

Praxistaugliche Weiterentwicklung eines Frontend-Leitstellen-Konfigurators durch Backend-Integration und Erweiterung um zusätzliche Features

Enhancing Practical Applicability: Advancing a Frontend Command Post Configurator through Backend Integration and Additional Feature Integration

Fabian Thomas

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Tilo Mentler

Auderath, DD.MM.YYYY

Kurzfassung

Die vorliegende Bachelorarbeit behandelt die Entwicklung und Erweiterung eines Leitstellenkonfigurators, 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 Leitstellen 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 Daten 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 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.

Insgesamt bietet der erweiterte Leitstellenkonfigurator eine benutzerfreundliche, sichere und leistungsfähige Plattform zur Erstellung, Verwaltung und Zusammenarbeit an Leitstellenkonfigurationen. Die praktische Anwendung dieses Konfigurators in Leitstellen kann die Effizienz und Effektivität der Einsatzplanung und -koordination verbessern.

Abstract

This bachelor's thesis presents the expansion of a control room configurator, originally comprising only a frontend without backend functionality. The objective of the work was to augment the existing configurator with a backend component, enabling practical utilization in control room settings. Additionally, meaningful features were implemented to enhance user experience and improve efficiency in the creation and management of control room configurations.

The process of enhancement commenced with the integration of the existing project into a Next.js framework to facilitate the implementation of API routines and backend capabilities. Subsequently, a database was established using Prisma to enable basic data storage and management. The user interface was refined using React MUI to ensure an appealing and user-friendly interaction.

To ensure security and access control, user management and permission systems were implemented. An admin area was established, permitting authorized individuals to manage all settings and assign permissions. Authentication was guaranteed through the implementation of sessions and rights, ensuring that only authorized personnel could communicate with the server.

Overall, the extended control room configurator offers a user-friendly, secure, and powerful platform for creating, managing, and collaborating on control room configurations. The practical application of this configurator in control room environments has the potential to enhance the efficiency and effectiveness of deployment planning and coordination.

Inhaltsverzeichnis

1 Einleitung	1
2 Stand der Technik	3
2.1 Der Frontend-Konfigurator	3
2.2 Analyse der Datenstruktur	7
3 Konzeption	10
3.1 Zielsetzung	10
3.2 Funktionale Anforderungsanalyse	11
3.3 Technologien und Architektur	12
3.4 Datenmodellierung	14
3.5 Schnittstellen	15
3.5.1 Zugriff auf das Dateisystem	16
3.5.2 Zugriff auf die Datenbank	16
3.5.3 Authentifizierung	16
3.6 Sicherheit und Sessions	17
4 Realisierung	18
4.1 Version 1	18
4.1.1 Integration des Frontendkonfigurator in ein Projekt mit Backend	18
4.1.2 Konfigurationen auf Server speichern	19
4.1.3 Raum mit eigenen Objekten erstellen	20
4.1.4 TreeView	21
4.2 Version 2	22
4.2.1 Datenbank	22
4.2.2 Benutzer und Login	22
4.2.3 Chat	23
4.3 Version 3	23
4.3.1 Adminarea	23
4.3.2 Farben und Texturen	24
4.3.3 Objekt ein-/ausblenden	25
4.4 Version 4	26

4.4.1 Memberships	26
4.4.2 Datenstrukturen der Scene überarbeitet	27
4.4.3 Texturen selber hinzufügen	28
4.5 Version 5	28
4.5.1 Rechte überarbeitet	29
4.5.2 Adminbereich erweitert	30
4.5.3 Scene Versionen	30
4.6 Version 6	31
4.6.1 CurrentSceneEdit	31
4.6.2 Synchronisieren einer Scene	32
4.6.3 Licht	34
4.7 Version 7	34
4.7.1 Mit E-Mail Registrieren	35
4.7.2 Passwort verschlüsselt speichern	36
4.7.3 Passwort vergessen und Passwort ändern	36
4.7.4 Anzeigename	37
5 Ergebnisdarstellung	38
5.1 Login und Register	38
5.2 Home und Navigatebar	39
5.3 Scenelist	39
5.4 Leitsellenkonfiguration	40
5.4.1 Threejs Scene	41
5.4.2 Chat	43
5.4.3 TreeView	43
5.4.4 Toolbar	43
5.4.5 Eigenschaften	44
5.4.6 Licht	44
5.4.7 WallList	45
5.4.8 ModelList	46
5.4.9 Menubar	47
5.5 Adminarea	47
5.6 FbxList	47
5.7 TextureList	47
5.8 Settings	49
Literaturverzeichnis	50
Selbstständigkeitserklärung	51

1

Einleitung

Eine Leitstelle stellt eine zentrale Komponente dar, die eine effiziente Koordination von Einsätzen 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, Leitstellen virtuell zu konfigurieren und an die individuellen Bedürfnisse anzupassen.

Diese Bachelorarbeit konzentriert sich auf die Entwicklung und Erweiterung eines Leitstellenkonfigurators, 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 Leitstellen 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 Leitstellenkonfigurator ermöglicht es, Konfigurationen einer Leitstelle in einer virtuellen Umgebung zu erstellen, siehe Abb. 1.1. Dadurch können Einsätze 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 [Ver23] migriert, um eine solide Backend-Unterstützung zu gewährleisten. Prisma [Pri23] wird zur Erstellung und Verwaltung der Datenbank verwendet, während React MUI [Fac23] die Oberflächenkomponenten bereitstellt. Zusätzlich wird socket io [Soc23] verwendet um bestimmte Daten 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.

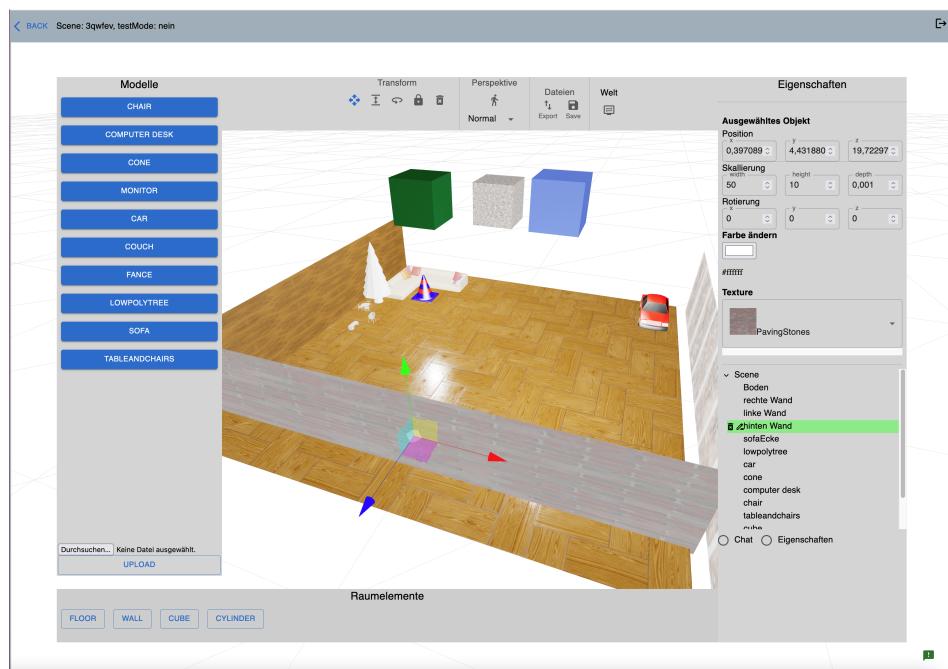


Abbildung 1.1: Beispiel einer Konfigurierten Leitstelle (hier kommt noch ein Bild von einer richtigen konfogurierten leitestelle hin)

2

Stand der Technik

Vor der Umsetzung der neuen Funktionalitäten wurde eine Analyse des vorhandenen Konfigurators durchgeführt, der lediglich über ein Frontend verfügte. Ziel war ein besseres Verständnis, um die Erweiterung durchführen zu können. Der erste Abschnitt beschreibt die bestehenden Fähigkeiten des Konfigurators, während der zweite Abschnitt die Datenstruktur einer Konfiguration näher erläutert.

2.1 Der Frontend-Konfigurator

Ein wichtiger Aspekt des vorhandenen Leitstellen-Konfigurators, welcher in Abb. 2.1 zu sehen ist, ist die Nutzung von React [Fac23], 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.

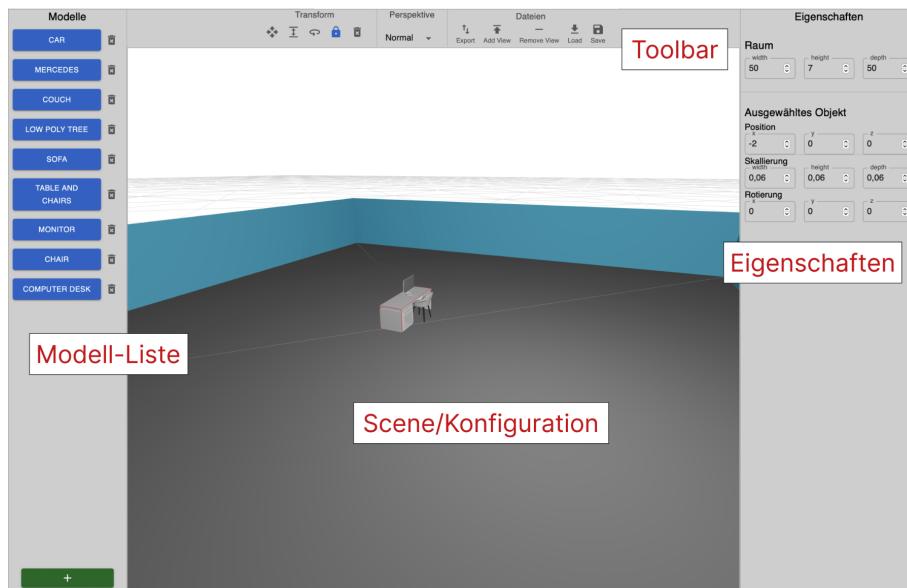


Abbildung 2.1: Der frontend-Leitstellenkonfigurator

Ein weiteres zentrales Element des Leitstellen-Konfigurators ist die Verwendung von Three.js [Thr23a] bzw. Three.js Fiber [Thr23b]. 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.

Mithilfe von React und Three.js wird eine spezielle Canvas-Komponente, von React Three Fiber verwendet, siehe Abb. 2.2 Zeile 38, um eine WebGL-Leinwand zu erzeugen. Auf dieser Leinwand können 3D-Elemente gerendert werden. Diese Canvas-Komponente dient als Container für die gesamte 3D-Szene und ihre darin enthaltenen Objekte, wie Licht, Kamera und 3D-Modelle.

```

38  <Canvas>
39    /* Kamera */
40    <Camera
41      controlsRef={props.controlsRef}
42      orthogonal={props.ortho}
43      perspektive={props.perspektive}
44    ></Camera>
45
46    /* Licht */
47    <ambientLight intensity={0.1} />
48    <pointLight position={[10, 10, 10]} />
49
50    /* Modelle */
51    {props.models.map((model) =>
52      <SceneModel
53        controlsRef={props.controlsRef}
54        key={model.id}
55        id={model.id}
56        isSelected={model.id === props.currentObjectProps?.id}
57        setCurrentObjectProps={props.setCurrentObjectProps}
58        editMode={model.editMode}
59        modelPath={model.modelPath}
60        showXTransform={model.showXTransform}
61        showYTransform={model.showYTransform}
62        showZTransform={model.showZTransform}
63        position={model.position}
64        scale={model.scale}
65        rotation={model.rotation}
66        removeBoundingBox={model.removeBoundingBox}
67        camPerspektive={props.perspektive}
68      ></SceneModel>
69    )}
70
71    /* Raum */
72    <Room
73      height={props.roomDimensions.height}
74      width={props.roomDimensions.width}
75      depth={props.roomDimensions.depth}
76      leftWall={props.wallVisibility.leftWall}
77      rightWall={props.wallVisibility.rightWall}
78    />
79  </Canvas>
```

Abbildung 2.2: Code um Objekte in die Szene hinzuzufügen

Alles was sich in einer Konfiguration befindet, befindet sich innerhalb der Canvas Komponente. Man kann auch in Zeile 40-44 sehen das in der Szene sich im-

mer eine Kamera befindet. Die Kamera in Three.js definiert, wie die Szene "gesehen" wird, ähnlich wie eine Kamera in der realen Welt. Zusätzlich befinden sich noch 2 Lichtquellen in der Konfiguration. Einmal das Ambilight in Zeile 47, um eine allgemeine Helligkeit und Beleuchtung in einer Konfiguration zu erzeugen. Danach das Pointlight in Zeile 48, es simuliert das Licht, das von einer einzelnen Lichtquelle, wie z.B. einer Glühbirne, ausgeht.

Dann werden in Zeile 51 alle 3D-Modelle, in die Konfiguration gerendert. Dafür gibt es das Array Models, welches alle Daten der 3D-Modelle, die sich in der Konfiguration befinden, enthält. Ein 3D-Modell wird durch die TypeObjectProps beschrieben, siehe dazu Kapitel 4.1.3. Mithilfe der map-Funktion wird über diese Array iteriert und für jedes Element eine bestimmte Komponente erstellt und zurückgegeben.

Der Code in Abb. 2.5, ist eine React-Komponente, die eine 3D-Modellansicht rendernt und es ermöglicht, das Modell mithilfe der "TransformControls" zu Transformieren. 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. Der Pfad gibt eine FBX-Datei an, in der Daten eines 3D-Modells enthalten sind.

```

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 2.3: Code um ein FBX-Modell zu laden (vereinfacht)

Die Komponente erstellt Referenzen für das geladene Mesh und des TransformControls - Objekt, um später auf sie zugreifen zu können. Das "TransformControls"-Element enthält die Steuerelemente, die es ermöglichen, das Modell zu verschieben, zu drehen und zu skalieren, siehe Abb. 2.4a.

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 geändert werden. Durch die Umschaltung zwischen Translation (verscheiben), Rotation und Skalieren wird ein anderes Transformcontrols, aus three drei, angezeigt. Three drei ist eine wachsende Sammlung nützlicher Hilfsfunktionen und voll funktionsfähiger, einsatzbereiter Abstraktionen für three fiber. [?]

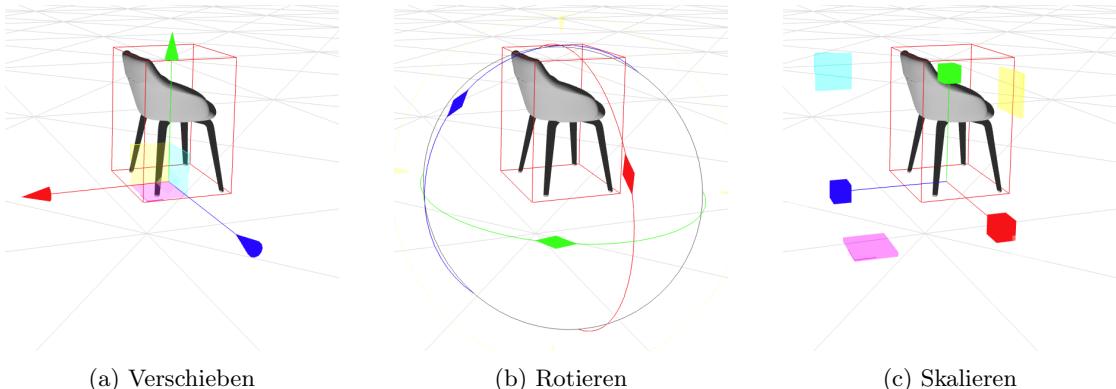


Abbildung 2.4: Das TransformControl

Darüber hinaus bietet der Konfigurator die Möglichkeit, zwischen verschiedenen Kameraperspektiven in der Toolbar 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.



Abbildung 2.5: Kameraperspektiven

Zusätzlich ermöglicht der Leitstellen-Konfigurator das Exportieren der Szenen als GLTF-Datei, um sie in anderen Anwendungen anzuzeigen oder weiterzuverarbeiten. Die Save und Load Funktionen erlauben es, die erstellte Leitstellenkonfiguration zu downloaden und diese später wieder zu laden. Beim laden muss man

die Datei wieder im Dateisystem auswählen und dann kann der Konfigurator diese laden. Des Weiteren besteht die Möglichkeit, die Raummaße anzupassen, indem die Breite, Tiefe und Höhe des virtuellen Raums verändert werden können.

2.2 Analyse der Datenstruktur

Jedes 3D Modell 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:

Basierend auf der JSON-Datei ist ersichtlich, dass der Raum eine Höhe von 7 Einheiten, eine Breite von 50 Einheiten und eine Länge von 50 Einheiten aufweist. Zusätzlich befinden sich in dieser Szene insgesamt 3 Objekte.

```
{
  "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"
    }
  ]
}
```

Abbildung 2.6: JSON Daten einer Konfiguration

Damit die unterschiedlichen Komponenten wie zum Beispiel die Toolbar, die in Abbildung 2.1 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.

3

Konzeption

Im folgenden Kapitel "Konzeption" werden die grundlegenden Leitlinien und strategischen Entscheidungen für die Entwicklung des erweiterten Leitstellen-Konfigurators erläutert. Hierbei werden die Zielsetzung der Arbeit, die ermittelten funktionalen Anforderungen, die gewählte Systemarchitektur und die angewandten Technologien vorgestellt. Ebenso erfolgt eine Darstellung der Datenmodellierung, um die Strukturierung und Organisation der Informationen im System zu verdeutlichen. Zusätzlich werden auf die Schnittstellenfunktionen zwischen Server und Client sowie Sicherheitsmechanismen eingegangen. Dieses Kapitel legt somit die konzeptuelle Basis für das Projektes dar.

3.1 Zielsetzung

Das Hauptziel dieser Bachelorarbeit besteht darin, den bestehenden Leitstellen-Konfigurator weiterzuentwickeln und um ein Backend zu erweitern. Der Fokus liegt darauf, eine praktikable Lösung zu schaffen, die in Leitstellen 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 [Ver23] 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:** Ein zentraler Adminbereich wird implementiert, um berechtigten Personen die Verwaltung aller relevanten Einstellungen und Benutzerrechte zu ermöglichen.

4. **Authentifizierung:** Authentifizierung wird verwendet, um sicherzustellen, dass nur registrierte und angemeldete Benutzer Zugang zu geschützten Bereichen oder Funktionen der Anwendung haben.
5. **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.
6. **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.
7. **Weitere sinnvolle Funktionalitäten:** Weitere nützliche Funktionen werden entwickelt, um die Benutzererfahrung zu verbessern und die Effizienz bei der Konfigurationserstellung zu steigern.

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

3.2 Funktionale Anforderungsanalyse

Die funktionale Anforderungsanalyse definiert grundlegenden Funktionen und Aufgaben, die das zu entwickelnde System erfüllen muss. In diesem Kapitel liegt der Fokus auf der Beschreibung der Anforderungen für den erweiterten Leitstellenkonfigurator. Ziel dieses Analyseschrittes ist es, einen klaren und präzisen Überblick über die essentiellen Funktionen zu gewinnen, die den Benutzern ermöglichen, ihre Leitstellen in einer effizienten Umgebung zu konfigurieren und zu verwalten.

1. **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 Leitstellen mit verschiedenen Objekten enthalten, die transformiert werden können (Skalierung, Positionierung, Rotation).
2. **Speichern und Laden von Szenen** Benutzer sollen die Möglichkeit haben, erstellte Szenen zu speichern und jederzeit wieder aufrufen können.
3. **Registrierung** Die Verwendung von E-Mail-Adressen bei der Registrierung dient dazu, Benutzer eindeutig zu identifizieren, Kommunikation für wichtige Informationen zu ermöglichen und die Identität zu überprüfen.
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. **Wiederherstellen von Szenendaten** Benutzer sollen Änderungen an der Szene 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.
10. **Passwort vergessen** Benutzern die Möglichkeit bieten, ihr Passwort zurückzusetzen, falls sie es vergessen haben, um wieder auf ihr Konto zugreifen zu können.

3.3 Technologien und Architektur

Die geplante Architektur der erweiterten Anwendung umfasst eine Kombination aus Technologien, die eine robuste und skalierbare Lösung ermöglichen. Als Backend-Framework habe ich mich für **Next.js** entschieden, da es eine leistungsstarke Plattform für die Entwicklung von React-Anwendungen bietet. Next.js ermöglicht es, 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 Datenbankstruktur aufzubauen und die Konfigurationen und andere wichtige Daten 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 werde ich in der Lage sein, eine leistungsstarke Lösung für den Leitstellen-Konfigurator zu schaffen.

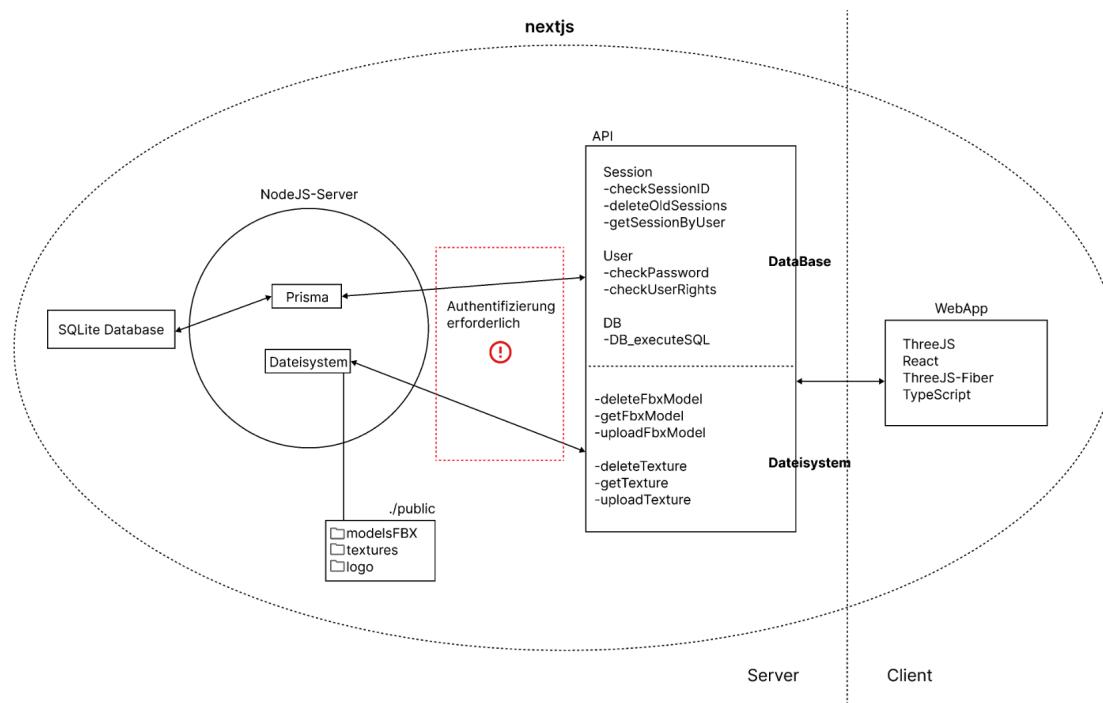


Abbildung 3.1: Architektur

Die verwendeten Technologien sind in Abbildung 3.1 dargestellt. Durch die Verwendung von Next.js ist es möglich, sowohl den Server- als auch den Client-Teil in einem einzigen Projekt zu entwickeln.

Die Clientseite, also die Web-App, wird mithilfe von .tsx-Dateien erstellt, was für TypeScript XML steht und hauptsächlich in der Entwicklung von React-basierten Webanwendungen verwendet wird. Zusätzlich kommt Three.js bzw. Three.js Fiber für die Darstellung der Konfiguration zum Einsatz.

Die Serverseite besteht aus einer Reihe von API-Funktionen, die für drei unterschiedliche Zwecke verwendet werden. Ein Teil greift auf die Datenbank zu, ein Teil auf das Dateisystem und ein Teil wird zur Authentifizierung verwendet, mehr dazu im Abschnitt Schnittstelle 3.5.

Texturen, Bilder und FBX-Dateien werden im Dateisystem abgelegt. Alle anderen Daten werden in einer SQLite-Datenbank, welche durch prisma verwaltet wird, gespeichert.

3.4 Datenmodellierung

Die Beschreibung der Tabellen liefert ein Verständnis für die Struktur und Organisation der Daten im erweiterten Leitstellenkonfigurator. Jede Tabelle erfüllt eine spezifische Rolle und ermöglicht das Speichern und Verwalten relevanter Informationen.

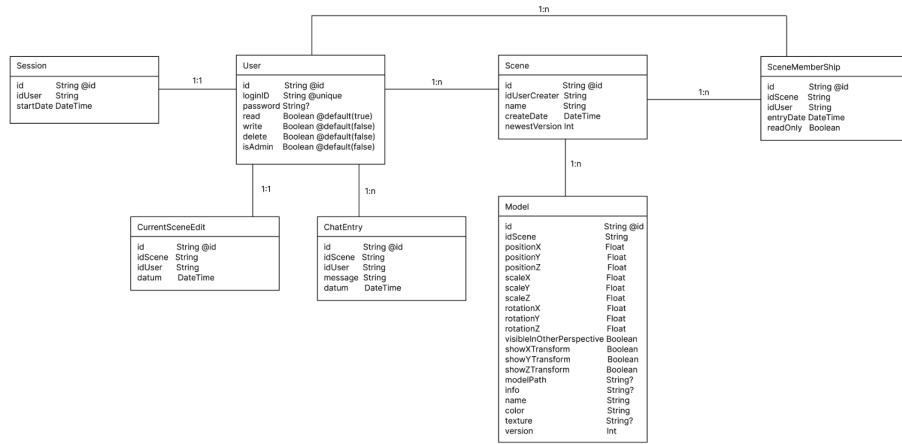


Abbildung 3.2: Datenstruktur

- User:** Die Tabelle User enthält Informationen über jeden Benutzer, einschließlich einer eindeutigen ID, eine E-Mail, Passwort und Zugriffsrechte. 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. Zudem hat verfügt jeder Benutzer über eine eindeutige Session, die ihn identifiziert. Zudem besteht eine 1:1-Verbindung zwischen einem Benutzer und dem aktuellen CurrentSceneEdit-Datensatz. Dieser Datensatz gibt an, an welcher Konfiguration der Benutzer derzeit arbeitet.
- 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.
- ChatEntry:** Jede 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.
- 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:** Jedes Objekt in der Szene wird als Modell in der Datenbank gespeichert. Ein Modell erweitert die in Kapitel 4.1.3 des Frontend-Konfigurators beschriebenen TypeObjectProps um zusätzliche Werte. Die bestehenden Felder wie id, position, scale, rotation, editMode, showXTransform, showYTransform, showZTransform und modelPath bleiben erhalten. Neu hinzugefügt wurden die folgenden Felder:
 - **idScene:** gibt an in welcher Konfiguration sich das Objekt befindet
 - **Info:** Das ist ein Feld um eine beliebige Information zum Model hinzuzufügen
 - **Name:** Jedes Model hat einen Namen.
 - **Color:** Ein Model, welches kein 3D-Modell ist, kann eine Farbe erhalten. Diese wird hier gespeichert.
 - **Texture:** Ein Model, welches kein 3D-Modell ist, kann eine Textur erhalten. Die Texturinformationen liegen im Dateisystem, also wird hier ein Pfad zu den Texturinformationen gespeichert.
 - **Version:** Jedes Model gehört zu einer Version welche hier gespeichert wird.
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:** Die Tabelle ermöglicht es, die aktuellen bearbeitenden Benutzer einer Szene zu identifizieren, sodass andere Benutzer über die laufende Bearbeitung informiert werden können. 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.

3.5 Schnittstellen

Die API-Funktionen lassen sich in drei Hauptkategorien aufteilen: Zugriff auf das Dateisystem des Servers, Zugriff auf die Datenbank und Funktionen für die Authentifizierung. 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 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.

DExecuteSQL(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 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.

3.5.3 Authentifizierung

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 das letzte mal aktiv war. Anhand des Datums kann man erkennen, wie lange der Benutzer schon inaktiv ist. Dieses Datum wird mit einer API-Routine `sessionKeepAlive(idSession: string)` bei bestimmten Funktionen geupdated. Sessions die länger als eine bestimmte Zeit nicht geupdated wurden, werden gelöscht. So werden inaktive Nutzer aus ihrer Sitzung geworfen.

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: Zugriff verweigert: Ungültige SessionID.** an den Client gesendet.

Nach einem erfolgreichen SessionID-Check wird der Benutzer anhand der mitgelieferten Benutzer ID 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 ein Objekt **error: Zugriff verweigert: Keine Rechte** 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.

4

Realisierung

Im Verlauf dieser Bachelorarbeit wurde der Leitstellenkonfigurator 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 Leitstellenkonfigurators erfolgte in aufeinanderfolgenden Versionen, von denen jede neue Version zusätzliche Features und Verbesserungen einführt. Dieses Kapitel beleuchtet die wichtigsten Entwicklungsmeilesteine 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 genutzt werden, um den Leitstellenkonfigurators kontinuierlich zu verbessern. Im Folgenden werden die einzelnen Versionen des Leitstellenkonfigurators präsentiert, wobei auf die jeweiligen Funktionalitäten und Verbesserungen eingegangen wird, die in jeder Version vorgenommen und implementiert wurden. Die schrittweise Entwicklung verdeutlicht die Evolution des Konfigurators und unterstreicht Erfolge, die während des Entwicklungsprozesses bewältigt wurden.

4.1 Version 1

In Version 1 wurde der bestehende Konfigurator in eine Umgebung mit Backend integriert. Eine weitere Neuerung war die Möglichkeit, die Szene auf dem Server zu speichern, sowie die Anpassung der Raumanordnung. Zudem wurde das TreeView-Element hinzugefügt, das eine Liste aller verfügbaren Objekte darstellt.

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



Abbildung 4.1: Next.js Struktur

er dazu dient, Routen und Seiten für die Anwendung zu definieren. Der Inhalt des "pages"-Ordners wird automatisch in eine Serverseite gerendert und zur Verfügung gestellt.

Im "pages"-Ordner werden die Dateien mit den Erweiterungen .js, .jsx, .ts oder .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 "api" innerhalb des "pages"-Ordners. Dieser "api"-Ordner dient dazu, serverseitige API-Endpunkte für die Anwendung zu definieren. Eine Datei im "api"-Ordner 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 Frontendkonfigurator 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 Konfigurationen 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 Verantwortlichkeiten 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 Abbildung 4.2, 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, welche in Abbildung 4.2 ganz unten zu sehen ist.

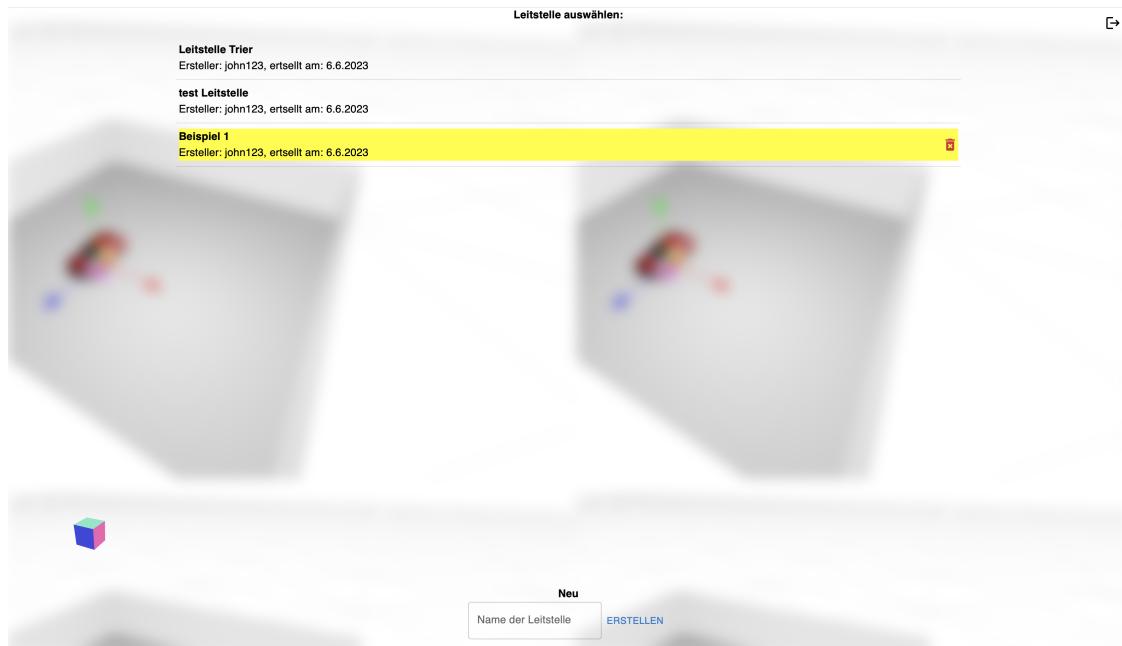


Abbildung 4.2: Liste der Szenen in Version 0

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 TypeObjectProps , jedoch mit einem leeren Path-Wert (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 models-Array durchlaufen, und für jedes Modell in models wird überprüft, ob es ein Wand-, Boden-, Box-Element oder ein 3D-Modell ist. Abhängig davon wird das entsprechende 3D-Objekt oder -Modell hinzugefügt und in der Szene angezeigt. Die Box hat die Besonderheit, dass er als einziges der Objekte (Wand, Boden, Box) 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, siehe Abbildung 4.3 rechte Seite, 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 noch mit seiner id 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.

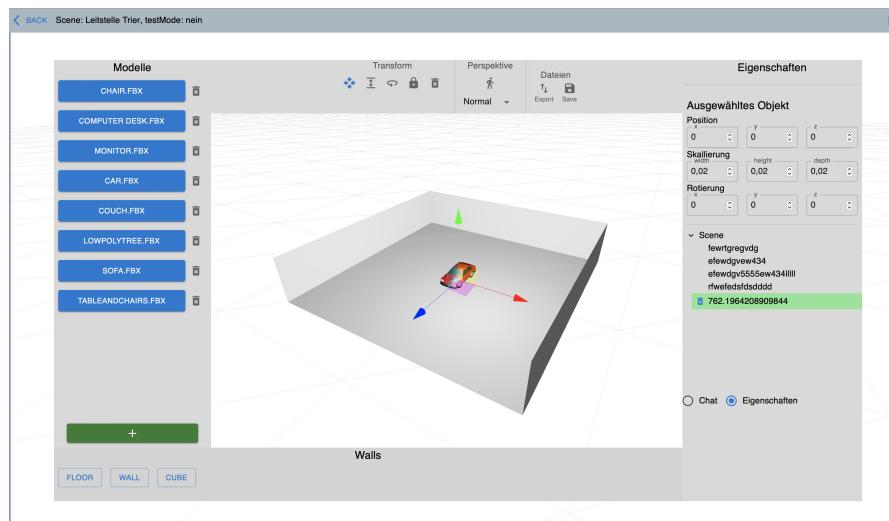


Abbildung 4.3: Version 0 mit TreeView und WallList

4.2 Version 2

Hier wurde die Datenbank integriert und die ersten Tabellen erstellt, um Benutzer und Chatbeiträge zu speichern. Dies ermöglicht die Realisierung des Logins und des Chats.

4.2.1 Datenbank

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 sein können.

Mit dem Befehl npx prisma migrate dev wird 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.

4.2.2 Benutzer und Login

Es wurde ein einfacher Login implementiert und muss sich nun mit einem Benutzer anmelden. Dazu gibt es die loginID (wird später durch eine E-Mail ersetzt) und ein Passwort. Da es noch keine Registrierungsfunktion gibt, wurde zum Testen ein dummy-Datensatz, für einen Benutzer in der Datenbank angelegt.

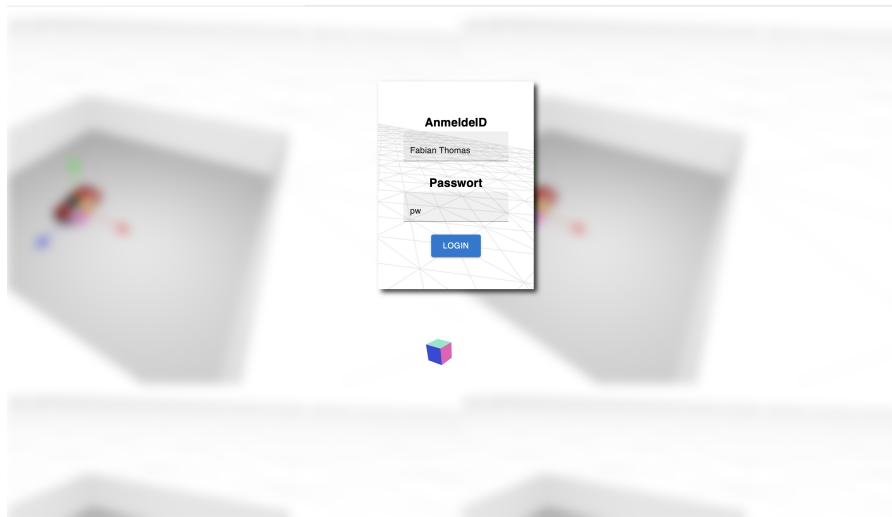


Abbildung 4.4: Login in Version 2

4.2.3 Chat

Nach der Integration der SQLite-Datenbank in Version 2 wurde als nächstes ein simpler globaler Chat mit Socket.IO erstellt. Dieser Chat ermöglicht es allen Benutzern eine Nachricht zu senden. Eine Nachricht, die von einem Benutzer gesendet wurde, wird an alle anderen Benutzer weitergeleitet. Dies wird in den späteren Versionen noch geändert, das es kein globaler Chat sein soll, sondern ein Chat pro Konfiguration geben wird.

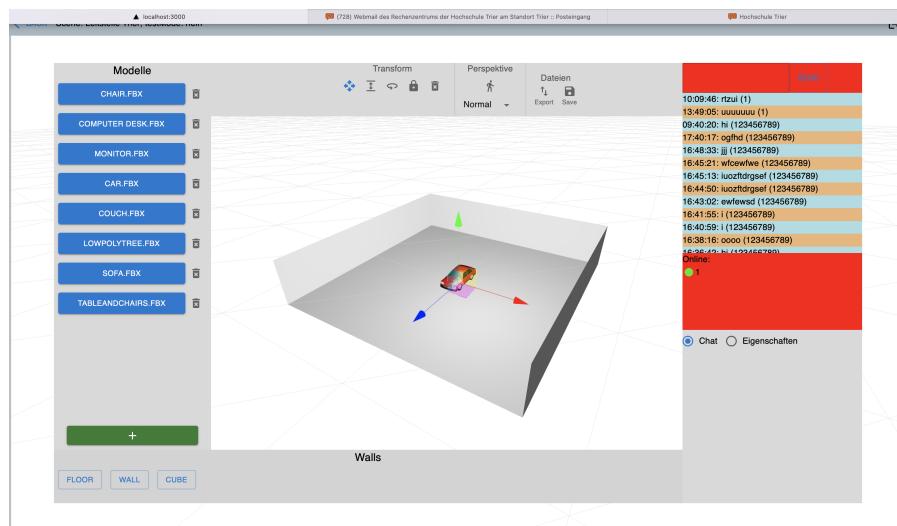


Abbildung 4.5: Chat

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 sofort 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

In Version 3 wurde ein Adminbereich eingeführt, der es Administratoren ermöglicht, Einstellungen zu ändern und auf alle Daten zuzugreifen. Außerdem wurde die Funktion hinzugefügt, Objekte die kein 3D-Modell sind, Farben oder Texturen zuzuweisen. Abschließend wurde eine Hilfsfunktion integriert, um Objekte in bestimmten Ansichten auszublenden.

4.3.1 Adminarea

Der Adminbereich wurde eingeführt, in dem der Administrator alle Benutzer anzeigen, bearbeiten und neue Benutzer hinzufügen kann. Zusätzlich besteht die

		WECHSELN	
		ZURÜCK	
		user	
id		loginID	password
1234		1234	12345
44444444		4444444444	4444444444
y		y	aysy
d		d	d
1b27243c-128a-4cd2-9b24-e0d43d4f0a87		eeeeecowas	eee32dnewcx
x		rr	rr
s		s	s
ss		ss	ss
1af20a09-8b83-43ee-bf02-c086e2b1187c		ttt	ttt
qqqweidcsd		wqdcasc	vddox
qqqweidcsd554654		wqdcasc65654	vddox5546
qqqweidcsd5546544cf		wqdcasc65654dfs	vddox5546
ww		ww	ww
wwwsay		wwwsayc	wwwsay
www		www	www
xwqgr		xdwefgr!	x
Neuen user erstellen		ANSETZEN	
id , string		password , string	
<input type="text"/>		<input type="text"/>	
loginID , string		readOnly , boolean	
<input type="text"/>		<input type="checkbox"/>	
password , string		password	
<input type="text"/>		<input type="text"/>	
readOnly , boolean		readOnly	
		AUFLESEN	

Abbildung 4.6: AdminArea Datatable user



Abbildung 4.7: AdminArea FBXList

Möglichkeit, sich die FBX-Modelle anzusehen und diese hinzuzufügen oder zu löschen.

4.3.2 Farben und Texturen

In der 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 texture-Feld 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 Möglichkeit ist, eigene Texturen, mit dem Programm hinzufügen. Dies kommt in einer späteren Version hinzu.

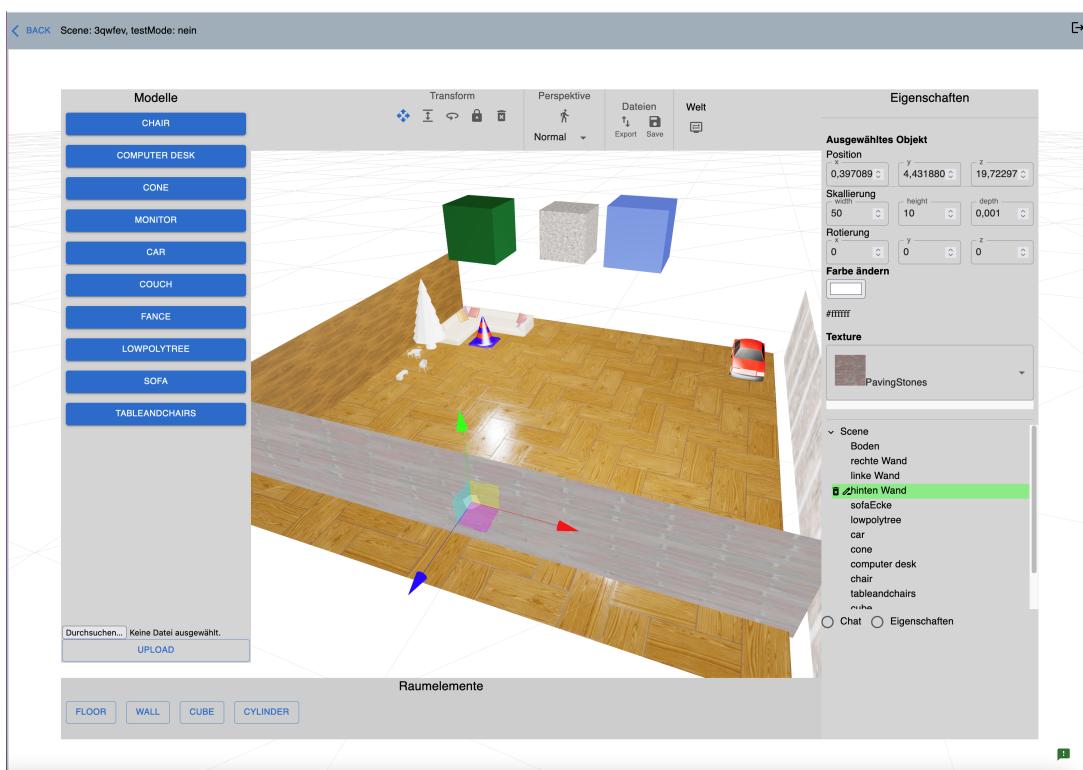


Abbildung 4.8: Texturen und Farben

4.3.3 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.

4.4 Version 4

Mit der Einführung von Version 4 erfolgten Veränderungen im Design und der Datenstrukturen der Konfigurationen. In dieser Version wurde auch die Möglichkeit eingebaut, dass Benutzer ihre eigenen Texturen hochladen können, wodurch eine höhere Anpassbarkeit ermöglicht wird. Des Weiteren erfolgte eine Anpassung der Datenstruktur zur Speicherung von Konfigurationen. Zudem wurden Memberships eingeführt, um Benutzer als Mitglieder zu Konfigurationen hinzuzufügen.

4.4.1 Memberships

Nach Erstellung einer Szene ist diese zunächst nur für den Ersteller sichtbar und bearbeitbar. Um anderen Benutzern die Möglichkeit zu geben, an der Szene mitzuarbeiten, wurden die SceneMemberships eingeführt. Wie bereits im Abschnitt zur Datenmodellierung beschrieben, kennzeichnet eine SceneMembership, welcher Benutzer Zugriff auf welche Szene hat. Darüber hinaus besteht die Option, einen Benutzer für eine Szene in den Read-only-Modus zu versetzen, was bedeutet, dass er die Szene zwar einsehen, aber nicht bearbeiten kann.

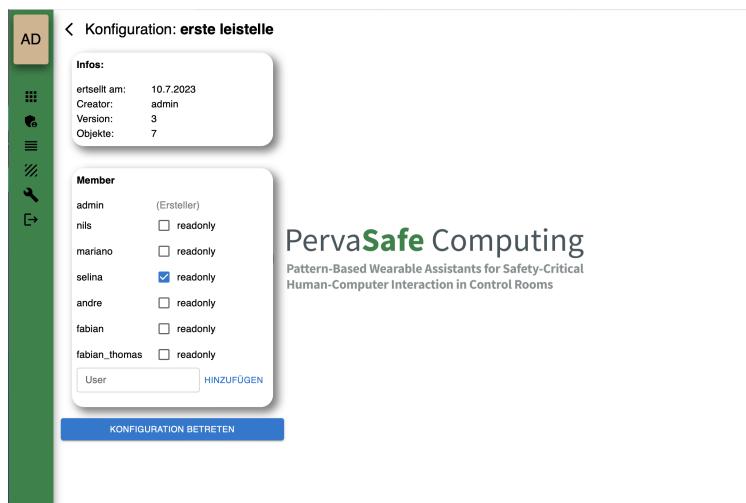


Abbildung 4.9: Mitglieder einer Konfiguration

Nur der Ersteller der Szene hat die Befugnis, Benutzer zur Szene hinzuzufügen. Andere Szene-Mitglieder sehen eine Liste der Szene-Mitglieder und können erkennen, wer als "Read-only"Mitglied gekennzeichnet ist. Beim Hinzufügen eines Benutzers wird ein neuer Eintrag in der Tabelle SSceneMembership mit den entsprechenden Berechtigungen erstellt. Standardmäßig wird ein Benutzer ohne "Read-only"Rechte hinzugefügt. Wenn der Mauszeiger über einen Eintrag bewegt wird, erscheint ein Lösch-Symbol, um den Benutzer zu entfernen.

Um einen Benutzer hinzuzufügen, muss die LoginID des Benutzers in das Textfeld eingegeben werden. Es ist nicht möglich, einen bereits bestehenden Szene-Mitglied erneut hinzuzufügen. Während der Eingabe wird bereits überprüft, ob der Nutzer existiert.



Abbildung 4.10: Benutzer suchen

4.4.2 Datenstrukturen der Scene überarbeitet

Zuvor wurden Konfigurationen als JSON-Strings in Textdateien auf dem Server gespeichert. Diese Vorgehensweise wurde geändert, und stattdessen werden nun die einzelnen Modelle separat in der Datenbank gespeichert. Hierfür wurde eine neue Tabelle namens `model` erstellt, und jedem Modell wurde das Feld `idScene` hinzugefügt, um zu kennzeichnen, welches Modell in welcher Szene verwendet wird.

Diese Umstellung ermöglicht eine einfachere Handhabung der Modelle innerhalb einer Szene und erlaubt nun auch die Selektion von Modellen, was zuvor aufgrund der gegebenen Datenstruktur nicht möglich war. Zusätzlich befinden sich nun alle Daten zentral in der Datenbank, im Gegensatz zur vorherigen Verteilung, bei der die Szenendaten im Dateisystem des Servers und die Metadaten der Szene in der Datenbank gespeichert waren. Die neue Datenstruktur bietet somit eine effizientere Verwaltung und vereinfacht die Handhabung der Szenen und Modelle.

Scene hinzufügen

Bei der Erstellung einer Szene wird zunächst ein Datensatz für die Szene selbst angelegt. Anschließend werden vier Datensätze für Modelle erstellt, die den Raum bilden, siehe Abbildung 4.11. Dieser Raum setzt sich aus einem Boden und drei Wänden zusammen. Zum Abschluss wird ein Eintrag für die Mitgliedschaft in der Szene erstellt, um den Benutzer der Szene zuzuordnen. Im Szene-Datensatz ist vermerkt, dass dieser Benutzer der Ersteller der Szene ist.

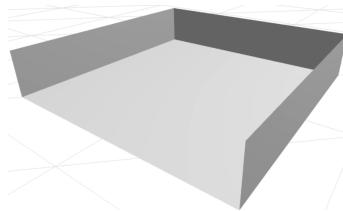


Abbildung 4.11: Start Konfiguration

Scene löschen

Wenn eine Szene gelöscht wird, sind zwei Szenarien zu differenzieren: einmal wenn der Ersteller die Szene löscht und einmal wenn ein Mitglied der Szene dies tut. Im

einfachen Fall, wenn ein Mitglied die Szene löscht, wird lediglich der Eintrag des SceneMemberships entfernt, der den Benutzer mit der Szene verknüpft. Die Szene selbst bleibt erhalten, wird jedoch nicht mehr im Benutzerprofil angezeigt.

Im Falle, dass der Ersteller die Szene löscht, erfolgt eine umfassendere Löschung. Zunächst werden sämtliche SceneMemberships gelöscht, die die Verbindung der Benutzer zur Szene herstellen. Anschließend erfolgt die Löschung aller Modelle, die zur Szene gehören. Schließlich wird der Datensatz der Szene selbst entfernt, wodurch die gesamte Szene gelöscht wird.

Hier ist wichtig zu beachten das die Models nicht mit einer map funktion durchlaufen und gelöscht werden. Dies ist in Abbildung ?? (a) dargestellt. Das löschen ist nämlich asynchronen. Die map-Funktionen startet alle lösch requests an den Server gleichzeitig, was zu einer großen Anzahl von gleichzeitigen Datenbankanfragen führen kann. Dies führt zu einem Timeout fehler von Prisma. Um das zu vermeiden werden die Objekt mit einer For-Schleif durchlaufen?? (b). Die For-Schleife hingegen arbeitet die lösch-Requests sequentiell also nacheinander ab. Wodurch die lösch-Request nacheinander ausgeführt werden.

```
requestedModelsFromScene.map(async (model: Model) => {
  // models der scene löschen
  const requestDelete1 = await fetchData(
    props.userId,
    props.sessionId,
    "model",
    "delete",
    { id: model.id },
    null,
    null
  );
});

for (const model of requestedModelsFromScene) {
  await fetchData(
    props.userId,
    props.sessionId,
    "model",
    "delete",
    { id: model.id },
    null,
    null
  );
}
```

Abbildung 4.12: Löschen mit map
(Bild noch anpassen const Request1)

Abbildung 4.13: Löschen mit for

4.4.3 Texturen selber hinzufügen

In den vorherigen Versionen waren die Texturen fest im Programmcode verankert und konnten nicht dynamisch verändert werden. Mit der Erweiterung ist es nun möglich, neue Texturen dynamisch hochzuladen. Dadurch können Benutzer nun ihre eigenen Texturen hochladen und auf ihre individuellen Bedürfnisse anpassen. Dazu wurde im Homebildschirm eine neue Seite TextureList hinzugefügt, wo berechtigte Personen Texturen löschen und hinzufügen können. Mehr dazu in Kapitel TextureList.

4.5 Version 5

Die Rechteverwaltung wurde überarbeitet und umfasst nun Berechtigungen wie read, write, delete und update. Der Admin-Bereich wurde erweitert, um dem Administrator die Möglichkeit zu geben, Mitglieder einer Konfiguration einzusehen und zu bearbeiten. Des Weiteren wurde die Funktionalität hinzugefügt, zwischen den verschiedenen gespeicherten Versionen einer Konfiguration zu wechseln.

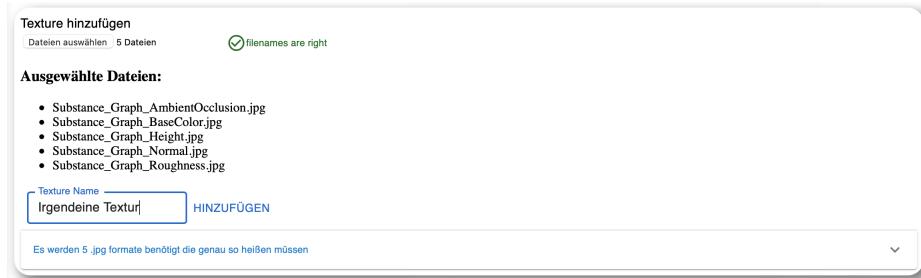


Abbildung 4.14: Textur hinzufügen

4.5.1 Rechte überarbeitet

Die Rechteverwaltung wurde überarbeitet. Während in den älteren Versionen lediglich die Berechtigungen `readonly` und `isAdmin` existierten, umfasst das neue Rechtesystem nun die Abstufungen `read`, `write`, `delete`, `update` und `isAdmin`. Zusätzlich wurde eine Funktion implementiert, die vor jeder Serveranfrage die Berechtigungen überprüft. Neben den Berechtigungen wird auch die Session-ID vorab geprüft, da ohne gültige Session-ID keine Daten vom Server abgerufen werden können, wie bereits im Kapitel zur Sicherheit beschrieben wurde. Das Recht `read` ermöglicht das Abrufen von Konfigurationen, FBX-Modellen und Texturen. Fehlen dem Benutzer diese Rechte, ist ihm der Zugriff auf die entsprechenden Daten und Funktionen verwehrt. Das Recht `write` bestimmt, ob Nutzer neue Daten erstellen können. Dazu gehören das Erstellen und Modifizieren von Konfigurationen, das Hochladen neuer FBX-Modelle oder das Hinzufügen von Texturen. Das Recht `delete` legt fest, ob Benutzer Konfigurationen, FBX-Modelle oder Texturen löschen dürfen. In Abbildung 4.15 wird ein Beispiel gezeigt, bei dem ein Nutzer ohne `read`-Rechte versucht, auf die 3D-Modelle zuzugreifen.



Abbildung 4.15: Keine Rechte um einen Bereich zu öffnen

Die zugehörigen Steuerelemente für read, write oder delete werden entsprechend ausgeblendet. Zusätzlich erfolgt eine serverseitige Überprüfung dieser Rechte, um sicherzustellen, dass die Aktionen nur von Benutzern mit den entsprechenden Berechtigungen ausgeführt werden können, siehe dazu Abbildung 4.16.

```
export default async function DB_executeSQL(
  req: NextApiRequest,
  res: NextApiResponse
) {
  const { tableName, action, where, data, include, sessionID, idUser } =
    req.body;

  // SESSION
  const check = await checkSessionID(sessionID);
  if (!check) {
    res.status(403).json({
      error: "Zugriff verweigert: Ungültige SessionID.",
    });
    return;
  }

  // RECHTE
  const rights = await checkUserRights(idUser, action);
  if (!rights) {
    res.status(403).json({
      error: "Zugriff verweigert: Der Nutzer hat keine Rechte für diese Aktion.",
    });
    return;
  }

  //...
}
```

Abbildung 4.16: Session und Rechte Serverseitig prüfen

Die Abbildung 4.16 zeigt die API-Funktion DB-executeSQL um Daten aus der Datenbank anzufragen oder zu bearbeiten, wie in Kapitel 3.5.2 beschrieben. Die Funktion checkSessionID überprüft, ob die übergebene Session-ID in der Anwendung registriert ist. Die Funktion checkRights hingegen dient zur Überprüfung, ob der Benutzer berechtigt ist, die angeforderte Aktion auszuführen. Zum Beispiel wird bei einem Löschvorgang geprüft, ob der Benutzer die erforderlichen Löschrechte besitzt

4.5.2 Adminbereich erweitert

Der Administrationsbereich wurde erweitert, um dem Administrator die Möglichkeit zu geben, alle Szenen einzusehen, einschließlich der Mitglieder, und diese auch zu löschen oder zu bearbeiten.

Zusätzlich musste das löschen eines Benutzers angepasst werden. Wenn ein Benutzer gelöscht wird, werden auch alle seine Daten wie ChatEntries und Memberships gelöscht. Wegen foreignKey contrains.

4.5.3 Scene Versionen

Jede Szene enthält mehrere Modelle. Der Datensatz der Szene hat ein neues Feld namens "newestVersion", das die neueste Version der Szene angibt. Nach jedem Speichern wird die neueste Version um eins erhöht und im Szene-Datensatz aktualisiert. Anschließend werden alle Objekte durchlaufen und mit der neuen Version

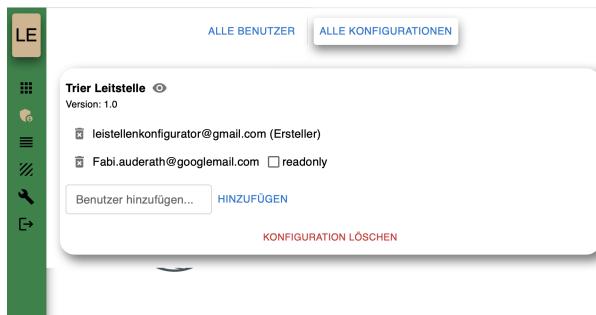


Abbildung 4.17: Scenes plus Members

in die Datenbank eingetragen. Die alten Objekte mit der alten Version bleiben in der Datenbank erhalten, um zwischen den verschiedenen Versionen wechseln zu können. Wenn beispielsweise die neueste Version 8 ist, können ältere Versionen wie Version 4 problemlos geladen werden, indem man einfach alle Modelle in der Szene auswählt, die der Version 4 zugeordnet sind. In Abbildung ist die Speicherung von den Objekten in der Szene mit unterschiedlichen Version dargestellt.

Id	name	color	texture	version
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 4.18: Beispiel Model Version

Diese Konzept erlaubt ein einfaches umschalten zwischen den Versionen.

4.6 Version 6

In dieser Version wurde die Möglichkeit hinzugefügt, gleichzeitig mit anderen an einer Szene zu arbeiten. Dazu wurde eine neue Tabelle CurrentSceneEdit angelegt. Dort wird gespeichert wer gerade an welcher Konfiguration arbeitet. Zudem wurden neue Funktionen für die Einstellung von Lichtquellen implementiert.

4.6.1 CurrentSceneEdit

Bei jedem Betreten einer Konfiguration wird ein neuer Datensatz CurrentSceneEdit in der Datenbank angelegt. Dieser Datensatz speichert Informationen darüber,

welcher Benutzer gerade an welcher Konfiguration arbeitet. Dadurch wird die Möglichkeit geschaffen, zu verfolgen, welcher Benutzer an welcher Konfiguration arbeitet. Dieser Datensatz wird mit den anderen verbunden Clients geteilt. Nur Clients die sich gerade in der selben Konfiguration befinden verarbeiten die Daten. Clients die in der selben Konfiguration arbeiten laden alle CurrentSceneEdits der Konfiguration, und haben so Daten die angeben wer gerade an derselben Konfiguration arbeitet. Anhand dessen werden die Benutzer visuell in der Konfiguration dargestellt. Die Visuelle Darstellung ist in Abbildung 5.6 verdeutlicht. Verlässt ein Benutzer die Konfiguration, teilt er dies den anderen Clients über socket io mit, und diese entfernen dann die Visuelle Darstellung des Benutzers welcher die Konfiguration verlassen hat. Zusätzlich werden alle CurrentSceneEdits nach einer bestimmten inaktiven Zeit gelöscht und werden somit nicht mehr angezeigt.

Der Client betritt Scene und legt CurrentSceneEdit DATensatz an und speichert ihn ab und schickt ein Ereigniss mit socket io (sceneOnEnter) mit dem CurrentSceneEdit Datensatz.

Jede SCene hört auf das Ereignis getSceneOnEnter welches gefeuert wird wenn eine Benutzer eine Scene betritt. Diese Funktion ändert den Status der ein Laden der neuen CurrentSceneDatensätze erzwingt. Dadurch wird der neue Datensatz vom beigetretenen Benutzer geladen und gespeichert in workers. Die Scene durchläuft worker und stellt sie visuell da, das sind die UserCams.

CurrentSceneEdits werden bei verlassen der Szene in der menubar (backbutton) gelöscht oder nach 20min Inaktivität. Die 20min Inaktivität werden gelöscht wenn die SceneList gerendert wird. Ein Benutzer kann ja nur eine scene bearbeiten, also einen CurrentSceneEdit Datensatz haben. Um das sicherzustellen, werden vor jedem Betreten einer scene alle Datensätze mit der idUser im CurrentSceneEdit gelöscht. DAs stellt sicher dass alle Datensätze gelöscht werden und dann ein neuer angelegt wird.

Das folgende Bild zeigt, welche Benutzer gerade an der Konfiguration arbeiten.

4.6.2 Synchronisieren einer Scene

Beim Betreten einer Szene werden die Objekte aus der Datenbank geladen und im Programm im Status models gespeichert. Wenn ein Benutzer ein neues Objekt zur Szene hinzufügt, wird dieses zunächst in den models Status aufgenommen, ohne dass es bereits in der Datenbank gespeichert ist, wie in Abbildung 5.7 dargestellt. Um sicherzustellen, dass andere Benutzer, die an derselben Szene arbeiten, das hinzugefügte Objekt ebenfalls sehen, wird es über Socket.IO an die anderen Clients verteilt. Dabei verarbeiten nur die Benutzer das Objekt, die sich in der selben Szene befinden. Alle anderen Clients verwerfen es.

Um die Synchronisierung zu gewährleisten, muss das Objekt mit derselben ID auf den anderen Clients hinzugefügt werden. Dadurch wird sichergestellt, dass die Objekte auf allen beteiligten Clients eindeutig identifiziert und richtig platziert werden. Diese Implementierung ermöglicht eine Echtzeit-Kollaboration, bei der die Benutzer gleichzeitig an derselben Szene arbeiten und ihre Änderungen sofort mit anderen teilen können.

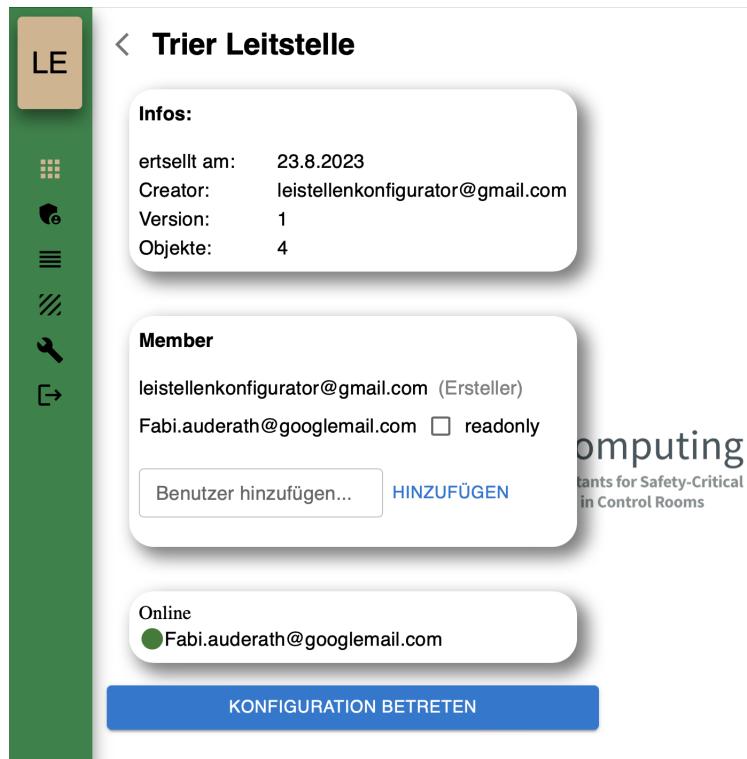


Abbildung 4.19: Aktive Benutzer in einer Konfiguration

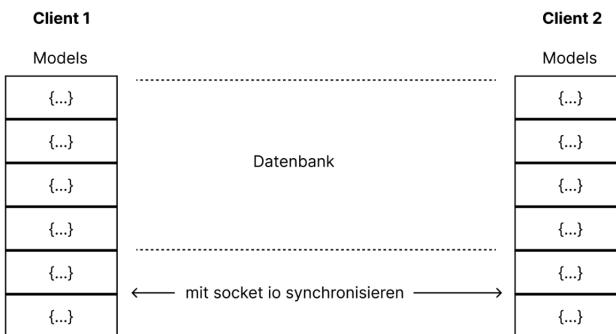


Abbildung 4.20: Synchronisation der Objekt in einer Konfiguration

UserCam

Beim Betreten einer Szene hat der Benutzer die Möglichkeit, diese mithilfe der Orbitcontrols aus Three.js zu erkunden. Wenn ein weiterer Benutzer sich mit derselben Szene verbindet, wird die Position der Kamera zwischen den beiden Benutzern synchronisiert, indem Socket.IO verwendet wird. Für jeden Benutzer, der an der Szene arbeitet, wird ein rotes Viereck in der Szene platziert, das die aktuelle Position der Kamera des jeweiligen Nutzers repräsentiert. Dies kann die Interaktion und Kommunikation zwischen den Benutzern in der Leitstellen-Konfiguration verbessert.

USerCam-Control lauscht auf das socket io ergebnis getUserCamData. DAs ergebnis wird geschickt wenn ein Benutzer die Kamera bewegt. Wied die Kamera bewegt, wird die Position und skalierung mit den anderen Client mit socket io geteilt. Es wird ein Objekt (currentWorkerID: pos rot) verteilt. Alle anderen clients in der Scene haben ja eine UserCam, mit der id des currentWordEdit Datensatz, als control engelgt, deswegen wissen die was die aktualisieren müssen.



Abbildung 4.21: Sicht von admin



Abbildung 4.22: Sicht von nils

Sobald ein Benutzer die Kamera bewegt, wird die position an die anderen Clients gesendet, damit die die position synchronisieren können.

4.6.3 Licht

Es wurden Funktionen eingebaut um die Lichteinstellungen der Konfiguration zu verändern. Es existieren verschiedene Arten von Beleuchtungen in der Konfiguration. Erstens die Umgebungsbeleuchtung, die eine grundlegende Beleuchtung der Szene darstellt und alle Flächen beleuchtet. Dann gibt es das gerichtete Licht (directional light), das Lichtstrahlen in einer bestimmten Richtung bündelt und in die Szene ausstrahlt. Diese Ausrichtung nun je nach Bedarf verschoben werden. Das verschiebbare Licht ist in Abbildung 4.23 dargestellt.

Die Richtung dieses Lichts ist immer zur Mitte gerichtet. Zusätzlich können PointLights hinzugefügt werden und in der Szene verschoben werden. Jede Lichtquelle außer dem Umgebungslicht (ambient light) wird durch ein Lichtsymbol in der Szene gekennzeichnet.

4.7 Version 7

In den vorherigen Versionen registrierte sich ein Benutzer mit einer LoginID und einem Passwort und konnte sich mit diesen Daten direkt anmelden. Ein Nachteil dabei war, dass wichtige Informationen außerhalb des Programms, wie beispielsweise ein neues Passwort oder eine Verifizierungsmaile, nicht an den Benutzer

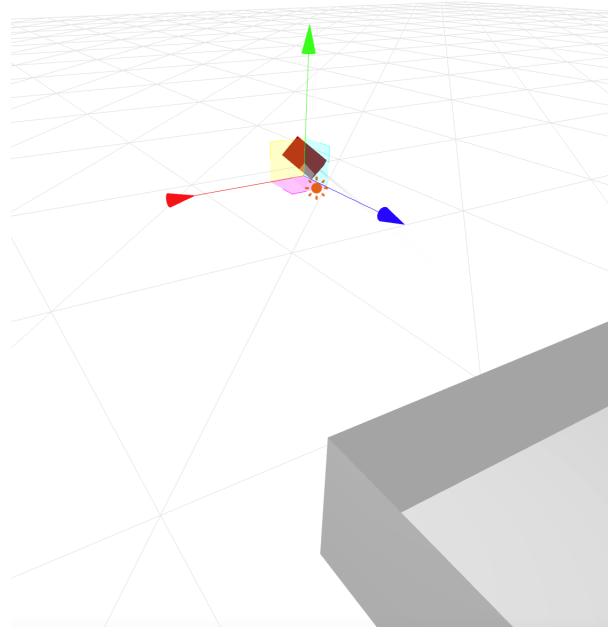


Abbildung 4.23: Beispiel Model Version

übermittelt werden konnten. Um diese Nachteile zu beheben, wurde die LoginID durch eine E-Mail-Adresse ersetzt, wodurch die Kommunikation und Benachrichtigungen außerhalb der Anwendung umgesetzt werden kann.

4.7.1 Mit E-Mail Registrieren

Zusätzlich wurde eine Funktion eingeführt, mit der Benutzer ihr Passwort ändern können. Die Umsetzung dieser Funktion erforderte die Verwendung einer API-Funktion, um E-Mails zu versenden. Dies wurde mit Hilfe von Nodemailer realisiert, einer weitverbreiteten Node.js-Bibliothek, die speziell für den E-Mail-Versand entwickelt wurde. Diese Bibliothek erleichtert das Erstellen und Senden von E-Mails über das SMTP (Simple Mail Transfer Protocol). Um E-Mails senden zu können, wurde ein eigener Account bei einem E-Mail-Provider erstellt, in diesem Fall Google Mail (Gmail). Zur Nutzung dieser Funktion musste die Zwei-Faktor-Authentifizierung (2FA) aktiviert werden, um ein App-Passwort zu generieren. Dieses App-Passwort wird benötigt, um E-Mails über Gmail verschicken zu können, aufgrund der von Google implementierten Sicherheitsmaßnahmen. Bei diesem Prozess sendet der Client eine Anfrage an den Server mit einer E-Mail-Adresse als Parameter, und der Server versendet dann die entsprechende E-Mail an die angegebene Adresse.

Der vorliegende Code nutzt die Nodemailer-Bibliothek in Node.js, um die Versendung von E-Mails zu ermöglichen. Die Funktion `sendEmail` nimmt die Empfängeradresse, den Betreff sowie den Textinhalt der E-Mail entgegen. Ein

```

1  const nodemailer = require("nodemailer");
2
3  const sendEmail = async (to1, subject1, text1) => {
4    // Konfiguration des Transporters
5    const transporter = nodemailer.createTransport({
6      service: "gmail",
7      auth: {
8        user: "leistellenkonfigurator@gmail.com",
9        pass: "mchjrlhsvosdzjo",
10       },
11     });
12
13    // E-Mail-Details
14    const mailOptions = {
15      from: "leistellenkonfigurator@gmail.com",
16      to: to1,
17      subject: subject1,
18      text: text1,
19    };
20
21    transporter.sendMail(mailOptions, function (error, info) {
22      if (error) {
23        throw new Error(error);
24      } else {
25        console.log("Email Sent");
26        return true;
27      }
28    });
29  };
30
31  module.exports = sendEmail;

```

Abbildung 4.24: Code um eine E-Mail zu versenden

Transporter wird konfiguriert, der speziell auf den Gmail-Dienst abzielt. Für die Authentifizierung werden die E-Mail-Adresse und das App-Passwort verwendet.

Die Details der zu sendenden E-Mail werden in den mailOptions spezifiziert, darunter die Absenderadresse, die Empfängeradresse, der Betreff und der Textinhalt.

Die sendEmail-Funktion wird abschließend exportiert, wodurch sie in der API-Funktion eingesetzt und verwendet werden kann.

4.7.2 Passwort verschlüsselt speichern

Das Passwort war in den vorherigen Versionen immer als Klartext gespeichert. Das hat sich nun geändert. Das Passwort wird im Hashverfahren verschlüsselt und das gehashte Passwort wird in der Datenbank gesichert. Beim Login wird auch nun das eingegebene Passwort gehashet und mit dem gehasteten Wert aus der Datenbank verglichen.

4.7.3 Passwort vergessen und Passwort ändern

Für den Fall, dass ein Benutzer sein Passwort vergisst, wurde auch eine Passwort vergessen Funktion implementiert. Hierbei wird ein neues Passwort per E-Mail an die registrierte Adresse gesendet. Dazu wurde auch eine neue API-Funktion implementiert. Zudem wurde noch eine Funktion implementiert, um sein Passwort ändern zu können. Dazu wird das neu eingegebene Passwort gehashet und in der Datenbank eingetragen.

4.7.4 Anzeigename

Es wurde eine Anzeigename eingeführt. Weil E-Mail war zu lange und man weiß nicht genau wer gemeint ist. Dieser kann in den Einstellungen geändert werden.

5

Ergebnisdarstellung

Nachdem im vorherigen Kapitel auf die Umsetzung eingegangen wurde werden in diesem Kapitel die Ergebnisse der verschiedenen Seiten des Konfigurators betrachtet und ihre jeweiligen Funktionalitäten erläutert. Die Abbildung 5.1 dient als Überblick und zeigt die wichtigsten Hauptkomponenten des Konfigurators.

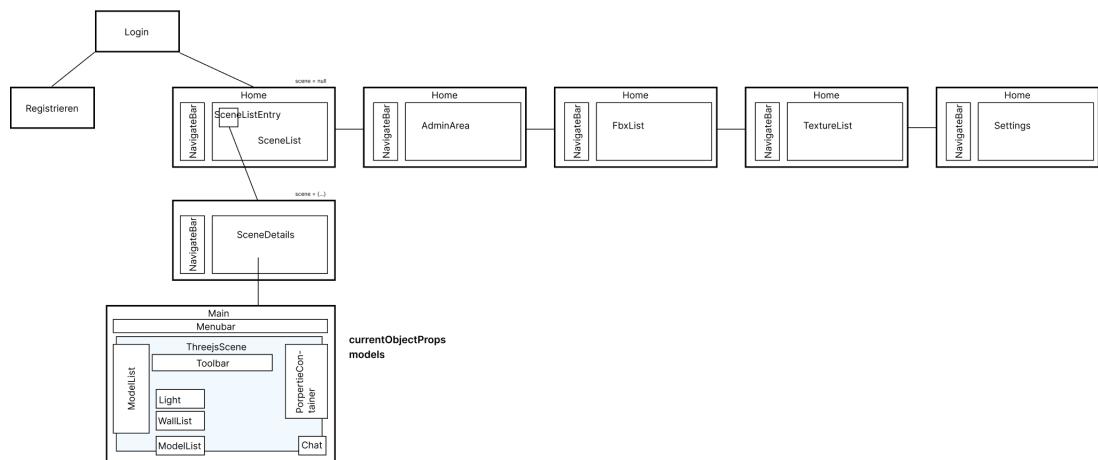


Abbildung 5.1: Sitemap

5.1 Login und Register

Um den Leitstellen-Konfigurator zu nutzen, ist zunächst ein Account erforderlich. Die Registrierung erfolgt durch die Angabe einer E-Mail-Adresse. Nach der Registrierung wird eine Bestätigungs-E-Mail mit einem generierten Passwort an diese E-Mail-Adresse gesendet. Mit diesem Passwort kann sich der Benutzer dann anmelden. Für den Fall, dass der Benutzer sein Passwort ändern möchte, steht ihm diese Möglichkeit in den Einstellungen zur Verfügung.

Sollte der Benutzer sein Passwort vergessen, kann er über die Option "Passwort vergessen" ein neues Passwort anfordern. In diesem Fall wird ihm ein neues Passwort an seine registrierte E-Mail-Adresse gesendet.

Nach einer erfolgreichen Anmeldung wird automatisch eine Sitzung (Session) für den angemeldeten Benutzer erstellt. Diese Sitzung ist essenziell, da sie für die Authentifizierung erforderlich ist. Ohne eine gültige Sitzung kann kein Zugriff auf Daten vom Server erfolgen. Bei jeder Serveranfrage wird die Sitzung in der Anfrage mitgeliefert und auf Serverseitiger Ebene überprüft.

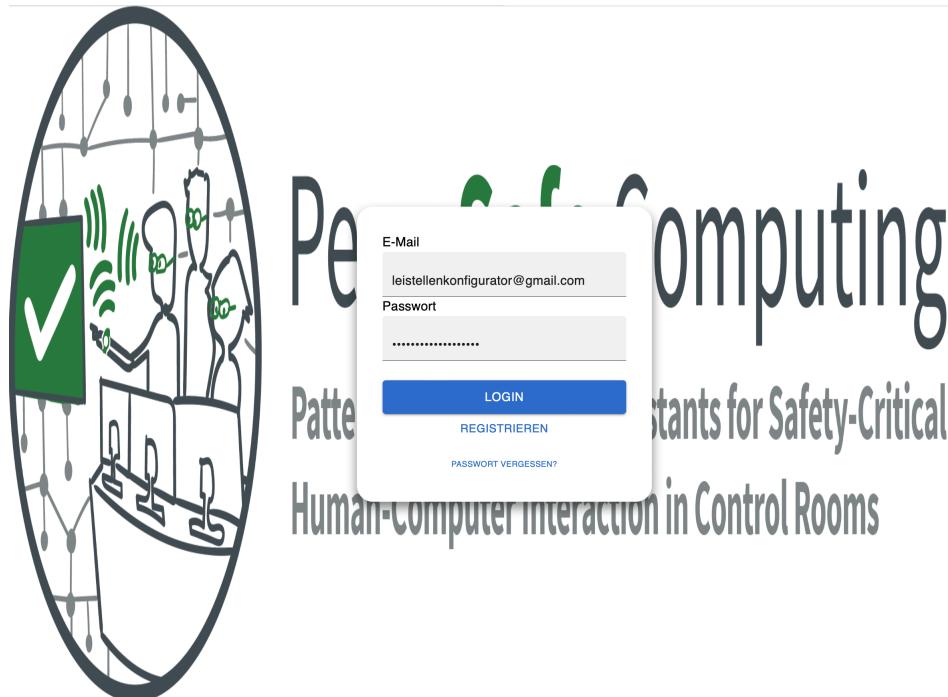


Abbildung 5.2: Login

5.2 Home und Navigatebar

Nach erfolgreichem Anmelden wird der Benutzer zur Startseite weitergeleitet, wie in Abbildung 5.3 zu sehen ist. Die Startseite ist in zwei Hauptbereiche unterteilt: die Navigationsleiste (Navigatebar) und die aktuell ausgewählte Komponente. Über die Navigationsleiste kann der Benutzer zwischen verschiedenen Komponenten wie SceneList, AdminArea, FbxList, TextureList und Settings navigieren. Die Standardauswahl ist die SceneList-Komponente. Zusätzlich befindet sich auf der Startseite ein Logout-Button. Im weiteren Verlauf werden die einzelnen Komponenten genauer erläutert.

5.3 Scenelist

Auf dieser Seite werden sämtliche erstellten Konfigurationen angezeigt, die vom Benutzer entweder eigenständig erstellt oder zu denen er hinzugefügt wurde. Beim

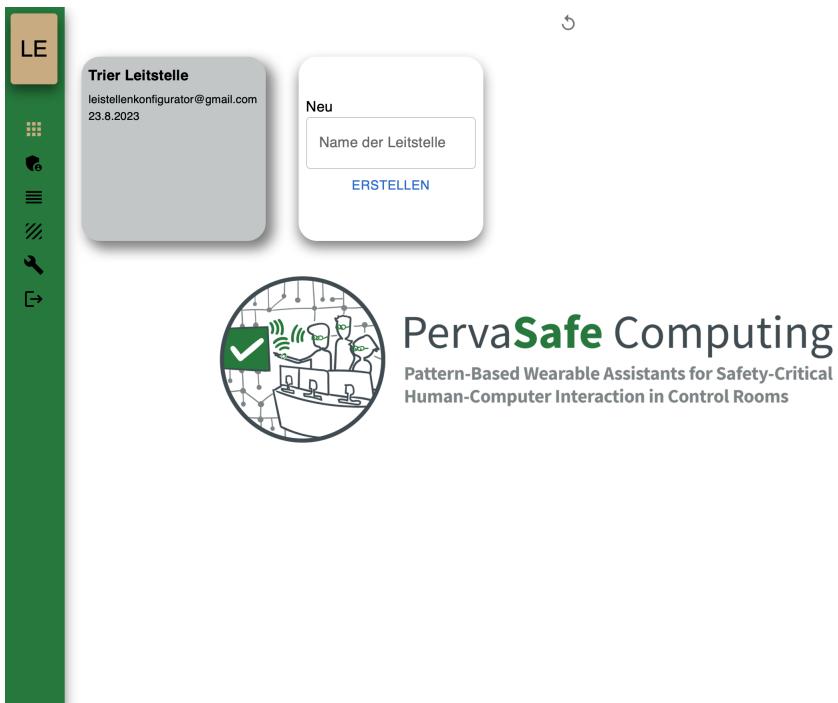


Abbildung 5.3: Home

Hovern über die jeweiligen Kacheln werden weitere Informationen zur entsprechenden Konfiguration angezeigt.

Am Ende der Liste befindet sich eine spezielle Kachel, die zur Erstellung einer neuen Konfiguration dient. Hierbei vergibt der Benutzer einen Namen und bestätigt die Erstellung durch Klicken auf den **Erstellen**-Button.

Durch Klicken auf eine Kachel gelangt der Benutzer zur Übersicht der ausgewählten Konfiguration, diese ist in Abbildung zu 5.4 sehen. Hier werden Details der Szene erneut zusammengefasst und dargestellt. Zudem enthält die Übersicht eine Liste der Benutzer, die zu dieser Konfiguration gehören. In dieser Liste können Benutzer hinzugefügt oder entfernt werden. Für Benutzer, die lediglich die Szene betrachten dürfen und keine Bearbeitungsrechte haben sollen, kann der Status **readonly** zugewiesen werden. Um die Konfiguration zu betreten, steht ein entsprechender Button zur Verfügung. Falls andere Benutzer gerade die Konfiguration bearbeiten, wird das auch angezeigt, dies ist in Abbildung 4.16 zu sehen.

5.4 Leitsellenkonfiguration

Dies ist das Herzstück des Leitstellen-Konfigurators. Es vereint 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(1), Toolbar(4), PropertieContainer(5), TreeView(3), ModelList(8), WallList(7), Light(6) und der Chat(2).

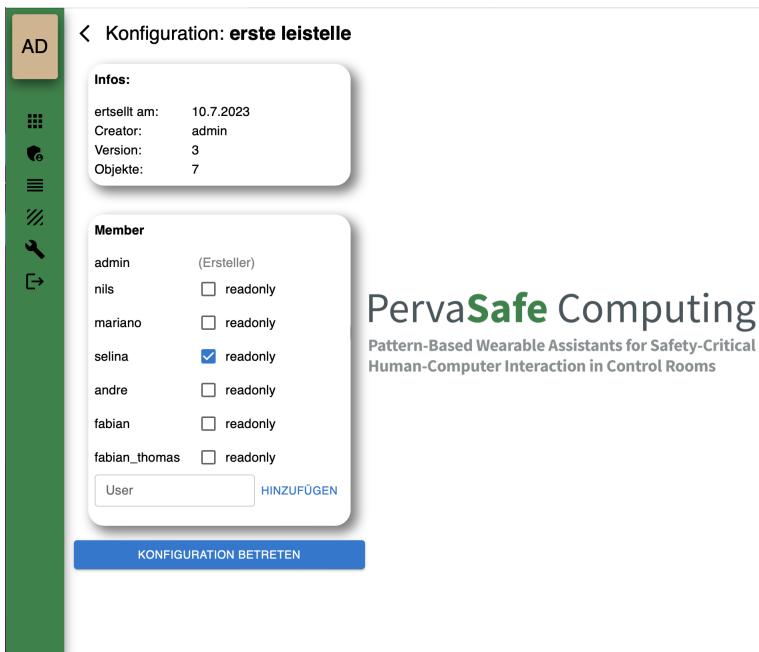


Abbildung 5.4: Konfigurationsübersicht

Die UI-Elemente Chat, WallList, ModelList und Light sind standardmäßig minimiert und müssen explizit eingeblendet werden. Wenn sie eingeblendet sind, können sie 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. Sie können nicht verschoben werden.

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 die Funktionsweise und Interaktionen der einzelnen UI-Elemente erläutern.

5.4.1 Threejs Scene

Die Three.js Scene, im Bild als 1 markiert, bildet das zentrale Element der Konfiguration und besteht aus verschiedenen Komponenten, darunter eine Kamera, ambientLight, ein pointLight, Usercams sowie eine Vielzahl von Objekten, einfache Geometrien oder 3D-Modellen, die dynamisch in die Konfiguration hinzugefügt werden können.

Innerhalb der Scene können die einzelnen Objekte durch Transformationen verändert werden. Wenn ein Benutzer auf ein bestimmtes Objekt klickt, wird dieses als aktuelles Objekt 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, besteht die Möglichkeit, Farben oder Texturen zu verwenden, wie im Kapitel "PropertieContainer" näher erläutert

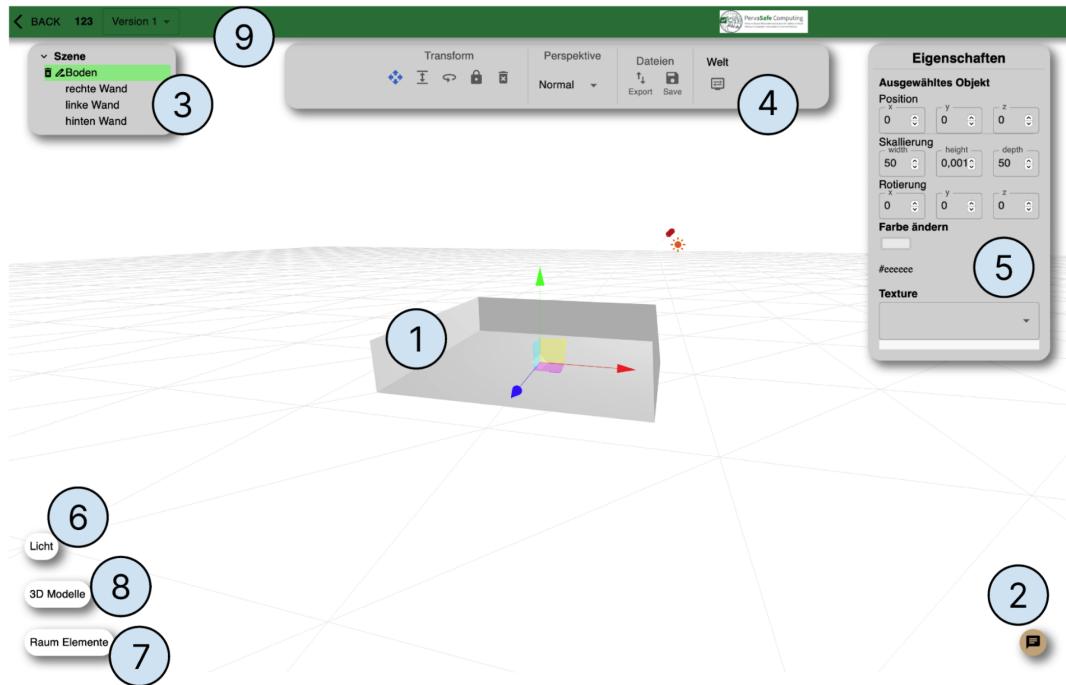


Abbildung 5.5: Leitstellenkonfiguration

wird. Die Three.js Scene bildet somit das zentrale Arbeitsfeld für die Konfiguration der Leitstelle und ermöglicht eine Manipulation der Objekte und Eigenschaften.

Synchronisation

Der Konfigurator ermöglicht die gleichzeitige Zusammenarbeit mehrerer Benutzer an einer Konfiguration/Szene. Dadurch werden Änderungen, wie das Transformieren (Verschieben, Drehen und Skalieren), Hinzufügen oder Löschen von Objekten, in Echtzeit für alle beteiligten Benutzer sichtbar. Zusätzlich wird die aktuelle Position eines anderen Benutzers in der Szene visualisiert



Abbildung 5.6: Mehrer Benutzer gleichzeitig in einer Szene

5.4.2 Chat

Wie schon in der Zielsetzung erwähnt, gibt es pro Szene einen Chat, welcher in echt Zeit kommuniziert. Der Chat ist Standardmäßig minimiert und kann mit dem Button, welcher in Abbildung 5.5 mit der Nummer 2 markiert ist, geöffnet werden. Geöffnet sieht der Chat, wie in Abbildung 5.5 zu sehen ist, aus.

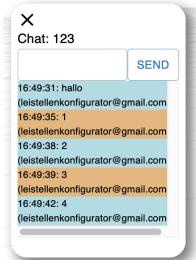


Abbildung 5.7: Chat pro Szene

5.4.3 TreeView

Der TreeView listet alle in der Szene vorhandenen Objekte auf. Das TreeView ist mit der Nummer 3, in Abbildung 5.5, markiert. Für jedes Objekt in der Szene gibt es einen entsprechenden Eintrag im TreeView. Durch Anklicken eines Eintrags wird das zugehörige Objekt in der Szene ausgewählt. Hier können Objekte nicht nur ausgewählt, sondern auch gelöscht oder umbenannt werden, was der besseren Organisation dient. Gleichzeitig wird die Auswahl eines Objekts in der Szene auch im TreeView markiert.

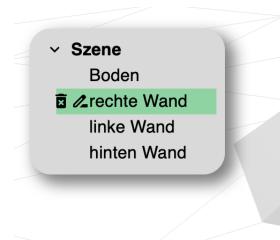


Abbildung 5.8: TreeView

5.4.4 Toolbar

Die Toolbar, mit der Nummer 4 in Abbildung 5.5 markiert, war bereits im ursprünglichen Frontend-Konfigurator vorhanden, jedoch wurden einige Änderungen und zusätzliche Funktionen integriert. Sie ist in vier Bereiche unterteilt: Transform, Perspektive, Dateien und Welt.

Der Bereich "Transform" bezieht sich auf das aktuell ausgewählte Objekt in der Szene und bietet Funktionen für seine Transformation. Dazu gehören Verschieben, Skalieren, Drehen, Sperren und Löschen. Beim Verschieben, Skalieren und Drehen wird der entsprechende Modus des Transformcontrols angepasst. Durch Klicken auf "SSperren" wird das Objekt unveränderlich gemacht, während das Klicken auf "Löschen" es aus der Szene entfernt.

Im Bereich "Perspektiven" besteht die Möglichkeit, die Ansicht zu verändern. Verschiedene Ansichten stehen zur Auswahl: leftMid, rightMid, frontal, topdown und Normal. Bei der Ansicht "Normal" wird die Szene in einer perspektivischen Darstellung betrachtet, während bei den anderen Ansichten eine orthogonale Ansicht verwendet wird. Die orthogonale Ansicht mag weniger realistisch erscheinen, erlaubt jedoch eine bessere Erkennung der Anordnung der Objekte im Raum.

Im Bereich "Dateien" besteht die Möglichkeit, die Szene entweder als GLTF-Datei zu exportieren oder die Szene zu speichern.

Im Bereich "Welt" besteht die Möglichkeit, Einstellungen für einzelne Objekte zu ändern. Hier kann ausgewählt werden, welche Objekte in den orthogonalen Ansichten ein- oder ausgeblendet werden sollen. Diese Funktion dient dazu, zu verhindern, dass bestimmte Objekte andere verdecken und somit eine präzise Anordnung im Raum erschwert wird.



Abbildung 5.9: Toolbar

5.4.5 Eigenschaften

Dieses UI-Element, welches in Abbildung 5.5 mit der Nummer 5 markiert ist, bezieht sich ebenfalls auf das aktuell ausgewählte Objekt in der Szene. Hier können die Verschiebung, Drehung oder Skalierung über numerische Werte angepasst werden, anstelle des Pivot-Controls. Des Weiteren können für Objekte, die keine 3D-Modelle sind, Texturen oder Farben zugewiesen werden.

Um Texturen Hochzuladen siehe Kapitel TextureList.

5.4.6 Licht

In diesem UI-Element, welches in Abbildung 5.5 mit der Nummer 6 markiert ist, haben Sie die Möglichkeit, Lichteinstellungen vorzunehmen. Sie können die Intensität des Ambientlight anpassen, um die Beleuchtung heller oder dunkler zu gestalten. Darüber hinaus können Sie weitere Lichtquellen (Pointlights) in die Szene einfügen. Diese Lichtquellen können verschoben werden und werden im TreeView

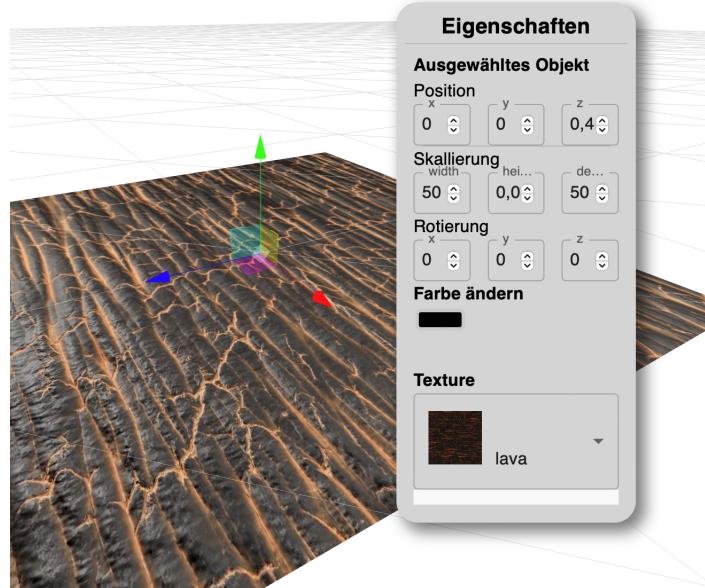


Abbildung 5.10: PropertieContainer

angezeigt. Es ist jedoch wichtig zu beachten, dass die Lichtquellen nicht im eigentlichen Szenendatensatz gespeichert werden. In der Szene werden Lichtquellen durch ein Sonnen-Icon visualisiert.

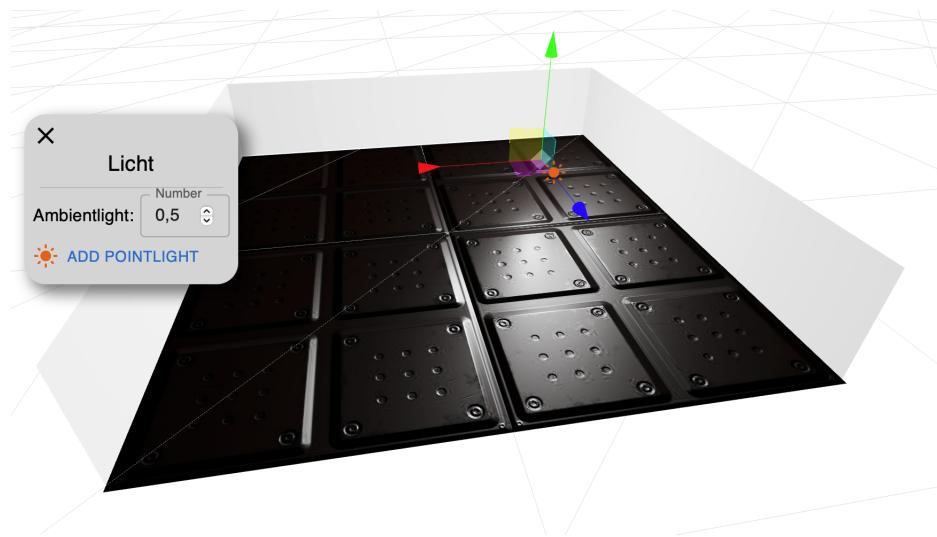


Abbildung 5.11: Licht

5.4.7 WallList

In diesem UI-Element, welches in Abbildung 5.5 mit der Nummer 7 markiert ist, können einfache Geometrien wie Wände, Böden, Boxen und Zylinder hin-

zugefügt werden. Dies ermöglicht den Benutzern, individuelle Räume nach ihren Bedürfnissen zu erstellen. Die Box und der Zylinder können in alle Richtungen skaliert werden, wobei die Wand und der Boden beim Skalieren Einschränkungen haben und ihre Form beibehalten.

So sehen die einzelnen Geometrien aus, die hinzugefügt werden können.

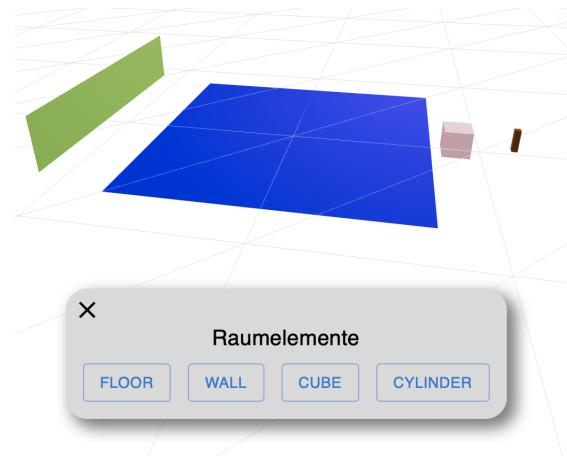


Abbildung 5.12: WallAdd plus Walls

5.4.8 ModelList

Die ModelList, welche in Abbildung 5.5 mit der Nummer 8 markiert ist, enthält eine Übersicht aller verfügbaren 3D-Modelle, die in die Szene eingefügt werden können. Durch Anklicken eines Eintrags wird das entsprechende Objekt in die Szene eingefügt. Gleichzeitig wird es als aktuell ausgewähltes Objekt markiert, was bedeutet, dass es automatisch nach dem Einfügen selektiert ist.

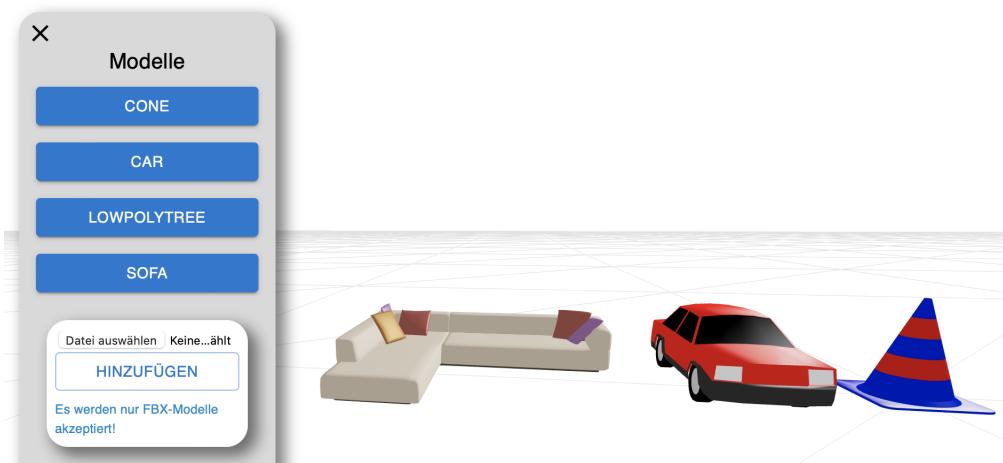


Abbildung 5.13: ModelAdd

5.4.9 Menubar

Die Menubar, welche in Abbildung 5.5 mit der Nummer 9 markiert ist, zeigt den Namen der aktuellen Konfiguration an, zusammen mit der aktuellen Version und einem Dropdown-Menü, das es ermöglicht, zwischen verschiedenen Versionen zu wechseln. Zusätzlich gibt es einen Button, der es ermöglicht, die Konfiguration zu verlassen und zur Übersicht zurückzukehren.



Abbildung 5.14: Menubar

5.5 Adminarea

Im Adminbereich können Benutzer und ihre Rechte verwaltetet werden. Die Rechte können angepasst werden. Benutzer können gelöscht und es können neue Benutzer manuell hinzugefügt werden. Zusätzlich kann der Admin alle Konfigurationen und ihre Mitglieder sehen. Er kann Mitglieder hinzufügen, entfernen oder sie auf readonly setzen. Zu dem kann er auch Konfigurationen löschen.

5.6 FbxList

In diesem Bereich können Benutzer 3D-Modelle hinzufügen, entfernen und anzeigen. Das Hinzufügen neuer Modelle erfordert eine FBX-Datei, ein gebräuchliches Format für den Austausch von 3D-Modellen. Die Verwaltung von 3D-Modellen ist beschränkt auf befugte Personen. Wenn ein 3D-Modell hinzugefügt wird, steht es für die Verwendung in einer Konfiguration zur Verfügung.

5.7 TextureList

In dieser Seite werden Texturen verwaltetet, einschließlich der Optionen zum Löschen, Hinzufügen und Benennen von Texturen. Nach dem Hochladen einer Textur steht sie automatisch im PropertyContainer zur Verfügung. Für den Upload einer Textur werden fünf Dateien benötigt.

1. **Substance-Graph-AmbientOcclusion.jpg:** Die Umgebungsokklusions-Textur. Sie simuliert abgedunkelte Bereiche, die aufgrund der Nähe von Objekten oder Oberflächen auftreten.
2. **Substance-Graph-BaseColor.jpg:** Die Basiskolor-Textur. Sie definiert die Grundfarbe des Modells.

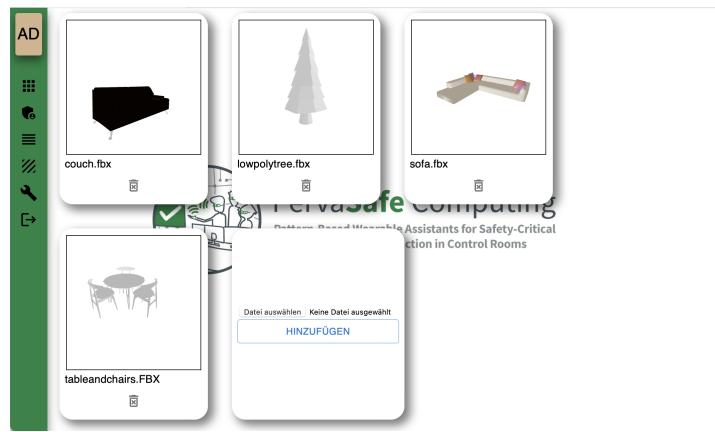


Abbildung 5.15: TreeView

3. **Substance-Graph-Height.jpg:** Die Höhen- oder Bump-Textur. Sie beeinflusst die Oberflächenhöhen des Modells, um eine erhabene oder eingedrückte Struktur zu erzeugen.
4. **Substance-Graph-Normal.jpg:** Die Normalen-Textur. Sie erzeugt Details in der Oberflächenstruktur, indem sie die Normalenrichtungen auf subtile Weise verändert.
5. **Substance-Graph-Roughness.jpg:** Die Rauheits-Textur. Sie steuert, wie stark das Licht auf der Oberfläche gestreut wird, was die Rauheit oder Glätte der Oberfläche beeinflusst.

Nachdem die Dateien für den Upload ausgewählt wurden, erfolgt eine Überprüfung, ob exakt fünf Dateien ausgewählt wurden und ob sie jeweils die korrekten Bezeichnungen tragen. Nachdem die korrekten Dateien erfolgreich ausgewählt wurden, wird ein Name für die Textur festgelegt. Anhand dieses Namens erfolgt der Upload der Dateien auf den Server.

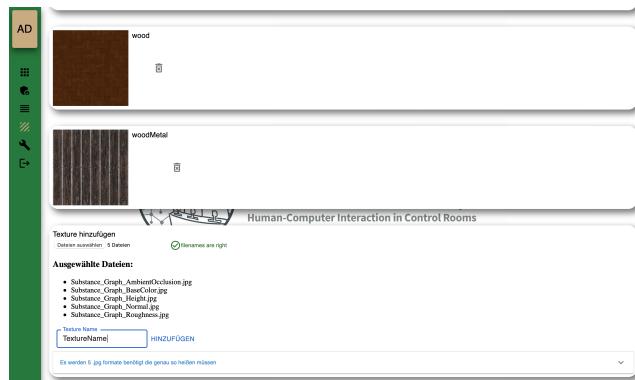


Abbildung 5.16: Texturen

5.8 Settings

Hier können verschiedene Einstellungen geändert werden.

Literaturverzeichnis

- Fac23. FACEBOOK, INC.: *React - A JavaScript library for building user interfaces.* <https://reactjs.org/>, 2023.
- Pri23. PRISMA LABS: *Prisma - Database toolkit for TypeScript and Node.js.* <https://www.prisma.io/>, 2023.
- Soc23. SOCKET.IO CONTRIBUTORS: *Socket.IO - Real-time communication library.* <https://socket.io/>, 2023.
- Thr23a. THREE.JS CONTRIBUTORS: *Three.js - JavaScript 3D library.* <https://threejs.org/>, 2023.
- Thr23b. THREE.JS CONTRIBUTORS: *Three.js Fiber - Reactive 3D library.* <https://docs.pmnd.rs/react-three-fiber/getting-started/introduction>, 2023.
- Ver23. VERCEL: *Next.js - The React Framework.* <https://nextjs.org/>, 2023.

A

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