients and backend was needed, and it was decided to use JSON[2]. This also played nicely together with our requirement to implement the API server in Node.JS[3], using the Express web framework[4] which provided us a convenient way to define the RESTful resources of our API.

3.3 Workload Split Between Clients and Backend

Our system setup required a certain workload split between the two logical parties, the client side and the backend side. The modification of data on backend side was designed to be very transparent to clients but still some values submitted on the endpoints needed preprocessing to be done on mobile application side.

The general idea is to have the clients handle all direct user interaction with the service in a consistent, user-friendly manner. The other important task on client side is to gather the position information of the requesting user and performing prioritization logic on all available information. This was needed to deliver only the most accurate position to backend where it would replace the old value. The following position accuracy prioritization order was chosen:



Position prioritization order: pin-pointed better than Bluetooth better than WiFi position.

On backend side received data would then be sanitized, error and security checked and processed in its according request handler function. Therefore backend would handle everything concerning long term user state management, data persistence and API response aggregation.

[1] https://en.wikipedia.org/wiki/Create,_read,_update_and_delete
[2] http://json.org/
[3] https://nodejs.org/en/
[4] http://expressjs.com/

3.4 Authentication and Session Management

The CYCLONE Federation Provider provided us with the ability to integrate a well-tested user authentication method with only little effort into our service. CYCLONE is based on the JBoss Keycloak[5] project which in turn is based on the OpenID Connect standard[6], OAuth 2.0[7], JSON Web Token[8] and SAML 2.0. In the following a short overview of the involved technologies will be given.

The OpenID Connect standard provides two important functionalities in focus of our project. The first one is the addition of an identity layer on top of the OAuth 2.0 Authorization Framework that enables developers to reliable verify what person is using the authenticated service, no matter the used client, be it web or native applications. OpenID Connect does this without the need to maintain password storages on developers' side. Specifically, the CYCLONE Federation Provider uses the Authorization Code Flow[9] of the OpenID Connect standard. The other feature OpenID Connect brings into the project is that it already is built as a RESTful HTTP API based on JSON as the transport format and therefore perfectly integrated into the implementation and also provides the functionality to extend the specification in order to, for example, encrypt the transported identity data.

The OAuth 2.0 Authorization Framework is an IETF RFC [8] which introduces an abstraction layer, the authentication layer, in distributed web application environments. Therefore the standard is designed for the HTTP protocol and does not specify other protocols. OAuth 2.0 provides the ability to differentiate between resource owners (e.g. end users on a service, the resource server) and requesting third parties that need access to some or all resources of the resource owner. Third parties will never need to be authenticated by the resource owner's own credentials but will instead request an access token issued for specific scope, access duration and further attributes from an authorization server. This flow lets the resource owner be in full control of all allowed access requests from third parties, enabling her/him to eventually revoke granted access. Furthermore the attack vector on each involved participants in the service is isolated by separating all authorizations via the access tokens. An extensive threat model analysis on OAuth 2.0 can be found in [15].

JWTs (pronounced: "jots") are another IETF RFC [1] standardizing the transfer of a set of claims as a JSON object in a compact and URL-safe manner. The standard defines three parts of an JWT with the first one being the algorithm and token part, the second being the payload and the third used to verify the transported JWT. Encoding and concatenating these fields with dots yields the mentioned URL-safe representation predestined to be transported in HTTP Authorization headers or URI query parameters.

This authentication environment helped us to achieve multiple goals in our implementation. First of all, users were able to authenticate via their university account as it is part of the edugain collaboration for which CYCLONE acts as an federation provider. Therefore we had a single-sign-on mechanism available by simply integrating CYCLONE. The other major advantage was that we never saw the user's credentials used to authenticate. We were able to

[5] http://keycloak.jboss.org/
[6] https://openid.net/connect/
[7] http://oauth.net/2/
[8] https://jwt.io/
[9] https://openid.net/specs/openid-connect-core-1_0.html#CodeFlowAuth

rely on the well-tested user authentication code base provided implicitly through the layered mechanisms the CYCLONE Federation Provider contained.

3.5 Database Design

Part of the template we received at the beginning of the project from our supervisors at the SNET department was the integration of a MongoDB[10] as the database for our service. The communication with the database was not done directly via the officially supported Node.JS MongoDB drivers but instead via another layer in between, an object mapping library called Mongoose[11].

MongoDB is a representative of the class of the NoSQL databases. These kind of databases reject the traditional database approach of laying out the whole data landscape in a relational, table-based fashion prior to implementing an architecture. In the class of NoSQL databases again multiple different data models are used, including columns, documents, key-value stores, graphs or multi-modal databases. MongoDB itself is a document-oriented data store based on the "Binary JSON" (BSON, pronounced "bison") data format.

The NoSQL schema provides multiple features favourable for distributed, scalable environments such as big web applications are. NoSQL databases are generally simpler in design, are able to scale horizontally through shading and provide greater operation speed in some settings. Also MongoDB incorporates a replication mechanism for high availability in which sets of at least two representations of data exist, one being the primary, the other ones being secondary replicas. In case of a failure of the primary replica, the remaining members perform a leader election in order to retain availability.

In our case we defined models consistent to the representation of data in our architecture. The models included an `user`, a `location`, a `companionrequest` and a `hotspot` model which in turn got converted into MongoDB collections. Via Mongoose we were able to perform operations on these collections by invoking methods on such a specified model. Further discussion of our model structure will be given in chapter **??**.

Of course this setup did not only provide advantages but also yielded quite a lot of headaches during the project. We will be reviewing our issues with MongoDB, or more specifically Mongoose, in our chapter about the evaluation.

[10] `https://www.mongodb.org/`
[11] `http://mongoosejs.com/`

4 Implementation

4.1 Backend

4.1.1 Architecture

In this section we will focus on the architecture and our decisions on backend side and its communication. We built a RESTful API which allows clients to communicate with the backend via the JSON format. The figure 3.1 shows what the architecture consisted of. We had two different client applications which were communicating with the backend and a federation provider, further backend specific information about the federation provider will follow in chapter 4.1.7. The backend persisted the data into a MongoDB database. The SNET department provided us a virtual machine at the address `http://piazza.snet.tu-berlin.de`, where we could setup our environment.

We developed the backend with Node.JS[1] with the purpose that the SNET department could use or integrate it later into other projects which are mostly written in Node.JS. For our web application we used the Express web framework which influenced us in the structure of how our application is built. Our file structure consisted of five main directories:

- **Controllers** are the specific implementation for an API endpoint request.
- **Routes** link an endpoint to a controller.
- **Models** are schemas for the persistent data in the database.
- **Middleware** takes care of our authentication.
- **Tests** are provided for some of the important controllers.

4.1.2 Controllers

We designed four controllers which are also part of our API endpoints to handle all requests to our API. `companionrequests` handles the creation and modification of a companion request. This means that a user wants to add a colleague to her/his friends list and she/he asks her/his friend for permission. If she/he accepts it, both parties are added to each others friends list. If she/he denies it, the requesting user will be notified about the changed status. The controller `hotspot` gives back all information about the defined hotspots like mensa or library. This includes GPS coordinates, companyUUID, major and minor of the Estimote Bluetooth Beacons contained in the area. Here we want to mention one specific function of the `hotspot` controller.

[1] `https://nodejs.org/en/`

The `hotspots/.../active_friends` (the ...stands for a specific hotspot ID) searches in the current users friendlist all active users which shared the location with the current user. For retrieving or modifying user specific information like her/his location, groups and settings the `users` controller is the handler for these kind of requests. The last controller is the `login` controller which on successful login returns some information about the logged in user.

4.1.3 Routes

Routes specifies a specific endpoint for a defined URI[2]. Our routes link mostly with the same name to the controllers which we defined above. Since we have a RESTful API the Express web framework also allows us to define GET, POST, PUT and UPDATE functionality. All endpoints are secured via our middleware which we will describe in section 4.1.5.

4.1.4 Models

Mongoose[3] is a powerful plugin for Node.JS which allows us to have an easier connection to the MongoDB database. For this we have to define schema's for our models. So we defined the four schemes `companionrequest`, `hotspot`, `location` and `user`. But not every information has to be an extra collection in our MongoDB. We defined sub-documents for those information which do not need an extra collection. For example the beacons are always in a hotspot so there is no need for an extra collection.

4.1.5 Middleware

The middleware is provided by the CYCLONE project which handles the authentication and session management to the federation provider. If the user does not exist in our database the authentication middleware will add her/him with the information which it gets from the federation provider. This includes the name, email and the ID which is also given from the authentication token of the user. We overloaded this middleware for the test section below. This allowed us to have a valid authentication against the database for testing purposes.

4.1.6 Test

We wrote tests for the `users/me`, `users/me/groups` and `hotspot` endpoints. In Node.JS a popular test framework is supertest which allows tests for HTTP GET, POST, PUT and UPDATE functionality. We combined that with the chai framework which is a BDD (Behavior Driven Development) / TDD (Test Driven Development) assertion library. For tests we had to overload the middleware because we could not test against the federation provider with a dummy user. With the command **npm test** all ten written tests will start to be executed. Our ten tests are the following:

1. delete the user
2. get the login page

[2] Uniform Resource Identifier
[3] `http://mongoosejs.com/`

3. update user information
4. create a group with the name “Some Groupname here”
5. get the created group back
6. update the group name
7. delete the created group
8. get hotspots back and check structure of hotspot, also inner structures
9. get one specific hotspot back
10. get one specific hotspot and there beacons back

4.1.7 CYCLONE Federation Provider

Authentication and session management is necessary in mostly every software project that deals with the web. But in our case we included the CYCLONE Federation Provider which lead to less development overhead for our project. Because the federation provider takes care of the authentication of a user. It also gave us the ability to integrate a single-sign-on solution. We will focus in this section mostly on the including development part and not the concept of the CYCLONE Federation Provider which is written in chapter 3.4.

On development side we dug into a bug in the implementation of `keycloak-nodejs`. But thanks to Mathias Slawik, who fixed it fast, the integration of Keycloak was quickly done. The bug was that the redirection of a not yet logged in user was broken. But after this we secured each endpoint with the `authenticate` function of Keycloak which redirects the user automatically to the federation provider’s login page (figure 4.1). After a successful login we received the **user ID**, **user name** and **user mail** in a `JSON Web Token` wish we persisted in our MongoDB. In the same step we also created a default group called “All friends” and saved it, too. Into this default group all future friends of the user will be added.

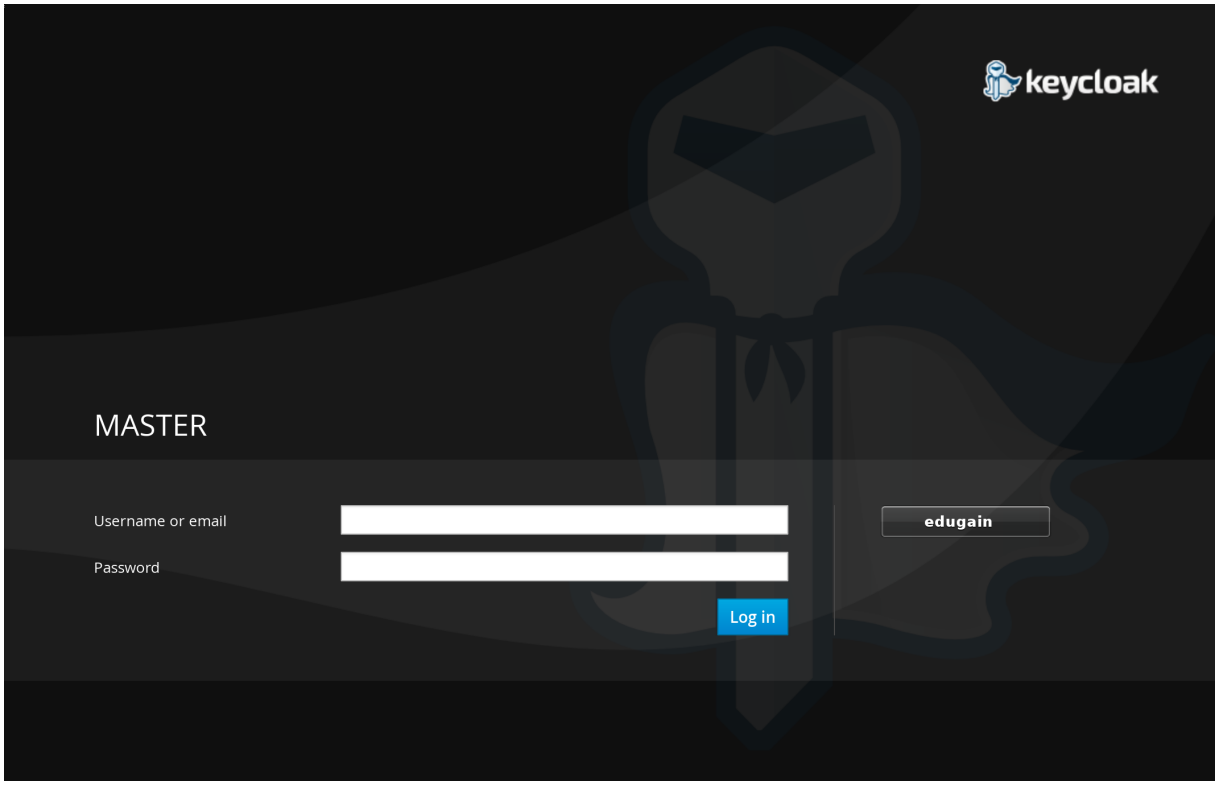


Figure 4.1: Login screen to CYCLONE Federation Provider.

4.1.8 Deployment

The SNET department wished to have an easy way to deploy our software. So we looked into the recommended Docker[4] application. Docker is an open source application which automates deployment of software applications in Linux containers. One of the biggest advantages of Linux containers in Docker is that you have an isolated Linux container for each application. Another positive effect of Docker is the Dockerfile. This allowed us to define how the software should be deployed and also the dependencies in the application. Our software depends on the MongoDB database so we also decided to have the MongoDB as an own Linux container, because we were not sure where the database will run in a production environment. Docker gave us the ability to define a sequence of how to start the created containers, because the database has to run before our application starts. For this we made use of Docker Compose and added a docker-compose.yml file which contained all necessary settings for the containers to start in the desired order.

4.2 Android

Our Android application enables the user to manage companions, automatically and manually retrieve the user's indoor position using different localization techniques available, share position with friends and show friend's positions on indoor maps. The application currently

[4] https://www.docker.com/

includes indoor maps of the mensa in Hardenbergstraße at TU Berlin.

Our application requires Android version 5.0 or higher. It uses API level 21. The application was tested on Nexus 4 and Nexus 5 devices.

4.2.1 Floor Plans

The application shows floor plans as overlays over Google Maps. We use floor plans in the PDF format provided by *Studentenwerk Berlin* for purpose of this project. PDF files were converted to PNG files and shown on top of Google Maps using *GroundOverlay* class. Different floors are represented with multiple floor plans.

4.2.2 Login

To authenticate with the backend server, the application has to send an authentication token with every request it sends. The application retrieves an authentication token during login procedure. Login is done with the federation provider project. During login procedure, the user is shown a login page in web view inside the mobile application. After successful login the application stores the retrieved authentication token in *shared preferences* and uses it for future requests.

4.2.3 Communication with Backend Server

Our application communicates with the backend server using RESTful requests. For performing RESTful requests the application uses the *Volley* library.

4.2.4 Menu View

Menu view is the main screen of application as shown in figure 4.2. It enables user to open a map of a hotspot (1), show groups (2), add companions (3) and manage companion requests (4). User can add friends using email address as shown in figure 4.3.

4.2.5 Retrieving User's Position

4.2.5.1 CISCO MSI API

Retrieving position from CISCO MSI API provided by tubIT is implemented as a background service. If mobile device has WiFi enabled and it is connected to *eduroam*, service scans the API in 30 seconds intervals. API response is processed and building name and floor information are stored in *LocationSharingSingleton* instance.

4.2.5.2 Estimote Beacons

Searching for nearby Estimote Beacons and Stickers is done using ranging method and nearable discovery method provided by Estimote SDK. All detected Beacons and Stickers are rep-

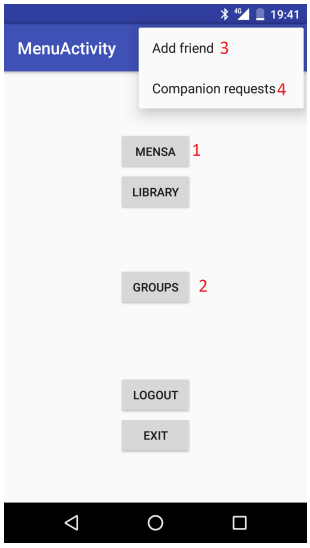


Figure 4.2: Menu friend

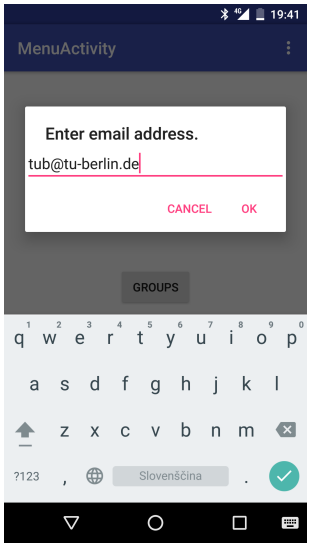


Figure 4.3: Add friend

resented as internal *Beacon* class. Closest beacon is determined as the one with the best signal strength (RSSI). Information about detected beacons is then stored in *LocationSharingSingleton* instance.

4.2.5.3 Manual Position Pinning

User can manually pinpoint position on the indoor map of selected hotspost. With checkbox *Share position* (1) user enable manual pinpointing. Position can then be set with marker (2). Manual pinpointing is shown in figure 4.4. Pinpointed coordinates and floor information are stored in *LocationSharingSingleton* instance.

4.2.6 Sharing User's Position

Sharing user's position is implemented with a background service. Every 15 seconds service sends the most accurate location data stored in *LocationSharingSingleton* instance to the backend server. Determining the most accurate position goes as follows; most accurate are manually pinpointed coordinated and floor information, followed by beacon identifier that defines a region. The least precise is the MSI retrieved building name and floor information.

4.2.7 Showing Friends

Friends in hotspots are shown on the indoor map as shown in figure 4.5. For different accuracies different markers are used. Pinpointed position is shown with single marker (1). Position shared with beacon ID is shown with marker surrounded by circle representing the region defined by beacon signal strength (2).

Figure 4.4: Manual pinpointing

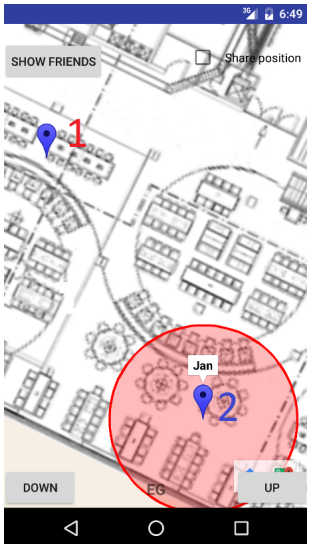


Figure 4.5: Showing friends

4.2.8 Future Work

Implemented client application provides basic functionalities for indoor positioning. Proposed future work suggest the functionalities that should be considered in further development of the project.

Remote notifications
: Backend and could implement sending remote notifications to inform mobile clients about events such as friend entering hotspot.

Friends and group Management
: Current application enables user to list all available groups and friends inside them. Adding and deleting friends is already implemented. Group management is already supported on backend, so mobile application should implement options to add, edit and delete groups and to assign user to one or more groups.

4.3 iOS

System Requirements:

- **Operating System:** Version: iOS9; Build:13A340
- **Screen Size** 640 x 1136 pixels (326 ppi pixel density); 4 inches
- **Bluetooth:** v4.0, A2DP, LE
- **Internet Technology:** Wi-Fi

The iOS application provides the user with a map of the specific hotspot. For this project the example of indoor navigation inside the mensa has been implemented. The app shows both floors of the mensa, together with additional information about the location.

4.3.1 Floor Plans

In order to show positions of the user and friends with the exact coordinates on the map, the floor plan material is added as MKOverlay on top of the MKMapView provided by Apple. We use the PDF files provided by *Studentenwerk Berlin* for this project.

Apple also provides registered iOS developers with a framework to manage the mapping of x/y coordinates of the PDF file with the exact Latitude/Longitude real world coordinates of the map **(see Figure: 4.6)**.

This framework has been used for this project to integrate the floor plan of the mensa [10]. The GeoAnchor class provided by Apple converts each corner of a PDF rectangle into a MKMapPoint. The collection of MKMapPoints is combined to a MKPolygon which is used to draw elements and annotations on top of a map. This technology is used to divide the floor plan material into multiple map-tiles that can be added to the map **(see Figure: 4.7)**.

4.3.2 Login

In order to use the application, the user has to login via Federation Provider, to get a security token which grants access to the application server.

The app handles the login process using a UIWebView. The login request will then be redirected to the tubIT login page **(see Figure: 4.8)**, in which the user finally can type in tubIT username and password. If the user credentials are correct, the web view gets redirected to the piazza application server.

If the web view was able to access the application server successfully, the web view closes and grants the user access to the application. The web view can not be bypassed without a successful login via Federation Provider **(see Figure: 4.9)**. Each request to the application server re-opens the web view again, if the used security token is invalid or if the user has been logged out.

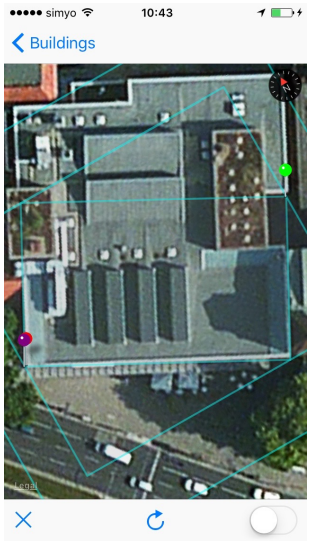


Figure 4.6: Defined Anchor Points and Boundaries

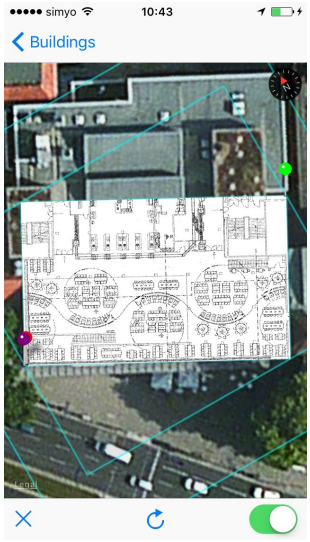


Figure 4.7: Applied Floor plan to the defined Boundaries

4.3.3 Hotspots/ Buildings

The Buildings View **(see Figure: 4.10)** is the first view the user sees after login. This view enlists all available hotspots delivered by the backend.

1. The reload button, calls the backend to get a list of available hotspots. The list contains the hotspots and the available beacon and floor information.
2. The Table-view lists all available hotspots from the backend. Each cell contains the name of a Hotspot.
3. The Application is divided into three contextual distinct parts: Buildings, Friends and Groups. This subdivision is implemented in a UITabBar which is the leading element of the application architecture.

4.3.4 Indoor Positioning

If the user taps on one of the buildings enlisted, the app opens respectively the corresponding floor plan in an MKMapView **(see Figure: 4.12)** that shows the mensa floor plan on a large scale. This improved scale functionality was added additionally in order to increase the usability and accuracy of manual pin-pointing.

1. The the map view shows the first floor of the mensa by default, if the tubIT MSE API which provides the floor and building information is not available. In cases of availability of the tubIT MSE API, the map view shows the corresponding floor as soon as the map view appears on the screen.
2. The manual pin-pointing of the user's position is triggered by a so called long press on the map which was additionally implemented. The long press is a distinct user-interaction mode. It is applied as solution for manual pin-pointing on iOS in order to prevent the user

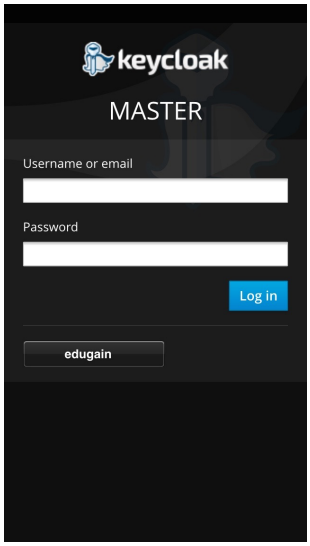


Figure 4.8: Keycloak/Federation Provider

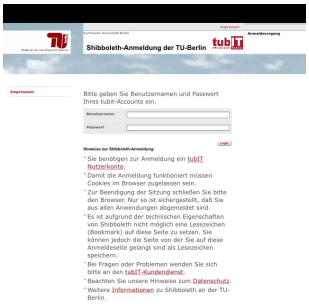


Figure 4.9: Tubit Login Page

from mistakenly share positions by inadvertently touching the map. After a long press is registered by the system, it drops a red pin on the position where the user pressed. This switches the Map-view into a share position mode. In this mode the user can touch on any place on the map to easily edit or remove the pin, if the position to share is not correct.

3. The information panel is used to give the user the possibility to change the floor by pressing the plus (up) or minus (down) button. The information panel also flips with an animation, signalling the user of the changed mode of the map view. The flip animation is triggered as soon as the system successfully registers a long press. In Share Position Mode the information panel prompts the user to share the location.
4. The file button modally opens a settings view with further options.

Figure: 4.13 shows the map view after a user has successfully shared the position either via Bluetooth beacon, automatically using the Apple CLLocation Framework or manually via pinpointing.

1. As soon as the backend has received the new location of the user successfully, the red pin is exchanged with a marker annotation. This marker shows the provided synchronized location of the user. The switch of the marker gives the user a hint, that his position is now synchronized with the server state. This marker is also used for the positions of friends, however the user position is highlighted with a star.
2. If the user taps on his own marker, it reveals an annotation view that includes the user name as well as a remove button that deletes the shared position from the server permanently (in this project the username “Synchronized Me” was used for debugging).
3. The information panel is now flipped back to normal mode or to Synchronized Position Mode if the user previously shared the position manually.

The settings view (**see Figure: 4.14 and 4.15**) provides additional options to indoor position-

Figure 4.10: List of available Hotspots

Figure 4.11: Access Location Request

ing and information markers that are shown on the floor plan.

1. **Show all Friends** activates markers for friends in the same hotspot. **(Note: Friends will only be shown on the map if the user has shared his own position before).**
2. **Automatic Update** activates a routine that constantly requests the server for location updates of friends.
3. **WiFi Indoor Positioning** activates the CLLocation Framework positioning. This will automatically update the user position while moving through the building.
4. **Bluetooth-Positioning** activates location sharing via Bluetooth beacons.
5. **Show Beacons/Regions** activates annotations showing beacons placed in the hotspot.
6. **Show Specific Hotspot-Information** activates annotations showing specific hotspot informations.

If the user activates **Show all Friends** on settings and has already shared the own position, the map-view adds markers for each friend located in the same hotspot. The friend markers are equipped with Accuracy-Circles placed underneath the marker. The radius of the Accuracy-Circles decreases with the accuracy of the shared position. If a position is manually shared, no Accuracy-Circle will be shown. However, if the position is shared via Bluetooth or WiFi, the Accuracy-Circle increases to show the region the friends position **(see Figure: 4.17)**.

If the user activates **Show Specific Hotspot Informations** on settings, the map-view will show the user specific informations regarding the Hotspot such as Cash-register or Foodstands. This functionality has been added in order to improve the usability of the floor plan and helps the user to orientate using the map while navigating in the building **(see Figure: 4.16)**.

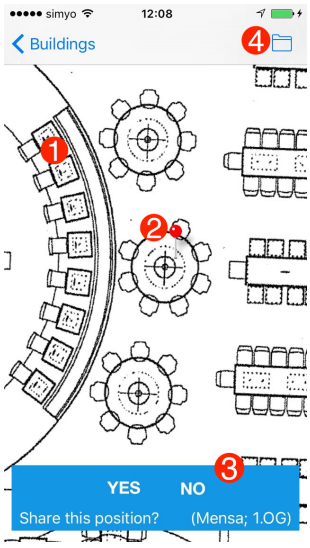


Figure 4.12: Share Position Mode

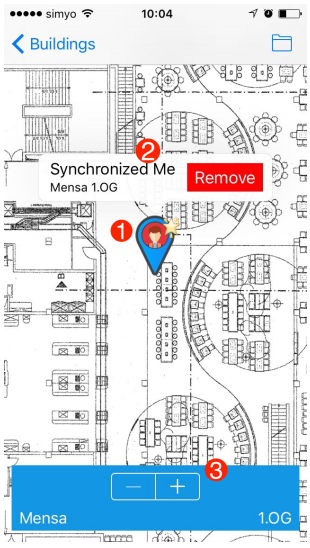


Figure 4.13: Synchronized Position Mode

4.3.5 Friends

As soon as the user taps on the All Friends button of the TabBar, the application shows the All my Friends List **(see Figure: 4.18)**. This view shows either the friends of the user or opens companion requests that the user needs to accept in order to add a person into his friends list. Note that all available friends will be shown in this list regardless of their group affiliation.

1. The Reload-button is implemented in order to update the list of friends or new companion requests. In future work the server should be able to send remote notifications in order to inform the user of new companion requests.
2. The Plus button is implemented to open a new view **(see Figure: 4.19)** where the user can add new friends and send new companion requests.
3. The list-view is managed by a UISegmentControl. The content is divided into available friends and available companion requests which are shown by numbers. A tap on one of the segments will either change the content of the Table-view to friends or companion requests. The entry of the list contains the provided name of a friend.

As seen on **(Figure: 4.19)**, the app opens modally a new view that enables the user to add new friends.

1. The UITextfield is used in order to enable the user to type in the email address of a friend. This request will be handed by the backend. The response of the backend about the success or fail of the request will be directly shown underneath the textfield.

4.3.6 Groups

The backend enables the user to manage groups. Each user has a group named All friends, containing all of the users friends. Additionally the user may have different friends that can be added to own groups. These groups will be enlisted in this Table View as shown in **(see**

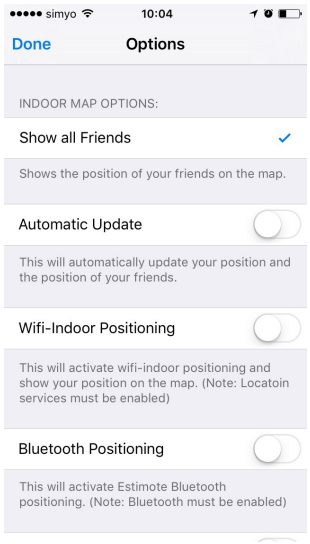


Figure 4.14: Map Settings 1

Figure 4.15: Map Settings 2

Figure: 4.20).

1. The plus-button will open a new view modally that enables the user to create a new group (**(see Figure: 4.21**).
2. This tableview shows all available groups of the user. Each cell contains of the Name of the group together with the GroupID as subtitle. For future work if the user taps a group, it will open a new list showing the members of this group. The user should also be able to add specific friends to a group as well as edit or delete a group.

4.3.7 Geofencing

The iOS client supports Geofencing abilities for registered hotspots. Geofencing notifies the app when its device enters or leaves the geographical region of a hotspot. For this project the triggered notification greats the user and acts as a reminder to use the app if the user enters the mensa. The notification can also be used to take the user in charge of the decision whether he wants to download updates about the hotspot. The notification will also act as a reminder to deactivate Bluetooth if the user leaves the mensa to save battery **(see Figure: 4.22 and 4.23**).

4.3.8 Future Work

The iOS client build for the project shows a proper solution of mobile indoor navigation backed with a scalable backend system. However some functionalities may be added to the client as they have been out of scope to this project.

Remote Notifications
: Remote notification handling is state of the art in mobile applications of these days. It should be considered to equip the backend with the ability to notify a user if one of his friends shared his position in a specific hotspot, instead of constantly polling for updates.

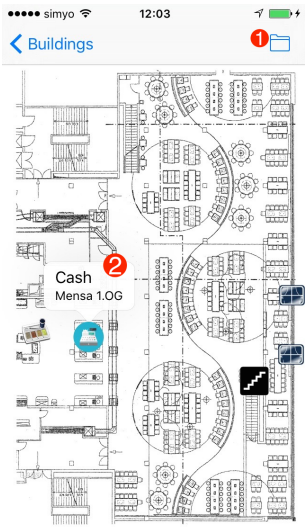


Figure 4.16: Additional Hotspot Informations

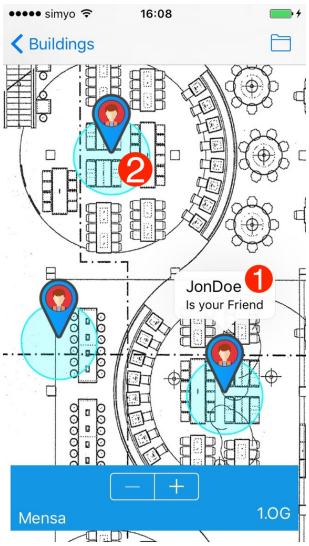


Figure 4.17: Friends in this Hotspot with Accuracy

Remote Notifications would also be used to notify the user for new companion requests. Also scheduled polling for hotspot updates is not a best practice and should be exchanged by a publish-subscribe pattern.

CLLocation Framework

In future work, the new CLLocation framework should be considered as first choice technology for indoor navigation on iOS devices. Authorities of locations that are interested in indoor navigation on iOS should consider an enrolment in Apples Map Connect program [13]. With a successful enrolment, floor information as well as the accuracy of the users position would be improved.

Friends/Group Management

In future work, the client should give the user more possibilities to manage his friends as moving friends to specific groups or edit groups. The map should also be equipped with the possibilities to show specific groups instead of only all friends in a hotspot.

Figure 4.18: Available Friends and Open Requests

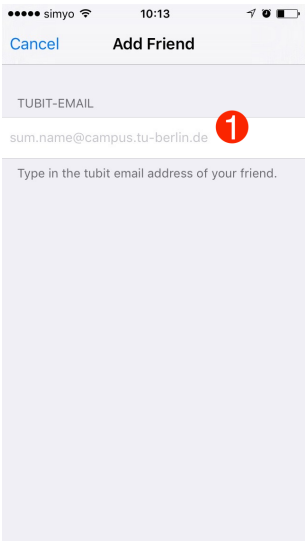


Figure 4.19: Add Friends

Figure 4.20: Show available groups

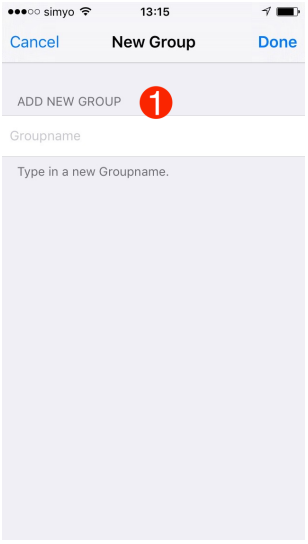


Figure 4.21: Add new group

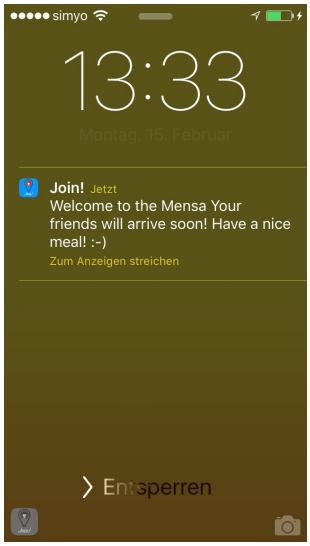


Figure 4.22: Greating on entering a Hotspot

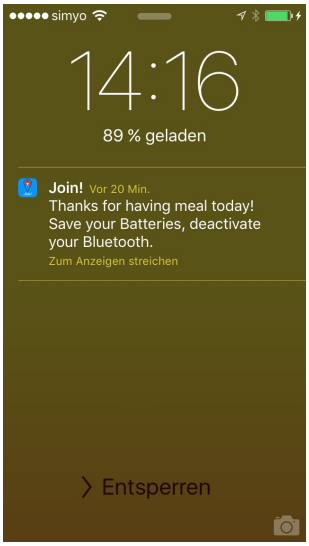


Figure 4.23: Reminder on leaving a Hotspot

5 Evaluation

The CISCO MSE's lack of providing user coordinates was a huge drawback for this project. This disadvantage is due to the not completed installation of the API by tubIT. The information the MSE API provides has been shown as only these that the user is already aware of while being at the location, instead of coordinates which has not been aware for indoor positioning.

The use case to provide the service to TU Berlin students with access to eduroam restrained the efforts of taking more development time into the actual part of indoor mobile client based navigation. Having access to eduroam via Federation Provider was mandatory in order to get access to the MSE API. However the implementations of the backend together with the multiple issues faced while working with the Federation Provider had a huge time consuming impact into the backend work and so to the whole project.

The project requirements to automatically download and update hotspot information from the server once a week did not pay off as expected. A user without an ongoing session on the server would be forced to actively open the app to login into the Federation Provider in order to get access to the servers.

An early renunciation of the Federation Provider as well as the CISCO MSE requirements would have given more possibilities to focus on the indoor navigation it self as the main idea of the project.

The project requirements to implemented a permanently polling mobile client to update active friends positions in a building, instead of implementing a remote notification scheme that works on update of new positions, forces the clients to download data, multiple times per minute even though the retrieved data is redundant. This client behaviour might get even worse since the client must also automatically upload its position on the server repeatedly at the same time. Additionally the client must also query the MSE API multiple times per minute for information that are redundant if the user is not even changing the floor.

The excessive data workload combined with monitoring of Bluetooth beacons tends to a huge impact on especially older mobile clients batteries, causing a massive drain that could empty the battery even before the user has finished his meal.

Nevertheless the project benefits from the usage of the federation provider in that way, that the service can be now provided to multiple universities in Europe. The mobile client also does not handle or store user credentials locally which is a good security approach.

The Estimote beacons have indeed been shown as very useful to locate the user indoors without having coordinates. The combination of ranging and monitoring methods offered by the Estimote SDK enables the clients to detect available beacons and share the position of the

beacon regarding the closest range. This technology would also be useful in future work if the mensa would be equipped with multiple Stickers which are cheaper than Beacons. Using BLE did not reveal itself as extremely battery draining on the client side, rather than on the beacon side. The latter one has constantly been an issue through the whole project.

While monitoring and ranging beacons, the system does not give the user active feedback about a current state. This system specific behaviour could not have been altered on either clients. The delays of detecting a beacon and recognizing the unavailability of a beacon had an negative influence on usability. This is shown in cases where the user is not informed if a beacon is already detected, how long it takes until the correct beacon is detected or which beacon has been shared to the server.

Against our expectations, the manual sharing of positions has revealed it self as most accurate and user friendliest way of sharing positions. It is more accurate, since the user exactly decides which position is shared rather then the system. The user is also in charge to delete his shared position from the servers when he wants. The direct feedback of interaction, and the possibility to actively engage into the process instead of relying backend or automated services improves the users experience in indoor navigation.

We have been successful in implementing floor plans on Android and iOS that indeed helps a user to find friends and join them in the mensa. Our clients have shown that this technology is now also possible indoors as expected for comparable applications outdoors.

The Apple indoor Framework on iOS has been shown as a proficient solution for indoor navigation. Although the framework used for this project leaves some serious issues with memory management that can lead to crashes on older devices it shows that it is not yet ready for a release. However in this case it is, regarding the fact that there is more to come from, a really good solution to show the users position in an indoor location.

At the backend we tried to implement some geofencing possibilities because the mongoDB should offer us some of these functionalities but the mongoose framework was not implementing it right and we discovered some bugs that hindered us to implement it. The backend technology of nodejs was quite interesting because we had no experience with an asynchronous programming language but our learning curve was high and it would have been possible that we implement functionalities which are now future work. The CYCLONE Federation Provider was a nice project to include in our work but sometimes we had some struggles with them but mostly to get specific information from the tubIT which we didn't get until now.

In total we did not get everything implemented what we thought we could do at the beginning but we got the main functionalities of sharing a position manually and automatically working on two different devices including the communication to the backend.

6 Conclusion

In our project we focused on locating people in indoor environments. We firstly defined a use-case to state the importance of our work and provide motivation. We tried to constrain our project so that the final solution could be used by as many different users as possible. Our typical user is a TU student or employee with a WiFi and Bluetooth enables smart phone in their pocket.

To solve problem of positioning people in indoor environments, we firstly made a research on available indoor location technologies. We chose the technologies that were available by the TU infrastructure and by smart phones of our typical users.

Our indoor positioning solution consists of backend application and client applications. Backend application handles user authentication, persisting of data and location data exchange between users. Client applications were developed for Android and iOS mobile devices. They provide functionalities of automatically locating users, sharing their locations and showing their friends on indoor maps.

Implemented indoor positioning solution fulfilled all major goals that we defined at the beginning of the project. We successfully implemented automatic indoor location retrieving, with manual fall-back option, sharing of retrieved location with friends and showing friends on indoor maps.

Overall, the topic of indoor navigation turned out to be very interesting one from both research and implementation perspective. It helped us to get to know the ropes of indoor navigation and understand the user's and the technology perspective of it.

6.1 Future Work

One of the scenarios we prepared at the beginning of the project defined the case, in which a user shares his location with only a specified subset of his friends. Implemented backend application already provides functionalities to create and modify different user groups and to classify friends into them.

Functionalities to manage user groups and to classify friends into them are not jet fully implemented in both client applications. Future work should implement an intuitive and user friendly way to achieve that.

Furthermore, when sharing his position, user should be able to select a group with which he want his position to be shared. This functionality should become part of client applications. Backend application should extend its functionalities so that users location is shared only with

groups of friends he specified and not with all of his friends.

List of Figures

Bibliography

[1] John Bradley, Nat Sakimura, and Michael Jones. "JSON Web Token (JWT)". In: (2015). URL: https://tools.ietf.org/html/rfc7519.

[2] Allan Brimicombe and Chao Li. *Location-based services and geo-information engineering*. Vol. 21. John Wiley & Sons, 2009.

[3] Solomon Chan and Gunho Sohn. "Indoor localization using wi-fi based fingerprinting and trilateration techiques for lbs applications". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 38 (2012), p. 4.

[4] Veronika Doljenkova. *Beacons: Exploring Location-Based Technology in Museums*. Mar. 2015. URL: http://www.metmuseum.org/about-the-museum/museum-departments/office-of-the-director/digital-media-department/digital-underground/2015/beacons (visited on 03/30/2015).

[5] Nick Farina. *Why indoor navigation is so hard*. Nov. 2011. URL: http://radar.oreilly.com/2011/10/indoor-navigation.html (visited on 11/12/2011).

[6] Silke Feldmann et al. "An Indoor Bluetooth-Based Positioning System: Concept, Implementation and Experimental Evaluation." In: *International Conference on Wireless Networks*. 2003, pp. 109–113.

[7] Roy Thomas Fielding. "Architectural styles and the design of network-based software architectures". PhD thesis. University of California, Irvine, 2000. URL: http://jpkc.fudan.edu.cn/picture/article/216/35/4b/22598d594e3d93239700ce79bce1/7ed3ec2a-03c2-49cb-8bf8-5a90ea42f523.pdf.

[8] Dick Hardt. "The OAuth 2.0 authorization framework". In: (2012). URL: https://tools.ietf.org/html/rfc6749.

[9] Robin Heydon. *Bluetooth Low Energy: The Developer's Handbook*. Prentice Hall, 2012.

[10] Apple Inc. "Core Location Framework". In: (2015). URL: https://developer.apple.com/library/prerelease/ios/samplecode/footprint/Introduction/Intro.html.

[11] Apple Inc. "Core Location Framework Reference". In: (2015). URL: https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CoreLocation_Framework/.

[12] Apple Inc. "Getting Started with iBeacon". In: (2014). URL: https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf.

[13] Apple Inc. *Maps Connect*. Feb. 2015. URL: https://mapsconnect.apple.com (visited on 02/09/2016).

[14] Estimote Inc. *Developer Docs*. Feb. 2015. URL: http://developer.estimote.com (visited on 02/09/2016).

[15] Torsten Lodderstedt, Mark McGloin, and Phil Hunt. "OAuth 2.0 threat model and security considerations". In: (2013). URL: https://tools.ietf.org/html/rfc6819.

[16] Vitali Lovich. "Building an Indoor Application WWDC 2015". In: (2015). URL: https://developer.apple.com/videos/play/wwdc2015-714/.

[17] American Museum of National History. *Explorer Mobile Application*. Mar. 2015. URL: http://www.amnh.org/apps/explorer (visited on 03/30/2015).

[18] IoSL-INav team. *Project repository of IoSL-INav*. Feb. 2016. URL: https://github.com/IoSL-INav/project (visited on 02/09/2016).

Appendices

1 PDF Floor plans Mensa

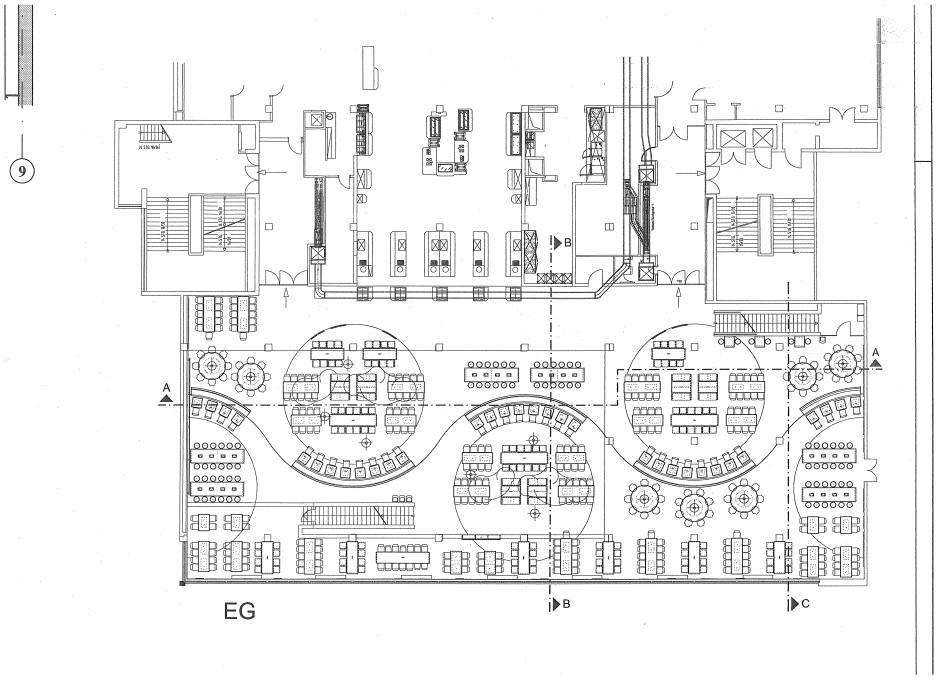


Figure 1: Floor plan Mensa EG

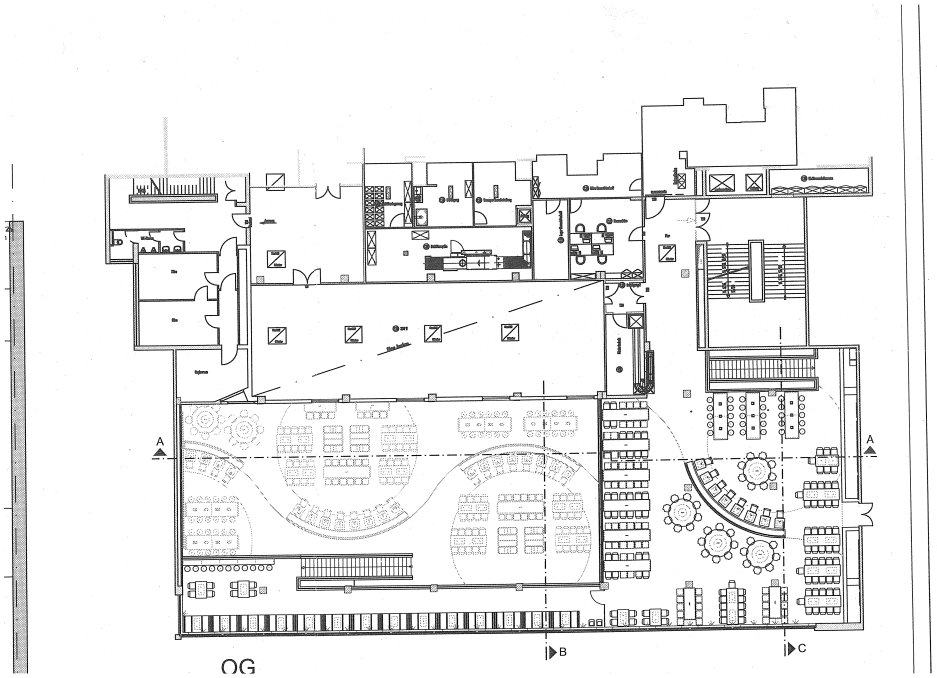


Figure 2: Floor plan Mensa OG

2 Tool for performing tests on the CISCO MSE API wrapper

```
1  package main
2
3  import (
4      "fmt"
5      "log"
6      "os"
7      "syscall"
8      "time"
9
10     "encoding/xml"
11     "io/ioutil"
12     "net/http"
13     "os/signal"
14 )
15
16 type WiFiInfo struct {
17     XMLName           xml.Name `xml:"Info"`
18     ChangedOn         string   `xml:"changedOn,attr"`
19     ConfidenceFactor  float32  `xml:"confidenceFactor,attr"`
20     Building          string   `xml:"building,attr"`
21     Floor             string   `xml:"floor,attr"`
22     Network           string   `xml:"WLAN-Status,attr"`
23     UserName          string   `xml:"username,atrr"`
24     Longitude         float64  `xml:"lon,attr"`
25     Latitude          float64  `xml:"lat,attr"`
26 }
27
28 type LogWiFiStruct struct {
29     Timestamp int     `json:"timestamp"`
30     Latitude  float64 `json:"latitude"`
31     Longitude float64 `json:"longitude"`
32     Building  string  `json:"building"`
33     Floor     string  `json:"floor"`
34 }
35
36 func handleUserExit(signalChannel chan os.Signal) {
37
38     for _ = range signalChannel {
39
40         // Define a useful result file name (format: "wifi-measurement-year-
               month-day-hour-minute-seconds.json")
41         fileName := fmt.Sprintf("wifi-measurement-%s.json", fileNameTime)
42
43         // Open measurement result file
44         fileResult, fileError := os.OpenFile(fileName, os.O_CREATE|os.O_RDWR|os.
               O_APPEND, 0666)
45
46         if fileError != nil {
47             log.Fatal(fileError)
48         }
```

```
49
50          // Close open logging file
51          defer fileResult.Close()
52
53          log.Printf("\nWritten measurement values to file %s. Good bye.\n",
                fileName)
54
55          os.Exit(0)
56      }
57  }
58
59  func main() {
60
61      // Put in here the API endpoint to the wifi location service.
62      const apiURL = "PUT THE API ENDPOINT IN HERE"
63
64      // Get time data
65      startTime := time.Now()
66      fileNameTime := startTime.Format("2006-1-2-3-4-5")
67
68      // Define a clean up channel
69      signalChannel := make(chan os.Signal)
70      signal.Notify(signalChannel, os.Interrupt, syscall.SIGTERM)
71      go handleUserExit(signalChannel)
72
73      log.Printf("Starting to measure WiFi API.\n")
74
75      for {
76
77          // Make a GET call on that URL
78          apiResp, apiError := http.Get(apiURL)
79
80          if apiError != nil {
81              log.Fatal(apiError)
82          }
83
84          // Read in all body content we got and close connection
85          xmlData, ioError := ioutil.ReadAll(apiResp.Body)
86          apiResp.Body.Close()
87
88          if ioError != nil {
89              log.Fatal(ioError)
90          }
91
92          // Our go struct representation of the tubIT XML
93          wInfo := WiFiInfo{}
94
95          // Parse received XML into struct
96          xmlError := xml.Unmarshal([]byte(xmlData), &wInfo)
97
98          if xmlError != nil {
99              log.Fatal("XML parsing error: %v.\n", xmlError)
```

```
100         }
101
102         // Get the current UNIX epoch timestamp for logging
103         timeResult := time.Now().Unix()
104
105         // Build up the log string
106         logResult := fmt.Sprintf("\t\t{\"timestamp\": %d, \"latitude\": \"%.15f
                \", \"longitude\": \"%.15f\", \"building\": \"%s\", \"floor\": \"%s
                \"},\n", timeResult, wInfo.Latitude, wInfo.Longitude, wInfo.Building,
                 wInfo.Floor)
107
108         // Write log string to opened file
109         fileResult.WriteString(logResult)
110
111         log.Printf("Received: \"long\": %v, \"lat\": %v.\n", wInfo.Longitude,
                wInfo.Latitude)
112
113         // Let the execution wait for 2 seconds
114         time.Sleep(2 * time.Second)
115     }
116 }
```