Bachelor's Thesis

Design and Implementation of an Interactive Floorplan for an Access Management System

Design und Implementierung eines interaktiven Gebäudeplans für ein Zugangskontrollsystem

by

Tim Hehmann

Supervisors

Prof. Dr. Christoph Meinel, Eric Klieme, Christian Tietz

Fachgebiet für Internet-Technologien und Systeme

Philipp Berger, Stephan Schultz, Uwe Leppler

neXenio GmbH

Hasso Plattner Institute at University of Potsdam

July 26, 2019

Disclaimer

I certify that the material contained in this dissertation is my own work and does not contain significant portions of unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation.

Hiermit versichere ich, dass diese Arbeit selbständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projektes zu.

Potsdam, July 26, 2019

(Tim Hehmann)

Abstract

Guaranteering the safety in facilities is one of the main tasks of the facility management. This includes ensuring a safe evacuation in case of an alarm, the quick reaction to possible threats and also the protection of the doors and gates from intruders. Especially with the deployment of *Behavioural Authentication* the safety level for each of these access points can be finely set. In large office buildings this management and the guarantee of the safety can only be done with the help of visual aids.

This thesis presents an implementation of one possible visual aid: the interactive floorplan. Furthermore it will showcase the tools that are available for creating such a plan and also evaluate how good it performs in different simulation environments.

Zusammenfassung

Die Sicherheit in Gebäuden zu gewährleisten gehört zu einer der Kernaufgaben der Gebäudeverwaltung. Dazu zählt die Gewährleistung einer Evakuierung im Notfall, die schnelle Reaktion auf mögliche Gefahren, aber auch die Absicherung der einzelnen Türen vor Eindringlingen. Bei letzterem ist besonders mit dem Einsatz von verhaltensbasierter Authentifizierung es möglich, feingranulare Sicherheitsstufen für die einzelnen Türen festzulegen. Bei großen Bürokomplexen kann diese Verwaltung und Sicherheitsgewährleistung nur mit visuellen Mitteln bewältigt werden.

Diese Arbeit präsentiert eine Umsetzung einer dieser Mittel: den interaktiven Gebäudeplan. Dabei wird darauf eingegangen mit welchen Werkzeugen ein solcher Plan implementiert werden kann und auch gleichzeitig evaluiert, wie performant dieser ist in verschiedenen Simulationsumgebungen.

Contents

1 Introduction **1**
1.1 Context of the project 1
1.2 Context of this thesis 2

2 Related Work **4**
2.1 Infsoft 4
2.2 Moca 4
2.3 Roommaps 6
2.4 Summary 6

3 Concept **7**
3.1 BAAM Overview 7
3.1.1 Gates 7
3.1.2 Backend 8
3.1.3 Frontend 8
3.1.4 Logging Server 8
3.2 External Input 9
3.3 Requirements 9

4 Implementation **10**
4.1 Technology Discussion 10
4.1.1 Google Maps 10
4.1.2 OpenLayers 11
4.1.3 Leaflet 11
4.1.4 GeoJSON 11
4.1.5 Summary 12
4.2 Display of indoor features 12
4.3 Interactive Floorplan 13
4.4 Logging of Gate Events 14
4.4.1 Elasticsearch Server 14
4.5 Real-time Floorplan 18
4.5.1 Backend 18
4.5.2 Frontend 20

4.5.3 Heatmap 21
4.5.4 Access Decision Information 22
4.5.5 Alarm 24

5 Evaluation **25**
5.1 Setup 25
5.2 Simulation Sizes 25
5.2.1 Empire State Building Mittagspause 25
5.2.2 Small 25
5.2.3 Medium 25
5.2.4 Large 25
5.3 Data Privacy 25

6 Future Works **26**

7 Conclusion **28**

Bibliography **29**

List of Figures

1 Infsoft Analytics Web Application 5
2 Moca Web Application 5
3 Component diagram 7
4 Example of visualization options in Google Maps 10
5 Implemented interactive floorplan 23

List of Code Listings

1 Setup map from GeoJSON data 12
2 Adding indoor layer to map 13
3 Handling clicks on a room 13
4 cURL script for creating gates index template 16
5 cURL script for creating gates index 16
6 Interface of Elasticsearch server to create gate event logs 16
7 Example search request to Elasticsearch server 17
8 Creation of Socket.IO server 18
9 Middleware of Socket.IO server 19
10 Function that handles client connections to Socket.IO server . . . 19
11 Helper function for emitting notifications 19
12 Emission of notification to all admins 20
13 Setup of Socket 20
14 Function that is called when gate event happened 21
15 Function for colorizing the room 21
16 Function for calculating color based on occupancy 22
17 Function for adding message to Activity Monitor 23
18 Gate Alarm happened 24
19 Notify Admin on Alarm 24

1 Introduction

In order to protect critical areas from unauthorized access, most office buildings use an access control system that grants or denies access to gates and doors based on the permissions of the employee.

The most common way to authenticate in these systems is either by knowledge (keypad with pin login) or ownership (NFC chipcard). But not only office buildings, also gyms, public transportation services and universities use access control systems with the same ways to authenticate. This results in a lot of different cards and passwords for the user. The management of these can easily be overwhelming and once a thief obtained one of these there is a possibility for an attack.

1.1 Context of the project

The bachelors project from 2016 'Passwords Are Obsolete - User Authentication Using Wearables And Mobile Devices' tried to solve this problem by building an app that makes it possible to authenticate the user solely on his behaviour. This is done by continously analysing the sensor data from smartphone/-watch and calculating a *trustlevel*, a value that determines how certain it is, that the device is in possession of the correct owner. Together with our project partner *neXenio*[1], some members of the bachelors project develop this app continously further by the name of *BAuth*[2],

Besides authenticating the user by behavior, the smartphone (with BAuth installed) is also already able to communicate with the gates via Bluetooth. But the management of the access rights of each employee for each single gate is currently done by hardcoding a list of allowed employee directly on to the hardware inside the gates. Although there are a lot of access management systems out there that provide automation for the management of access rights, none of them fits the needs for the app of our predecessors. Additionally these systems

[1] https://www.nexenio.com/

[2] https://play.google.com/store/apps/details?id=com.nexenio.behaviourauthentication&hl=de

are all closed source and cannot be extended to also work for this use case.

The goal of our project was to create an access management platform that is suited to work with BAuth and makes the access management more comfortable. The facility management and also companies should be able to define which employees can access which gates. It should also be possible to set the minimal trustlevel that is needed to enter or leave a certain gate, which enables to define which rooms need more protection than others.

With this solution, BAuth could be used in a real world scenario.

1.2 Context of this thesis

The management of an office building with multiple floors and multiple gates can be a challenging task for the facility management team. To prevent loosing the overview of the facility, the usage of an interactive floorplan can be helpful. The implementation of such a plan was also part of our projects scope and forms also the main topic of this thesis.

In our access management system this graphical plan gives insight about the different gates in the building, including the access decisions made at these and information about the person that tried to access. Furthermore it visualises how many persons are approximately in a room and at which gates an alarm occured.

This information could be used by the facility management team to see how heavily the gates are used, where a possible security threat exists and also if a room is currently at risk of not being evacuated safely. In general it improves the overall view of the facility and could lead to a faster use of our other access management tools.

This bachelor thesis will showcase an approach for the design and implementation of such a floorplan. To accomplish this it will be guided by the following structure:

In the second chapter, we will present the requirements that we received for the implementation of the interactive floorplan and what external input we have at disposal.

In the third chapter, related work gets discussed. This will showcase different solutions from software companies that already offer services for similar requirements.

The fourth chapter describes our chosen approach and the architecture and components behind it.

In the fifth chapter we will present our implementation for an interactive floorplan. This will present the solutions used to fulfill the requirements explained in the Background chapter.

The topic of the sixth chapter is evaluation. In this chapter the performance of the floorplan for different scenarios gets analysed. Furthermore it will discuss the protection of data privacy in a floorplan, especially focussing on the personal informations that get shown in the floorplan.

The seventh chapter will present what further features could be implemented in the future and what needs to be done before actually deploying it into production.

The eigth and last chapter will wrap the thesis up.

2 Related Work

Our project of creating an interactive floorplan with its features described in the Introduction chapter is similar to projects of other software companies. In this chapter we will look at the solutions from different companies and discuss if our project can be based on this work.

2.1 Infsoft

Infsoft is a german software company, which specializes in indoor tracking, positioning and navigation. They use custom *infsoft Locator Nodes*[3], which enable to detect the position of devices trough WiFi or Bluetooth, but can also track the location of RFID chips or utilize Ultra-wideband technology.

This location data is then analyzed to track the path of employees, visitors or objects. These analytics are available in real-time over a web interface, which includes a live rendering of a heatmap, showing locations with heavy or low traffic (figure 1). But also a history of location data can be displayed. The location data of each device is anonymized and can't reveil information about the owner of the device.

2.2 Moca

Moca is a platform for helping companies find out insights about the shopping behavior of their customers. This contains also analytics about the movement paths of the customers in a shopping mall. This can be utilized to learn how the customer gets to the retailshop or what they do after leaving the shop.

To achieve this, they use the existing WiFi network setup in the building to track the location of the devices, which has an accuracy of approximately 10 meters[4]. The data is then displayable in a real-time floorplan view with a heatmap over-

[3] https://www.infsoft.com/de/technologie/hardware/infsoft-locator-nodes

[4] According to https://www.mocaplatform.com/blog/moca-indoor-location-mobility-flows-for-venues

lay which can be seen in figure 2. They also allow real-time playbacks of location data from the past days and import of floorplans.

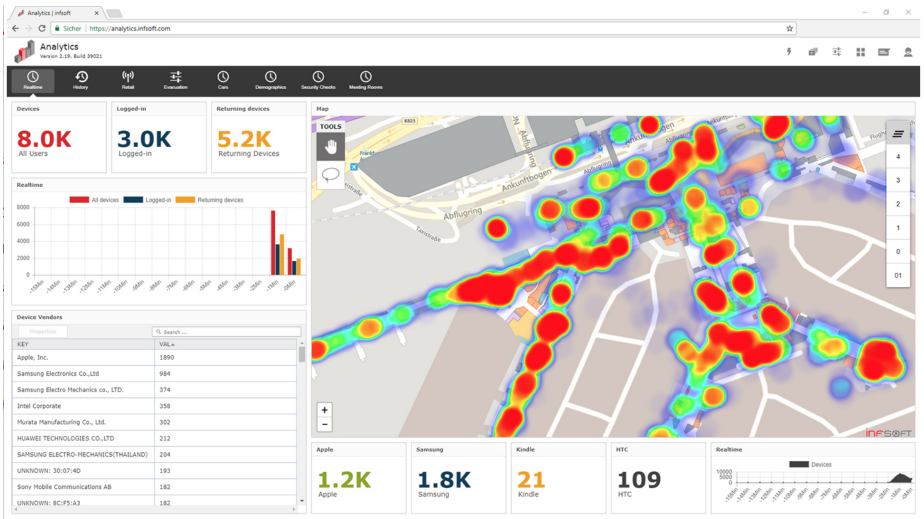

Figure 1: Infsoft Analytics Web Application

Figure 2: Moca Web Application

2.3 Roommaps

Roommaps is another indoor information and navigation system, which is primarly focussing on apps for Android and iOS, but is also available as a web application.

The system is also able to track device locations via WiFi signals or Bluetooth beacons and display a heatmap based on this data. But unlike the other solutions they support setting detailed data privacy options for each floor. This helps protecting data privacy even more, as it is possible to precisely configure the view access rights for each user for each floor. Additionally, the services are also available for deployment in third party applications.

2.4 Summary

All these solutions already offer a great set of functionality. But these are based on the location of devices, which is determined through an indoor positioning sensor network, like a WiFi network or a set of Bluetooth beacons. This makes it unapplicable to our project, because the only data that is available for us are the locations of the gates and the access decisions made at these and not the location of the devices themselves.

Although Roommaps provides their services for third party applications, these services cannot be extended or customized for our use case.

Additionally, all these services are not open-source products and come with costs regarding hardware and software. Since our project partner requires the use of open-source and free software, the interactive floorplan of our project cannot be based on the related work that was presented and needs to be build from the ground up.

3 Concept

3.1 BAAM Overview

To be able to meet the requirements of our interactive floorplan, the events from the gates need to be logged and displayed in real-time. For this to work, there are multiple components involved. A general overview about the architecture can be seen in figure 3.

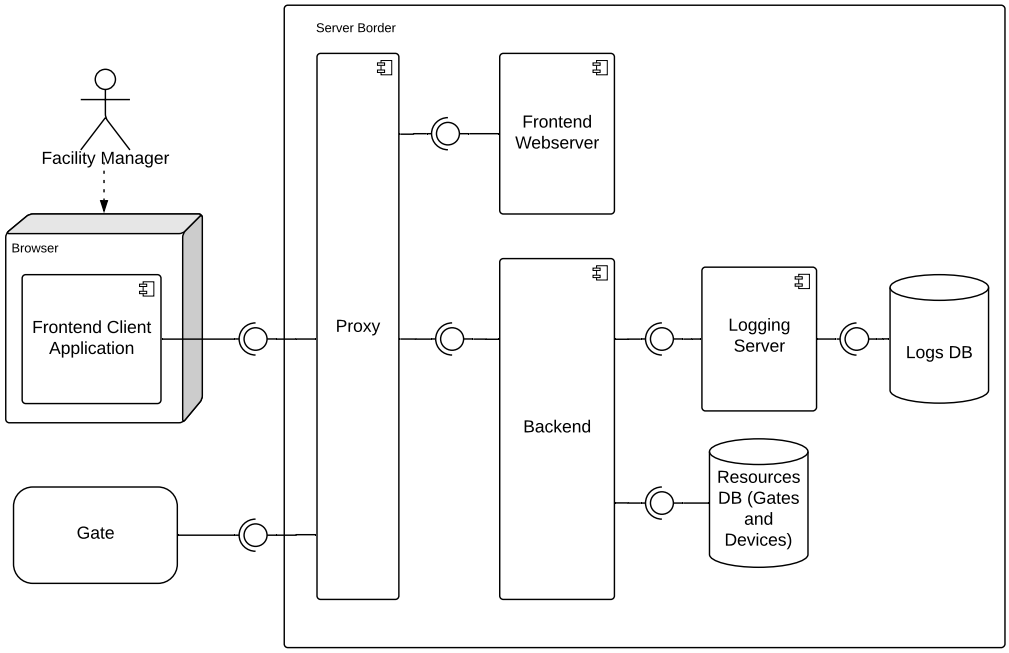

Figure 3: Component diagram

3.1.1 Gates

The gates take care of capturing different informations once a person tries to access through a gate. They gather information about the device that communicates with them, if they granted or denied the access and if the person tries to enter or exit through the gate. In case of an alarm they need to store other useful information about that incident.

The gate then is responsible of sending this information to our system via an API[5].

3.1.2 Backend

The backend offers interfaces for the gates and also the frontend. For the gates it provides an interface which allows the gates to send events to, which then will be persisted as logs. For the frontend it provides routes for retrieving the logs and also analytical results based on these.

Once a gate transmits data through the available interface, the backend emits a notification that an event occured together with the event data.

3.1.3 Frontend

The frontend presents the interactive floorplan to the facility management with clickable markers for each gate. It features the adding and deleting of markers and the possibility to link them to a gate. A click on a specific marker shows useful informations about the gate, such as the minimal entry and minimal exit trust level required for this gate.

For the possibility of displaying data in real-time, it listens for the gate event notification sent from the backend. When a notification is emitted the frontend visualizes the event data to the user. This also triggers a recolorization of the room that is connected to the gate where the event occured, thereby showing the up-to-date occupancy of that room.

3.1.4 Logging Server

The logging server takes care of storing the events in a log database. It offers endpoints for creating event logs and also for searching through the logs. This allows to calculate the number of people in a room for example.

[5] Application Programming Interface

3.2 External Input

Multiple gates can be installed inside an office. We receive data about the access decisions made at these gates. We also get geospatial data for each indoor feature inside the office.

3.3 Requirements

- **Interaction**: The floorplan should be an interactive map. This means that the user is able to interact with the map by zooming, panning and clicking, thereby altering the state and look of the map. This includes the possibility to set markers at a specific location on the map, which then can be linked with gates. A click on a marker then shows information about that specific gate.
- **Real-time Data Visualization**: The events from the gates should be displayed in real-time on the map. This includes information about the access, which enables to quickly view who entered at what gate at what time.
- **Eventlogging**: There needs to be an interface for the gates to send these events to. These need to be persisted as logs for later analytics.
- **Heatmap**: The plan should visualize the number of people in a room and render a "heat" on the map, thereby presenting the occupancy status of each room.
- **Alarm Localization**: The floorplan needs to visualize alarms, which is needed to locate threats on the map quickly.
- **Upload Functionality**: Because of the fact that every user has different floorplans the upload of own floorplans must be possible.

4 Implementation

As this was part of our access management platform we implemented this floorplan also as a web application.

4.1 Technology Discussion

4.1.1 Google Maps

In March 2011, Google introduced the first indoor floorplans in their map. The intention was to increase the overview in public areas like train stations, malls and airports. Users can upload own floorplans (valid formats include for example PNG, PDF or JPEG) to the map, with restriction to only publicly available areas.

Google Maps also offers a very popular API for their services. This allows the integration of the Google Maps Services in your own website. The usage is free for commercial use up until 28000 calls per day[6] and requires an API key. The Maps JavaScript API comes with direct support for importing GeoJSON and can be customized with own content. Its designed to load maps quickly and is optimized for mobile use. Aside from that it also offers a versatile visualization library, which also includes a Heatmap Layer that helps with visualizing a heatmap (Figure 5).

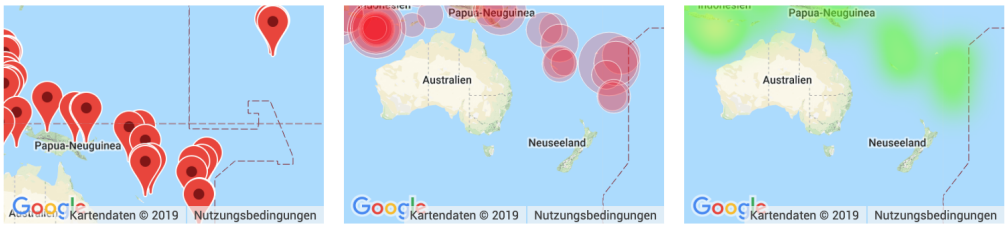

Figure 4: Example of visualization options in Google Maps

[6] According to `https://cloud.google.com/maps-platform/pricing/sheet/?hl=de`

4.1.2 OpenLayers

OpenLayers is an open-source JavaScript library for displaying interactive maps. Out of the box it comes with various features like map rotation, direct mobile support and import of GeoJSON, TopoJSON, KML[7] or GML[8] data [9]. Unlike Google Maps, OpenLayers is a pure client-side library with no server-side dependencies.

4.1.3 Leaflet

Leaflet is another open-source JavaScript library for creating interactive maps. With it's first version released in 2011, it is a well established and tested library.

By only including core features for map visualization, it only has a bundled size of 138.6KB[10], making it a very lightweight library.

Furthermore Leaflet supports every browser and can easily be extended by plugins from the community or own plugins. This also includes plugins for creating indoor maps [1] and real-time maps with Socket.IO.

4.1.4 GeoJSON

GeoJSON is a format for interchanging geospatial data and is based on JavaScript Object Notation. With the publishing of RFC 7946 in August 2016 it has a standardized format specification. Different geometries can be represented in a GeoJSON file. These include for example Lines, Linestrings, Polygones or Multipolygones[11].

This can be used to encode the geometry for countries, houses and streets on a map, but also for encoding data for indoor rooms, stairs and hallways.

[7] Keyhole Markup Language
[8] Geography Markup Language
[9] `https://openlayers.org/`
[10] `https://bundlephobia.com/result?p=leaflet@1.5.1`
[11] `https://tools.ietf.org/html/rfc7946`

4.1.5 Summary

Although Google Maps looks very promising for creating our indoor floorplan, there are quite a few problems for us: Because the API needs an internet connection, offline development is not possible. Furthermore the Google Maps API would request payment after hitting the threshold of API calls mentioned above and therefore needs to be linked to an account where billing is activated. Although hitting this threshold could only happen in production, our project partner set the requirement to only use free and also open-source software and linking a billing account of our partner to use the API was not possible.

Unlike Google Maps, OpenLayers provides an open-source library, which provides a lot of features for interactive maps out of the box. But because it has a lot of features already bundled together the OpenLayers module is a heavyweight module. With a minified bundle size of 330.1 kB (Version 5.3.3) it takes up to 1.6 seconds to download on 3G[12]. Although this can be lowered for production by deleting unused modules, it takes extra effort to see which modules are really not used.

Leaflet solves this problem by providing only a core set of functionalities which can easily be extended by self-written or community plugins. This reduces the size of the module. All in all, we decided to build our interactive map with the Leaflet library, because of its leightweightness and its extensibility, which is useful for configuring the interactive floorplan exactly for our needs. Besides that we will also use GeoJSON as the format for the external input of the geometric data of each room in the office. We base this decision on the fact that GeoJSON is a widely used and standardized format for encoding geospatial data.

4.2 Display of indoor features

The indoor map plugin extends Leaflet by an `Indoor` class, which is able to load in geospatial data from a GeoJSON file (Listing 1):

```
1 ...
```

[12] `https://bundlephobia.com/result?p=ol@5.3.3`

```
// leaflet map gets rendered inside a div that has the id of the first parameter
this.map = L.map('floorplan-container', {
  center: new L.LatLng(49.41873, 8.67689),
  zoom: 20,
});

const indoorLayer = new L.Indoor(this.props.geoJSON, {
...
```

Listing 1: Setup map from GeoJSON data

This creates a layer with the GeoJSON data where each feature is represented as a HTML element. This indoor layer is then added on top of an instance of the Leaflet `Map`[2] class ((Listing 2):

```
indoorLayer.addTo(this.map);
```

Listing 2: Adding indoor layer to map

4.3 Interactive Floorplan

The floorplan handles mouse clicks on each room that is displayed (Listing 3). If a click event occurs, it adds a new marker at this position, which is then automatically linked with the room where the click occured.

```
handleClickOnRoom = (event) => {
  this.addMarker(
    event.latlng,
    { gateId: undefined, assignedRoomId: event.target.feature.id },
    true);
  };
```

Listing 3: Handling clicks on a room

To create this connection between the room and the marker, each room represented in the GeoJSON file has to have a unique member attribute *id*.[13]

[13]This is still following the GeoJSON format specification according to `https://geojson.org/geojson-spec.html#feature-objects`

After setting the marker it needs to be linked with a gate, which is done by a selection input presenting all available gates. After successfully linking it to a gate the marker gets persisted in our database.

The map is capable of also handling pan events for moving to another location and scroll events for zooming in and out. This functionality is part of the `Map` class from Leaflet.

4.4 Logging of Gate Events

To support the functionality to log gate events we make use of the Elastic Stack[14]. The Elastic Stack consists of mainly three open-source projects: *Elasticsearch*, *Logstash* and *Kibana*. Together they form a pipeline that can be used to analyze, search and visualize logs created from different sources.

The component of the first stage of this pipeline is Logstash, which is responsible for collecting data from different locations and transforming it for the next step. Elasticsearch then indexes these logs and provides a RESTful API for searching. Kibana uses this API from Elasticsearch to provide meaningful visualization of the logs. Through a smart indexing technology the Elastic Stack promises a fast response time even for large data sets and is used by companies like Netflix for monitoring security related logs or Medium for debugging production issues [15].

To ensure a fast display of the access decision data and to also ensure the possibility in the future to search and visualize logs not only from the gates, but from other sources also, we decided to utilize the Elastic Stack.

4.4.1 Elasticsearch Server

Since we're working with own custom visualization, we will ignore Kibana for our implementation. And since we're only receiving events from one source - the gates, we can skip the Logstash pipeline step and send event data directly to

[14] https://www.elastic.co/de/products/elastic-stack

[15] `https://hackernoon.com/elastic-stack-a-brief-introduction-794bc7ff7d4f`

the Elasticsearch server, which then will be stored in a database and indexed.

Since data of any form can be sent to the Elasticsearch server, it performs an automatic type detection for each property that is sent. This can result in wrong datatypes, causing the server to reject future events because they're not fulfilling the required datatypes[16]. To prevent this, we first have to create a template for our gate event objects, which strictly declares the datatypes for each property in the event object (Listing 4):

[16]For example: The first event has a member *id* with the value "1234". The Elasticsearch server will automatically interpret this value as an Integer and set it as required datatype. If the following event now has an id of "fe123d" (String) the server will not accept this event.

```
curl -v -X PUT 'localhost:9200/_template/gates' -H 'Content-Type: application/json' -d '
{
  "index_patterns" : ["gates"],
  "settings": {
  /* Shards allow to split up the content volume that is taken by the documents that are
     inside a single index. This enables good scaling, by allowing the documents not to
     reside on just one single harddrive but multiple. Since we are currently not receiving
     that much data yet it is sufficient to only work with one shard for now.*/
    "number_of_shards": 1
  },
  "mappings": {
    "_source": {
    // allows for example updating a document
      "enabled": true
    },
    "properties": {
      "timestamp": { "type": "date" },
      "loglevel": { "type": "keyword" },
      "gateId": { "type": "keyword" },
      "deviceId": { "type": "keyword" },
      "accessType": { "type": "keyword" },
      "wasSuccessful": { "type": "boolean"},
      "message": { "type": "text"}}
  }
}'
```

Listing 4: cURL script for creating gates index template

We then create a *gates* index (Listing 5). All logs for the gates will be stored under that index.

```
curl -v -X PUT 'localhost:9200/gates'
```

Listing 5: cURL script for creating gates index

This server then offers an REST API endpoint for creating gate event logs (Listing 6):

```
function postGateEvent(eventData) {
    return fetch(`http://${elasticsearchBasePath}/gates/_doc`, {
        headers: {
            'Content-Type': 'application/json',
        },
```

```
        body: eventData,
        method: 'POST',
    })
        .then(response => errorHandling.checkResponseOk(response, msg.getCreateEventFailMsg
    (response)));
}
```

Listing 6: Interface of Elasticsearch server to create gate event logs

This endpoint is exposed to the outside so gates can communicate to our system. To protect this route from being exploited from other sources than the gates we use a preshared token that needs to be set in the *Authorization* header. The endpoint requires a transfer of the ID of the device that tried to access, the access type (entry or exit), the gate ID and if the entry or exit was successful. Moreover the optional parameters loglevel and message can be send. The loglevel can be used to signalize an alarm event and the message parameter can be used to send over more detailed information.

The Elasticsearch server also offers an interface to search for logs satisfying specific conditions. This is how we can look up all the events at a gate that are of a specific access type (Listing 7):

```
function fetchAllEventsAtGateWithAccessType(gateId, accessType) {
    const now = new Date().toISOString();
    const officeOpening = getOfficeOpeningDatetime().toISOString();

    const url = `http://${elasticsearchBasePath}/gates/_search?sort=timestamp:desc&`
        + `q=gateId:${gateId}`
        + `%20AND%20`
        + `accessType=${accessType}`
        + `%20AND%20`
        + `timestamp:[${officeOpening}+TO+${now}]`;

    return fetch(url, {
        headers: {
            Accept: 'application/json',
        },
    })
        .then(response => errorHandling.checkResponseOk(response));
}
```

Listing 7: Example search request to Elasticsearch server

4.5 Real-time Floorplan

To implement a real-time floorplan we utilize the *Socket.IO* JavaScript library, a library which enables to implement real-time applications. This is achieved through an event-based, bidirectional communication between the client and the server.

Although Socket.IO also uses WebSockets for transportation it is not an implementation of the WebSocket-protocol. It extends and combines multiple real-time protocols and switches between them if needed. Therefore a connection can only be established between a Socket.IO client and a Socket.IO server[17].

We decided to use Socket.IO, because of its easy to understand API and its functionality set. It provides the creation of realiable connections by having different fallback real-time methods, auto reconnection support and the detection of disconnections.

In the following sections we will explain the setup of Socket.IO in the backend and frontend and how this setup is used to implement a real-time floorplan.

4.5.1 Backend

We first have to create a Socket.IO `Server`[3] instance by binding it to the existing HTTP server we have for our backend (Listing 8):

```
1  //app.js
2  const server = http.createServer(app);
3  socketHelpers.handleSockets(server);
4
5  //socket.helpers.js
6  function handleSockets(server) {
7    // creates Socket.IO Server
8      const io = socketIo(server);
9      ...
10 }
```

Listing 8: Creation of Socket.IO server

[17] `https://socket.io/docs/index.html`

To ensure that we only send data to clients that are authenticated, we need to verify the token that is sent with each packet. This is done by installing a middleware on to our Socket.IO server that includes a function that gets executed for every packet that is sent. This function verifies that the token sent is a valid access token (Listing 9).

```
io.use((socket, next) => {
    const { token } = socket.handshake.query;
    const verifyToken = keycloakHelpers.verifyToken(token);
    return verifyToken
        .then(() => next())
        .catch(err => next(err));
});
```

Listing 9: Middleware of Socket.IO server

Everytime a frontend client now connects to the server, the server extracts the userId from the token that was sent by the client. We then create a *Room*[4] - a seperated communication channel - with this userId, so that we can emit notifications only to that specific user. If the logged in user is also an admin, he gets added to an admin room, a room where all other admins that are connected are also inside. The implementation of this can be seen in Listing 10.

```
io.on('connection', (socket) => {
  const { token } = socket.handshake.query;
  const parsedToken = jwt.decode(token);

  const isAdmin = parsedToken.realm_access.roles.includes('admin');
  const userId = parsedToken.sub;

  if (isAdmin) socket.join('admin');
  socket.join(userId);
});
```

Listing 10: Handling client connections to Socket.IO server

We can then emit messages to specific users or to all admins through a helper function (Listing 11):

```
module.exports.emitMessage = (user, emitType, message) => {
```

```
  // user can be either a userId or 'admin'
  // all current client sessions with logged in user will receive message
  io.to(user).emit(emitType, message);
};
```

Listing 11: Helper function for emitting notifications

This method gets called everytime the gates send an event through our backend interface that we presented earlier. We use the room 'admin' to notify all admins at the same time (Listing 12).

```
function notifyAboutEvent(data) {
    if (data.loglevel.toUpperCase() === constants.ALARM_LOG_LEVEL) {
        notificationHelpers.notifyAdminOnAlarm(data);
    } else {
        socketHelpers.emitMessage('admin', socketHelpers.GATE_EVENT, data);
    }
}
```

Listing 12: Emission of notification to all admins

4.5.2 Frontend

The frontend has to install the Socket.IO JavaScript library and then initialize a `Socket`[5]. Handlers are then registered to that `Socket` for the different notification events from the backend (Listing 13).

```
listenForGateEvents = () => {
        const socket = io({
            secure: true,
            // only use WebSocket as transportation method
            transport: ['websocket'],
            query: {
                token: sessionStorage.getItem('kctoken'),
            },
            jsonp: false,
        });

        socket.on('gateEvent', (event) => { this.gateEventHappened(event); });
        socket.on('gateAlarm', (event) => { this.gateAlarmHappened(event); });
    };
```

Listing 13: Setup of Socket

4.5.3 Heatmap

Everytime an event occurs the marker that is linked with the gate Id of the event object is searched. Then the room id that is connected with this marker is recolorized based on the updated number of people that are behind this gate (Listing 14):

```
gateEventHappened = (event) => {
        const gateMarker = this.findGateMarkerWithId(event.gateId);

        if (gateMarker) {
            this.applyPulseEffectToMarker(gateMarker);

            const { assignedRoomId } = gateMarker.options;
            getNumberOfPeopleForGateWithId(event.gateId)
                .then((data) => {
                    this.updateGateInfoOfCurrentSelectedMarker(gateMarker, data);
                    this.colorizeRoom(assignedRoomId, data.count);
                });
        }
    };
```

Listing 14: Handling gate events in frontend

To calculate the number of people we use an endpoint in our backend that looks at the datetime of the office opening at the day the request to the endpoint is made. At this time we expect that no people are inside the office. It then counts all entries and exits at the gate with the given id that were successful and after the office opening. By subtracting the exits from the entries we get the number of people that are behind this gate.

This result is then the input for colorizing the room (Listing 15):

```
colorizeRoom = (roomId, numberOfPeople) => {
    let occupancyRate = numberOfPeople / constants.ROOM_PERSON_COUNT_LIMIT;
```

```
    if (occupancyRate > 1) occupancyRate = 1;
    if (occupancyRate < 0) occupancyRate = 0;

    const room = this.findRoomWithId(roomId);

    room.setStyle({ fillColor: this.getColorForOccupancyRate(occupancyRate) });
};
```

Listing 15: Function for colorizing a room

Because we can only work with events from the gates and no indoor positioning technology, we're unable to locate the exact position of single device. Therefore we colorize the entire room evenly.

We follow the common convention in heatmaps of representing the occupancy of a room by using colors on a scale from green to red, with green representing a room with no people inside and with red a room that hit its maximum capacity. The calculation for the color can be seen in Listing 16.

```
    getColorForOccupancyRate = (rate) => {
        /*
            input: value from 0 to 1
            returns: a hsl color on a scale from green to red
        */
        const hue = ((1 - rate) * 120).toString(10);
        return ['hsl(', hue, ',100%,50%)'].join('');
    };
```

Listing 16: Function for calculating color based on occupancy

4.5.4 Access Decision Information

More information about the access decisions get also displayed in real-time in a table (*Activity Monitor*) below the interactive floorplan (Figure 4).

Because the gates transmit the deviceId in the event object, the relationship to the owner of the device can be made through our backend. This happens each time an gate event occurs and we add the information as a new row into the Activity Monitor (Listing 17):

```
addMessage = (logMessage) => {
        const { logMessages } = this.state;

        // only show 100 log messages at a time
        if (logMessages.length >= 100) {
            logMessages.shift();
        }

      // get user information with the device ID provided in the event object
        getDeviceById(logMessage.deviceId)
            .then(device => getUserById(device.userId))
            .then(user => {
                logMessages.push({ ...logMessage, username: user.username});
                this.setState({ logMessages });
                this.scrollToBottom();
            });
    };
```

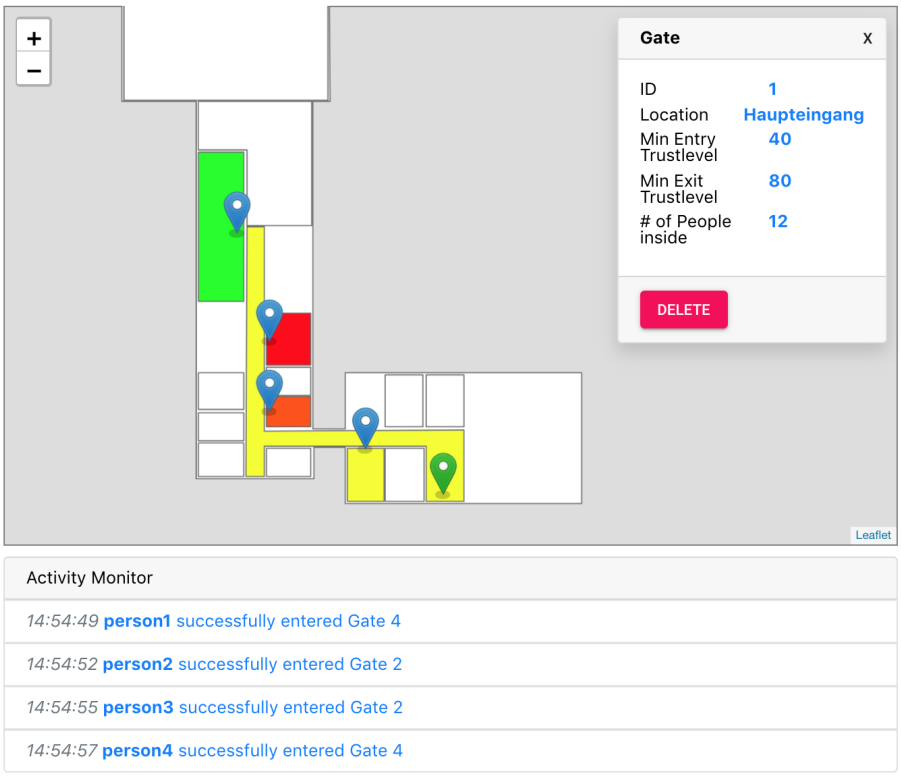

Figure 5: Implemented interactive floorplan

Listing 17: Function for adding message to Activity Monitor

For each event we display information about the time, the person that is connected with the event, the access type, the success of the access and the gate Id where the event happened.

4.5.5 Alarm

To visualize an alarm event at a gate we colorize the marker red and display a red log message in the Activity Monitor (Listing 18):

```
gateAlarmHappened = (event) => {
        const gateMarker = this.findGateMarkerWithId(event.gateId);
        if (gateMarker) {
            this.pulsateMarker(gateMarker, 'red');
        }
    };
```

Listing 18: Handling gate alarm events in frontend

Furthermore the backend automatically sends an email to all admins with the information about the incident (Listing 19):

```
function notifyAdminOnAlarm(alarmEvent) {
    socketHelpers.emitMessage('admin', socketHelpers.GATE_ALARM, alarmEvent);
    // send mail to every admin
    return getMailsOfAdmins()
        .then((mails) => {
            for (const mail of mails) {
                if (mail) mailHelpers.sendAlarmMail(mail, alarmEvent);
            }
        });
}
```

Listing 19: Notifying admins on alarm event

5 Evaluation

5.1 Setup

5.2 Simulation Sizes

Zeigen wie schnell die anfragen zu elastic sind etc

Wie lange dauert rendern

Verschiedene groesse der waittime

5.2.1 Empire State Building Mittagspause

5.2.2 Small

5.2.3 Medium

5.2.4 Large

5.3 Data Privacy

6 Future Works

This work leaves a lot of tasks open before the deployment into a production environment would be possible.

The most important feature is the functionality to link gates together, thereby creating a gate graph. Currently, gates are directly connected to one room and define the number of people for that room. But in an office where multiple gates are installed, some gates can only be reached by passing another gate first. In this situation the current version of the implemented floorplan would lead to wrong calculations for the occupancy statuses. An entry event through a gate g_2 that is behind another gate g_1 would add one person to the room connected to g_2 but not subtract the person from the room connected to g_1, where the person came from. To implement this, we would need further information from the user about the predecessors and successors of each gate.

Further, the current implementation only supports connecting a gate to exactly one room, but of course multiple rooms could be protected by a single gate. Therefore the user should have the ability to connect multiplerooms to a gate.

Office buildings usually have more than one floor. But the implemented floorplan is not designed to work with multiple floors yet. This could be implemented through the already mentioned indoor plugin for Leaflet, which features the option to handle GeoJSON files with integrated data about the floorlevel of each feature and includes map controls to switch between the different floors. We would then also need to persist the floorlevel for each marker, so we can add them to the correct floors.

This also reveals another current usability issue: the manual positioning of the gates. A better solution would be to integrate with already existing GeoJSON data where gates are also included as features.

The next improvement would be the option for the facility manager to set the maximum limit of persons in a room themselves or define a person per squaremeter limit (which would require to know or calculate the squaremeters of each room). Currently this maximum value is hardcoded and therefore is the same

for large and small rooms.

Every time an event happens, a request is send to the backend for the updated number of people that are behind the gate where the event occured. This results in a lot of traffic at each notification about an event. By only requesting the number of people for each gate at startup and the client updating the occupancy state by itself (simply adding to or subtracting from the amount of people in the room), this traffic could be reduced and improve the performance of the floorplan.

Some of the work presented in the Related Work chapter include the option to review and replay events from the past. This could also be useful for our use case. Through this option the facility manager could view how the heatmap developed over time and also replay missed events, giving them insights about which time intervals and areas are critical for the safety of the employees.

7 Conclusion

References

[1] Christopher Baines: *Provides basic tools to create indoor maps with Leaflet: cbaines/leaflet-indoor*, July 2019.
`https://github.com/cbaines/leaflet-indoor`, visited on 2019-07-18, original-date: 2014-04-08T13:43:38Z.

[2] *Documentation - Leaflet - a JavaScript library for interactive maps*.
`https://leafletjs.com/reference-1.5.0.html#map`, visited on 2019-07-21.

[3] *Socket.IO — Server API*.
`https://socket.io/docs/server-api/index.html`, visited on 2019-07-18.

[4] *Socket.IO — Rooms and Namespaces*.
`https://socket.io/docs/rooms-and-namespaces/index.html`, visited on 2019-07-18.

[5] *Socket.IO — Client API*.
`https://socket.io/docs/client-api/index.html`, visited on 2019-07-18.