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 30, 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 30, 2019

(Tim Hehmann)

Abstract

Guaranteeing 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

List of Figures

List of Code Listings

List of Tables

1 Introduction

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 chip card). Not only office buildings, but 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 bachelor's 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 behavior. This is done by continuously analyzing 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 bachelor's project develop this app continuously 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 every single gate is currently done by hardcoding a list of authorized employees 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

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

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

systems 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 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 minimum trust level that is needed to enter or leave a certain gate, which enables them 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 losing the overview of the facility, the usage of an interactive floorplan can be helpful. The implementation of such a plan was also part of the scope of our project and forms 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 visualizes how many persons are approximately in a room and at which gates an alarm occurred.

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 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, related work gets discussed. This will showcase differ-

ent solutions from software companies that already offer services for similar requirements.

The third chapter describes our chosen approach and the architecture and components behind it. Besides that, 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 fourth chapter, we will present our implementation for an interactive floorplan. This will present the solutions used to fulfill the requirements explained in the Concept chapter.

The topic of the fifth chapter is evaluation. In this chapter, we will evaluate how well our floorplan fulfills its requirements. Furthermore, it will discuss the protection of data privacy in a floorplan, especially focussing on the personal information that gets shown in the floorplan.

The sixth 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 seventh 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 through 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 reveal 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 retail shop 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 the import of floorplans.

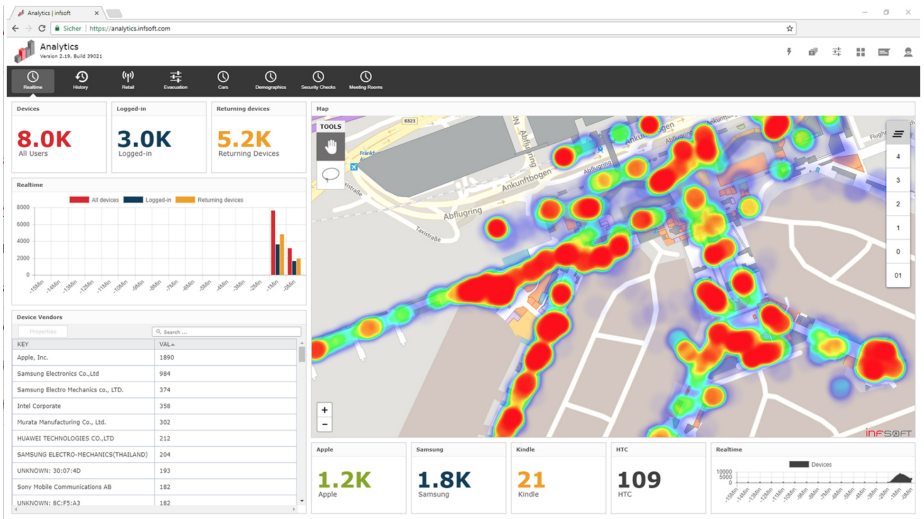

Figure 1: Infsoft Analytics Web Application

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 to protect 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 its 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 built from the ground up.

3 Concept

In this chapter, we want to present the general concept of our approach by giving an overview of our architecture and explain the roles of each component. Afterwards, we will present the external data that is available for us and list all the requirements that we received from our external partner for the implementation of the interactive floorplan.

3.1 System 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 of the architecture can be seen in figure 3.

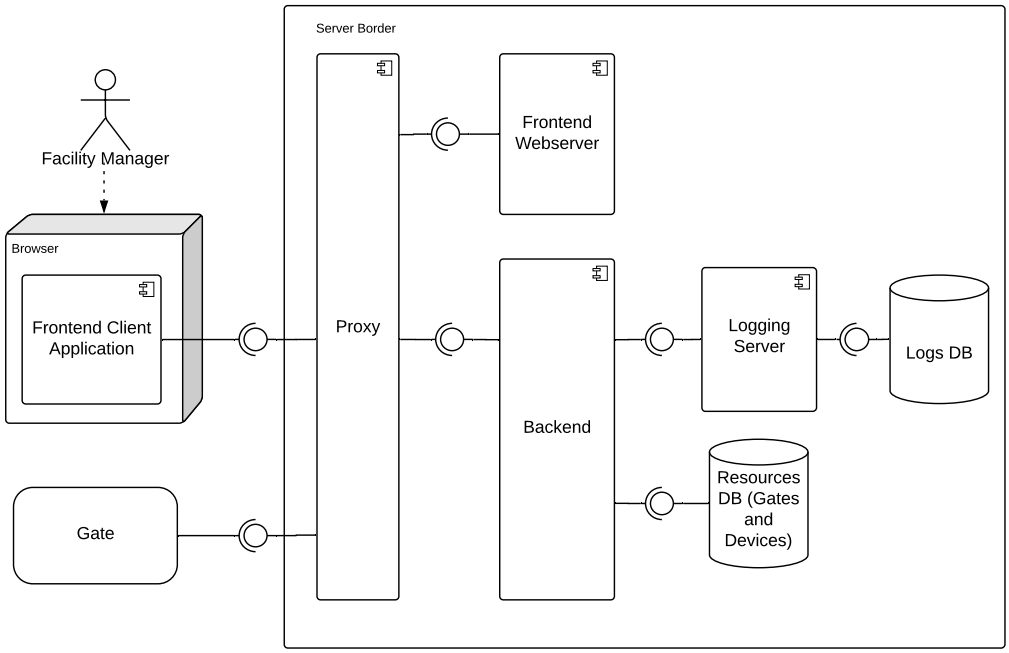

Figure 3: Component diagram

3.1.1 Gates

The gates take care of capturing different pieces of information once a person tries to access through a gate. They gather information about the device that communicates with them, if they granted or denied 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 for 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 that 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 occurred 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 information about the gate, such as the minimum entry and minimum 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 occurred, thereby showing the up-to-date occupancy of that room.

[5] Application Programming Interface

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 calculating the number of people in a room for example.

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 can interact with the map by scrolling, dragging of 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.

4 Implementation

In this chapter, we will present how we implemented the features that were necessary for fulfilling each requirement described in the Concept chapter. This will be guided by the code of our current implementation.

4.1 Choosing technology as a basis

There are multiple libraries available that provide tool kits for creating interactive maps. In this section, we want to showcase these different libraries and explain on what library our interactive floorplan will be built up on. Furthermore, we want to clarify what format of geospatial data will be used in this floorplan.

4.1.1 Google Maps

In March 2011, Google introduced the first indoor floorplans on 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 Google Maps Services on your 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. It's 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 4).

[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 its 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 own plugins or community plugins. The community has already plugins for creating indoor maps [1] and real-time maps with Socket.IO.

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

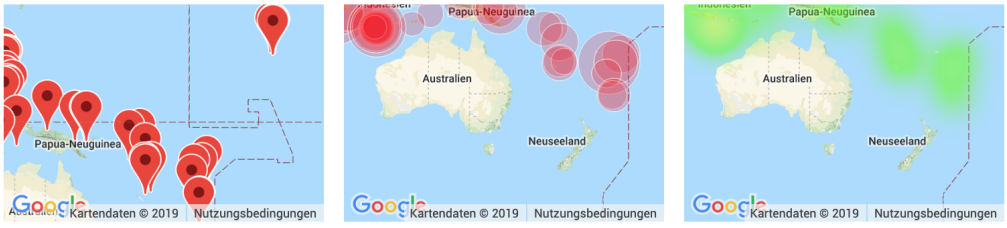

Figure 4: Example of visualization options in Google Maps

4.1.4 GeoJSON

GeoJSON is a format for interchanging geospatial data and is based on JavaScript Object Notation. Since 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.

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 not used.

Leaflet solves this problem by providing only a core set of functionalities that 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

[11] `https://tools.ietf.org/html/rfc7946`
[12] `https://bundlephobia.com/result?p=ol@5.3.3`

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):

```
...
// 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 an 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 occurred.

```
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]

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 drag 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

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

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

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 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):

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

[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 a 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 sent. 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:${accessType}`
        + `%20AND%20`
        + `wasSuccessful:true`
        + `%20AND%20`
        + `timestamp:[${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 reliable connections by having different fallback real-time methods, auto reconnection support and the detection of dis-

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

connections.

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):

```
//app.js
const server = http.createServer(app);
socketHelpers.handleSockets(server);

//socket.helpers.js
function handleSockets(server) {
    // creates Socket.IO Server
    const io = socketIo(server);
    ...
}
```

Listing 8: Creation of Socket.IO server

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

Every time a frontend client now connects to the server, the server extracts the

user id from the token that was sent by the client. We then create a *Room*[4] - a separate communication channel - with this user id, 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 every time 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.IO socket

4.5.3 Heatmap

Every time an event occurs the marker that is linked with the gate id of the event object is searched. Then the room that is connected with this marker is re-colorized 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 date and time 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 was 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 gets also displayed in real-time in a table (*Activity Monitor*) below the interactive floorplan (Figure 5).

Because the gates transmit the device id in the event object, the relationship to the owner of the device can be made through our backend. This happens each time a 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();
```

```
16                  });
17      };
```

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):

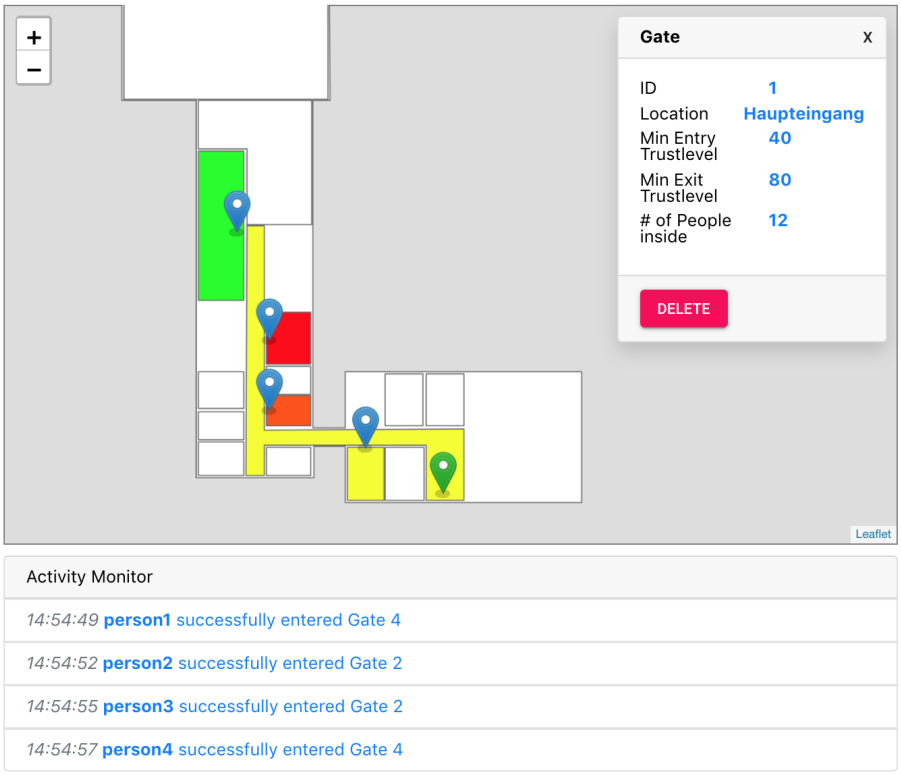

Figure 5: Implemented interactive floorplan

```
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

In this chapter, we will evaluate our interactive floorplan by looking at how well it fulfills the requirements from the Concept chapter. The evaluation will be based on the result of several different measurements. We will explain how these measurements are set up and put the results into context. Besides that, we will also discuss the protection of data privacy in our floorplan.

5.1 Real-time Data Visualization

To measure how performant the real-time data visualization in our system is, we wrote a bash script that fires fake gate events to our backend at different time intervals (Listing 20). The script will execute 100 requests to our backend, which are running asynchronously.

```
#!/bin/bash
GATE_COUNT=10
DEVICE_ID_WHICH_ACCESSED="c8ce80bc-c238-44bf-bc8b-fe68a3dbb957"
SLEEP_TIME=0.1

createGateEventLog() {
    // picks a random gate id
    GATE_ID=$RANDOM
    let "GATE_ID %= $GATE_COUNT"
    //fires event to development server
    curl --insecure -v -X POST https://dev.baam.nexenio.com/api/logs
        -H "Content-Type: application/json"
        -H "Authorization: Basic zdWIiOiIxMjM0NTY3ODkwIiwi"
        -d '{"gateId":'"$GATE_ID"', "wasSuccessful": "true", "accessType": "ENTRY", "
    loglevel": "DEBUG", "deviceId":'"$DEVICE_ID_WHICH_ACCESSED"'}'
}
// make 100 calls to our backend
for ((i=0;i<100;i++));
do
    // sleep time determines the interval at which the events are sent
    sleep $SLEEP_TIME;
    // the usage of the ampersand makes this call asynchronous
    createGateEventLog &
done
```

Listing 20: Bash script for creating fake gate events at intervals

The script will result in multiple notifications from the backend about gate events. In the frontend, we will then look at the delay time between the emission of these notifications and the completion of certain methods. This measurement delivers a good metric for evaluating the real-time data visualization of our application because we can determine the time gap between the actual gate event and the associated display in our floorplan.

To measure the delay time the frontend client is looking at the timestamp of the event object that is received through the notification from the backend and then creates a new timestamp once the desired function has been successfully terminated. By looking at the difference between the timestamps we can then calculate the delay time. The frontend client will keep track of all the 100 single delay measurements and calculate the average.

We set up two points of measurement in the frontend. The first measures the delay time for the completion of the method that re-colorizes the room where the event happened and thereby updates the visual occupancy state of the room. The second is set up in the Activity Monitor and measures the delay for the completion of adding a new log message to the table. The results of this can be seen in Table 1.

	Low Frequency	**Medium Frequency**	**High Frequency**
Entries per minute	6	60	600
Delay time in milliseconds between notification emission (backend) and recolorization of the room (frontend)	166	138	343
Delay time in milliseconds between notification emission (backend) and adding of log message to Activity Monitor (frontend)	219	190	14494

Table 1: Delay times between backend and frontend

As we can see, our application updates the color of the room quite fast in the low frequency and medium frequency simulations with a delay time of just 166ms at low frequency and 138ms at medium frequency. Also, the addition of the log message happens fast with a delay time of 219ms at low frequency and 190ms at medium frequency. But at a high frequency, we can see a drop in performance. Although the updating of the room color still happens quite fast with an average delay time of 343ms, the addition of the log message happens here with a delay time of 14.5 seconds.

This could be because in the high-frequency simulation the script from Listing 20 is sending a request to our backend every 0.1 seconds (600 entries per minute), but the average completion time for the create event log POST request is 0.3 seconds. This means that our backend has to handle multiple POST requests at the same time, which could be critical if the setup of our backend server is not capable of handling this case properly. But also the high load of work that has to happen almost simultaneously in the frontend could be the cause of the performance drop.

But although there are performance issues at high frequency, the average build-

ing will likely not have that many entries per minute, which makes our floorplan performant in displaying data in real-time for the average use case. We can also conclude that the update process of the heatmap each time an event happens is fast and reliable. Moreover, we can conclude that the display of alarm events on the map and in the Activity Monitor is also fast and thereby helps the facility management to quickly react to security threats.

5.2 Logging Server Response Time

To measure how good the event logging in our system works, we will look at the response time from the search request we've seen in Listing 7 at different amounts of logs in our log database. With this measurement, we can see how performant searching through gate logs is in small and large environments.

We measured the response time by executing a bash script which executes a cURL[18] request 100 times and calculates the average response time. The result of this measurement can be seen in Table 2.

Amount of logs	**1000**	**10000**	**100000**	**1000000**
Average response time (in seconds)	0.33	0.33	0.31	0.34

Table 2: Elasticsearch server response times

We see that while the amount of logs is growing exponentially the average response time stays constant. Even for 1 million logs we still have the same response time as for 1000 logs, which makes the display of the heatmap on the floorplan fast, even for a large set of data. The usage of the Elasticsearch server with its fast searching mechanism build inside makes this performance possible.

[18] https://curl.haxx.se/

5.3 Interactive Floorplan

Since there is no clear metric for evaluating interactivity we will look at the performance of the floorplan in click, drag and scroll events at different amounts of gate markers (10, 100 and 1000 markers in the same view frame) on the plan. We utilize the Google Chrome Performance tab for measuring the performance for different events. The tests are done on a MacbookPro with an Intel Iris Graphics 6100 graphics card and a 2,7 GHz Intel Core i5 processor on Google Chrome (Version 75.0.3770.142). The results can be seen in Table 3.

Amount of gate markers	**10**	**100**	**1000**
Frames per second at drag event (changing location in floorplan)	49	19	2
Frames per second at scroll event (changing zoom level of floorplan)	70	66	2
Execution time in milliseconds of click event handler	58	56	56

Table 3: Web application performance for different events

The performance of handling the click events stays constant in our tests no matter the number of gate markers on the map. But we can see that the rendering performance of the floorplan if the user drags or scrolls is dependant on the number of gate markers. At an amount of 1000 gate markers in the view frame the frames per second at dragging or scrolling drop to 2 frames per second. This makes the interactive floorplan unresponsive and therefore uncomfortable to use at large amounts of gate markers. Better handling of the map rendering is necessary here.

5.4 Data Privacy

Our implementation protects data privacy by using a local logging server, which is storing and analyzing gate event data in an internal database. The data is thereby not shared with any third parties. Furthermore, our API endpoint for

retrieving gate event data is guarded by Keycloak[19], an open-source access management solution. This protects the gate event data from unauthorized access. Also, the Socket.IO server is only sending data to users that are authenticated by verifying that the Socket.IO client is requesting data with a valid Keycloak access token. Additionally, we only share this data with trustworthy admin personnel and no other users by using separated communication channels for users and admins.

Moreover, the data that is visible in the frontend about the user that enters or exits a gate is coming from user data that is stored in our system. The gates are only sending us the id of the device, which is then mapped to the stored user data. Therefore we don't reveal any further information about the device or any sensitive data about the owner that could be stored on that device.

[19]https://www.keycloak.org/

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. Although a better solution would be to integrate with an indoor positioning system as this limits the inputs the user has to give our system.

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 multiple rooms 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 floor-level of each feature and includes map controls to switch between the different floors. We would then also need to persist the floor-level 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 square

meter limit (which would require to know or calculate the square meters 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 sent to the backend for the updated number of people that are behind the gate where the event occurred. 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 number 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 includes 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

We had the goal of designing and implementing an interactive floorplan that features the real-time display of occupancy statuses of the rooms in a building as a heatmap. Furthermore, it should also visualize the access decisions made at the gates in real-time.

We only had access to the access decision data coming from the gates. Therefore all software that is working with indoor positioning technologies for displaying indoor heatmaps cannot be used. For that reason, we implemented this floorplan web application from the ground up. As a basis, we used the GeoJSON format together with the JavaScript Leaflet to implement the interactive floorplan. For logging of gate events and analytics we used the Elastic Stack and for the implementation of the real-time functionality Socket.IO.

Our solution offers a performant real-time display by having a delay time between the gate event and the respective update of the heatmap of an average 138ms (at 60 requests per minute). Our solution also provides a fast creation and retrieval of gate event log data. We achieved a constant retrieval response time of 0.33 seconds for small (one thousand) and large (one million) amount of data. As a result, the display of the heatmap happens fast even for large data sets. Furthermore, our floorplan achieves a respectable rendering performance at 100 gate markers on the map.

References

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

[2] *Leaflet — Map API*.
`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.