

Webasiertes Leitstellen-Konfigurator System

control center configurator

Fabian Thomas

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Tilo Mentler

Auderath, DD.MM.YYYY

---

## Kurzfassung

Die vorliegende Bachelorarbeit behandelt die Entwicklung und Erweiterung eines Feuerwehrleitstellenkonfigurators, der ursprünglich nur ein Frontend ohne Backend-Funktionalitäten umfasste. Das Ziel der Arbeit bestand darin, den bestehenden Konfigurator um eine Backend-Komponente zu erweitern, um eine praktische Anwendung in Feuerwehrleitstellen zu ermöglichen. Zusätzlich wurden sinnvolle Funktionen implementiert, um die Benutzererfahrung zu verbessern und die Effizienz bei der Erstellung und Verwaltung von Leitstellenkonfigurationen zu steigern.

Der Prozess der Erweiterung begann mit der Integration des bestehenden Projekts in ein Next.js-Projekt, um die Implementierung von API-Routinen und Backend-Funktionalitäten zu ermöglichen. Daraufhin wurde eine Datenbank mit Prisma erstellt, um die grundlegende Speicherung und Verwaltung von Szenendaten zu ermöglichen. Die Benutzeroberfläche wurde mithilfe von React MUI verbessert, um eine ansprechende und benutzerfreundliche Bedienung zu gewährleisten.

Um die Sicherheit und Zugriffsrechte zu gewährleisten, wurde eine umfassende Benutzerverwaltung und Rechteverwaltung implementiert. Dabei wurde ein Adminbereich eingerichtet, der es autorisierten Personen ermöglicht, alle Einstellungen zu verwalten und Berechtigungen zu vergeben. Die Authentifizierung wurde durch die Implementierung von Sessions und Rechten sichergestellt, sodass nur berechtigte Personen mit dem Server kommunizieren können.

Ein wesentlicher Aspekt der Arbeit bestand in der Erstellung einer effizienten Datenstruktur zur Speicherung und Verwaltung von Szenen auf dem Server. Eine Szeneverwaltung wurde entwickelt, um Szenen übersichtlich zu organisieren und schnell darauf zugreifen zu können. Zusätzlich wurde ein Echtzeit-Chat pro Szene implementiert, um die Kommunikation zwischen den Benutzern während der Zusammenarbeit an einer Szene zu erleichtern.

Besonders herausragend ist die Implementierung eines Systems, das es mehreren Benutzern ermöglicht, gleichzeitig an einer Szene zu arbeiten. Hierbei wird die Szene mithilfe von Socket.io synchronisiert, um Echtzeitaktualisierungen zu gewährleisten und die kollaborative Arbeit zu erleichtern.

Insgesamt bietet der erweiterte Feuerwehrleitstellenkonfigurator eine benutzerfreundliche, sichere und leistungsfähige Plattform zur Erstellung, Verwaltung und Zusammenarbeit an Leitstellenkonfigurationen. Die praktische Anwendung dieses

Konfigurators in Feuerwehrleitstellen kann die Effizienz und Effektivität der Einsatzplanung und -koordination erheblich verbessern.

---

## Abstract

The present bachelor's thesis deals with the development and expansion of a fire brigade control center configurator, which originally only included a frontend without backend functionalities. The aim of the work was to extend the existing configurator with a backend component to enable practical use in fire brigade control centers. Additionally, meaningful features were implemented to enhance user experience and increase efficiency in the creation and management of control center configurations.

The process of expansion began with integrating the existing project into a Next.js project to enable the implementation of API routines and backend functionalities. Subsequently, a database was created using Prisma to enable basic storage and management of scene data. The user interface was improved using React MUI to ensure an appealing and user-friendly interface.

To ensure security and access rights, a comprehensive user management and permission system was implemented. An admin area was set up, allowing authorized personnel to manage all settings and assign permissions. Authentication was ensured through the implementation of sessions and rights, allowing only authorized users to communicate with the server.

A crucial aspect of the work was the creation of an efficient data structure for storing and managing scenes on the server. A scene management system was developed to organize scenes in a clear manner and allow quick access. Additionally, a real-time chat per scene was implemented to facilitate communication between users collaborating on a scene.

A particularly outstanding feature is the implementation of a system that allows multiple users to work on a scene simultaneously. This is achieved by synchronizing the scene using Socket.io to ensure real-time updates and facilitate collaborative work.

Overall, the expanded fire brigade control center configurator provides a user-friendly, secure, and powerful platform for creating, managing, and collaborating on control center configurations. The practical application of this configurator in fire brigade control centers can significantly improve the efficiency and effectiveness of mission planning and coordination.

---

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	1
<b>2</b>	<b>Stand der Technik</b>	2
2.1	Datenstruktur Analyse	4
<b>3</b>	<b>Konzeption</b>	7
3.1	Zielsetzung	7
3.2	Funktionale Anforderungsanalyse	8
3.3	Architektur und Technologien	9
3.4	Datenmodellierung	11
3.5	Schnittstellen	13
3.5.1	Zugriff auf das Dateisystem	13
3.5.2	Zugriff auf die Datenbank	13
3.6	Sicherheit und Sessions	14
<b>4</b>	<b>Realisierung</b>	16
4.1	Version 1	16
4.1.1	Integration des Frontendkonfigurator in ein Projekt mit Backend	16
4.1.2	Scene auf Server speichern	17
4.1.3	Raum mit eigenen Objekten erstellen	18
4.1.4	TreeView	18
4.2	Version 2	19
4.2.1	Database, Login, Benutzer und Chat	19
4.3	Version 3	21
4.3.1	Objekt ein-/ausblenden	21
4.3.2	Farbe und Texturen	21
4.3.3	Adminarea	21
4.4	Version 4	22
4.4.1	Datenstrukturen überarbeitet	22
4.4.2	Texturen selber hinzufügen	22
4.4.3	Memberships	22
4.5	Version 5	23

4.5.1 Rechte überarbeitet .....	23
4.5.2 Adminbereich erweitert .....	23
4.6 Version 6 .....	23
4.6.1 CurrentWorkingScene .....	23
4.6.2 Speichern und Synchronisieren einer Scene .....	24
4.6.3 Licht .....	25
<b>5 Ergebnisdarstellung .....</b>	<b>27</b>
5.1 Login und Register .....	27
5.2 Home und Navigatebar .....	28
5.3 Scenelist und SceneMemberships .....	28
5.3.1 Leitsellenkonfiguration Main .....	29
5.4 FbxList .....	34
5.5 TextureList .....	34
5.6 Settings .....	36
<b>Glossar .....</b>	<b>37</b>
<b>Selbstständigkeitserklärung .....</b>	<b>38</b>

# 1

---

## Einleitung

Die Feuerwehrleitstelle stellt eine zentrale Komponente im Feuerwehrwesen dar, die eine effiziente Koordination von Rettungseinsätzen ermöglicht. Sie bildet das Herzstück der Einsatzsteuerung und stellt sicher, dass alle erforderlichen Ressourcen und Informationen zur richtigen Zeit am richtigen Ort verfügbar sind. In der Vergangenheit wurde bereits eine Softwarelösungen entwickelt, die es ermöglicht, Feuerwehrleitstellen virtuell zu konfigurieren und an die individuellen Bedürfnisse anzupassen.

Diese Bachelorarbeit konzentriert sich auf die Entwicklung und Erweiterung eines Feuerwehrleitstellenkonfigurators, der ursprünglich nur ein Frontend ohne Backend-Funktionalitäten umfasste. Ziel dieser Arbeit ist es, den bestehenden Konfigurator um eine Backend-Komponente zu erweitern, um eine praktische Anwendung in Feuerwehrleitstellen zu ermöglichen. Dabei sollen zusätzlich sinnvolle Funktionen implementiert werden, um die Benutzererfahrung zu verbessern und die Effizienz bei der Erstellung und Verwaltung von Leitstellenkonfigurationen zu steigern.

Der Feuerwehrleitstellenkonfigurator ermöglicht es, komplexe KOnfigurationen einer Feuerwehrleitsetelle in einer virtuellen Umgebung zu erstellen. Dadurch können Feuerwehreinsätze effektiver und effizienter geplant werden, was im Ernstfall lebensrettend sein kann. Da der bestehende Konfigurator bisher kein Backend aufwies, sind seine Anwendungsmöglichkeiten begrenzt. Daher ist die Erweiterung um eine Backend-Komponente von entscheidender Bedeutung,

Im Rahmen dieser Arbeit wird das gesamte Projekt auf Next.js migriert, um eine solide Backend-Unterstützung zu gewährleisten. Prisma wird zur Erstellung und Verwaltung der Datenbank verwendet, während React MUI die Oberflächenkomponenten bereitstellt. Zusätzlich wird socket io verwendet um bestimmte Daten zu mit anderen Clients des Systems zu synchronisieren.

Die vorliegende Arbeit wird zunächst den Entwicklungsprozess von der Integration des bestehenden Projekts in ein Next.js-Projekt bis zur Implementierung einer Benutzerverwaltung und Rechteverwaltung sowie einer Szeneverwaltung beschreiben. Weiterhin wird die Implementierung einer Echtzeit-Kollaborationsfunktion, die es mehreren Benutzern ermöglicht, gleichzeitig an einer Szene zu arbeiten, vorgestellt. Zum Schluss werden die erreichten Ergebnisse des erweiterten Konfigurators dargestellt.

## 2

---

### Stand der Technik

Ein wichtiger Aspekt des vorhandenen Leitstellen-Konfigurators ist die Nutzung von React, einem beliebten JavaScript-Framework zur Entwicklung von Benutzeroberflächen. React ermöglicht es, interaktive und reaktive Komponenten zu erstellen, die eine schnelle und effiziente Aktualisierung der Benutzeroberfläche ermöglichen. Durch die Verwendung von React wurde die Steuerung und Verwaltung der Benutzerinteraktionen realisiert.

Ein weiteres zentrales Element des Leitstellen-Konfigurators ist die Verwendung von Three.js bzw. Three.js Fiber. Three.js ist eine JavaScript-Bibliothek für die Erstellung von 3D-Grafiken im Web. Three.js Fiber ist eine spezielle Version von Three.js, die speziell für die Verwendung mit React entwickelt wurde. Sie bietet Funktionen zur Darstellung von 3D-Modellen, Kameraeinstellungen und Szenenmanipulation. Die Verwendung von Three.js ermöglicht es, 3D-Darstellungen der Leitstellenumgebung zu erzeugen. 3D-Modelle können mit einer FBX-Datei geladen werden. Das sieht im Code so aus:

```
1 import { useRef } from "react";
2 import { TransformControls } from "@react-three/drei";
3 import { useLoader } from "@react-three/fiber";
4 import { FBXLoader } from "three/examples/jsm/loaders/FBXLoader";
5 import * as THREE from "three";
6
7 function SceneModel(props: { modelPath: string }) {
8   const fbx: THREE.Group = useLoader(FBXLoader, props.modelPath);
9
10  const refMesh = useRef<THREE.Mesh>(null);
11  const tcRef = useRef<any>(null);
12
13  return (
14    <TransformControls ref={tcRef} getObjectsByProperty={undefined}>
15      | <primitive ref={refMesh} object={fbx.clone(true)}></primitive>
16    </TransformControls>
17  );
18}
19
20 export default SceneModel;
```

Abbildung 2.1: Code um ein FBX-Model zu laden

Der vorliegende Code ist eine React-Komponente, die eine 3D-Modellansicht rendernt und es ermöglicht, das Modell mithilfe der "TransformControls" zu bearbeiten. Die Komponente verwendet verschiedene Importe, darunter "useLoader" von React Three Fiber, um das 3D-Modell mit dem "FBXLoader" von three.js aus dem angegebenen Pfad zu laden.

Die Komponente erstellt Referenzen für das geladene Mesh und das TransformControls-Objekt, um später auf sie zugreifen zu können. Das "TransformControlsElement" enthält die Steuerelemente, die es ermöglichen, das Modell zu verschieben, zu drehen und zu skalieren.

Die Hauptfunktionen des Leitstellen-Konfigurators umfassen das Hinzufügen und Löschen von FBX-Modellen sowie die Manipulation der Modelleigenschaften wie Verschieben, Skalieren und Rotieren, welche auch zusammendfassend als Transformationen genannt werden. Diese funktionen können in der Toolbar (im roten Kreis im Bild XYZ) geändert werden. Durch die Umschaltung zwischen Translation (verschieben), Rotation und Skalieren werden andere Transformcontrols, aus three drei, angezeigt. Three drei ist eine wachsende Sammlung nützlicher Hilfsfunktionen und voll funktionsfähiger, einsatzbereiter Abstraktionen für three fiber.

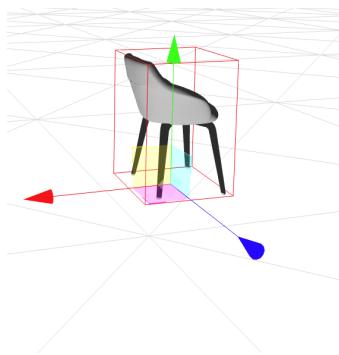


Abbildung 2.2: Bild 1

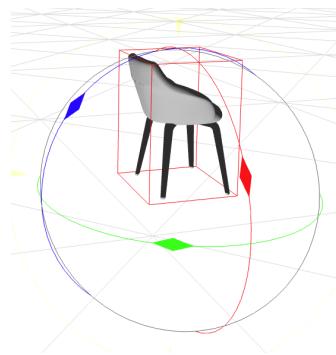


Abbildung 2.3: Bild 2

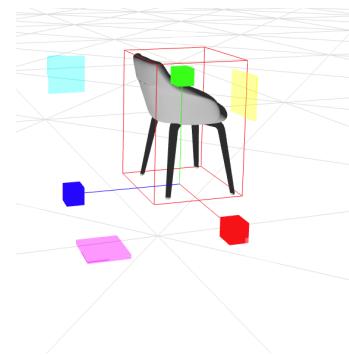


Abbildung 2.4: Bild 3

Abbildung 2.5: Drei Bilder nebeneinander

Darüber hinaus bietet der Konfigurator die Möglichkeit, zwischen verschiedenen Kameraperspektiven (orangener Kreis) zu wechseln, einschließlich einer perspektivischen und orthogonalen Ansicht. Die perspektivische Kamera wird zur Simulation einer realistischen Ansicht verwendet, während die orthogonale Kamera vor allem für eine Top-Down-, Frontal-, Left-Mid-, Right-Mid-Ansicht verwendet wird. Dies bietet die Möglichkeit, Objekte präzise an die gewünschte Position zu transformieren, ganz nach den individuellen Vorstellungen.

Zusätzlich ermöglicht der Leitstellen-Konfigurator das Exportieren der Szenen als GLTF-Datei (pinker Kreis), um sie in anderen Anwendungen anzuzeigen oder weiterzuverarbeiten. Die Save und Load Funktionen (grüner Kreis) erlauben es, die erstellte Leitstellenkonfiguration zu speichern und später wieder zu laden. Des Weiteren besteht die Möglichkeit, die Raummaße anzupassen, indem die Breite,

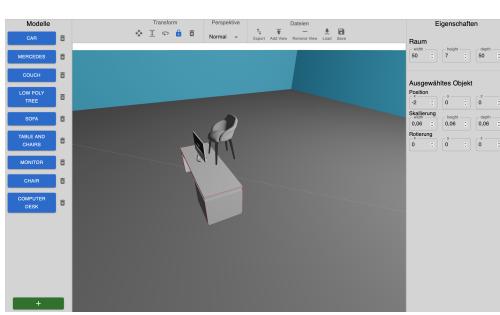


Abbildung 2.6: Orthogonale Kamera

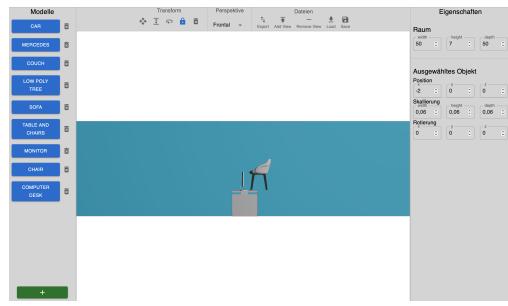


Abbildung 2.7: Perspektivische Kamera

Abbildung 2.8: Zeigt die unterschiedlichen Kameraeinstellungen

Tiefe und Höhe des virtuellen Raums verändert werden können (blauer Kreis).

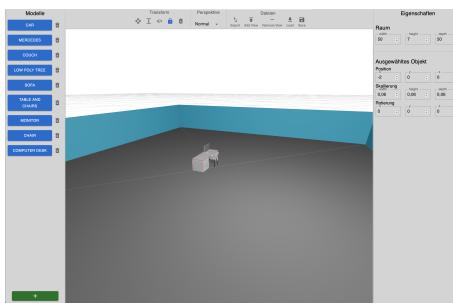


Abbildung 2.9: Der frontend-Leitstellenkonfigurator

## 2.1 Datenstruktur Analyse

Jedes Objekt in der Scene hat bestimmte Werte, die TypeObjectProps. TypeObjectProps hat die Felder:

1. **id**: Ein eindeutiger Bezeichner in Form einer Zeichenkette (String), der jedes Objekt im Raum identifiziert
2. **position**: Eine Objektspezifikation vom Typ "TypePosition", welche die Koordinaten der Position des Objekts im dreidimensionalen Raum mit den Werten für X, Y und Z enthält.
3. **scale**: Ebenfalls eine Objektspezifikation, hier vom Typ "TypeScale", welche die Skalierung des Objekts in den X, Y und Z Achsen definiert.
4. **rotation**: Eine Objektspezifikation "TypeRotation", die die Rotationswerte für das Objekt in den X, Y und Z Achsen angibt.
5. **editMode**: Eine Variable vom Typ SString oder undefined, die angibt, in welchem Bearbeitungsmodus sich das Objekt befindet. Die möglichen Werte sind

„scale“ für Skalierung, „translate“ für Positionierung, „rotate“ für Rotation oder „undefined“, wenn das Objekt nicht im Bearbeitungsmodus ist.

6. **showXTransform, showYTransform und showZTransform:** Diese Felder sind boolesche Variablen (Ja/Nein-Felder), die anzeigen, welche der Achsen (X, Y und Z) im TransformControls des Objekts angezeigt werden sollen.
7. **modelPath:** Ein String-Feld, das den Pfad zur FBX-Datei des 3D-Modells des Objekts angibt.
8. **removeBoundingBox:** Eine Funktion, die dazu dient, die rote BoundingBox des Objekts zu entfernen.

Diese Eigenschaften ermöglichen es, jedes Objekt eindeutig im Raum zu positionieren und die Transformationen (Skalierung, Positionierung, Rotation) entsprechend anzupassen. Der „editMode“ dient zur Unterscheidung und Steuerung des Transformationsmodus für das Objekt, während die „showXTransform“, „showYTransform“ und „showZTransform“ Felder die Sichtbarkeit der entsprechenden Achsen im TransformControls steuern.

Die „TypePosition“, „TypeScale“ und „TypeRotation“ binden spezifische Objektspezifikationen, die jeweils die Koordinaten für Position, Skalierung und Rotation in den dreidimensionalen Raum abbilden. Durch die Kombination dieser Eigenschaften wird eine präzise und eindeutige Platzierung und Anpassung der Objekte innerhalb der Szene ermöglicht.

Als Beispiel eine gespeicherte Scene:

Anhand der JSON Datei kann man erkennen, dass der Raum 7 Einheiten hoch, 50 Einheiten breit und 50 Einheiten lang ist und sich 3 Objekte in der Scene befinden.

Damit die unterschiedlichen Komponenten wie zum Beispiel die Toolbar, die in Abb. XYZ zu sehen ist, das aktuelle Object in der Scene verändern kann, bekommen die Komponenten dies als currentObjectProps übergeben. CurrentObjectProps enthalten immer die aktuellen TypeObjectProps des aktuell ausgewählten Objekt in der Scene.

```
{
  "roomDimensions": {
    "height": 7,
    "width": 50,
    "depth": 50
  },
  "models": [
    {
      "id": "123567",
      "position": {
        "x": -2,
        "y": 0,
        "z": 0
      },
      "scale": {
        "x": 0.06,
        "y": 0.06,
        "z": 0.06
      },
      "rotation": {
        "x": 0,
        "y": 0,
        "z": 0
      },
      "showXTransform": false,
      "showYTransform": false,
      "showZTransform": false,
      "modelPath": "./ModelsFBX/Computer Desk.FBX"
    },
    {
      "id": "12321321367",
      "position": {
        "x": -1,
        "y": 0,
        "z": 0
      },
      "scale": {
        "x": 0.03,
        "y": 0.03,
        "z": 0.03
      },
      "rotation": {
        "x": 0,
        "y": -1.6,
        "z": 0
      },
      "showXTransform": false,
      "showYTransform": false,
      "showZTransform": false,
      "modelPath": "./ModelsFBX/Chair.FBX"
    },
    {
      "id": "123211231233321367",
      "position": {
        "x": 2.0517650695421015,
        "y": 1.83353328885948,
        "z": 3.489659672608047
      },
      "scale": {
        "x": 0.03,
        "y": 0.03,
        "z": 0.03
      },
      "rotation": {
        "x": 0,
        "y": 1.6,
        "z": 0
      },
      "showXTransform": false,
      "showYTransform": false,
      "showZTransform": false,
      "modelPath": "./ModelsFBX/Monitor.FBX"
    }
  ]
}
```

# 3

---

## Konzeption

Für das Kapitel "Konzeption" Ihrer Bachelorarbeit, in dem Sie einen webbasierten Three.js-Konfigurator um ein Backend erweitern möchten, gibt es mehrere wichtige Aspekte, die Sie berücksichtigen sollten. Hier sind einige Punkte, die Sie in dieses Kapitel aufnehmen können:

### 3.1 Zielsetzung

Das Hauptziel dieser Bachelorarbeit besteht darin, den bestehenden Feuerwehrleitstellenkonfigurator weiterzuentwickeln und um ein Backend zu erweitern. Der Fokus liegt darauf, eine umfassende und praktikable Lösung zu schaffen, die in Feuerwehrleitstellen effektiv eingesetzt werden kann. Die Entwicklung des Backends ermöglicht die Speicherung und Verwaltung von Konfigurationsdaten in einer Datenbank, wodurch die Konfigurationen dauerhaft gespeichert und von berechtigten Benutzern problemlos abgerufen werden können.

Die Zielsetzung der Erweiterung beinhaltet folgende Aspekte:

1. **Integration eines Backend-Systems:** Das bestehende Frontend soll um ein Backend erweitert werden, um die persistenten Speicher- und Verwaltungsfunktionen für die Szenenkonfigurationen zu ermöglichen. Hierfür wird ein geeignetes Backend-Framework wie Next.js verwendet, um API-Routinen und Datenbankverbindungen zu implementieren.
2. **Benutzer- und Rechteverwaltung:** Die Erweiterung umfasst die Implementierung einer Benutzerverwaltung und Rechteverwaltung. Dies ermöglicht es, Benutzer mit spezifischen Zugriffsrechten zu definieren und so die Sicherheit und Integrität der Szenenkonfigurationen zu gewährleisten.
3. **Adminbereich und Authentifizierung:** Ein zentraler Adminbereich wird implementiert, um berechtigten Personen die Verwaltung aller relevanten Einstellungen und Benutzerrechte zu ermöglichen. Die Integration einer sicheren Authentifizierung mit Sessions gewährleistet, dass nur autorisierte Personen mit dem Server kommunizieren können.
4. **Datenstruktur und Szenenverwaltung:** Eine effiziente Datenstruktur wird entwickelt, um die Szenen auf dem Server zu speichern und zu organisieren.

Eine Szenenverwaltung ermöglicht die einfache Erstellung, Bearbeitung und Abrufung von Szenenkonfigurationen.

5. **Chat-Funktion und Echtzeit-Kollaboration:** Ein Chatsystem pro Szene wird integriert, um die Kommunikation zwischen Benutzern während der gemeinsamen Arbeit an einer Szene zu erleichtern. Zusätzlich wird eine Echtzeit-Kollaborationsfunktion implementiert, die es mehreren Benutzern ermöglicht, synchron an derselben Szene zu arbeiten.
6. **Weitere sinnvolle Funktionalitäten:** Weitere nützliche Funktionen werden entwickelt, um die Benutzererfahrung zu verbessern und die Effizienz bei der Konfigurationserstellung zu steigern. Dazu gehören beispielsweise die Möglichkeit xxxxx des Szenenexports im GLTF-Format und die Möglichkeit, die Ansicht der Szene zu wechseln, um beispielsweise eine Top-Down-Ansicht zu betrachten.

Die Zielsetzung dieser Arbeit ist es, einen erweiterten und leistungsfähigen Feuerwehrleitstellenkonfigurator zu schaffen, der im praktischen Betrieb effektiv eingesetzt werden kann und eine intuitive und sichere Umgebung für die Planung der Feuerwehrleitstelle bereitstellt.

## 3.2 Funktionale Anforderungsanalyse

Die funktionale Anforderungsanalyse bildet das fundamentale der Softwareentwicklung, da sie die grundlegenden Funktionen und Aufgaben definiert, die das zu entwickelnde System erfüllen muss. In diesem Kapitel liegt unser Fokus auf der Beschreibung der funktionalen Anforderungen für den erweiterten Feuerwehrleitstellenkonfigurator. Ziel dieses Analyseschrittes ist es, einen klaren und präzisen Überblick über die essentiellen Funktionen zu gewinnen, die den Benutzern ermöglichen, ihre Feuerwehrleitstellen in einer intuitiven und effizienten Umgebung zu konfigurieren.

1. **CurrentSceneEdit:** Jeder Eintrag in der Tabelle enthält eine eindeutige ID für die aktuelle Bearbeitung, die ID des Benutzers, der die Szene bearbeitet, die ID der zugehörigen Szene und das Datum, an dem die Bearbeitung begonnen wurde. Die Tabelle ermöglicht es, die aktuellen bearbeitenden Benutzer einer Szene zu identifizieren, sodass andere Benutzer über die laufende Bearbeitung informiert werden können.

### 2. Szenenerstellung und -bearbeitung

Benutzer sollen in der Lage sein, neue Szenen zu erstellen und bestehende Szenen zu bearbeiten. Die Szenen sollen eine dreidimensionale Darstellung der Feuerwehrleitstellen mit verschiedenen Objekten enthalten, die transformiert werden können (Skalierung, Positionierung, Rotation). Wenn mehrere Leute dran arbeiten dann synchron.

### 3. Speichern und Laden von Szenen

Benutzer sollen die Möglichkeit haben, erstellte Szenen zu speichern und jederzeit wieder aufrufen können.

**4. Benutzerverwaltung und Rechteverwaltung**

Es soll eine Benutzerverwaltung geben, die die Registrierung neuer Benutzer und die Anmeldung von bereits registrierten Benutzern ermöglicht. Administratoren sollen die Möglichkeit haben, die Benutzerrollen und -rechte zu verwalten, um den Zugriff auf bestimmte Funktionen und Szenen zu kontrollieren.

**5. Chat-Funktion pro Szene**

Jede Szene soll über einen Chat verfügen, der es den Benutzern ermöglicht, in Echtzeit zu kommunizieren, während sie an derselben Szene arbeiten.

**6. Echtzeit-Kollaboration**

Benutzer sollen gleichzeitig und synchron an derselben Szene arbeiten können, wobei Änderungen in Echtzeit für alle Beteiligten sichtbar sind.

**7. Rückgängig und Wiederherstellen**

Benutzer sollen Änderungen an der Szene rückgängig machen und wiederherstellen können, um versehentliche Fehler zu korrigieren.

**8. Szenenverwaltung**

Es soll eine übersichtliche Verwaltungsoberfläche geben, um alle erstellten Szenen anzuzeigen und zu organisieren.

**9. Authentifizierung und Sicherheit**

Das System soll eine sichere Authentifizierung implementieren, um unbefugten Zugriff auf die Funktionalitäten zu verhindern.

### 3.3 Architektur und Technologien

Die geplante Architektur der erweiterten Anwendung umfasst eine Kombination aus bewährten Technologien, die eine robuste und skalierbare Lösung gewährleisten. Als Backend-Framework habe ich mich für **Next.js** entschieden, da es eine leistungsstarke Plattform für die Entwicklung von React-Anwendungen bietet und eine nahtlose Integration von Serverseitigem Rendern ermöglicht. Next.js ermöglicht es uns, effizient APIs zu erstellen und das Backend mit der vorhandenen Three.js-Anwendung zu verbinden.

Für die Datenbankverwaltung habe ich mich für **Prisma** entschieden. Prisma ist ein leistungsfähiges ORM (Object-Relational Mapping)-Tool, das die Interaktion mit der Datenbank vereinfacht und eine einfache Modellierung der Daten ermöglicht. Es ermöglicht uns, eine effiziente und zuverlässige Datenbankstruktur aufzubauen und die Szenen, 3D-Modelle und Benutzerdaten zentralisiert zu speichern.

Die Integration des Chatsystems wird mit Hilfe von **Socket.io** realisiert. Socket.io ist eine Bibliothek, die Echtzeitkommunikation zwischen Client und Server ermöglicht und sich ideal für die Implementierung eines Echtzeit-Chats eignet. Dadurch können die Benutzer während der Konfiguration miteinander interagieren und sich in Echtzeit austauschen. Nicht nur der Chat wird mithilfe von Socket.io implementiert, sondern auch die Echtzeit-Kollaboration. Mehrere Benutzer sollen gleichzeitig an derselben Szene arbeiten können und ihre Aktionen werden in

Echtzeit mit anderen geteilt und synchronisiert. Dies schafft eine dynamische Umgebung, in der die Benutzer ihre Änderungen und Anpassungen in Echtzeit sehen können,

Die Oberfläche wird mit **React MUI** (Material-UI) gestaltet, einer beliebten React-Komponentenbibliothek, die ein modernes und ansprechendes Design ermöglicht. Durch die Verwendung von React MUI können wir eine konsistente Benutzeroberfläche mit vorgefertigten Komponenten erstellen und das Benutzererlebnis verbessern.

Die Entscheidung für diese Technologien basiert auf ihrer Stabilität, Flexibilität und ihrer Unterstützung für die Anforderungen der erweiterten Anwendung. Sie ermöglichen eine effiziente Backend-Integration, eine zuverlässige Datenbankverwaltung, Echtzeitkommunikation und eine ansprechende Benutzeroberfläche. Durch die Integration dieser Technologien werden ich in der Lage sein, eine leistungsstarke und umfassende Lösung für den Feuerwehrleitstellen-Konfigurator zu schaffen.

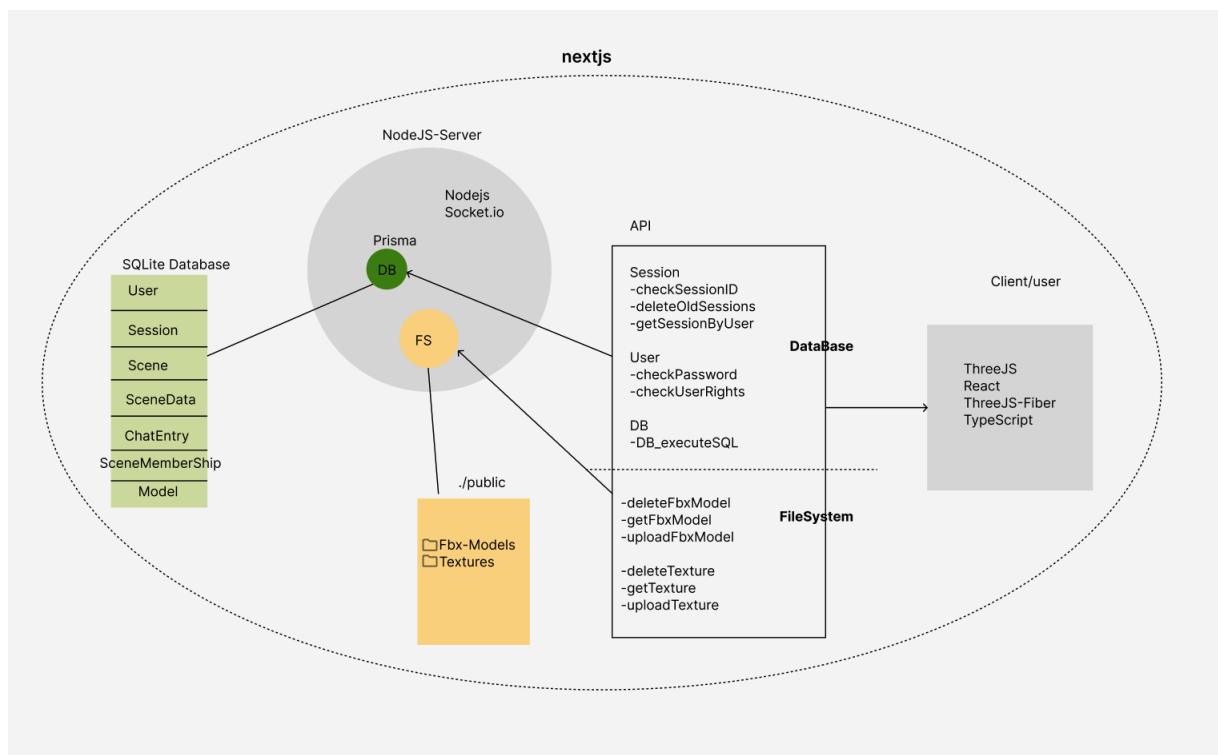


Abbildung 3.1: Architektur

Damit der Client mit dem Server kommunizieren kann gibt es die sogenannten Schnittstellen- bzw. API-Funktionen. Die API kann man in 2 Teile aufteilen. Ein Teil greift auf die Datenbank zu und der andere reift auf das Filesystem zu. Für die Datenmanipulation in der Datenbank gibt es eine API-Funktion, DB-executeSQL. Die funktion hat ein paar Parameter über die man die Datenmanipulation steuern kann. Daten können Daten geladen, erstellt, geloescht und geaendert werden.

Rechte und sessionID werden serverseitig geprüft!!!

## 3.4 Datenmodellierung

Es gibt verschiedene Tabellen in der Datenbank um verschiedenste Daten zu speichern. Erstmal gibt es eine Tabelle User. Dort werden Daten über den angemeldeten Benutzer gespeichert, wie loginID und Passwort. Jeder User hat Rechte, die angeben ob er Daten lesen, schreiben, ändern oder löschen darf. Zudem gibt es noch ein Feld das angibt ob der Benutzer ein Admin ist. Zu dem kann ein User viele ChatEntries haben. Also gibt es eine 1:n Beziehung zu der Tabelle ChatEntries. Ein ChatEntry hat einen Verweis auf den Benutzer der die Nachricht geschrieben hat plus wann die Nachricht verschickt wurde.

Dann gibt es noch die Tabelle Scene. Dort werden die erstellten Leistellen konfigurationen gespeichert. Zu jeder Konfiguration wird der Creator, der Name der Konfiguration, wann sie erstellt wurde und eine Version gespeichert. Jede Scene/-Konfiguration hat viele Memberships. Also besteht auch hier eine 1:n Beziehung zwischen Scene und SceneMembership.

Eine SceneMembership gibt an welcher User welche Konfiguration sehen kann. Man kann jedem Membership, angeben, ob man sich die Konfiguration nur ansehen kann, oder ob man diese auch bearbeiten kann.



Abbildung 3.2: Datenstruktur

In jeder Konfiguration sind Objekte enthalten, wie zum Beispiel eine Wand oder ein FBX-Modell. Diese werden alle in der Tabelle Model gespeichert. Für jedes Modell werden Position, Skalierung und Rotation im Raum gespeichert. So kann man immer feststellen wo sich die Objekte im Raum befinden. Zudem wird noch gespeichert, zu welcher Scene das Objekt gehört und in welcher Version der Scene das Objekt gehört. Zusätzlich werden noch andere wichtige Daten für das Handling im Code gespeichert.

Es gibt die Felder und es wird beschrieben wozu die Felder benötigt werden.

1. **User:** Die Tabelle User enthält Informationen über jeden Benutzer, einschließlich einer eindeutigen ID, Login-ID, Passwort und Zugriffsrechten. Die Felder "read", "write" und "delete" geben an, ob der Benutzer Lese-, Schreib- und Löschrechte besitzt, während "isAdmin" den Administratorstatus kennzeichnet. Die Tabelle speichert auch die erstellten Chat-Einträge des Benutzers und seine Mitgliedschaften in verschiedenen Szenen.
2. **Session:** Die Tabelle Session enthält Informationen über die Sitzungen der Benutzer, einschließlich einer eindeutigen ID für jede Sitzung, der ID des zugehörigen Benutzers und dem letzten Startzeitpunkt der Sitzung. Die Tabelle ermöglicht die effiziente Verwaltung und Zuordnung von Sitzungen zu den entsprechenden Benutzern im System.
3. **ChatEntry:** Jeder Nachricht enthält eine eindeutige ID, die ID der zugehörigen Szene, die ID des Benutzers, der die Nachricht gesendet hat, den Nachrichtentext, das Datum und eine Verbindung zum Benutzer, der die Nachricht gesendet hat.
4. **Scene:** Jeder Eintrag in der Tabelle enthält eine eindeutige ID für die Szene, die ID des Benutzers, der die Szene erstellt hat, den Namen der Szene, das Erstellungsdatum und die neueste Version der Szene. Das Feld "idUserCreator" bildet eine Verbindung zum Benutzer, der die Szene erstellt hat, und ermöglicht somit die Zuordnung jeder Szene zu ihrem Ersteller. Die Tabelle speichert auch Informationen über die Mitgliedschaften von Benutzern in den verschiedenen Szenen.
5. **Model:** entspricht den TypeObjectProps
6. **SceneMembership:** Dient zur Verwaltung der Mitgliedschaften von Benutzern in den verschiedenen Szenen. Jeder Eintrag in der Tabelle enthält eine eindeutige ID für die Mitgliedschaft, die ID der zugehörigen Szene, die ID des Benutzers, das Eintrittsdatum in die Szene und eine Angabe, ob der Benutzer nur Lesezugriff (read-only) auf die Szene hat. Die Felder "idScene" und "idUser" stellen Verbindungen zu den Szenen und Benutzern her und ermöglichen es, die Mitgliedschaften entsprechend zuzuordnen.
7. **CurrentSceneEdit:** Jeder Eintrag in der Tabelle enthält eine eindeutige ID für die aktuelle Bearbeitung, die ID des Benutzers, der die Szene bearbeitet, die ID der zugehörigen Szene und das Datum, an dem die Bearbeitung begonnen wurde. Die Tabelle ermöglicht es, die aktuellen bearbeitenden Benutzer einer Szene zu identifizieren, sodass andere Benutzer über die laufende Bearbeitung informiert werden können.

Die Beschreibung der Tabellen liefert ein Verständnis für die Struktur und Organisation der Daten im erweiterten Feuerwehrleitstellenkonfigurator. Jede Tabelle erfüllt eine spezifische Rolle und ermöglicht das Speichern und Verwalten relevanter Informationen. Die Datenstruktur bildet das Fundament für den Betrieb des Konfigurators.

## 3.5 Schnittstellen

Beschreiben Sie die Schnittstellen, die zwischen dem Frontend und dem Backend Ihrer Anwendung benötigt werden. Erklären Sie, welche Daten übertragen werden müssen und wie die Kommunikation zwischen den Komponenten erfolgen soll, z. B. über REST-APIs oder Websockets.

Die API-Funktionen des erweiterten Feuerwehrleitstellenkonfigurators lassen sich in zwei Hauptkategorien aufteilen: Zugriff auf das Dateisystem des Servers und Zugriff auf die Datenbank. Jeder dieser Teile spielt eine wesentliche Rolle bei der Speicherung und Verwaltung von Ressourcen und Daten für die reibungslose Funktionalität der Anwendung.

### 3.5.1 Zugriff auf das Dateisystem

Der Zugriff auf das Dateisystem ist von entscheidender Bedeutung, um Texturen, FBX-Modelle und Bilder effizient zu speichern und zu verwalten. In diesem Zusammenhang werden folgende API-Funktionen zur Verfügung gestellt:

1. **loadTexture**: Ladefunktion, um eine Textur vom Server zu laden.
2. **deleteTexture**: Löscht eine Textur vom Server.
3. **uploadTexture**: Lädt eine Texture auf den Server hoch.
4. **loadFBXModel**: Ladefunktion, um ein FBX-Modell vom Server zu laden.
5. **deleteFBXModel**: Löscht eine FBX-Datei vom Server.
6. **uploadFBXModel**: Lädt eine FBX-Datei auf den Server hoch.

### 3.5.2 Zugriff auf die Datenbank

Für den Datenbankzugriff wird eine allgemeine Funktion bereitgestellt, die durch Parameter gesteuert werden kann. Diese Funktion ermöglicht das Laden, Erstellen, Aktualisieren und Löschen von Daten aus der Datenbank.

**DBeexecuteSQL(tableName, action, where, data, include, sessionID, idUser)**

1. **tableName**: Name der Tabelle, von der Daten geladen, erstellt, geändert oder gelöscht werden sollen.
2. **action**: Steuert die Aktion, die auf den Daten ausgeführt wird. Zulässige Werte sind `Select` für Laden von Daten, `Delete` für Löschen von Daten, `Create` für Erstellen von Daten und `Update` für Aktualisieren von Daten.
3. **where**: Bedingung, um die Daten bei `Select`, `Update` oder `Delete` zu filtern oder zu manipulieren.
4. **data**: Daten, die bei `Create` übergeben werden sollen.
5. **include**: Gibt an, ob Daten aus Beziehungstabellen mitgeladen werden sollen.
6. **sessionID**: Ist notwendig für die Authentifizierung auf dem Server.
7. **idUser**: Ist notwendig für die Authentifizierung auf dem Server.

Die allgemeine Datenbank-API-Funktion bietet eine flexible und leistungsstarke Möglichkeit, um auf die Datenbank zuzugreifen und die erforderlichen Datenoperationen durchzuführen. Sie bildet das Rückgrat der Datenbankanbindung und ermöglicht die nahtlose Integration und Verwaltung von Daten für den erweiterten Feuerwehrleitstellenkonfigurator.

Zusätzlich zu den beschriebenen API-Funktionen, die für die Datenverwaltung verantwortlich sind, spielen vier weitere API-Funktionen eine wesentliche Rolle in Bezug auf die Authentifizierung und Sicherheit. Diese Funktionen sind speziell darauf ausgerichtet, die Identität und Zugriffsrechte der Benutzer zu prüfen.

1. **checkSessionID**: Prüft die Session ID. Die Funktion wird bei jedem Request aufgerufen.
2. **deleteOldSessions**: Löscht alle inaktiven Sessions.
3. **getSessionByIdUser**: Lädt die angelegte Session nach dem Login anhand der User ID.
4. **CheckUserRights**: Prüft die Zugriffsberechtigung. Die Funktion wird bei jedem Request aufgerufen.
5. **checkLogin**: Prüft die Login-Daten.

## 3.6 Sicherheit und Sessions

Nachdem ein Benutzer sich erfolgreich angemeldet hat, wird eine Session in der Datenbank angelegt. Eine Session hat eine ID, die **sessionID**, einen Verweis auf den angemeldeten Benutzer und ein Datum, das angibt wann die Session angelegt wurde.

Bei jedem Request an den Server wird der Body des Requests die SessionID und die ID des Benutzers enthalten. Auf der Serverseite wird zuerst überprüft, ob die mitgelieferte SessionID gültig ist und im System registriert wurde. Falls dies nicht der Fall ist, wird der HTTP-Statuscode "403 Forbidden" zurückgesendet, um anzudeuten, dass der Zugriff verweigert wird. Zusätzlich wird ein Objekt mit dem Inhalt **error: SZugriff verweigert: Ungültige SessionID.**" an den Client gesendet.

Nach einem erfolgreichen SessionID-Check wird der Benutzer anhand der mitgelieferten userID geladen. In den Benutzerdaten sind die zugehörigen Rechte hinterlegt, die angeben, ob der Benutzer berechtigt ist, den angeforderten Befehl auszuführen. Falls der Check nicht erfolgreich ist und der Benutzer nicht über die erforderlichen Rechte verfügt, wird erneut der HTTP-Statuscode "403 Forbidden" an den Client gesendet. Auch hier wird das Objekt **error: SZugriff verweigert: Ungültige SessionID.**" an den Client gesendet.

Wenn beide Tests erfolgreich waren und der Benutzer über die notwendigen Rechte verfügt, wird der eigentliche Request ausgeführt und die angeforderte Funktion oder Aktion durchgeführt. Diese doppelte Überprüfung stellt sicher, dass nur berechtigte Benutzer mit gültigen SessionIDs Zugriff auf die geschützten Funktionen und Daten haben und unerlaubte Zugriffsversuche verhindert werden.

Die Sessions werden mit einer API-Routine `SessionKeepAlive(idSession: string)` bei bestimmten Funktionen geupdatet. Das bedeutet das das Datum immer auf das aktuelle Datum gesetzt wird. Mit Datum meine das Datum plus Uhrzeit. Alle Sessions die älter als eine bestimmte Zeit XY sind, werden gelöscht. SO werden inaktive Nutzer aus ihrer Sitzung geworfen.

## 4

---

### Realisierung

Im Verlauf dieser Bachelorarbeit wurde der Feuerwehrleitstellenkonfigurator kontinuierlich erweitert, um den gestellten Anforderungen gerecht zu werden. Das Kapitel "Realisierung" bietet einen Einblick in den Entwicklungsprozess dieses Projekts und stellt dar, wie schrittweise Funktionalitäten hinzugefügt wurden, um das Endprodukt zu erreichen.

Die Realisierung des Feuerwehrleitstellenkonfigurators erfolgte in aufeinanderfolgenden Versionen, von denen jede neue Version zusätzliche Features und Verbesserungen einführte.

Dieses Kapitel beleuchtet die wichtigsten Entwicklungsmeilensteine und den iterativen Ansatz, der es ermöglichte, den Konfigurator schrittweise zu erweitern. Von der Integration des Backends über die Implementierung der Benutzerverwaltung und Rechteverwaltung bis hin zur Echtzeit-Kollaboration und anderen sinnvollen Funktionen wird jede Version beschrieben.

Durch die iterative Vorgehensweise bei der Entwicklung konnten Erkenntnisse aus früheren Versionen effizient genutzt werden, um Feuerwehrleitstellenkonfigurator kontinuierlich zu verbessern.

Im Folgenden werden die einzelnen Versionen des Feuerwehrleitstellenkonfigurators präsentiert, wobei auf die jeweiligen Funktionalitäten und Verbesserungen eingegangen wird, die in jeder Version vorgenommen wurden. Die schrittweise Entwicklung verdeutlicht die Evolution des Konfigurators und unterstreicht Erfolge, die während des Entwicklungsprozesses bewältigt wurden.

#### 4.1 Version 1

##### 4.1.1 Integration des Frontendkonfigurator in ein Projekt mit Backend

Zuerst wurde der vorhandene Frontendkonfigurator um das Backend erweitert. Dazu wurde ein neues Nextjs Projekt erstellt um den Frontendkonfigurator dort zu integrieren. Ein Nextjs Projekt hat folgenden Aufbau:

In einem Next.js-Projekt befindet sich der "pages" Ordner auf oberster Ebene im Projektverzeichniss. Dieser Ordner hat eine besondere Bedeutung in Next.js, da er dazu dient, Routen und Seiten für die Anwendung zu definieren. Der Inhalt des



Abbildung 4.1: Datenstruktur

”pagesOrdners wird automatisch in eine Serverseite gerendert und zur Verfügung gestellt.

Im ”pagesOrdner werden die Dateien mit den Erweiterungen ”.js”, ”.jsx”, ”.tsöder ”.tsx abgelegt. In meinem Fall werden nur .tsx Dateien verwendet. Tsx ermöglicht es HTML ähnlichen Code in TypeScript einzubetten. Jede Datei in diesem Ordner wird zu einer eigenen Route, die von Next.js behandelt wird.

In einem Next.js-Framework gibt es einen speziellen Ordner namens äpiinnerhalb des ”pagesOrdners. Dieser äpiOrdner dient dazu, serverseitige API-Endpunkte für deine Anwendung zu definieren. Eine Datei im äpiOrdner, wird automatisch zu einem serverseitigen API-Endpunkt, der von der Next.js-Anwendung bereitgestellt wird.

Next.js basiert auf wiederverwendbaren Komponenten, und der Frontend-Konfigurator wurde mithilfe von React erstellt. Dementsprechend kann das gesamte Projekt als eine React-Komponente betrachtet werden. Durch die Zusammenführung aller Komponenten kann die Hauptkomponente, die den gesamten Frontend-Konfigurator darstellt, nahtlos in das Next.js-Projekt integriert werden.

In der Datei ”index.tsx”, die als Einstiegspunkt fungiert, kann die Hauptkomponente in das Projekt eingefügt werden, und darum herum können weitere Elemente und Funktionalitäten hinzugefügt werden. Diese Integration ermöglicht es, den alten Konfigurator in eine neue Umgebung mit Backend-Funktionalitäten zu überführen. Nun befindet sich das Frontendprojekt in einem neuen Projekt mit Backend.

#### 4.1.2 Scene auf Server speichern

Eine der ersten funktionalen Änderungen bestand darin, dass der SceneData JSON-String nicht mehr direkt im Frontend behandelt wurde, wie es im Stand der Technik beschrieben wurde. Stattdessen werden die JSON-Daten nun auf dem Server gespeichert. Um größere Änderungen am bestehenden Projekt zu vermeiden, wurde erstmal eine einfache Lösung implementiert, bei der der JSON-String in einer Datei auf dem Server abgelegt wurde.

Nun wurde die Speicherung der Scene in den Zuständigkeitsbereich des Servers verlagert, wodurch die Komplexität des Frontends reduziert wurde. Indem der JSON-String nun auf dem Server gespeichert wird, kann das Frontend die Daten effizienter abrufen und bearbeiten, indem es API-Anfragen an den Server sendet, anstatt den JSON-String direkt herunterzuladen und später die Datei wieder ins Programm laden zu müssen.

Durch diese Änderung blieb die bestehende Struktur des Projekts weitgehend intakt, während gleichzeitig die Funktionalität verbessert wurde, indem die Ver-

antwortlichkeiten zwischen Frontend und Backend klarer abgegrenzt wurden. Dies erleichterte die Erweiterung und Skalierbarkeit der Anwendung für zukünftige Entwicklungen.

Um die erstellten Szenen in der Anwendung anzuzeigen, wurde eine Szenenliste (SceneList), siehe Bild ”Liste der Szenen in Version 0”, hinzugefügt. Diese Liste zeigt alle verfügbaren Szenen an und ermöglicht dem Benutzer, zwischen ihnen zu navigieren. Darüber hinaus wurde eine neue Komponente erstellt, mit der der Benutzer eine neue Szene erstellen kann.

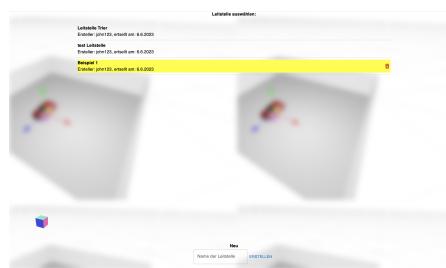


Abbildung 4.2: Liste der Szenen in Version 0

#### 4.1.3 Raum mit eigenen Objekten erstellen

Nach der Implementierung der Szenenliste wurde die Möglichkeit hinzugefügt, den Raum flexibler zu erstellen und anzupassen. Im vorherigen Konfigurator konnte man nur die Höhe, Breite und Tiefe des viereckigen Raums angeben. Mit den neuen Funktionen kann der Benutzer nun Wände, Böden und Boxen in die Szene einfügen, um seinen gewünschten Raum individuell zusammenzubauen.

Diese Elemente (Wände, Böden und Boxen) sind ebenfalls vom Typ ”Type-ObjectProps”, jedoch mit einem leeren ”PathWert (null)”, da sie keine externe Datei benötigen. Alle Objekte in der Szene werden in einem Array namens ”models” gespeichert, das (noch) aus dem JSON-Datenstring generiert wird.

In der Scene-Komponente wird das ”modelsArray” durchlaufen, und für jedes Modell wird überprüft, ob es ein Wand-, Boden- oder Box-Element ist. Abhängig davon wird das entsprechende 3D-Modell hinzugefügt und in der Szene angezeigt. Der Cube hat die Besonderheit, dass er als einziges der Objekte in alle Richtungen skaliert werden kann, um flexiblere Raumgestaltungen zu ermöglichen. Wenn der Benutzer in den Skalierungsmodus wechselt, wird angezeigt, dass bestimmte Objekte wie Wände oder Böden nicht in alle Richtungen skaliert werden können, da dies ihre Funktionalität beeinträchtigen würde. Beispielsweise bleibt eine Wand eine Wand und ein Boden ein Boden, und ihre Dicke wird nicht stark beeinflusst, um realistische Proportionen beizubehalten.

#### 4.1.4 TreeView

Zuletzt wurde in dieser Version das TreeView hinzugefügt, das eine Liste aller Objekte in der Szene enthält. Mit dieser neuen Funktion kann der Benutzer nun

Objekte im TreeView auswählen, und das ausgewählte Objekt wird sowohl in der Szene als auch im TreeView hervorgehoben.

Das TreeView dient als praktische Übersicht aller vorhandenen Objekte in der Szene. Jedes Objekt wird in der Liste mit seinem Namen oder einer Bezeichnung aufgeführt. Wenn der Benutzer ein Objekt im TreeView auswählt, wird dieses Objekt in der 3D-Szene hervorgehoben, sodass der Benutzer genau sehen kann, welches Objekt ausgewählt ist.

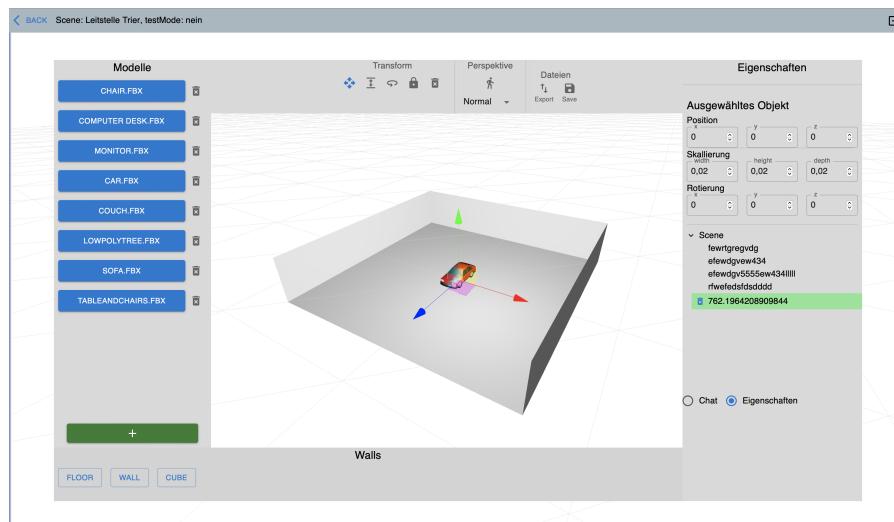


Abbildung 4.3: Version 0 mit TreeView und WallList

## 4.2 Version 2

### 4.2.1 Database, Login, Benutzer und Chat

In Version 2 wurde die SQLite-Datenbank in das Projekt integriert und mithilfe von Prisma erstellt. Zunächst wurde Prisma im Projekt installiert, um die Datenbankanbindung zu ermöglichen. Dafür wurde die `schema.prisma`-Datei erstellt, die die Datenbankkonfiguration und das Datenbankschema enthält.

In dieser `schema.prisma`-Datei wurden die ersten Tabellen definiert, nämlich 'User' und 'ChatEntries'. Die 'User'-Tabelle enthält Benutzerdaten, wie beispielsweise ihre Berechtigungen, wobei einige Benutzer 'Readonly'-Rechte haben und andere als Admin markiert sind.

Mit dem Befehl `npx prisma migrate dev` wurde das Datenbankschema in der SQLite-Datenbank erstellt bzw. aktualisiert. Dieser Befehl ermöglicht die Ausführung von Datenbankmigrationen, um das Schema gemäß den Änderungen in der `schema.prisma`-Datei zu aktualisieren.

## Login

Nun muss man sich mit einem Benutzer anmelden. Dazu gibt es die loginID und ein Passwort.

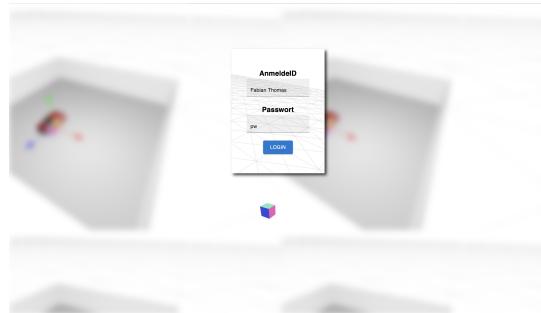


Abbildung 4.4: Login in Version 2

## Chat

Nach der Integration der SQLite-Datenbank in Version 2 habe ich als nächstes einen simplen Chat mit Socket.IO erstellt. Dieser Chat ermöglicht es allen Benutzern, Nachrichten zu sehen und Nachrichten, die von einem Benutzer gesendet werden, werden an alle anderen Benutzer weitergeleitet.

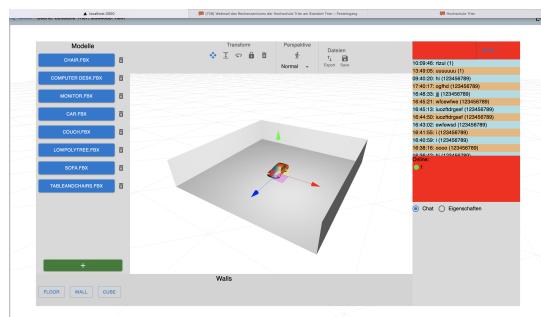


Abbildung 4.5: Chat

Die Socket.IO-Implementierung ermöglicht eine Echtzeitkommunikation zwischen den verbundenen Clients und dem Server. Sobald ein Benutzer eine Nachricht im Chat eingibt und abschickt, wird diese Nachricht an den Server gesendet. Der Server leitet die Nachricht dann an alle anderen verbundenen Clients weiter, sodass alle Benutzer die Nachricht sofort sehen können, ohne die Seite aktualisieren zu müssen.

### 4.3 Version 3

#### 4.3.1 Objekt ein-/ausblenden

In Version 3 wurde eine neue Funktion hinzugefügt, mit der der Benutzer auswählen kann, welche Wände/Objekte bei einer Orthogonalen Ansicht ausgeblendet werden sollen. Diese Funktion ist besonders nützlich, wenn sich Objekte hinter bestimmten Wänden/Objekten befinden und dadurch ihre Sichtbarkeit beeinträchtigt wird. Vorher wurde nur die rechte wand bei leftMid ausgeblendet, usw. ....

### 4.3.2 Farbe und Texturen

In der neuesten Version 3 wurden noch zwei Funktionen hinzugefügt, die es dem Benutzer ermöglichen, die visuelle Gestaltung der Szene noch weiter anzupassen: die Möglichkeit, Farben und Texturen der Objekte in der Szene zu ändern.

Dafür wurden die TypeObjectProps entsprechend angepasst und um zwei neue Felder erweitert: "color" und "texture". Das Feld "color" ist immer ausgefüllt, da jedes Objekt in der Szene eine Farbe besitzt. Dadurch kann der Benutzer die Farbe eines Objekts individuell anpassen, um eine gewünschte Ästhetik zu erzielen.

Das Feld "texture" ermöglicht es dem Benutzer, Texturen auf die Objekte anzuwenden oder zu entfernen. Wenn das "textureFeld" ausgefüllt ist, wird dem Objekt die entsprechende Textur zugewiesen.

Die Texturdateien befinden sich auf dem Server in einem speziellen Ordner namens "textures". In diesem Ordner sind bereits eine Reihe von Texturen vorhanden. Es besteht noch nicht möglich ist, eigene Texturen, mit dem Programm hinzufügen.

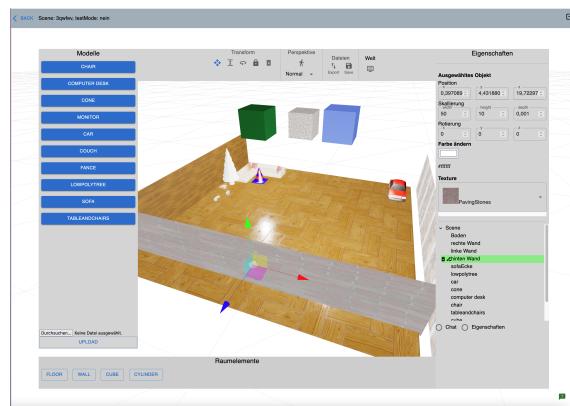


Abbildung 4.6: Texturen und Farben

### 4.3.3 Adminarea

Adminarea kam hinzu. Dort kann der Admin die User bearbeiten und FBX-Modelle hinzufügen oder löschen.



Abbildung 4.7: AdminArea Datatable user



Abbildung 4.8: AdminArea FBXList

## 4.4 Version 4

### 4.4.1 Datenstrukturen überarbeitet

In Version 3 wurde das Design und datenstrukturen geändert. zuerst wurden die scene als json string in einer textdatei auf dem server gespüeichert. das habe ich geändert und jedes Model einzeln in der DB gespeichert. dazu habe ich die entitäten der json file analysiert und ich eine tabelle model ertsellt. und dort das feld idScene zu jedem model hinzugefügt, damit man weis welche models in der scene sind. Das ermöglicht mir eine einfache handhabung der models in einer scxene. nun ist es möglich models zu selektieren, das war durch die gegeben datenstruktur als einen langen string nicht gegeben. Zu dem liegt jtz alles an einer stelle. Vorher lagen die daten im filesystem des servers und die meta daten der scene in der DB. vielleicht noch ModelToTypeObjectProps erklären.

### 4.4.2 Texturen selber hinzufügen

Neue Texturen dynamisch selber hochladen mit uploadTexture. Vorher waren die Texturen fest einprogrammiert.

### 4.4.3 Memberships

Memberships

## 4.5 Version 5

### 4.5.1 Rechte überarbeitet

Rechte überarbeitet. Alt: readonly, isAdmin; neu: read, write, delete und update und isAdmin checkSession ID, checkUser rights und alle api DB funktion zu einer DB-executeSQL zusammengefasst. neues design Admin berreich erweiter. Admin kann alle scenen sehen plus memberships

Wenn ein User ein request an den Server macht, wird immer eine SessionID und die id des Benutzers im body mitgeliefert. bevor irgendwelche dasten auf dem server abgeruft werden können, wird esrtaml nach einer gültigen sessionID geprüft. Ist keine sessionID oder eine falsche angegeben wird gesendet. Ist die sessionID gültig wird danach geprüft ob der nutzer, welcher anhand der idUser geladen wird, rechte hat um den request auszuführen. Erst wenn beide tests erfolgreich waren können Daten abgerufen werden.

### 4.5.2 Adminbereich erweitert

Der Admin berreich wurde erweitert. Nun kann der Admin alle scenen einsehen plus member und diese auch löschen und ändern.

## 4.6 Version 6

### 4.6.1 CurrentWorkingScene

Jedes mal wenn jemand eine Konfiguration betritt, wird ein Datensatz in der Datenbank eingetragen. Es wird gepsichert, wer an welcher Scene arbeiten. So kann man sehen, wie viele keute gerade an einer Scene am arbeiten sind. Das ieht man im folgenden BIld in Aktuell working.



Abbildung 4.9: Beispiel Model Version

Das spielt auch eine wichtige rolle für die UserCam.

### 4.6.2 Speichern und Synchronisieren einer Scene

add fbxModel, muss mit selben id auf den anderen client mit setModles hinzugefügt werden, sonst wird es nicht synchronisiert.

add wall wird auch synchronisiert

Jede Scene enthält mehrere Models. Jedes Model hat einen Verweis auf ihre Scene, so kann man alle Models laden die in einer scene sind. Der Scene Datensatz hat ein Feld newestVersion, zu Deutsch neueste Version. Das gibt an in welcher Version sich die Scene befindet. Nach jedem Speichern wird die neueste Version um eins hochgezählt und im SceneDatensatz aktualisiert. Danach werden alle Objekte durchlaufen und werden neu, mit der neuen Version, in die Datenbank eingetragen.

id	name	color	texture	version
1	Boden	#eeeeee	NULL	1
	rechte Wand	#eeeeee	NULL	1
	hinten Wand	#eeeeee	NULL	1
	linke Wand	#eeeeee	NULL	1
	linke Wand	#eeeeee		2
	Boden	#53d5...	NULL	2
	rechte Wand	#eeeeee		2
	hinten Wand	#eeeeee		2
	linke Wand	#eeeeee		3
	Boden	#53d5...		3
	hinten Wand	#eeeeee		3
	rechte Wand	#eeeeee		3
	linke Wand	#eeeeee		4
	Boden	#53d5...		4
	hinten Wand	#eeeeee		4
	rechte Wand	#eeeeee		4

Abbildung 4.10: Beispiel Model Version

Diese Konzept erlaubt ein einfaches umschalten zwischen den Versionen. Will ich an einer alten Version weiterarbeiten, kann ich einfach alle Modelle mit der Version XY laden und hab da weiter machen.

Im Programm haben Modelle den Typ TypeObjectProps. Da ist zum Beispiel die Position als eine 3 stelliges Array angegeben, wie in etwa so [x, y, z]. In der Datenbank aber ist jeder Wert einzeln gespeichert, also positionX, positionY und positionZ. Deswegen muss man vor dem Speichern die Modelle in der Scene die den Typ TypeObjectProps haben zu einem Object umwandeln welches in der DB gespeichert wird. Dafür gibt es die Funktion convertTypeObjectPropsToModel. Also werden in einer Schleife alle Objekte in der Scene durchlaufen. Dann wird die neue Version gestzt und zu einem object umgewandelt welche man in der Datenbank speichern kann.

Wenn mehrere Leute an der selben Scene arbeiten. Wenn man eine Scene betritt, kann man sich die Scene mithilfe des Orbitcontrols aus drei dreiecken angucken. Verbindet sich nun ein zweiter mit der selben Scene wo man gerade am arbeiten ist, werden die Positionen der Kameras synchronisiert mit socket io. Das sieht dann so aus.

Auf den 2 Bildern sind 2 Nutzer angemeldet, die gerade die selbe Scene bearbeiten. Da wo der rote Würfel ist befindet sich der andere. Das ist möglich weil die



Abbildung 4.11: Sicht von admin

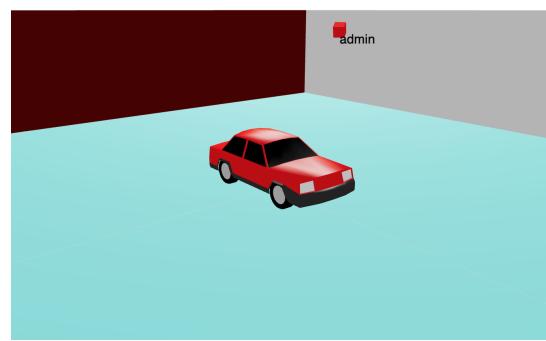


Abbildung 4.12: Sicht von nils

Position und Rotation der Kamera mit socket io emiten werden. Es wird ein Object mit einer id (refCurrentWorkingScene) verteilt. Es werden alle Worker einer Scene geladen und pro Worker wird ein Würfel in die Scene eingefügt. Der würfel hat die DAten des Workers gespeichert. Und immer wenn Daten eines Workers emitten werden und es der id des Würfels entspricht wird die position angepasst, also synchronisiert. Damit man sieht was der andere gerade bewegt, werden die Daten des bewegten objectes auch emittet. Werden Daten epmfangen und diese sind Daten aus der aktuellen Scene so wird dieses object bei allen Clients synchronisiert, aber nicht gespeichert. Beispielsweise bewegt jemand die Wand nach oben, wird sie bei allen die gerade die Scene offen haben die Wand nach oben verschoben.

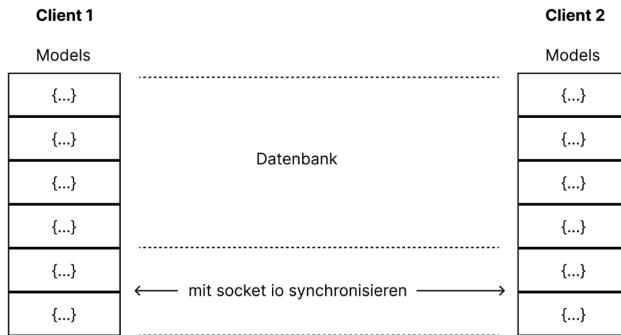


Abbildung 4.13: Models synchronisieren

### 4.6.3 Licht

Es gibt verschiedene Arten von Beleuchtungen. Einmal Ambiente Beleuchtung. Das ist eine Grundbeleuchtung der Scene. Jede Fläche wird angestrahlt, also quasi eine Grundbeleuchtung. Dann gibt es noch Directional Light, welches in eine Richtung kegelgebündigt Lichtstrahlen aussendet. In der Scene fest eingeprägt und kann nach belieben verschoben werden.

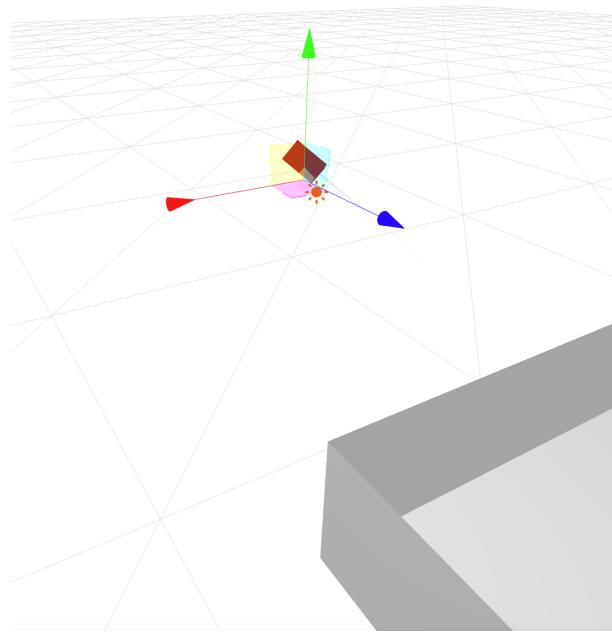


Abbildung 4.14: Beispiel Model Version

Diese Lichtquelle guckt immer richtung mitte. Dann kann man noch PointLight hinzufügen und auch verschieben. Jede Lichtquelle außer das ambient light erkennt man in der Scene an dem LichtIcon.

## 5

---

# Ergebnisdarstellung

Die Sitemap dient als Überblick über die verschiedenen Seiten und Komponenten des Konfiguratorprogramm. Die Sitemap dient als visuelle Darstellung der Struktur und Navigation des Projekts. Es werden die wichtigsten Hauptkomponenten des Konfigurators gezeigt.

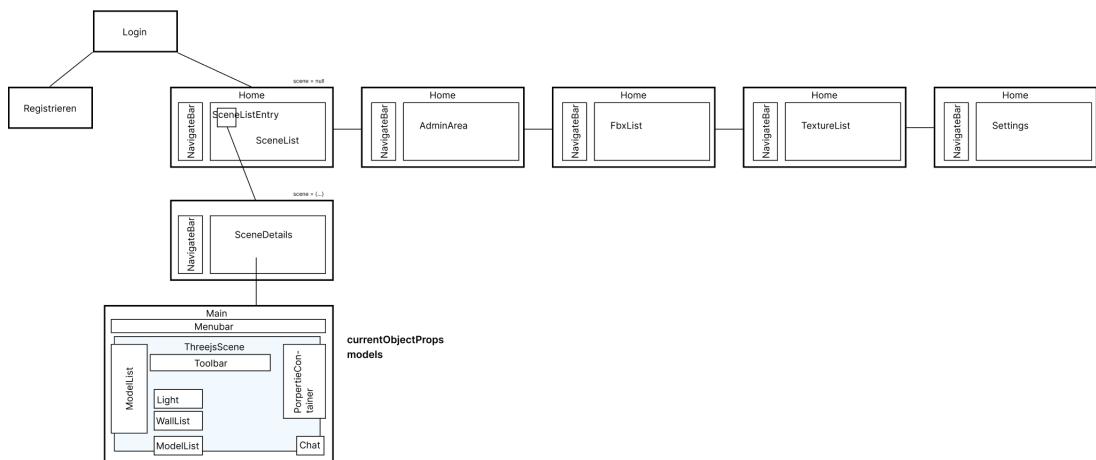


Abbildung 5.1: Sitemap

In den folgenden Abschnitten werden die Funktionsweise der einzelnen Komponenten beschrieben.

## 5.1 Login und Register

Erstmal benötigt man einen account um den konfigurator benutzen zu können. Um einen account zu ertsellen kann man sich mit einer loginID und einem passwort registrieren. Danach kann man sich mit diesen Daten im Login anmelden. Nach einer erfolgreichen anmeldung wird eine session für den angemeldeten benutzer ertsellt. Ohne diese session kann man keine daten vom server abrufen. Sie wird in jeder serveranfrgae im body mitgeliefert. Zusätzlich hat jeder Benutzer read, write und delete rechtre. Diese werden auch bei jeder serveranfrage abgefragt. dazu wird

die id des benutzers acuh im body mmitgeliefert. Ein neu registrieter Benutzer bekommt read, write und update rechte. Wie in **Kapitel 3.6 Sicherheit und Sessions** beschrieben.

## 5.2 Home und Navigatebar

Man kann im Homebildschirm mithilfe der Navigatebar zwischen den Seiten SceneList, AdminArea, FbxLis, TextureList und Settings. Zuätzlich ist dort auch der Logout button. Nach dem Login landet man automatisch bei der SceneList. So sieht das aus

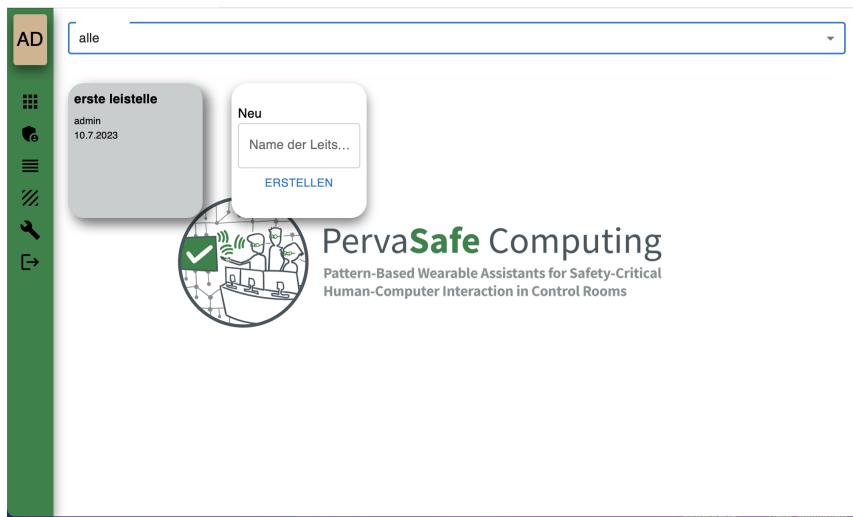


Abbildung 5.2: SceneList

An der linken Seite befindet sich die Navigatebar und rechts daneben wird die aktive Seite angezeigt

## 5.3 Scenelist und SceneMemberships

hier werden alle scenen angezeigt die ich erstellt habe oder zu ich hinzugefügt wurde. Das ist dann eine scenemembership. In der kachel werden die wichtigsten Informationen der leitsellen konfiguration. Ganz oben steht der name der Konfiguration. das wer sie ertsellt hat und dann noch wann sie erstellt wurde. Dann in welchewr version die sich befindet und wie viel Objekte da drin sind. Am ende der lsite ist eine Kachel um eine nbeue Konfiguration zu erstellen. Dazu muss man ur einen namen angeben und auf ertsellen klicken. Wenn man auf eine kachel klickt, gelangt man zur Konfigurationsübersicht. DOrt werden Dteilas der Scene nochmal zusammengefasst und dargestellt. Hier kann man leute zu seiner scene hinzufügen. Wird dies gemacht können diese bebutzer deine scene bearbeiten. Falls einer benutzer die scne nur sehen soll und nicht bearbeiten klann man ihn auf readonly

stellen. Benutzer können auch wieder entfernen werden. Hier ist auch ein Button zum die konfiguration zu betreten

Ganz zum schluss ist eine kachel um eine neu scene zu erstellen. Dazu wird ein neuer scenedatensatz mit dem namen der vorher vergeben wurde in der datenbank angelegt.

### 5.3.1 Leitsellenkonfiguration Main

Die Konfiguratorseite vereint alle UI-Elemente, die für die eigentliche Konfiguration notwendig sind. Im Folgenden werden die verschiedenen UI-Elemente in separaten Kapiteln beschrieben, darunter die Three.js Scene, Toolbar, Properties-Container, TreeView, ModelList, WallList, Light und der Chat.

Alle UI-Elemente können durch einfaches Drag-and-Drop verschoben werden, um eine flexible Anordnung der Benutzeroberfläche zu ermöglichen. Die Toolbar, PropContainer und das TreeView sind immer sichtbar und bieten somit jederzeit Zugriff auf alle relevanten Funktionen.

Die restlichen UI-Elemente sind standardmäßig minimiert und können je nach individuellen Bedürfnissen ein- oder ausgeblendet werden, um eine maßgeschneiderte Arbeitsumgebung zu schaffen. Die nachfolgenden Kapitel werden detailliert die Funktionsweise und Interaktionen der einzelnen UI-Elemente erläutern.

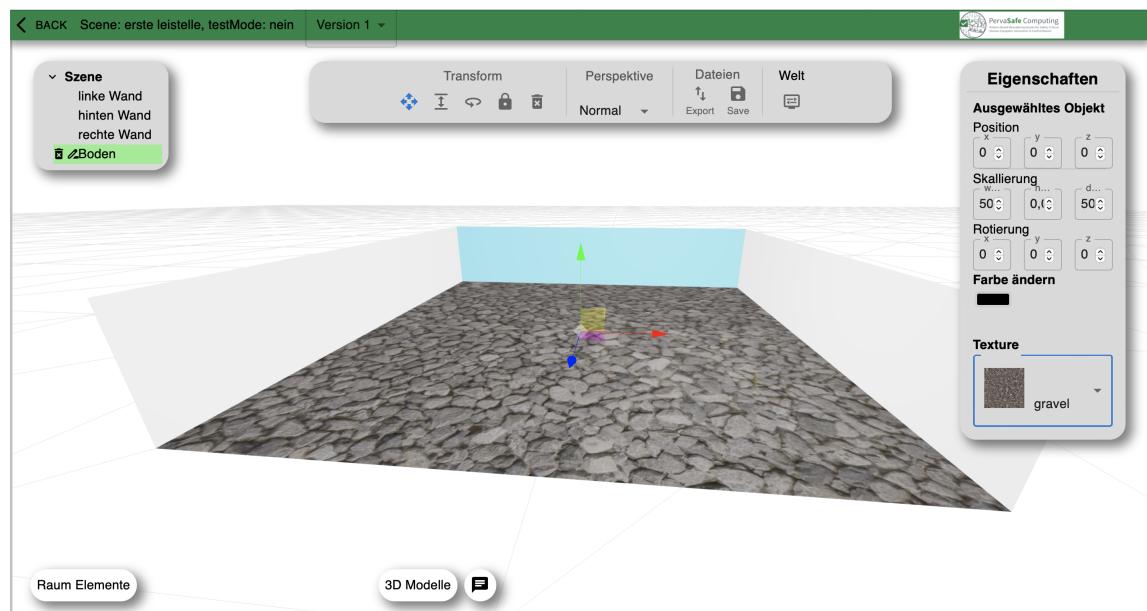


Abbildung 5.3: SceneList

## Threejs Scene

Die Three.js Scene wird in ein HTML-Element `<Canvas>` gerendert und bildet das zentrale Element der Benutzeroberfläche. Die Scene besteht aus verschiedenen

Komponenten, darunter eine Kamera, ambientLight, ein pointLight, Usercams sowie eine Vielzahl von Objekten, die dynamisch in die Szene geladen werden. Diese Objekte werden im Programm als "models" bezeichnet und können entweder 3D-Modelle oder einfache Geometrien repräsentieren.

Innerhalb der Scene können die einzelnen Objekte durch Transformationen verändert werden, wodurch eine interaktive Konfiguration ermöglicht wird. Wenn ein Benutzer auf ein bestimmtes Objekt klickt, wird dieses als "CurrentObjectProps" gespeichert und an die UI-Komponenten übergeben. Dadurch haben die UI-Elemente Zugriff auf die Eigenschaften des ausgewählten Objekts und können diese entsprechend verändern.

Für Objekte, die kein 3D-Modell repräsentieren können, besteht die Möglichkeit, Farben oder Texturen zu verwenden, wie im Kapitel "PropertieContainer" näher erläutert wird. Die Three.js Scene bildet somit das zentrale Arbeitsfeld für die Konfiguration der Feuerwehrleitstelle und ermöglicht eine intuitive und interaktive Manipulation der Objekte und Eigenschaften.

## Chat

Wie schon in der Zielsetzung erwähnt, gibt es pro Szene einen Chat, welcher in echt Zeit kommuniziert. Wenn man eine Nachricht sendet, wird zuerst ein Model ChatEntry, siehe Kapitel Datenmodellierung, ertsellt. Dies ist in Bild XY zu sehen.

Socket.emit ist eine Methode aus dem Socket.IO-Framework, mit der Daten (hier Nachrichten) an den Server und andere verbundene Clients (Benutzer) in Echtzeit gesendet werden können.

Im gegebenen Codeausschnitt wird socket.emit("emitChatEntry", chatEntry); verwendet, um den erstellten chatEntry zu "verteilen", d.h. an den Server zu senden, damit dieser die Nachricht an alle verbundenen Clients weiterleiten kann.

```
// socket IO chatEntry 'verteilen'
const onClickHandler = async () => {
  // model für chat eintrag erstellen
  const chatEntry: ChatEntry = {
    id: uuidv4(),
    idScene: props.scene ? props.scene.id : '',
    idUser: props.user.id,
    message: text,
    datum: new Date(),
  };

  // chatentry verteilen
  socket.emit("emitChatEntry", chatEntry);

  // test session keep alive
  await fetch("api/database/Session/DB_sessionKeepAlive", {
    method: "POST",
    body: JSON.stringify({
      sessionID: props.sessionID,
      idUser: props.user.id,
    }),
  });

  // textfeld leeren
  setText("");
};

// socket IO 'chatEntry' 'verteilen'
socket.on("emitChatEntry", (chatEntry) => {
  // hier wird der chatentry in die scene integriert
});
```

Abbildung 5.4: Codeausschnitt (Client) vom Chat

Nachdem der chatEntry erstellt wurde, wird er mit dem Ereignisnamen emit-ChatEntry an den Server gesendet. Der Server kann dann diesen eingehenden Chat-Eintrag verarbeiten und ihn an alle Clients übermitteln, die mit dem Socket.IO-Server verbunden sind. Dadurch können alle Benutzer in Echtzeit sehen, wenn neue Chat-Nachrichten gesendet werden.

Die socket.emit-Methode wird verwendet, um Ereignisse von einem Client an den Server zu senden, während die socket.on-Methode, siehe Bild XY, verwendet wird, um Ereignisse vom Server auf dem Client zu empfangen und darauf zu reagieren. Dadurch können Benutzer in einer Chatanwendung in Echtzeit miteinander interagieren.

```
// chat
socket.on("emitChatEntry", async (chatEntry) => {
  await prismaClient.chatEntry.create({
    data: chatEntry,
  });

  io.emit("getChatEntry", chatEntry.idScene);
});
```

Abbildung 5.5: Codeausschnitt (Server) vom Chat

In dem vorliegenden Code wird die SceneID an alle verbundenen Clients verteilt. Jeder Client erhält die SceneID, und wenn der Benutzer des Clients gerade dieselbe Szene bearbeitet wie die SceneID, werden alle Chat-Nachrichten neu nach ihrem Datum sortiert und anschließend geladen und auf dem Client dargestellt.

## TreeView

hier sind alle Objekt mit namen in der scene aufgelistet. Klickt man dort ein Eintrag an wird auch das entsprechende objekt in der scene ausgewählt. Man kann hier auch objekte löschen und ihnen auch einen anderen namen geben. Wähltr man ein Objekt in der scene aus wird das auch im treeView selektiert. Das TreeViewControl läuft durch die modles durch und erstell pro model einen eintrag. Mithilfe der übergebenen werte kann das TreeView Control gut mit der Scene zusammenarbeiten. Das SceneViewControl fängt ab einer bestimmten anzahl von einträgen auf zu wachsen und man muss scrollen. Man kann das TreeView auch zuklappen

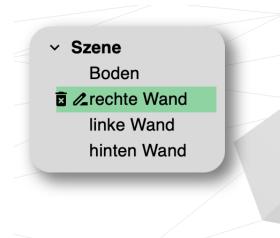


Abbildung 5.6: TreeView

Hier kann man sehen das in der Szene 4 Elemente sind.

## Toolbar

Die Toolbar war bereits im Frontend-Konfigurator vorhanden, jedoch wurden einige Änderungen und zusätzliche Funktionen implementiert. Ein Beispiel dafür ist die `showWalls`-Funktion, bei der durch Klicken auf den entsprechenden Button Checkboxen bei jedem Objekt erscheinen. Wenn diese Checkbox aktiviert wird, wird das Objekt in anderen Ansichten, wie "topDown" oder "leftMid", eingeblendet oder ausgeblendet. Dies ermöglicht es, Objekte, die andere Objekte verdecken, auszublenden und eine bessere Übersicht zu erhalten. Hierfür wurde das Feld "visibleInOtherPerspective" in "TypeObjectProps" eingeführt, das den Status speichert und ein Ja-/Nein-Feld ist.

Die Exportfunktion wurde angepasst, um die neuen Funktionen zu berücksichtigen. Anstatt die gesamte Szene als JSON-String in einer Textdatei auf dem Server zu speichern, werden nun alle Objekte einzeln in der Datenbank gesichert. Dadurch wird eine flexiblere Handhabung ermöglicht und gezielte Abfragen sind möglich.

Des Weiteren wurden die Modi "Transform", "Rotate", "SScale", "SSperren" und "Löschen" eingeführt, um eine präzisere Bearbeitung der Objekte zu ermöglichen. Zusätzlich kann die Perspektive geändert werden, um die Konfiguration aus verschiedenen Blickwinkeln zu betrachten und anzupassen. Durch diese Erweiterungen und Anpassungen der Toolbar wird die Benutzerfreundlichkeit und Flexibilität des erweiterten Feuerwehrleitstellenkonfigurators deutlich erhöht.



Abbildung 5.7: Toolbar

## PropertieContrainer

hier kann man translate, rotate und scale über numberinputs ändern, statt über das pivotcontrol. Hier kam neu hinzu das man objekten, die keine FBX-Modelle sind, Farben oder Texturen geben kann. Man kann Texturen auch selber hochladen wenn man entsprechende Berechtigungen hat.

Um Texturen Hochzuladen siehe Kapitel TextureList.

## Licht

In dem UI-Element kann man Lichteinstellungen vornehmen. Man kann das AmientLight heller oder Dunkler machen. Zusätzlich kann man noch weitere Lichtquellen (PointLights) in die Szene einfügen. Diese können auch verschoben werden und werden auch im TreeView angezeigt. Die Lichter werden aber nicht mitgespeichert. Weil beim Exportieren will man ja nur die Objekte haben und nicht noch die Lichtquellen dazu. Das PointLight muss über das TreeView control ausgewählt

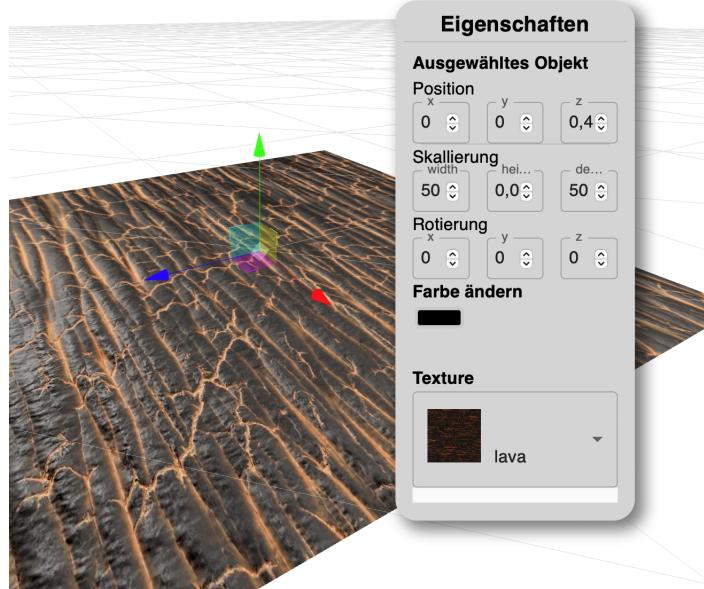


Abbildung 5.8: PropertieContainer

werden. Weil eine Lichtquelle kein Objekt ist muss man das Licht über das Tree-View control auswählen. Damit man das PointLight aber in der Szene sehen kann wird dort ein Sonne-Icon angezeigt, wie man im Bild XY sehen kann.

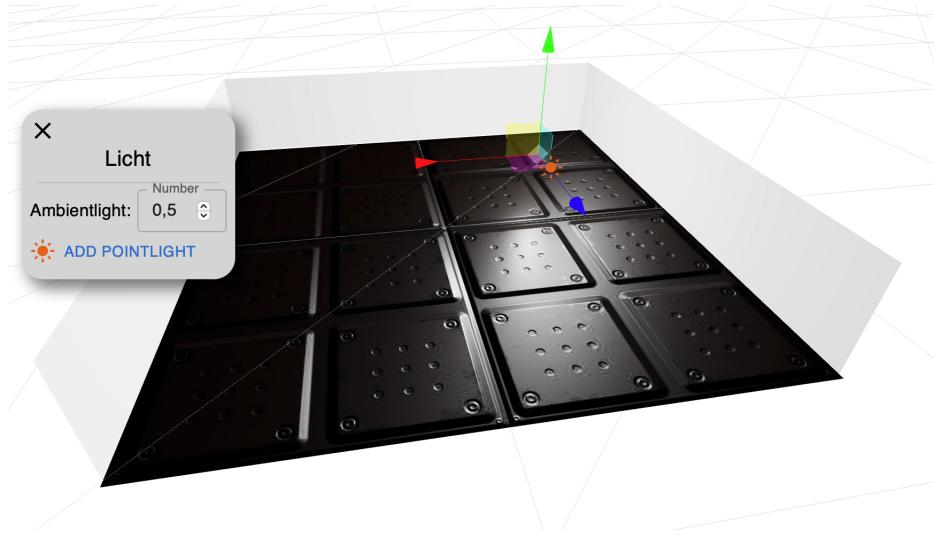


Abbildung 5.9: Licht

## WallList

hier kann man translate, rotate und scale über numberinputs ändern, statt über das pivotcontrol. Hier kam neu hinzu das man objekten eine farbe geben kann,

keine fxb model, weil die haben eine eigen texture. Zusätzlich kann man ohne auch eine texture geben.

## ModelList und Wallist

In der ModelList sind alle FBX-Modelle aufgelistet, welche in die Szene eingefügt werden können. Klickt man auf einen Eintrag wird das Objekt in die Szene im Nullpunkt eingefügt. Zusätzlich wird es noch als CurrentObjectProp festgelegt und ist damit automatisch selektiert nach dem einfügen. Im code wird ein neues Object vom Typ TypeObjectProps angelegt und in die models eingefügt.

In der Wallist können einfach Geometrien in die Scene hinzugefügt werden. Damit man sich einen Raum so gestalten kann wie man will. Auch hier wird ein neues Object vom Type TypeObjectProps angelegt, hier ist wichtig zu wissen dass das Feld Path leer ist. Path gibt ja den Pfad zum FBX-Modell an und da es keins ist bleibt es leer. DAs wird es auch in models eingefügt.

## Menubar

Ganz oben ist die menubar. dort kann man die version ändern.

Hier sind alle Objekte mit Namen in der Scene aufgelistet. Klickt man dort auf einen Eintrag wird auch das entsprechende Objekt in der Scene ausgewählt. Man kann hier auch Objekte löschen und ihnen auch einen anderen Namen geben. Wählt man ein Objekt in der Scene aus wird das auch im treeView ausgewählt.



Abbildung 5.10: Menubar

## 5.4 FbxList

Hier können FBX-Modelle hinzugefügt, gelöscht und angesehen werden. Nur berechtigte Personen können FBX-Modelle verwalten.

## 5.5 TextureList

Hier werden die Texturen verwaltet. Man kann Texturen Löschen, hinzufügen und ihnen einen Namen geben. Wenn eine Textur hochgeladen wurde, kann man sie automatisch im PropertieContainer benutzen. Um eine Textur hochzuladen, werden 5 Dateien benötigt:

1. Substance-Graph-AmbientOcclusion.jpg: xxx

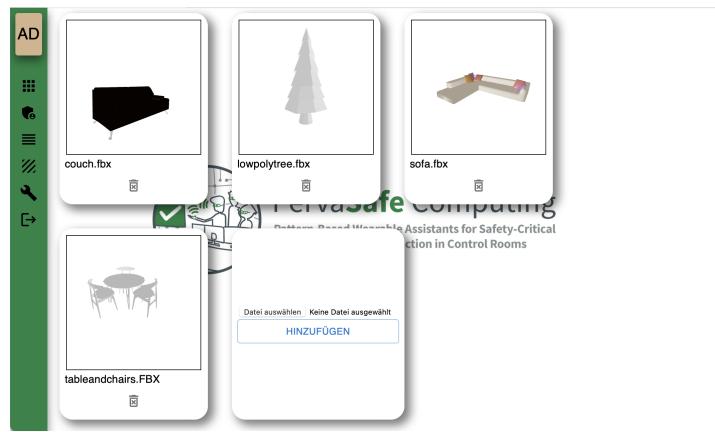


Abbildung 5.11: TreeView

2. **Substance-Graph-BaseColor.jpg:** xxx
3. **Substance-Graph-Height.jpg:** xxx
4. **Substance-Graph-Normal.jpg:** xxx
5. **Substance-Graph-Roughness.jpg:** xxx

Nach dem Dateien zum Hochladen ausgewählt wurden, werden diese auch geprüft ob es Dateien sind und ob sie auch genauso heißen. Nachdem die richtigen Dateien ausgewählt wurden sind, muss man einen Namen der Textur vergeben. Anhand dessen Namen werden die Dateien auf dem Server hochgeladen. Das sieht dann auf dem Server so aus:

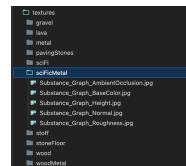


Abbildung 5.12: Texturen ablage auf dem Server

Wie schon erwähnt können nur die Walls Texturen haben. Wenn in den TypeObjectProps im Feld texture eine Wert steht, also der Name der Textur, wird diese geladen und auf das Objekt angewendet, wie man es im Codeausschnitt XY sehen kann.

```
    mat.setUniform('fTextureLoader', 1);
    fTextureLoader = new THREE.TextureLoader();
    fTextureLoader.load('textures/props/obj/props/textures/Substance_Graph_BaseColor.jpg',
    function (texture) {
        mat.setUniform('uBaseColor', texture);
    });
    fTextureLoader.load('textures/props/obj/props/textures/Substance_Graph_Height.jpg',
    function (texture) {
        mat.setUniform('uHeight', texture);
    });
    fTextureLoader.load('textures/props/obj/props/textures/Substance_Graph_Roughness.jpg',
    function (texture) {
        mat.setUniform('uRoughness', texture);
    });
    fTextureLoader.load('textures/props/obj/props/textures/Substance_Graph_DiffuseColor_01.jpg',
    function (texture) {
        mat.setUniform('uDiffuseColor', texture);
    });
});
```

Abbildung 5.13: Texturen im Code laden

Der erste Argument ist der Lader, der für das Laden der Ressource verwendet wird. In diesem Fall wird der TextureLoader aus "three.js" verwendet, um Texturen zu laden, die für die 3D-Objekte verwendet werden sollen. Der zweite Argument

ist ein Array von URLs oder Dateipfaden, die die zu ladenden Ressourcen repräsentieren. Hier werden die Pfade zu den Texturen für Farb-, Verschiebungs-, Normal-, Rauheits- und Umgebungsokklusionskarten übergeben. Nur berechtigte Personen können Texturen verwalten.

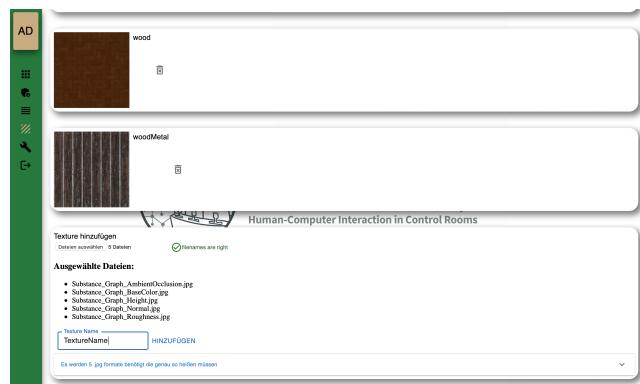


Abbildung 5.14: Texturen

## 5.6 Settings

Hier können verschiedene Einstellungen geändert werden.

# A

---

## Glossar

DisASTer	Distributed Algorithms Simulation Terrain, eine Plattform zur Implementierung verteilter Algorithmen [?]
DSM	Distributed Shared Memory
AC	Atomic Consistency (dt.: Linearisierbarkeit)
RC	Release Consistency (dt.: Freigabekonsistenz)
SC	Sequential Consistency (dt.: Sequentielle Konsistenz)
WC	Weak Consistency (dt.: Schwache Konsistenz)

## B

---

### Selbstständigkeitserklärung

- Diese Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.
- Diese Arbeit wurde als Gruppenarbeit angefertigt. Meinen Anteil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser:

Meine eigene Leistung ist:

---

Datum

---

Unterschrift der Kandidatin/des Kandidaten