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 13, 2016

Supervised by:
Sebastian Zickau and Mathias Slawik

Abstract

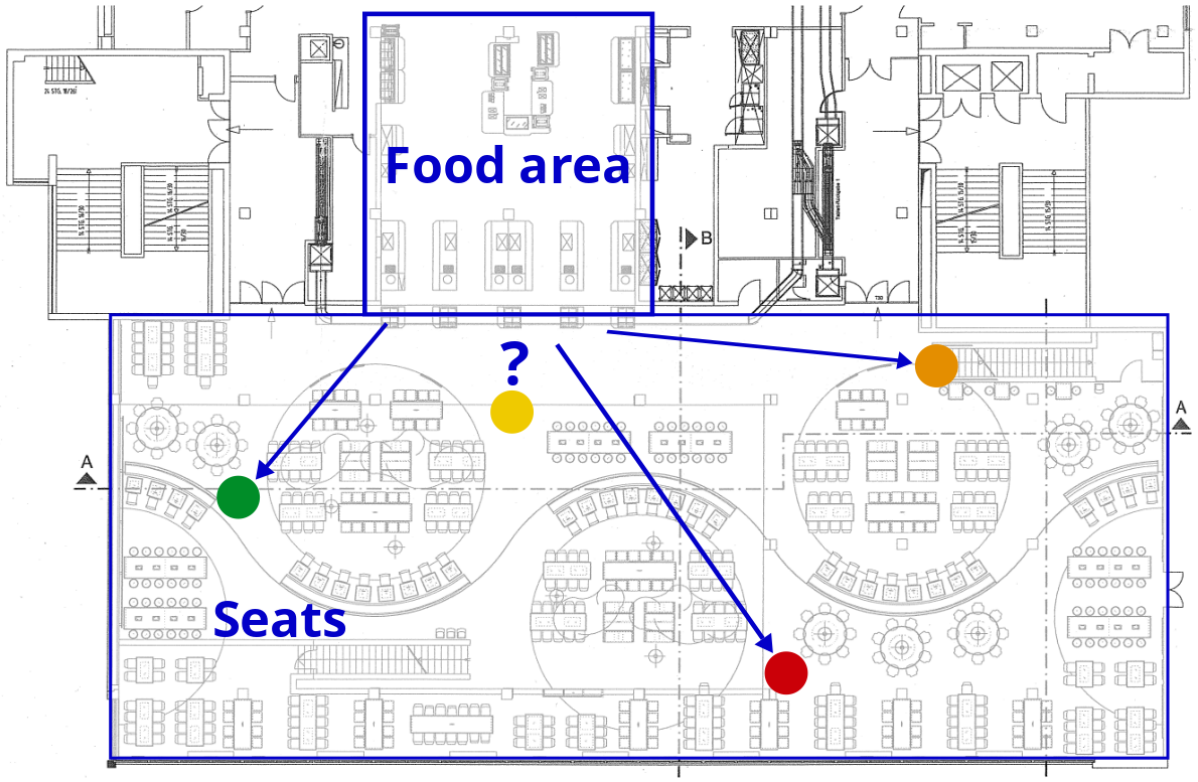
Short summary of project outcome.

Contents

1 Introduction

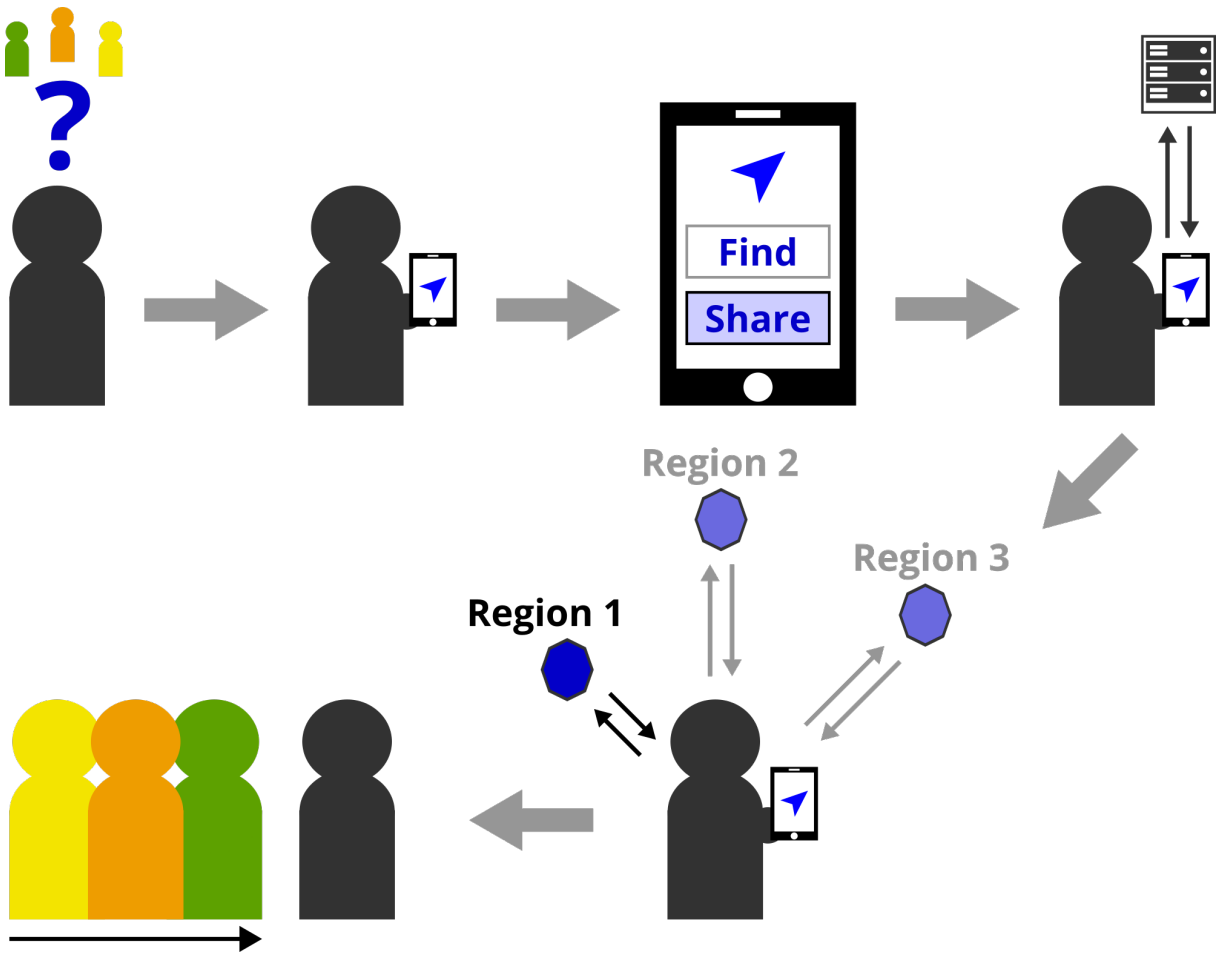
1.1 Motivation

The mensa in Hardenbergstraße at TU Berlin.



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

Imagined use of our app on mensa tray.



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. We do not, though, require the latest operating system versions. For Android, version 5.0 is required, for iOS, version X is required.
- **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[1]. Via this service users with an edugain (and therefore a TUB) account are able to authenticate against our backend.

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

- **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 striclty 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 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 [13]. They combine two methods in order to calculate a precise position. At first they use matching of pre-recorded 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.

[FelKyaZap2012] 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 [11]. 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 [7]. 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 [2]. However, this project aimed to provide additional locative 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 approach moves the calculation of a position away from a device [3]. Correctly installed cisco mse determines the location of any wireless device in a Building. The access points listen for Wi-Fi signals of devices and estimate its position also via trilateration. This solution can be seen at the American Museum of Natural History [12].

The Cisco MSE determines the location of any wireless device in a specific building. One approach is to make the building do the work instead of the device. Some Wi-Fi installations, such as the Cisco MSE, can determine the location of any wireless device in the building. The access points themselves listen for the Wi-Fi signals created by your phone, then estimate its position 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.

2.2 Bluetooth Low Energy

Multiple solutions are based on the already mentioned Bluetooh 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 [6]. 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 extremly 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).

2.3 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 accelerometer, 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 oftenly distracted under real world conditions, a range of about 40-50 and (Beacons) can be suspected. The battery can as stated by [9] 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 the BLE technology standard are implemented in all smartphones devices enabling them to support new technologys like Apple Watch or Fitness trackers.

Using the Estimote SDK [9] mobile applications are enabled to receive and understand BLE Estimote signals in order to calculate the proximity to nearby locations and objects. The beacons specifics provide informations about their type, ownership and approximate locations, temperature or motions.

By the detecting a beacon signal a phone can estimate the distance by measuring the received signal strength [9]. Since Bluetooth Low Energy does not need any pairing process between sender and receiver, the phone can constantly process new signals. this opens the doors for new technologic opportunities as indoor location/ indoor positioning.

2.3.1 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 which is sent by a beacon is called advertisement.

These advertisements provides a so called iBeacon identifier, that is 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 informations about the beacon. While the UUID can be destinguished to a corporation, major and minor values can be used to distinguish between regions and sub-regions of a corporation.

2.3.2 Region Monitoring

Region monitoring triggers actions on the device on entering or exiting a beacon defined regions range. This works in depending the devices capabilities while an app is in foreground background or suspended. 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 [7].

2.3.3 Ranging

Ranging however triggeres actions on the devise based on the proximity to a beacon. The iBeacon protocol applies filters to the accuracy of a advertisement of one beacon. The filterestimation 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 in example holding the Smartphone directly on to of a beacon.
- Near
  The Near proximity state indicates a proximity of round about 1-3 meters, if there are no obstructions between the device and the beacon which might cause distractions of the beacon.
- Far
  The Far proximity state indicates a detected beacon without much confidence in the accuracy that is to low to determine wether it is Near or Immediate. The Far proximity state relies on the accuracy property to determine the potential proximity to the beacon.
- Unknown

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

2.3.4 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, specifiyng 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.[9] This is a major drawback regarding a use-case of the project, that a 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 in additional applied ranging of all beacons that are monitored in order to process the "nearest" beacon as a users location. The devices report then the beacons in an order that is best guess of their proximity regarding issues of signal attenuation. This order however may still not be correct [7].

2.4 Apple Core Location Framework

As firstly presented on WWDC2015 [11] Apple presented new functionalities to the already existing Core Location Framework which is the major Framework on iOS devices for location service [8]. This Framework takes the user in charge of wether the app can use locations services on the device or not.

The Core Location Framework uses Cellular data to provide an aproximity 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 sorounding Wi-Fi signals.

As soon as the user enters an indoor venue, the iOS system turns down GPS and Cellular sensors and enlightens Motion and Wi-Fi sensors. These sensors are used in combination with the remaining GPS signals, comming 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 wi-fi signals are feeded 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 [8].

For the Project the new CLLocation Framework was used to show the users Positon in the Mensa. After the first indoor tests in the mensa revealed precise positioning of a user indoors. However the Framework also revealed large susceptibility in areas where the wi-fi signal was apparently weak. i.e. in the upper left corner of the Mensa which is surrounded by walls that are affecting the router wi-fi signals.

The Frameworks started in an early beta when this project started and is constantly improved. To enable the full abilities of CLLocation Framework indoors, the venue needs to be registered and enabled by apple, using Maps Connect program [14] in order to unlock CLLocation Frameworks full potential.

2.5 Evaluation of Available Technologies

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 1.

```
<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"},
        {"timestamp": 1445345872, "latitude": "0.000000000000000", "
           longitude": "0.000000000000000", "building": "BIB", "floor": "1.
            Obergeschoss"},
        {"timestamp": 1445345874, "latitude": "0.000000000000000", "
           longitude": "0.000000000000000", "building": "BIB", "floor": "1.
            Obergeschoss"},
        {"timestamp": 1445345876, "latitude": "0.000000000000000", "
           longitude": "0.000000000000000", "building": "BIB", "floor": "1.
            Obergeschoss"},
        ...
    ]
}
```

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"},
13          {"timestamp": 1445350847, "latitude": "52.516402095317481", "
            longitude": "13.323531401046099", "building": "MAR", "floor": "
            Erdgeschoss"},
14          {"timestamp": 1445350849, "latitude": "52.516402095317481", "
            longitude": "13.323531401046099", "building": "MAR", "floor": "
            Erdgeschoss"},
15          ...
16      ]
17  }
```

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.

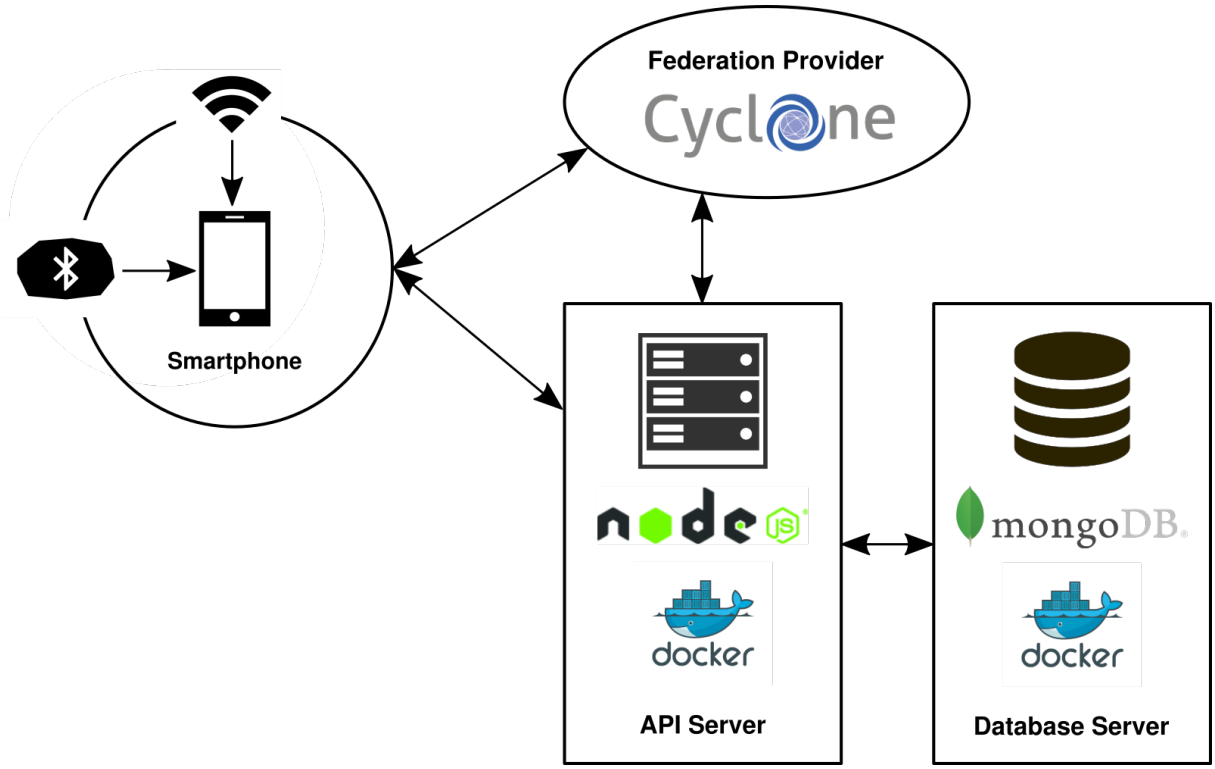
In the end, our conclusion at that point in the project progress was to use the CISCO MSE API wrapper provided by the tubIT in order to retrieve the rough position of a user. This means the building and floor the request originated from. We recommended back to our supervisors not to use this API for 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 [15].

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 ([4], 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 Server

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.

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

The OAuth 2.0 Authorization Framework is an IETF RFC [5] 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 [10].

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

3.5 Database Design

4 Implementation

4.1 Backend

4.1.1 Architecture

In this section we will focus on the architecture and our decision for the backend and its communication. We build a RESTful API which allows the clients to communicate with the backend in a JSON[1] format. The Figure 3.1 shows how the architecture is build up. We had two diffrent client applications which are communicating with the backend and a federation provider, we will describe the federation provider in chapter 4.1.2. The backend persists the data into a mongoDB database. The SNET department provided us a virtual machine with the url `piazza.snet.tu-berlin.de` where we could setup our environment.

We developed the backend with node.js[2] under the purpose that the SNET department can use or integrate it later to other projects which are mostly written in node.js. For our webapplication we uses the expressjs[3] framework which influnced us in the structure of how to build a application. Our file structure consists of five main directories:

- **controllers** are the specific implementation of a function
- **routes** links an endpoint to an controller
- **models** are schemas for the presistant data in the database
- **middleware** takes care for our authentication
- **test** are made for some of the important conrollers

4.1.1.1 controllers

We designed four controllers which are also part of our API endpoints to handle all requests to our API. Companionrequests handels the creation and modification of companion request which means, if a user wants to add a colleague to his friends list, he had first ask his friend/-companion for permission. If he accepted it, both parties get added to each others friend list. If he deny it, the requester will be notified that it was not acceped. The controller hotspot gives back all information of the defined hotspots like mensa or library. This contains GPS coordinates, companyUUID, major and minor of estimote bluetooth beacons. For getting or

[1] `http://json.org/index.html`
[2] `https://nodejs.org/en/`
[3] `http://expressjs.com/`

modifying user specific information like his location, groups and settings the users controller is the handler of this requests. The last controller is the login controller which gives back if the login was successful or not. He also adds some userinformation.

4.1.1.2 routes

Routes specifies a specific endpoint for a defined URI[4]. Our routes link mostly with the same name to the controllers which we defined above. Since we have an RESTful API the expressjs 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 4.1.1.4.

4.1.1.3 models

Mongoose[5] is a powerful plugin for nodejs 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 definded four schemas companionrequest, hotspot, location and user. But not every information have to be an extra collection in the mongoDB so we definded subdocuments for those information which dont need an extra collection. For example the beacons are always in a hotspot so there is now need for an extra collection.

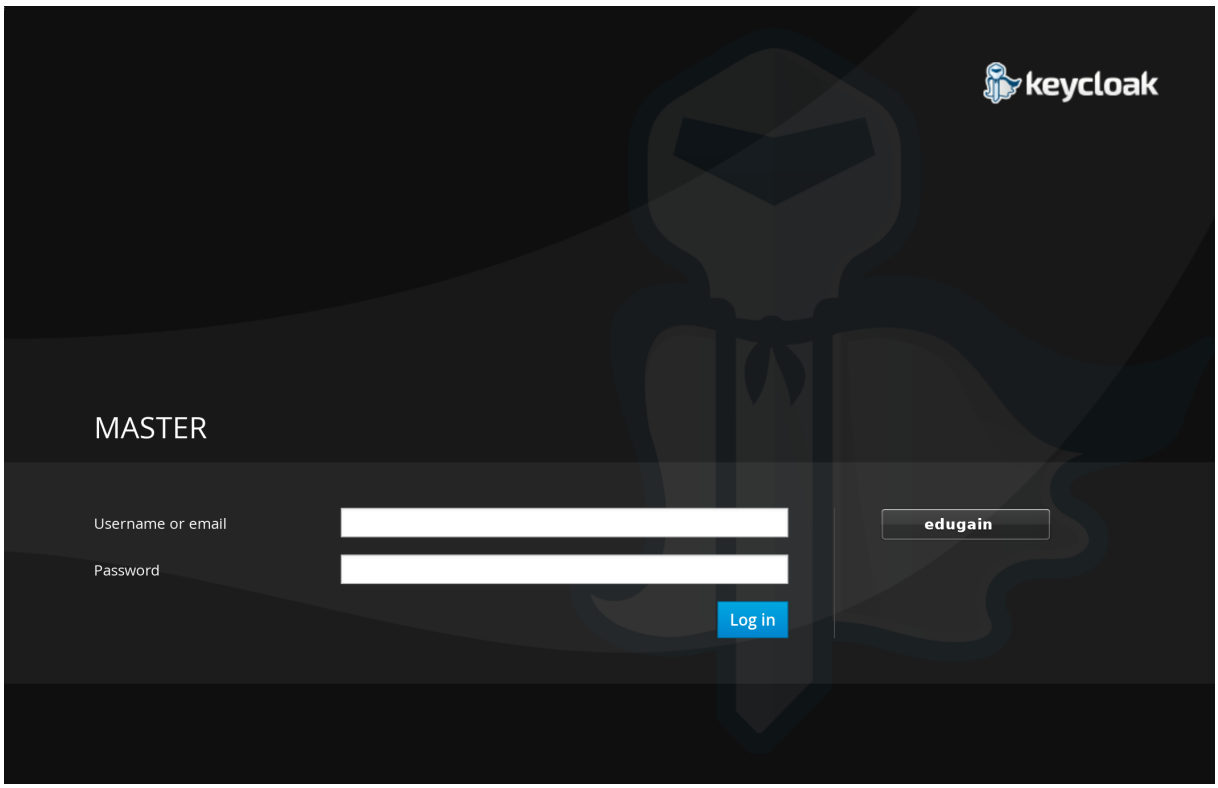
4.1.1.4 middleware

4.1.1.5 test

- Server setup
- Docker deployment
- List of important API endpoints
- Tests

[4] URI=Uniform Resource Identifier
[5] `http://mongoosejs.com/`

4.1.2 CYCLONE Federation Provider



Login screen to CYCLONE Federation Provider.

4.2 Android

4.3 iOS

5 Evaluation

Evaluate:

- What does work so far? What does not?
- What were the observed issues (see final presentation)?
- → MSE API, beacons, interplay server with clients, Node.JS, Mongoose (MongoDB), etc.

6 Conclusion

Conclusion:

- What did we do, how did it work out?
- Mention “team work issues”?
- Future work

6.1 Future Work

List of Tables

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] 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).

[3] 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).

[4] 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.

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

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

[7] 2014 Apple Inc. "Getting Started with iBeacon". In: (2014). URL: https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf.

[8] 2016 Apple Inc. "Core Location Framework Reference". In: (2015). URL: https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CoreLocation_Framework/.

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

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

[11] Vitali Lovich. "Building an Indoor Application WWDC 2015". In: (2015). URL: https://developer.apple.com/videos/play/wwdc2014-708/.

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

[13] Gunho Sohn1 Solomon Chan1*. "INDOOR LOCALIZATION USING WI-FI BASED FINGERPRINTING AND TRILATERATION TECHIQUES FOR LBS APPLICATIONS". In: (2012).

[14] IoSL-INav team. *Maps Connect*. Feb. 2015. URL: https://mapsconnect.apple.com (visited on 02/09/2016).

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

Appendices

1 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 }
```