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 17, 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 17, 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. 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 indifferent 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. 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

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 and came up with *BAuth*[1] (short for Behavioural Authentication), 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.

This new way of authenticating solves the management issue of cards and passwords by authenticating directly with the device. It also lowers the security risk in an event of a theft, because reading a *wrong* behaviour just for a few meters results in a significant drop in trustlevel, thus denying access almost immediately.

But due to the fact that existing access management solutions don't work with this authentication method, the desired protection of certain areas is left open. This is where the scope of this years bachelors project started.

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

The goal of our project was to create an access management platform that is suited to work with BAuth. 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

TODO: erklaeren was interactive genau heisst, Nachteile von Behavioral Auth (z.B. wenn Schuh sich aendert, oder generell verhalten, kommt man nicht mehr rein und so

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.

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 compare different approaches for implementing such a floorplan and present the chosen approach for this project. To accomplish this it will be guided by the following structure:

In the second chapter, related work gets discussed. This will showcase the different technologies that are available right now for creating an interactive floorplan, including the strengths and weaknesses for each one. In this chapter we will also give a short introduction to the tools and formats we will mention in

the further chapters.

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

In the fourth chapter we will present our implementation for a live indoor floorplan. This will show the solutions used for logging gate events and displaying a real-time floorplan with an integrated heatmap.

The topic of the fifth chapter is evaluation. In this chapter the performance of a live plan in differently sized simulation environments gets analysed. Furthermore it will discuss the protection of privacy in a live plan, especially focussing on the personal informations that get 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 Background

The following features were required for the floorplan:

- **Interaction**: The floorplan should be an interactive map. This means that the user is able to zoom, pan and click on the floorplan. It 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**: Events at the gates should be displayed in real-time on the map. This includes also 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, thus presenting which room are currently high occupancy.
- **Alarm Location**: The floorplan needs to visualize alarms, which is needed to locate threats on the map.
- **Upload Functionality**: Because of the fact that every user has different floorplans the upload of own floorplans must be possible.

3 Related Work

3.1 Mazemap

3.2 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[2] 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 2.1.).

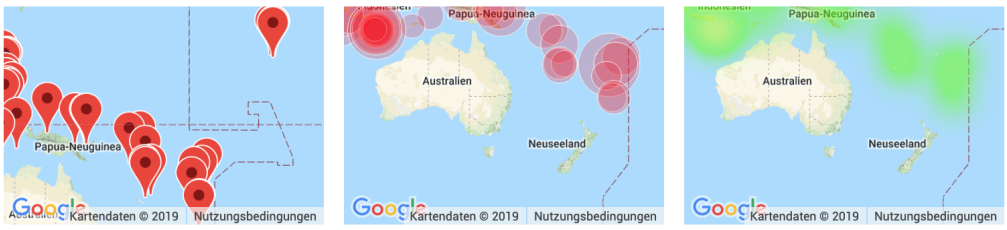

Figure 1: Example of visualization options in Google Maps

Although this 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

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

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.

Therefore Google Maps was not applicable to our task of creating an interactive floorplan.

3.3 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[3] or GML[4] data [5]. Unlike Google Maps, OpenLayers is a pure client-side library with no server-side dependencies.

Because it has a lot of features already bundled together, it offers far more functionality than we need to meet the requirements of our floorplan.

This also results in 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[6]. Although this can be lowered for production by deleting unused modules, it takes extra effort to see which modules are really not used.

Due to the limited time we had for the implementation of the floorplan (due to it being the last feature on our roadmap) and only beginner knowledge of JavaScript, we needed something that is easy to understand and lets beginners output something useful in a short time. Because the abstraction layer of OpenLayers is quite low, the learning curve can be pretty steap. Compared to Leaflet it takes a lot more code to get the same results.

Therefore we thought that OpenLayers doesn't fit our task.

[3] Keyhole Markup Language
[4] Geography Markup Language
[5] `https://openlayers.org/`
[6] `https://bundlephobia.com/result?p=ol@5.3.3`

4 Concept

To be able to meet the requirements of our interactive floorplan, the events from the gate need to be logged and displayed in real-time. For this to work, there a multiple component involved. A general overview about the architecture can be seen in figure 3.1.

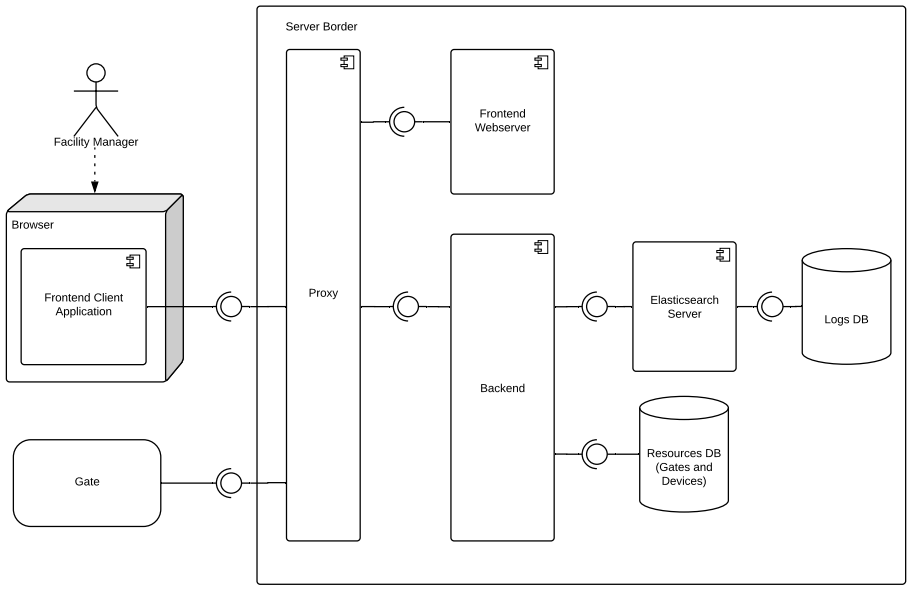

Figure 2: Component diagram

4.1 Gates

The gates take care of capturing different informations once a person tries to enter a gate. They retrieve information about the device that communicates with them, if they granted or denied the access and if the person tried to enter or exit through the gate.

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

If an alarm occured at a gate they need to provide specific information to our

backend.

4.2 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 logs and also other results based on these.

It emits a notification together with the event data once a gate event occured for a real time frontend application.

4.3 Frontend

The frontend presents the interactive floorplan to the facility management with clickable markers for each gate. It features the adding an deleting of marker 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 an event occurs the frontend reads in the gate event informationen. With this it finds out who accessed at which gate and presents that information to the facility management. It then asks for the backend for an updated number of people behind that specific gate. It than calculates the heat in that specific an rerenders the color of that room accordingly.

4.4 Elasticsearch Server

angeschlossen an DB, bietet interface fuer anlegen von event logs und auslesen von event logs + suche

5 Implementation

As this was part of our access management platform and requirement for our project, we implemented this floorplan also in our web application.

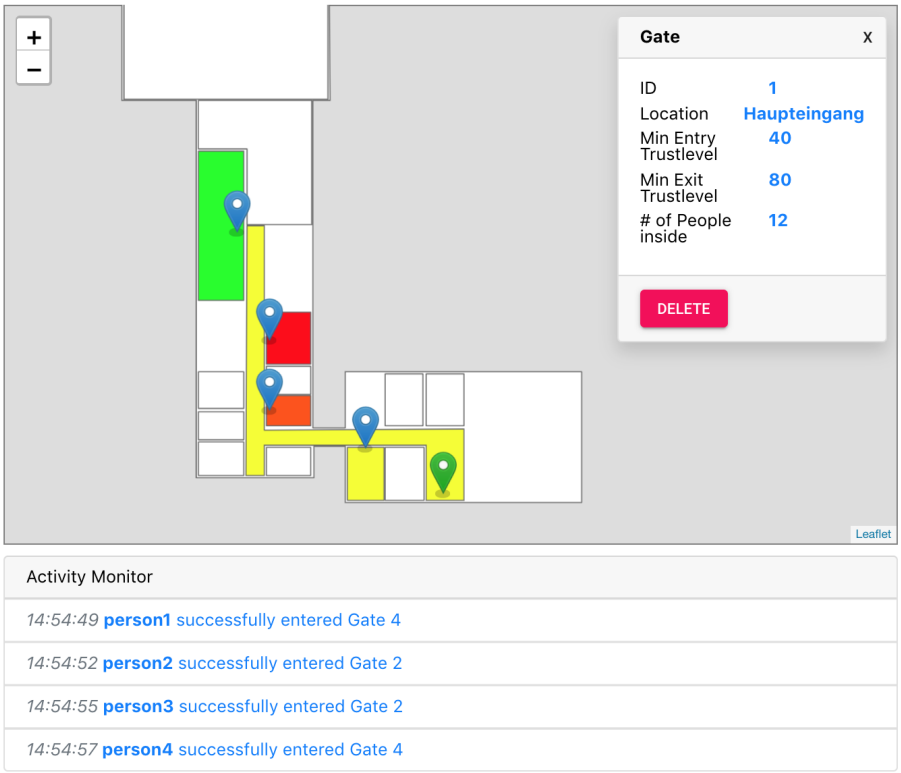

Figure 3: Screenshot of the implemented interactive floorplan

5.1 Display of indoor features

5.1.1 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 Multi-

polygones[7].

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.

5.1.2 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[8], making it a very lightweight library.

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

The indoor map plugin comes with support for loading in GeoJSON. We use the `L.Indoor` class to load in the geospatial data from a GeoJSON file:

```
...
this.map = L.map('floorplan-container', {
            center: new L.LatLng(49.41873, 8.67689),
            zoom: 20,
        });

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

This renders the geoJSON data on a map with a html layer for eachfeature.

TODO: latitude, longitude erklaeren

Hier erwaehnen das jedes feature eine ID benoetigt und so.

[7] https://tools.ietf.org/html/rfc7946
[8] https://bundlephobia.com/result?p=leaflet@1.5.1

5.2 Adding of markers

The floorplan handles mouse clicks on each room that is displayed. If a click event occurs, it adds a new marker at this position, which is then automatically linked with of the underlying room.

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

To create this connection between the room and the marker, each feature/room in the GeoJSON file has to have a distinguashble member-attribute *id*.

This is still following the GeoJSON format specification.

After setting the marker it needs to be linked with the gate. This is done via an Selection Input.

5.3 Logging of Gate Events

5.3.1 Elastic Stack

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 [9].

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

In our project logs sent from the gates need to be analyzed and visualized to give the FM Team more insights about the access decisions made. These informations get also displayed in the floorplan and the heatmap gets calculated based on the gate event logs. 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 include the Elastic Stack into our project.

Since were only working with events coming from one source, the gates, were not using Logstash. And since we provide own custom visualization Kibana is also not used.

Gates nutzen folgendes Endpunkt, ist abgesichert per pre shared token.

We only use the elasticsearch part. The elasticsearch exposes a RESTful API for searching. We use this interface to count the exits and entries.

```
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)));
}
```

The gates have to transmit the ID of the device that tried to access, the access type (entry or exit), the gate ID, if the entry or exit was successful

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

    const url = `http://${elasticsearchBasePath}/gates/_search?sort=timestamp:desc&`
        + `q=gateId:${gateId}%20AND%20accessType=${accessType}%20AND%20timestamp:[${
    officeOpening.toISOString()}+TO+${now.toISOString()}]`;
    return fetch(url, {
        headers: {
```

```
            Accept: 'application/json',
        },
    })
        .then(response => errorHandling.checkResponseOk(response));
}
```

Hier sagen wie man herauskriegt wieviele da sind

Elasticstack routen, indexes bla

5.4 Real-time Floorplan

5.4.1 Socket.IO

Socket.IO is a JavaScript library that makes it possible to implement real-time applications. This is achieved through an event-based, bidirectional communication between the client and the server.

For this to work the client-side needs to install a Socket.IO javascript library and the server needs to run a Socket.IO Node 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[10].

The frontend has to install and set the Socket.IO client and then listens for the events emitted from the backend.

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

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

```
10
11          socket.on('gateEvent', (event) => { this.gateEventHappened(event); });
12          socket.on('gateAlarm', (event) => { this.gateAlarmHappened(event); });
13      };
```

Socket.IO offers an easy to understand API and also comes with a lot of benefits like creating realiable connections by having different fallback real-time methods, auto reconnection support and the detection of disconnections.

As already mentioned in the Related Work Section, we use Socket.IO for creating a real-time interactive floorplan.

In order for this to work correctly the backend server has to emit

```
1  const data = { timestamp: new Date(), ...req.body };
2          return postGateEvent(JSON.stringify(data))
3              .then(() => {
4                  if (data.message === 'ALARM') {
5                      // here other things get done, like notifying the admin via email
6                      notificationHelpers.notifyAdminOnAlarm(data);
7                  } else {
8                      socketHelpers.emitMessage('admin', socketHelpers.GATE_EVENT, data);
9                  }
10                 res.send(msg.getCreateEventSuccessMsg());
11             })
12             .catch(err => next(err));
```

In the frontend we installed the Socket.IO JavaScript client library. This library listens for the Gate Events and then acts accordingly.

```
1  gateEventHappened = (event) => {
2          const gateMarker = this.findGateMarkerWithId(event.gateId);
3
4          if (gateMarker) {
5              this.applyPulseEffectToMarker(gateMarker);
6
7              const { assignedRoomId } = gateMarker.options;
8              getNumberOfPeopleForGateWithId(event.gateId)
9                  .then((data) => {
10                     this.updateGateInfoOfCurrentSelectedMarker(gateMarker, data);
11                     this.colorizeRoom(assignedRoomId, data.count);
12                 });
```

```
        }
    };
```

5.4.2 Alarm

5.4.3 Access Decision Information

More information about the access decisions get also displayed in real-time below the interactive floorplan.

```
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();
            });
    };
```

5.4.4 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 number of people that are behind this gate:

```
  colorizeRoom = (roomId, numberOfPeople) => {
        let occupancyRate = numberOfPeople / constants.ROOM_PERSON_COUNT_LIMIT;
        if (occupancyRate > 1) occupancyRate = 1;
        if (occupancyRate < 0) occupancyRate = 0;
```

```
5
6           const room = this.findRoomWithId(roomId);
7
8           room.setStyle({ fillColor: this.getColorForOccupancyRate(occupancyRate) });
9       };
```

To represent the occupancy of a room we use 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.

```
1       getColorForOccupancyRate = (rate) => {
2           /*
3               input: value from 0 to 1
4               returns: a hsl color on a scale from green to red
5           */
6           const hue = ((1 - rate) * 120).toString(10);
7           return ['hsl(', hue, ',100%,50%)'].join('');
8       };
```

The number of people get calculated by checking the number of succesful entries since the opening of the office minus the succesful exits since the opening of the office. It can be expected that before the opening of the office no one is in the office, thus calculating the correct amount of people

5.5 Persisting Data

Elasticsearch server Security (Hard Coded Token)

6 Evaluation

6.1 Setup

6.2 Simulation Sizes

Zeigen wie schnell die anfragen zu elastic sind etc

Wie lange dauert rendern

Verschiedene groesse der waittime

6.2.1 Empire State Building Mittagspause

6.2.2 Small

6.2.3 Medium

6.2.4 Large

6.3 Data Privacy

7 Future Works

7.1 Possible Improvements

Obwohl elasticsearch performant ist, waere folgendes besser:

Frontend erhoeht und sinkt den Counter fuer die tueren selbst bei erfolgreichem Eintritt und erfragt nicht nochmals alle Eintritte - alle Austritte..

8 Conclusion