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 11, 2016

Supervised by:
Sebastian Zickau and Mathias Slawik

Abstract

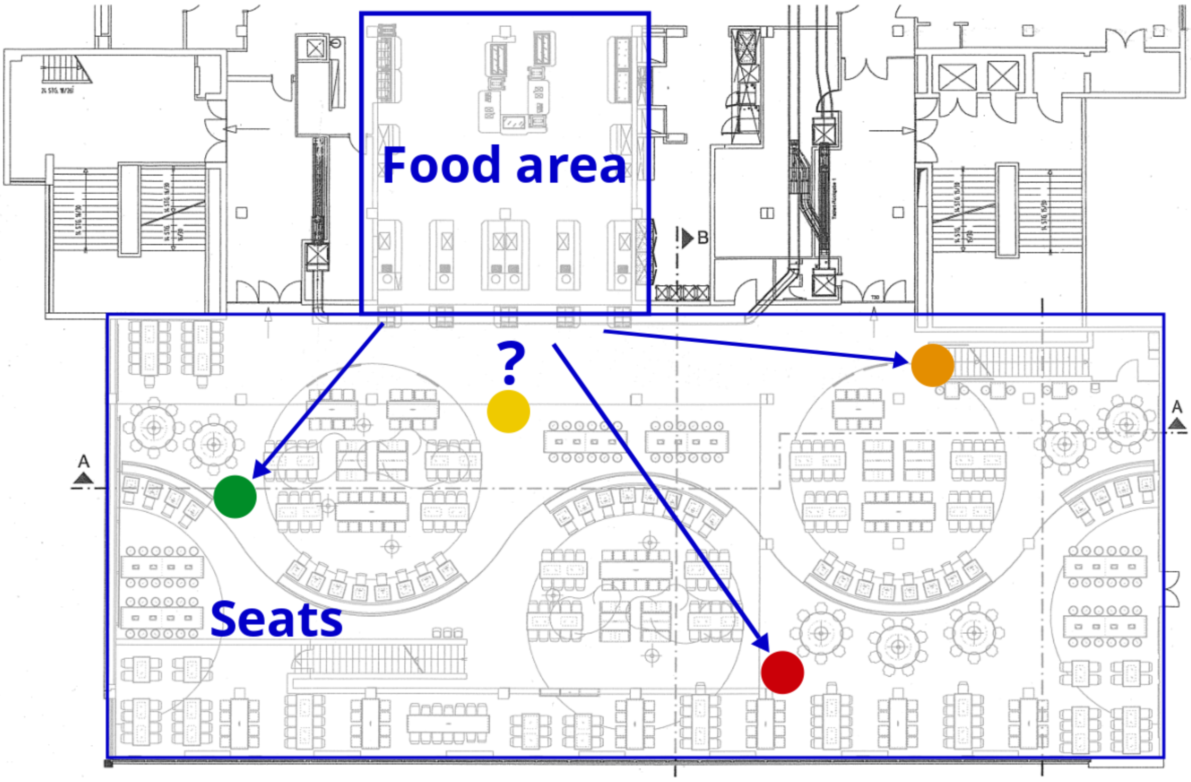
Short summary of project outcome.

Contents

1 Introduction

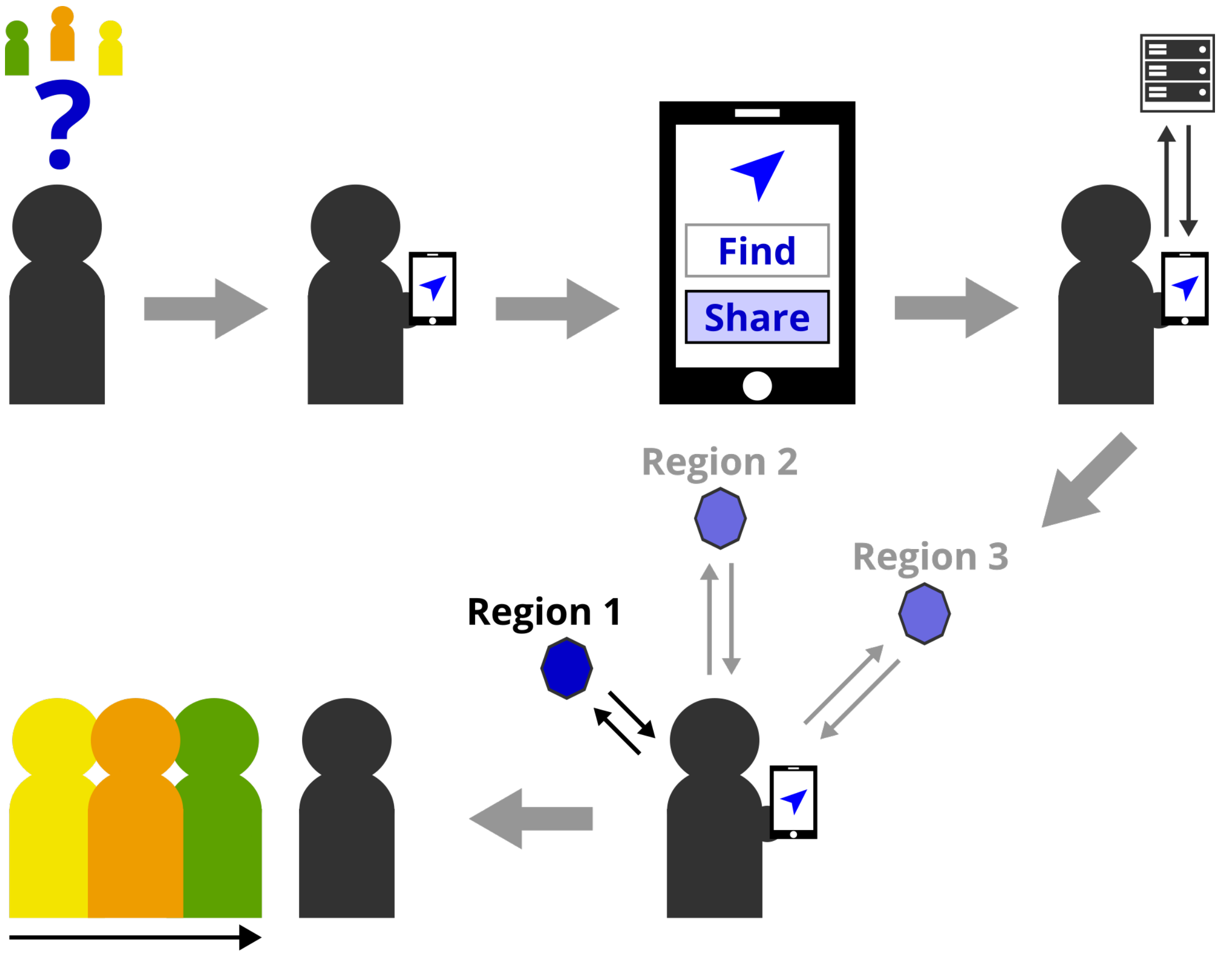
1.1 Motivation

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 Related Work

We are now going to take a look into what technology is available to accomplish our project goals defined in the previous chapter. First, we are starting with an overview of localization technologies that are in the reach of our project. After that we evaluate which ones are suitable for the circumstances our project is positioned in. And at the end one specific available solution will be examined more deeply in order to determine its possible value for our implementation.

2.1 Projects With Same Idea

2.2 Localization Technologies

2.3 Evaluation of Available Technologies

2.4 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:

```
{
    "signal": [
        {"timestamp": 1445344391, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "Mensa", "floor": "Mensa 1. OG"},
        {"timestamp": 1445344393, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "Mensa", "floor": "Mensa 1. OG"},
        {"timestamp": 1445344395, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "Mensa", "floor": "Mensa 1. OG"},
        {"timestamp": 1445344397, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "Mensa", "floor": "Mensa 1. OG"},
        {"timestamp": 1445344399, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "Mensa", "floor": "Mensa 1. OG"},
        {"timestamp": 1445344401, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "Mensa", "floor": "Mensa 1. OG"},
        {"timestamp": 1445344404, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "Mensa", "floor": "Mensa 1. OG"},
        {"timestamp": 1445344406, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "Mensa", "floor": "Mensa 1. OG"},
        {"timestamp": 1445344408, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "Mensa", "floor": "Mensa 1. OG"},
        {"timestamp": 1445344410, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "Mensa", "floor": "Mensa 1. OG"},
        ...
    ]
}
```

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:


```
{
    "signal": [
        {"timestamp": 1445345843, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "BIB", "floor": "Erdgeschoss"},
        {"timestamp": 1445345845, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "BIB", "floor": "Erdgeschoss"},
        {"timestamp": 1445345848, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "BIB", "floor": "Erdgeschoss"},
        {"timestamp": 1445345850, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "BIB", "floor": "Erdgeschoss"},
        {"timestamp": 1445345852, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "BIB", "floor": "Erdgeschoss"},
        {"timestamp": 1445345854, "latitude": "0.000000000000000", "longitude": "0.000000000000000", "building": "BIB", "floor": "Erdgeschoss"},
        ...
        {"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"},
        {"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.

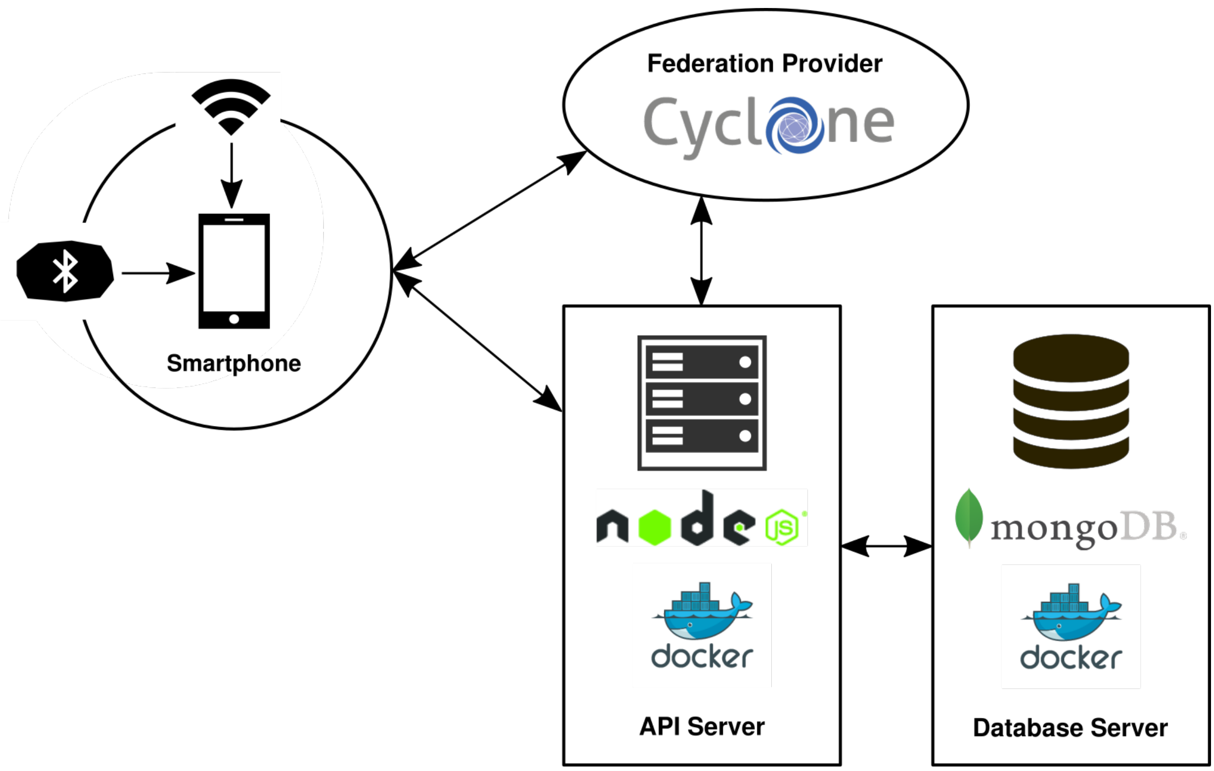
In the end, our conclusion at that point in the project progress was to use the CISCO MSE API wrapper provided by the tubIT in order to retrieve the rough position of a user. This means the building and floor the request originated from. We recommended back to our supervisors not to use this API for 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 [2].

3 Concept and Design

3.1 Big Picture

As already mentioned our system will consist of two parts, the mobile clients side and the backend side which interconnectedly exchange data. The backend part itself is split up again in three parts of which one is an "external" (means: SNET) resource, the CYCLONE Federation Provider. This entity provides us with user management and session handling functionalities so that we are able to out source these tedious and error-prone tasks to them. On the other side CYCLONE profits 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 is 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. 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 ([1], 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

3.5 Database Design

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

4 Implementation

4.1 Backend

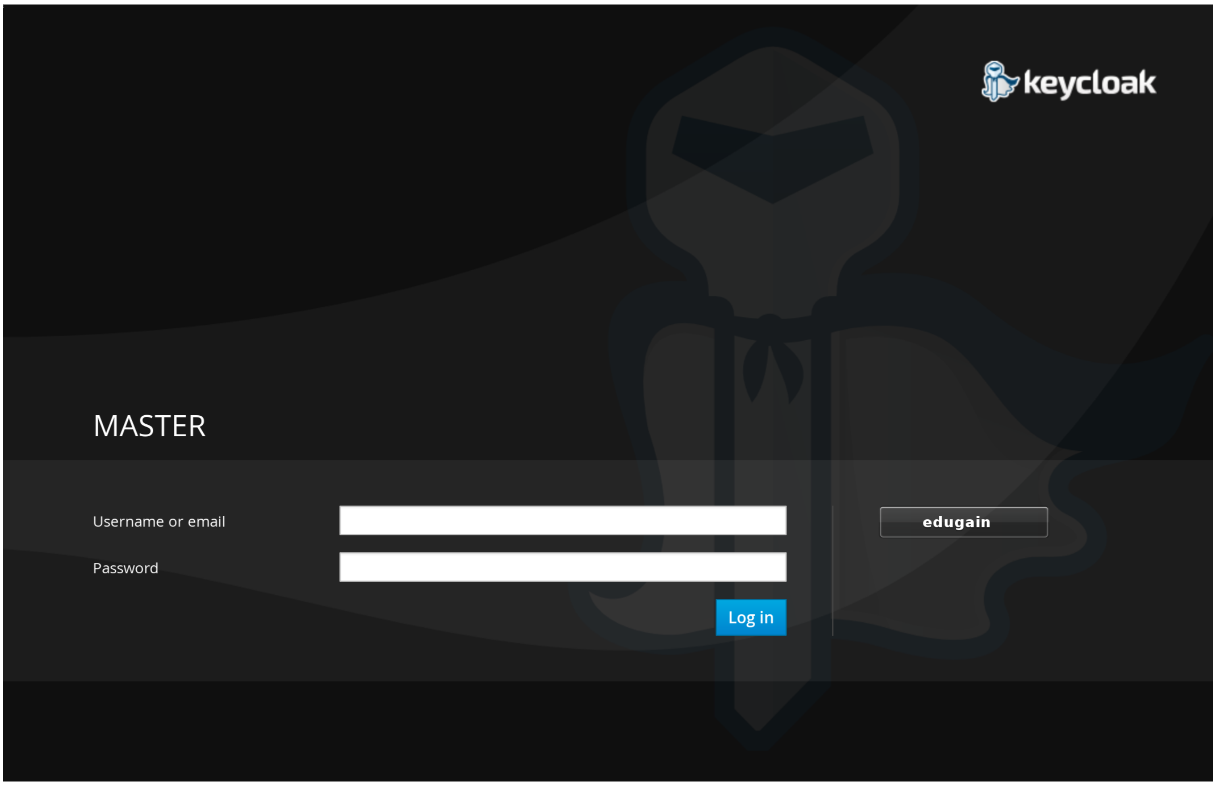
4.1.1 Architecture

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.

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

[1] http://json.org/index.html

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)?
- $\rightarrow$ 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] 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.

[2] 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

```go
package main

import (
    "fmt"
    "log"
    "os"
    "syscall"
    "time"

    "encoding/xml"
    "io/ioutil"
    "net/http"
    "os/signal"
)

type WiFiInfo struct {
    XMLName          xml.Name `xml:"Info"`
    ChangedOn        string   `xml:"changedOn,attr"`
    ConfidenceFactor float32  `xml:"confidenceFactor,attr"`
    Building         string   `xml:"building,attr"`
    Floor            string   `xml:"floor,attr"`
    Network          string   `xml:"WLAN-Status,attr"`
    UserName         string   `xml:"username,atrr"`
    Longitude        float64  `xml:"lon,attr"`
    Latitude         float64  `xml:"lat,attr"`
}

type LogWiFiStruct struct {
    Timestamp int     `json:"timestamp"`
    Latitude  float64 `json:"latitude"`
    Longitude float64 `json:"longitude"`
    Building  string  `json:"building"`
    Floor     string  `json:"floor"`
}

func handleUserExit(signalChannel chan os.Signal) {

    for _ = range signalChannel {

        // Define a useful result file name (format: "wifi-measurement-year-
            month-day-hour-minute-seconds.json")
        fileName := fmt.Sprintf("wifi-measurement-%s.json", fileNameTime)

        // Open measurement result file
        fileResult, fileError := os.OpenFile(fileName, os.O_CREATE|os.O_RDWR|os.
            O_APPEND, 0666)

        if fileError != nil {
            log.Fatal(fileError)
        }
```

```go

        // Close open logging file
        defer fileResult.Close()

        log.Printf("\nWritten measurement values to file %s. Good bye.\n",
            fileName)

        os.Exit(0)
    }
}

func main() {

    // Put in here the API endpoint to the wifi location service.
    const apiURL = "PUT THE API ENDPOINT IN HERE"

    // Get time data
    startTime := time.Now()
    fileNameTime := startTime.Format("2006-1-2-3-4-5")

    // Define a clean up channel
    signalChannel := make(chan os.Signal)
    signal.Notify(signalChannel, os.Interrupt, syscall.SIGTERM)
    go handleUserExit(signalChannel)

    log.Printf("Starting to measure WiFi API.\n")

    for {

        // Make a GET call on that URL
        apiResp, apiError := http.Get(apiURL)

        if apiError != nil {
            log.Fatal(apiError)
        }

        // Read in all body content we got and close connection
        xmlData, ioError := ioutil.ReadAll(apiResp.Body)
        apiResp.Body.Close()

        if ioError != nil {
            log.Fatal(ioError)
        }

        // Our go struct representation of the tubIT XML
        wInfo := WiFiInfo{}

        // Parse received XML into struct
        xmlError := xml.Unmarshal([]byte(xmlData), &wInfo)

        if xmlError != nil {
            log.Fatal("XML parsing error: %v.\n", xmlError)
```

```
        }

        // Get the current UNIX epoch timestamp for logging
        timeResult := time.Now().Unix()

        // Build up the log string
        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)

        // Write log string to opened file
        fileResult.WriteString(logResult)

        log.Printf("Received: \"long\": %v, \"lat\": %v.\n", wInfo.Longitude,
            wInfo.Latitude)

        // Let the execution wait for 2 seconds
        time.Sleep(2 * time.Second)
    }
}
```