POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Sviluppo di un'Applicazione Web per la Visualizzazione e la Gestione Interattiva di Modelli 3D

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Autore: **Elia Zito**

Matricola: 224584
Relatore: Prof. Luciano Baresi
Anno Accademico: 2023-24

Abstract

This work presents the development of a Web Application for the visualization and interactive management of 3D models, designed in response to a request from the World Health Organization (WHO), with the specific objective of providing a digital tool that can assist field operators in understanding the construction, organization and maintenance of a basic camp, which plays a crucial role in preventing the spread of infectious diseases in emergency scenarios and humanitarian interventions. The application, which has been built using React and integrates an advanced interactive 3D viewer, allows users to explore and interact with a virtual representation of the camp, offering the possibility to click on different structures, such as tents, bathrooms, and common areas, to access detailed information regarding their dimensions, functions, and specific sanitation protocols that must be followed to ensure compliance with health and safety regulations. The backend of the application has been developed using Firebase, which enables real-time updates, seamless data storage, and efficient content management, ensuring that all information remains accessible, synchronized, and up to date, even when multiple users interact with the system simultaneously. Beyond describing the functionalities of the application, this work also examines the technical challenges encountered during its development, particularly those related to the management of complex 3D models, the optimization of performance to ensure smooth rendering in a web-based environment, and the structuring of data to allow for efficient retrieval and manipulation. Furthermore, the modular and extensible nature of the application makes it adaptable to future developments and improvements, allowing the integration of additional features and optimizations based on evolving requirements and technological advancements. Using modern web technologies and combining 3D visualization with an interactive and user-friendly interface, this project demonstrates how digital tools can be used effectively in educational and operational contexts, offering a practical and innovative solution to support field operators.

Keywords: Web Application, 3D Visualization, React, Firebase, Interactive Management, 3D Model, Web Technologies, Raycasting

Abstract in italiano

Questo elaborato presenta lo sviluppo diun'Applicazione Web per la visualizzazione e la gestione interattiva di modelli 3D, progettata in risposta a una richiesta dell'Organizzazione Mondiale della Sanità (OMS), con l'obiettivo specifico di fornire uno strumento digitale in grado di supportare gli operatori sul campo nella comprensione della costruzione, dell'organizzazione e della manutenzione di un accampamento di base, il quale riveste un ruolo cruciale nella prevenzione della diffusione di malattie infettive in scenari di emergenza e interventi umanitari. L'applicazione, sviluppata utilizzando React e integrando un avanzato visualizzatore 3D interattivo, consente agli utenti di esplorare e interagire con una rappresentazione virtuale dell'accampamento. Cliccando sulle diverse strutture, come tende, bagni e aree comuni, è possibile accedere a informazioni dettagliate riguardanti le loro dimensioni, funzioni e specifici protocolli di sanificazione da seguire per garantire il rispetto delle normative igienico-sanitarie. Il backend dell'applicazione è stato realizzato utilizzando Firebase, che permette aggiornamenti in tempo reale, un'archiviazione dei dati fluida e una gestione efficiente dei contenuti, assicurando che tutte le informazioni rimangano accessibili, sincronizzate e sempre aggiornate, anche quando più utenti interagiscono con il sistema contemporaneamente. Oltre a descrivere le funzionalità dell'applicazione, questo lavoro analizza anche le sfide tecniche affrontate durante lo sviluppo, in particolare quelle relative alla gestione di modelli 3D complessi, all'ottimizzazione delle prestazioni per garantire un rendering fluido in un ambiente web e alla strutturazione dei dati per consentire un recupero e una manipolazione efficienti. Inoltre, la natura modulare ed estensibile dell'applicazione la rende adattabile a futuri sviluppi e miglioramenti, permettendo l'integrazione di funzionalità aggiuntive e ottimizzazioni in base all'evoluzione dei requisiti e dei progressi tecnologici. Sfruttando le moderne tecnologie web e combinando la visualizzazione 3D con un'interfaccia interattiva e intuitiva, questo progetto dimostra come gli strumenti digitali possano essere utilizzati in modo efficace in contesti educativi e operativi, offrendo una soluzione pratica e innovativa a supporto degli operatori sul campo.

Parole chiave: Applicazione Web, Visualizzazione 3D, Modello 3D, React, Firebase, Tecnologie Web, Raycasting, Gestione Interattiva

Indice

Introduzione

In contesti di emergenza sanitaria, come nel caso delle epidemie, uno degli aspetti più complessi e critici è la costruzione e la gestione degli accampamenti temporanei, spazi che devono ospitare personale medico, operatori e persone vulnerabili, come i malati. In quest'ottica gli accampamenti devono essere configurati in modo tale da rispondere a molteplici necessità, affinché non diventino focolai di malattie infettive, evitando che la scarsità di risorse o l'improvvisazione possano compromettere la salute degli occupanti e degli operatori.

La progettazione e gestione dell'accampamento coinvolge numerosi aspetti tecnici e logistici: dall'adeguata disposizione degli spazi alla sanificazione, dalla costruzione delle tende per il personale e i malati alla gestione di luoghi comuni. Ogni componente, dalle strutture di accoglienza fino agli impianti di trattamento dei rifiuti, deve essere pensata per garantire il massimo livello di sicurezza, efficienza e igiene.

É quindi fondamentale che gli operatori sul campo abbiano una conoscenza approfondita delle modalità di costruzione, gestione e sanificazione di ogni singolo elemento dell'accampamento.

La formazione tradizionale degli operatori sul campo, sebbene rappresenti un approccio consolidato e utile in circostanze ordinarie, rivela evidenti limitazioni nelle situazioni di emergenza: in contesti in cui i tempi a disposizione sono estremamente ristretti e le risorse a disposizione sono limitate, le metodologie convenzionali, come i manuali cartacei o le sessioni di formazione teorica, non risultano sufficienti a preparare adeguatamente il personale ad affrontare le difficoltà pratiche e imprevedibili che si presentano sul campo. La necessità di soluzioni formative più rapide, facilmente fruibili e altamente adattabili alla realtà dell'emergenza diventa dunque ancora più impellente, rendendo indispensabile l'adozione di strumenti tecnologici innovativi che possano colmare queste lacune e fornire un supporto immediato ed efficace agli operatori.

In risposta a questa crescente esigenza di soluzioni più efficaci e immediate, il progetto nasce con l'obiettivo di sviluppare uno strumento innovativo per la formazione e la gestione in tempo reale degli accampamenti. L'applicazione web interattiva sviluppata sfrutta le potenzialità delle tecnologie per la visualizzazione e l'interazione con modelli

3D, superando così i limiti delle metodologie tradizionali, permettendo agli operatori sul campo di esplorare in modo immersivo il modello 3D di un accampamento, interagendo direttamente con i singoli componenti.
Cliccando su ciascun elemento, l'utente ha la possibilità di esplorare informazioni dettagliate relative a ciascun componente dell'accampamento, che possono includere linee guida per la costruzione, la manutenzione, la gestione e la sanificazione dei vari spazi e strutture. Queste informazioni sono presentate in tempo reale, permettendo agli operatori di accedere a contenuti specifici al momento stesso in cui ne hanno bisogno, senza necessità di consultare manuali separati o risorse esterne. L'approccio utilizzato, immediato e contestualizzato, offre un supporto visivo e interattivo che facilita la comprensione e l'applicazione delle corrette procedure operative, ottimizzando così l'efficacia dell'intervento in situazioni di emergenza.

Struttura della tesi

La presente tesi è articolata in diversi capitoli, ognuno dei quali ha lo scopo di approfondire un aspetto chiave del progetto, dalle motivazioni alla realizzazione tecnica, fino ai possibili sviluppi futuri.
Nel primo capitolo vengono introdotte le tecnologie web fondamentali e le motivazioni che hanno portato alla scelta di sviluppare una Web App per affrontare le problematiche e raggiungere gli obiettivi sopra descritti. In questa sezione vengono analizzate le principali tecnologie utilizzate, tra cui React per la costruzione dell'interfaccia utente, Three.js per la visualizzazione 3D e Firebase per la gestione dei dati in tempo reale.
Nel secondo capitolo si entra nel dettaglio dell'implementazione del progetto, analizzando il codice delle componenti principali, le funzionalità sviluppate e le soluzioni adottate per superare le problematiche incontrate durante lo sviluppo.
Infine, nel terzo capitolo, vengono tratte le conclusioni del lavoro svolto, delineando possibili migliorie e sviluppi futuri.

1 | Fondamenti di Tecnologie Web

Alla luce degli obiettivi del progetto, che mirano a fornire uno strumento innovativo e facilmente accessibile per la formazione e la gestione di accampamenti sanitari in situazioni di emergenza, la scelta di sviluppare un'applicazione web si è rivelata la più adatta, in quanto offre una serie di vantaggi in termini di accessibilità, facilità di aggiornamento e distribuzione.
In primo luogo, una applicazione web non richiede l'installazione su dispositivi locali, il che la rende immediatamente fruibile da qualsiasi dispositivo connesso a Internet. Questo rappresenta un vantaggio significativo, poiché in situazioni di emergenza la possibilità di distribuire rapidamente l'applicazione agli operatori sul campo è cruciale. Inoltre, le Web App sono accessibili da vari dispositivi, dai computer desktop agli smartphone, garantendo una maggiore versatilità nelle operazioni quotidiane. Un altro aspetto fondamentale riguarda la facilità di aggiornamento: in scenari dinamici come quelli trattati nel progetto, è essenziale che le informazioni siano sempre aggiornate e che l'applicazione possa evolversi rapidamente in risposta a nuove esigenze o miglioramenti. L'adozione di una Web App consente, inoltre, una gestione centralizzata dei dati e delle risorse, risultando particolarmente utile per monitorare l'andamento delle operazioni in tempo reale, archiviare le informazioni e fornire supporto continuo agli utenti. Questo approccio si integra perfettamente con le tecnologie moderne, come React e Firebase, che permettono di realizzare interfacce reattive e altamente interattive. React, con la sua capacità di costruire componenti modulari e dinamici, e Firebase, con la gestione in tempo reale dei dati, garantiscono una fruizione ottimale del modello 3D e delle informazioni associate, assicurando un'alta reattività, scalabilità e una gestione efficiente dei dati, fondamentali per il successo di applicazioni interattive complesse come quella sviluppata nel progetto.
Per la realizzazione di questa Web App interattiva, sono state impiegate diverse tecnologie avanzate che hanno reso possibile la creazione di un'esperienza fluida e dinamica per gli utenti, in grado di visualizzare e interagire con un modello 3D. L'approccio scelto ha permesso di integrare il rendering di modelli 3D con funzionalità interattive, la gestione in tempo reale dei dati e una fruizione ottimale delle informazioni.
Le tecnologie utilizzate in questo progetto comprendono:

- React, una libreria JavaScript per la creazione di interfacce utente reattive;
- Three.js, una libreria per il rendering di grafica 3D nel browser;
- Raycasting, una tecnica fondamentale per l'interazione con gli oggetti 3D;
- Firebase, una piattaforma per la gestione dei dati in tempo reale e la sincronizzazione tra gli utenti.

In questo capitolo, verranno trattate nel dettaglio queste tecnologie, esplorando come ognuna di esse contribuisca a garantire la funzionalità dell'applicazione e la realizzazione di un'esperienza utente efficace e interattiva. In particolare, verranno approfonditi i motivi della scelta di ciascuna tecnologia, le relative funzionalità principali e il ruolo specifico nel progetto.

1.1. Applicazione Web

Un'applicazione Web, o Web App, è un software accessibile e fruibile tramite Internet attraverso un browser che non necessita un'installazione locale sul dispositivo dal quale si ha l'intenzione di accedervi.
Le applicazioni Web seguono ciò che è definita come architettura client-server dove, essenzialmente, il client non è altro che il dispositivo di chi vuole fruire la Web App, mentre il server è il sistema che ospita l'applicazione, gestisce i dati e risponde alle richieste del client.
Il codice delle Web App è diviso in due componenti principali:

- Script lato client: scritto in linguaggi come HTML, CSS e JavaScript, è responsabile della gestione dell'interfaccia utente (UI), del rendering grafico e dell'interattività. Il browser carica lo script lato client ed esegue il rendering degli elementi grafici e del testo per consentire l'interazione dell'utente.
- Script lato server: si occupa dell'elaborazione dei dati e della gestione delle richieste lato client. Nello specifico, le richieste includono il recupero dei dati da un database, l'autenticazione degli utenti e la logica di business dell'applicativo.

Grazie a questa architettura, le Web App offrono numerosi vantaggi, su tutti l'accessibilità universale e la scalabilità, il che la rendono la scelta migliore per realizzare il progetto in questione.

La creazione di applicazioni Web, indipendentemente dal framework scelto per lo sviluppo, coinvolge necessariamente tre linguaggi fondamentali: HTML per la struttura, CSS per la stilizzazione e JavaScript per la logica applicativa:

- HTML (HyperText Markup Language): linguaggio di markup che struttura e organizza il contenuto di una pagina web.
- CSS (Cascading Style Sheets): linguaggio per definire lo stile e il design di una pagina web, inclusi colori, layout, font e posizionamento degli elementi.
- JavaScript: linguaggio di programmazione utilizzato principalmente per creare e gestire le interazioni nelle pagine web; eseguito lato client, permette la manipolazione dinamica del contenuto della pagina, rispondere agli eventi e molto altro.

1.2. React

L'applicazione web sviluppata in questo progetto adotta un'architettura di Single Page Application (SPA), un modello che permette di offrire un'esperienza utente più fluida e interattiva; a differenza delle tradizionali web app, in cui ogni interazione richiede un nuovo caricamento, una SPA gestisce le interazioni in modo dinamico, aggiornando esclusivamente le parti dell'UI coinvolte nell'interazione, senza la necessità di ricaricare l'intera pagina. Questo si traduce in tempi di risposta più rapidi e in una navigazione più fluida. L'utilizzo di React è fondamentale in questo contesto, poichè consente di strutturare l'applicazione in componenti riutilizzabili, semplificando la gestione dello stato e l'aggiornamento in tempo reale dell'UI.
React è una delle librerie JavaScript più popolari e ampiamente utilizzate per la costruzione di interfacce utente; creata e mantenuta da Meta, React si distingue per il suo approccio dichiarativo e la gestione efficiente del Virtual DOM, che consente di ridurre il numero di operazioni sul DOM reale, migliorando significativamente le prestazioni dell'applicazione.
Il concetto centrale di React è che l'interfaccia utente è costituita da una serie di componenti, unità autonome che gestiscono il proprio stato e la logica, rendendo l'applicazione modulare.

1.2.1. Il Virtual DOM di React

Il DOM (Document Object Model) è una rappresentazione ad albero di una pagina web in cui ogni elemento HTML rappresenta un nodo; ogni volta che si verifica una modifica del documento, il DOM viene aggiornato per rappresentare tale modifica. Ad ogni interazione il browser deve quindi ricostruire e ridisegnare l'interfaccia e, in contesti in cui il numero di elementi interattivi risulta elevato, la necessità di aggiornare ogni volta il DOM può risultare inefficiente, causando rallentamenti e compromettendo l'esperienza di navigazione.
Il Virtual DOM di React è una rappresentazione virtuale, una copia leggera e in-memory del DOM effettivo, che nasce con l'esigenza di migliorare l'efficienza e ottimizzare le prestazioni delle applicazioni web, permettendo di minimizzare il numero di aggiornamenti effettuati dal browser.
Questa rappresentazione virtuale consente a React di fare una serie di operazioni in memoria, senza dover interagire direttamente con il DOM reale, riducendo così il numero di operazioni di rendering costose e migliorando le performance complessive dell'applicazione: quando lo stato di un componente cambia, React non aggiorna immediatamente il DOM reale ma applica la modifica al Virtual DOM, mantenendo in memoria la sua precedente versione.
A seguito della creazione di un nuovo albero avviene il confronto con la versione precedente, vengono apportate le effettive modifiche al rendering dell'applicazione agendo solo alla fine. Questo processo, noto come reconciliation, permette a React di minimizzare le modifiche reali al DOM, applicando solo le modifiche necessarie e riducendo così al minimo i costi in termini di prestazioni.

1.2.2. Componenti in React

Per comprendere appieno React, è fondamentale approfondire il concetto di componente, che rappresenta l'elemento cardine della sua architettura.
Un componente in React può essere visto come un vero e proprio "mattoncino" dell'applicazione: ciascun componente è un'unità autonoma che gestisce la propria logica, stato e rendering, incapsulando una specifica funzionalità o una porzione dell'interfaccia utente. Una volta creato, il componente può essere riutilizzato più volte all'interno della stessa applicazione, senza doverlo ridefinire ogni volta, promuovendo così la modularità e la riusabilità del codice e rendendo lo sviluppo delle applicazioni più efficiente.
I componenti giocano un ruolo centrale nello sviluppo di applicazioni React-based perché

è grazie alle combinazioni tra componenti che possiamo realizzare interfacce utente complesse ed altamente dinamiche.
Per comprendere meglio come un componente React funzioni, consideriamo l'esempio di un semplice contatore che gestisce uno stato interno e reagisce agli eventi dell'utente. In questo caso, il componente mostrerà un contatore che può essere incrementato ogni volta che l'utente preme un pulsante.

Codice 1.1: Componente Contatore in React

```
import React, {useState} from 'react';
function Counter() {
    const [count,setCount] = useState(0);
    const increment = () => {
        setCount(count+1);
    };
    return (
        <div>
            <h2> Contatore: {count} </h2>
            <button onClick={increment}> Incrementa </button>
        </div>
    );
}
export default Counter;
```

Nel Codice 1.1, il componente `Counter` gestisce uno stato interno `count` il cui valore è incrementato tramite la funzione `increment`, triggerata ad ogni click sul bottone "Incrementa".
Nell'esempio, il componente `Counter` è definito in un file separato e viene esportato tramite la parola chiave `export`; una volta esportato, può essere importato in un altro file, dove verrà utilizzato come parte dell'interfaccia utente, come mostrato nel Codice 1.2.

Codice 1.2: Codice per l'utilizzo del contatore

```
import React from 'react';
import Counter from './Counter';
function App() {
    return (
        <div>
            <h1>Benvenuto nell'app di esempio!</h1>
            <Counter />
        </div>
    );
}
export default App;
```

Come mostrato nel Codice 1.2, per utilizzare il componente Contatore, basterà utilizzare il tag `<Counter/>` utilizzando il nome definito nell' `export` del Codice 1.1; React si occuperà di eseguire il rendering del componente e delle sue interazioni con l'interfaccia utente.

1.2.3. Integrazione di Librerie Esterne in React

Una delle forze di React è la sua flessibilità e la possibilità di integrarsi facilmente con una vasta gamma di librerie esterne e proprio grazie alla sua architettura basata su componenti, React permette di estendere le sue funzionalità utilizzando diverse librerie per affrontare esigenze specifiche, come la visualizzazione di contenuti 3D, la gestione di dati in tempo reale, o l'implementazione di interazioni complesse.
In questo progetto, l'integrazione di librerie esterne è stata fondamentale per raggiungere gli obiettivi prefissati, in particolare, sono state scelte librerie come Three.js per il rendering e la manipolazione dei modelli 3D, Raycasting per l'interazione con gli oggetti 3D, e Firebase per la gestione dei dati in tempo reale.
Questa sezione esplorerà come ciascuna di queste librerie è stata integrata nell'applicazione React, evidenziando i vantaggi che ogni libreria ha portato in termini di prestazioni e funzionalità, e di come hanno contribuito a creare una soluzione robusta e scalabile per la visualizzazione interattiva dei modelli 3D.

1.2.4. Three.js: Visualizzazione 3D sul Web

Nel contesto della realizzazione di un'applicazione web che integra un visualizzatore 3D interattivo, è fondamentale disporre di strumenti capaci di gestire la grafica tridimensionale direttamente nel browser in modo efficiente e performante, senza imporre un'eccessiva complessità nello sviluppo. WebGL rappresenta la tecnologia di base che consente di renderizzare grafica 3D ma, alla base della complessità nell'effettuare un'implementazione diretta, che richiede una conoscenza approfondita delle pipeline grafiche e della gestione delle risorse a basso livello, è stata adottata Three.js, una libreria JavaScript che fornisce un'astrazione di alto livello su WebGL, permettendo di ottenere risultati avanzati con un codice più leggibile e mantenibile.
Three.js offre numerose funzionalità per la gestione della grafica tridimensionale, tra cui il caricamento, la visualizzazione e la manipolazione di modelli 3D in formati come GLTF/GLB, garantendo un'ottima resa visiva e supportando interazioni avanzate, come il raycasting e l'uso di animazioni. Tuttavia, poiché l'intera applicazione è sviluppata in React, si è reso necessario integrare Three.js in un ambiente basato su componenti, mantenendo la modularità e la riusabilità tipiche di questo framework.

Three.js è la libreria principale per la gestione della grafica 3D. Deve essere installata nel progetto per poter creare scene, caricare modelli e gestire luci e materiali. Il Codice 1.3 rappresenta il comando da terminale per installare la libreria Three.js

Codice 1.3: Comando da terminale per l'installazione di Three.js

```
npm install three
```

Per utilizzare Three.js in un ambiente React, la prima operazione da eseguire è l'installazione della libreria e di react-three-fiber, che permette di scrivere scene 3D in modo dichiarativo all'interno di componenti React (Codice 1.4)

Codice 1.4: Comando da terminale per l'installazione di react-three-fiber

```
npm install three @react-three/fiber
```

Per migliorare la gestione della scena 3D e ridurre la complessità del codice, oltre a @react-three/fiber, è stato necessario l'utilizzo di @react-three/drei, una libreria che fornisce componenti aggiuntivi utili per gestire illuminazione, controlli della telecamera, caricamento di modelli e altro ancora.

Per installare la libreria @react-three/drei è necessario inserire da terminale il Codice 1.5.

Codice 1.5: Comando da terminale per l'installazione di react-three-drei

```
npm install three @react-three/drei
```

Una volta installate le librerie necessarie, è fondamentale comprendere il concetto di scena in Three.js, che rappresenta il "contenitore" dove avviene tutta la magia della grafica 3D. In Three.js la scena è un oggetto principale al cui interno sono presenti tutti gli elementi necessari per il rendering, come luci, oggetti 3D, telecamere e sfondi.
Nel caso di un'applicazione React esiste un componente <Canvas> di react-three/fiber che si occupa del rendering dell'ambiente 3D; al suo interno si potranno poi definire gli oggetti, le luci, le telecamere e le animazioni che faranno parte della scena 3D.
Nel contesto di questo progetto, l'obiettivo principale è il caricamento e la visualizzazione di un modello 3D in formato .glb, che rappresenta una versione binaria del formato .gltf, ideale per la trasmissione e la visualizzazione di modelli 3D all'interno di applicazioni web grazie alla sua compattezza e velocità di caricamento.
In react-three/fiber, l'importazione di un modello .glb può essere facilmente gestita utilizzando l'hook `useGLTF` di @react-three/drei che è progettato per semplificare il processo di caricamento e gestione dei modelli 3D in formato .gltf o .glb all'interno di componenti React. La funzione `useGLTF` restituisce l'intera scena del modello, che può essere inserita direttamente nella scena di react-three/fiber senza necessità di ulteriore elaborazione.

Nel Codice 1.6 `useGLTF` viene utilizzato per caricare il modello in formato .glb direttamente da un percorso specifico.

Codice 1.6: Codice per l'utilizzo di useGLTF

```
import React from 'react';
import { Canvas } from '@react-three/fiber';
import { useGLTF } from '@react-three/drei';
const Model = ({ modelPath }) => {
    const { scene } = useGLTF(modelPath);
    return <primitive object={scene} scale={0.5} />;
};
const App = () =>
  return (
    <Canvas>
        <ambientLight intensity={0.5} />
        <spotLight position={[10,10,101}angle={0.15}penumbra={1}/>
        <Model modelPath="/path/to/your/model.glb" />
    </Canvas>
  );
};
export default App;
```

1.2.5. Raycasting

Il raycasting è una tecnica fondamentale per l'interazione con oggetti 3D; come suggerito dal nome, consiste nel tracciare un "raggio" virtuale da una sorgente, come ad esempio la posizione del cursore o del click dell'utente, e determinare gli oggetti con cui questo raggio interseca la scena 3D.
Il raggio viene "sparato" attraverso la scena 3D e, se incrocia un oggetto, viene registrato il punto di collisione, il quale verrà poi utilizzato per attivare eventi come la selezione di un oggetto o l'apertura di un'informazione.
In Three.js, il raycasting può essere facilmente integrato con la libreria, grazie al modulo Raycaster di Three.js. Nel Codice 1.7, il Raycaster viene utilizzato per proiettare un raggio sulla scena; ogni volta che il mouse si sposta, viene calcolata la posizione del mouse nella scena 3D utilizzando le coordinate 2D normalizzate. Una volta ottenuto il raggio si utilizza il metodo `intersectObject` per verificare se il raggio entra in contatto con l'oggetto, che in questo caso è un semplice cubo. Attraverso la variabile clicked viene stabilito se un oggetto è stato cliccato o meno, cambiando di conseguenza il proprio colore.

Codice 1.7: Codice Raycasting

```
import { useRef, useState } from 'react';
import { useThree, useFrame } from '@react-three/fiber';
import { Raycaster, Vector2 } from 'three';
function InteractiveObject (props) [
  const [clicked, setClicked] = useState(false);
  const meshRef = useRef ();
  const { camera, mouse } = useThree();

  const raycaster = new Raycaster();
  useFrame (( = {
    const mouseVector = new Vector2((mouse.x / window.innerWidth)
       * 2 - 1, -(mouse-y / window.innerHeight) * 2 + 1);
    raycaster. update(camera, mouseVector);
    const intersects = raycaster.intersectobject(meshRef.current);
    if (intersects. length > 0) {

        if (!clicked) {
            setClicked (true);

        }
    } else {
        if (clicked) {
            setClicked (false);
        }
    }
  }) ;
 return (
    <mesh {...props} ref={meshRef}>
        <boxGeometry args={[1, 1, 11} />
        <meshStandardMaterial color={clicked ? 'green' : 'red'} />
    </mesh>
);
}
```

1.2.6. OrbitControls in Three.js

OrbitControls è una libreria di Three.js che consente di controllare la visualizzazione del modello 3D da parte dell'utente, aggiungendo funzionalità di interazione alla telecamera, permettendo di ruotare, zoomare e spostarsi all'interno della scena 3D.
L'integrazione di OrbitControls in React è semplificata dall'uso della libreria @react-three/drei che fornisce direttamente il componente <OrbitControls /> compatibile con react-three/fiber. Tramite questa libreria, è sufficiente aggiungere il componente alla scena per attivare il controllo completo sulla telecamera, senza dover gestire manualmente la logica di movimento.

1.2.7. Firebase

Firebase è una piattaforma open source, ma supportata da Google, per lo sviluppo di applicazioni mobili e web, progettata per semplificare il processo di sviluppo e gestione delle applicazioni offrendo una serie di strumenti e servizi backend. Firebase è particolarmente utile per lo sviluppo di applicazioni cross-platform, ovvero quelle che possono essere eseguite su diverse piattaforme, come Android, iOS e Web.
Nel contesto del progetto, Firebase risulta particolarmente utile grazie alla sua capacità di gestire in modo efficiente il Realtime Database, il quale permette di utilizzare dati sincronizzati in tempo reale, aspetto fondamentale nella creazione di un'applicazione interattiva. Firebase, attraverso il suo Realtime Database, consente all'utente di avere un'esperienza fluida, dove ogni modifica, come l'aggiunta o l'aggiornamento di informazioni realtive agli oggetti presenti nel modello 3D, avviene in tempo reale.

2 | Implementazione del progetto

Nei precedenti capitoli sono state analizzate le tecnologie utilizzate e l'architettura dell'applicazione, delineando il contesto in cui è stato sviluppato il progetto e giustificando le scelte effettuate sulla base degli obiettivi prefissati.
In questa sezione verrà approfondita l'implementazione pratica dell'applicazione, illustrando nel dettaglio le soluzioni adottate per integrare e far interagire tra loro le diverse tecnologie descritte nei capitoli precedenti. Attraverso l'analisi del codice sorgente, la presentazione di screenshot dell'interfaccia e la spiegazione delle logiche di funzionamento, verranno mostrati i passaggi chiave che hanno portato alla realizzazione del visualizzatore interattivo con l'obiettivo di fornire una visione chiara e concreta del processo di sviluppo.

2.1. Architettura dell'applicazione

L'architettura dell'applicazione sviluppata per la visualizzazione interattiva del modello 3D di un accampamento si basa su una struttura modulare e scalabile, costruita con le tecnologie riportate per capitolo precedente che garantiscono una gestione fluida e performante.
L'architettura si basa su un modello client-server in cui il frontend si occupa della visualizzazione e dell'interazione in tempo reale con il modello 3D, mentre il backend si occupa della gestione dei dati. In questo scenario le due componenti si integrano strettamente per garantire che i dati siano aggiornati in tempo reale e visualizzati correttamente. Il frontend dell'applicazione è stato sviluppato utilizzando React, un framework JavaScript molto popolare, scelto principalmente per le sue capacità avanzate nella gestione dello stato e delle interazioni utente. React permette di costruire interfacce utente dinamiche e reattive in modo efficiente, consentendo la separazione della logica di presentazione e quella di gestione dello stato tramite componenti. Questo approccio modulare e dichiarativo risulta particolarmente utile quando si sviluppano applicazioni complesse, come quella in oggetto, che richiedono un'interazione continua con il modello 3D e un aggiornamento costante delle informazioni visualizzate.
Uno degli aspetti principali che ha reso React ideale per questo progetto è la sua abilità

di aggiornare selettivamente e in modo efficiente il DOM (Document Object Model) senza dover ricaricare l'intera pagina, consentendo una maggiore reattività e velocità nell'interfaccia utente, particolarmente importante per applicazioni interattive, dove ogni modifica o interazione deve essere visibile immediatamente, come nel caso della visualizzazione di un modello 3D in tempo reale.
Inoltre, React offre una grande flessibilità nell'integrazione con librerie esterne, come Three.js, una libreria JavaScript per la creazione e visualizzazione di grafica 3D, garantendo una combinazione potente per realizzare esperienze interattive 3D complesse, come quella di un modello di accampamento, senza dover ricorrere a tecnologie più pesanti o complesse.
Questa sinergia tra React e Three.js permette non solo di creare la scena 3D, ma anche di gestire in tempo reale le interazioni degli utenti, come il raycasting per selezionare gli oggetti nel modello o la navigazione attraverso la scena con OrbitControls. Il flusso di dati tra i componenti React e il modello 3D viene gestito in modo fluido, assicurando una visualizzazione interattiva e dinamica. D'altra parte, il backend dell'applicazione è stato progettato per gestire in modo efficace i dati relativi agli oggetti 3D e fornire un sistema scalabile per l'archiviazione e la sincronizzazione dei dati tra gli utenti.
I dati relativi agli oggetti 3D, alle informazioni associate e ad altre configurazioni vengono memorizzati nel Realtime Database di Firebase, che è facilmente accessibile dal frontend React. Grazie alla libreria Firebase SDK, i dati vengono sincronizzati automaticamente in tempo reale tra il client e il database e, quando un utente interagisce con il modello 3D, ad esempio cliccando su un oggetto, l'applicazione può aggiornare il database per registrare il cambiamento e notificare altri utenti in tempo reale.

Come mostrato in Figura 2.1, l'organizzazione delle cartelle del progetto segue una struttura modulare e chiara in cui le diverse aree funzionali sono separate per garantire la manutenibilità e la scalabilità dell'applicazione.

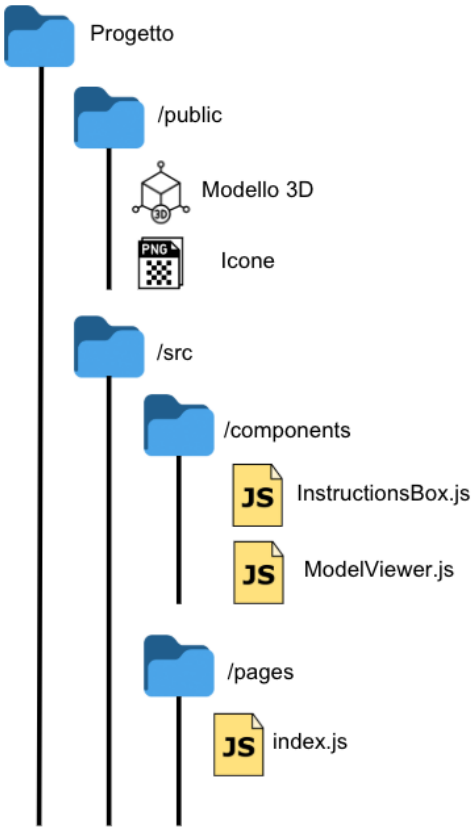

Figura 2.1: Struttura delle cartelle del progetto

- /**public**: questa cartella contiene risorse statiche come il modello 3D `ebola.glb` e tutte le icone utilizzate nel progetto.
- /**src**: all'interno di questa cartella risiedono i file sorgente del progetto, suddivisi in ulteriori sotto-cartelle.
  - /**components**: qui vengono archiviati i componenti React, tra cui:
    - `ModelViewer.js`, che si occupa della visualizzazione del modello 3D.
    - `InstructionsBox.js`, il componente che racchiude le istruzioni per l'utente.
  - /**pages**: la cartella che contiene i file delle pagine dell'applicazione. Trattandosi di una Single Page Application, l'unico file presente è:
    - `index.js`: il punto di ingresso principale della Web App e gestisce il rendering del visualizzatore 3D interattivo, integrando l'interfaccia utente e le funzionalità di navigazione.

Nei prossimi paragrafi verranno analizzate le sezioni principali del codice di ogni file, con particolare attenzione alle porzioni più rilevanti, al fine di fornire una comprensione approfondita delle soluzioni adottate e delle tecnologie utilizzate per realizzare il progetto.

2.2. Componente ModelViewer.js

Il compito principale di ModelViewer.js è quello di gestire e visualizzare il modello 3D (in formato .glb) in un'area dedicata dell'applicazione e, per fare ciò, utilizza principalmente la libreria three.js integrata con react-three/fiber.

2.2.1. Integrazione con Firebase

Attraverso l'integrazione con Firebase, è possibile gestire dinamicamente le informazioni relative agli oggetti 3D nel modello e sincronizzare in tempo reale i dati tra client e server, permettendo agli utenti di interagire con il modello 3D e visualizzare informazioni dettagliate su ciascun oggetto.

Codice 2.1: Configurazione Firebase

```
const firebaseConfig = {
apikey: "TUO_API_KEY" ,
authDomain: "TUO_AUTH_DOMAIN",
databaseURL: "TUO_DATABASE_URL" ,
projectId: "TUO_PROJECT_ID",
storageBucket: "TUO_STORAGE_BUCKET" ,
messagingSenderId: "TUO_MESSAGING_SENDER_ID"
appId: "TUO_APP_ID",
measurementId: "TUO_MEASUREMENT_ID",
};

const app = initializeApp(firebaseConfig);

const database = getDatabase(app) ;
```

Il Codice 2.1 contiene il necessario per la configurazione di Firebase, ovvero le credenziali necessarie per il progetto, ottenibili direttamente dalla console di Firebase:

- `apiKey`: la chiave API univoca che consente all'applicazione di interagire con Firebase, generata dalla console di Firebase e usata per autenticare le richieste verso i server di Firebase.
- `authDomain`: il dominio di autenticazione, utilizzato per configurare FirebaseAuthentication, utile per implementare la gestione degli utenti.
- `databaseURL`: l'URL del database Firebase, il riferimento al database utilizzato per leggere e scrivere i dati in tempo reale.
- `projectId`: il nome univoco del progetto su Firebase.
- `storageBucket`: l'URL del bucket di storage Firebase, utilizzato per caricare e memorizzare file.
- `messagingSenderID`: l'ID del mittente utilizzato per inviare notifiche push tramite Firebase Cloud Messaging.
- `appID`: l'ID univoco della tua applicazione, utile a identificare l'applicazione quando interagisce con i vari servizi di Firebase.
- `measurement`: l'ID di misurazione per Google Analytics, se la tua applicazione è configurata per raccogliere dati analitici tramite Firebase Analytics.

L'integrazione di Firebase consente il collegamento a un database in tempo reale, attraverso il quale è possibile gestire dinamicamente le informazioni associate agli oggetti del modello 3D. All'interno dell'applicazione sono state implementate diverse funzioni per l'inserimento, l'eliminazione e la visualizzazione dei dati, permettendo agli utenti di interagire direttamente con il modello per modificarne o consultarne il contenuto. Tuttavia, trattandosi di operazioni di base per la gestione dei dati in un database, il relativo codice non verrà riportato in dettaglio in questo elaborato.
Le operazioni di inserimento, modifica ed eliminazione dei dati sono state implementate in modo tale da poter essere eseguite direttamente dall'interfaccia utente, senza richiedere competenze specifiche in ambito di programmazione, rendendo così l'applicazione accessibile anche a utenti non esperti. Ad ogni modo, prima di illustrare nel dettaglio queste funzionalità, è fondamentale descrivere l'implementazione delle tecnologie utilizzate per gestire le interazioni con il modello 3D.

2.3. Visualizzazione ed interazione col modello 3D: Raycasting e OrbitControls

Come spiegato nel precedente capitolo, la scelta di utilizzare il Raycasting o l'OrbitControls è determinata dalla necessità di offrire un'esperienza di interazione fluida ed efficiace con il modello 3D. Il Raycasting permette di rilevare con precisione gli elementi del modello 3D soggetti all'interazione dell'utente, tramite un semplice click, mentre OrbitControls ha il compito di garantire una navigazione semplice ed intuitiva, permettendo di ruotare, gestire lo zoom e spostarsi all'interno del modello 3D, facilitando dunque l'espolarazione dell'accampamento virtuale.
Il Codice 2.2 mostra la modalità di importazione del modello 3D da visualizzare.

Codice 2.2: Importazione modello 3D

```
function Model () {
    const { scene } = useGLTF ('/ebola.glb');
    return <primitive object={scene} />;
}
```

In questo caso, la funzione `Model` è un componente React che utilizza `useGLTF` per caricare un modello 3D dal file `ebola.glb`, situato all'interno della cartella public del progetto, il cui percorso viene specificato direttamente nella chiamata a `useGLTF ('/ebola.glb')`. Il modello viene poi restituito come un elemento `primitive`, che inserisce l'oggetto `scene` nella scena 3D, rendendolo visibile.
Il prossimo passo consiste nell'implementare un sistema di interazione basato sull'uso del Raycaster, il quale permetterà di rilevare e gestire correttamente le interazioni con gli oggetti del modello 3D.

Il Codice 2.3 mostra il contenuto della funzione `RaycastHandler`, responsabile della gestione delle interazioni.

Codice 2.3: Codice di RaycastHandler

```
function RaycastHandler({ onObjectClick }) {
    const {camera, scene, gl} = useThree ();
    const raycaster = useRef (new THREE.Raycaster());
    const pointer = useRef(new THREE.Vector2());
    const initialPointerPosition = useRef ({x: 0, y: 0 });
    const threshold = 10;

    const handlePointerDown = (event) => {....
    };

    const handlePointerClick = (event) => {...
    };
    useEffect (() => {..
    },[]);
return null;
}
```

L'implementazione prevede l'utilizzo di un componente dedicato, denominato `RaycastHandler`, che gestisce l'interazione e si occupa di determinare se un clic corrisponde effettivamente alla selezione di un oggetto. Per garantire un'esperienza utente più precisa ed evitare selezioni accidentali, il sistema è stato progettato con un meccanismo di filtraggio: quando l'utente preme il tasto del mouse, la posizione iniziale viene memorizzata, e al rilascio viene calcolata la differenza con la posizione finale e, solamente se il movimento rilevato è inferiore a una soglia predefinita (10 pixel), il click viene considerato valido; in caso contrario, viene interpretato come un trascinamento e ignorato.
Per evitare interferenze con l'interfaccia utente, il sistema esclude automaticamente dalla gestione del Raycaster i click effettuati su pulsanti, campi di input o elementi popup. Questo accorgimento è fondamentale per garantire che l'interazione con il modello 3D non vada in conflitto con le altre funzionalità dell'applicazione. La posizione del puntatore del mouse è gestita attraverso la variabile pointer, che viene aggiornata in base alle coordinate dello schermo e trasformata in un sistema di riferimento normalizzato compreso tra -1 e 1. Questo è un passaggio essenziale per il corretto funzionamento del Raycaster, in quanto permette di allineare la posizione del puntatore con il sistema di coordinate della scena 3D.

Se il clic viene riconosciuto come valido, il Raycaster viene attivato per calcolare la direzione del raggio rispetto alla telecamera e verificare se uno degli oggetti presenti nella scena è stato intercettato. Nel caso in cui l'intersezione abbia successo, viene identificato l'oggetto selezionato e il suo nome viene passato a una funzione di gestione degli eventi, che può essere utilizzata per mostrare informazioni contestuali o attivare azioni specifiche. Questa soluzione offre un'interazione fluida ed efficace con il modello 3D, consentendo agli utenti di esplorare gli elementi del campo e ottenere dettagli aggiuntivi semplicemente cliccando sugli oggetti di interesse.

Nel Codice 2.4, la distinzione tra click effettivo, quindi valido, e il click effettuato con l'intenzione di muoversi nel modello, è gestita attraverso la definizione delle variabili `deltaX` e `deltaY`. Queste variabili calcolano la distanza tra la posizione iniziale del puntatore del mouse, memorizzata al momento in cui il tasto del mouse viene premuto, e la posizione attuale del mouse al termine del click. Ciò avviene tramite il confronto tra la posizione iniziale del mouse al momento del click, tramite `initialPointerPosition.current` e la posizione del mouse al momento del rilascio del click, con l'utilizo di `event.clickX` e `event.clickY`.

Codice 2.4: Normalizzazione coordinate

```
const deltaX = Math.abs (event.clientX - initialPointerPosition.
   current.x);
const deltaY = Math.abs (event.clientY - initialPointerPosition.
   current.y);

if (deltaX < threshold && deltay < threshold) {
    pointer.current.x = ((event.clientX - gl.domElement.
       getBoundingClientRect().left) / gl.domElement.clientWidth)
       * 2 - 1;

    pointer.current.y = -((event.clientY - gl.domElement.
       getBoundingClientRect().top) / gl.domElement.clientHeight)
       * 2 + 1;

    raycaster.current.setFromCamera(pointer.current, camera);
    const intersects = raycaster.current.intersectObjects(scene.
       children, true);
    if (intersects.length > 0) {
        const clickedObject = intersects [0].object;
        console.log('Clicked object: ${clickedObject.name}');
        onObjectClick(clickedObject.name);
    } else {
        console.log('No object clicked.');
        onObjectClick(null);
    }
}
```

A questo punto, per gestire correttamente l'interazione del mouse con la scena 3D, è necessario mappare le coordinate del mouse in un sistema di coordinate compatibile con

Three.js, cioè che vanno da -1 a 1 su entrambi gli assi X e Y, con il centro della finestra che corrisponde al punto (0, 0). Per ottenere la normalizzazione delle coordinate, le posizioni del mouse in pixel, ottenute tramite `event.clickX` ed `event.clickY`, vengono prima trasformate in valori compresi tra 0 e 1, che rappresentano la posizione del mouse all'interno della finestra di rendering del modello 3D, successivamente moltiplicando per 2 e sottraendo 1, si ottiene il range di coordinate normalizzate tra –1 e 1.
Una volta calcolate le coordinate normalizzate del puntatore, viene utilizzato un raycaster per determinare se il raggio lanciato dalla fotocamera interseca qualche oggetto nella scena 3D. Questo avviene tramite il metodo `raycaster.setFromCamera()`, che imposta il raggio in base alle coordinate del puntatore e alla fotocamera. La funzione `intersectObjects()` viene poi chiamata per verificare se il raggio incroci uno o più oggetti presenti nella scena. Il risultato di questa funzione è un array chiamato `intersects`, che contiene tutti gli oggetti della scena che sono stati colpiti dal raggio.
Se il raggio intercetta uno o più oggetti, l'array `intersects` non sarà vuoto, e il primo oggetto nell'array (l'elemento con indice 0) sarà quello più vicino alla fotocamera, quindi quello effettivamente cliccato dall'utente.
Questo passaggio è essenziale per garantire che il sistema riconosca correttamente gli oggetti sulla scena e permetta di visualizzare o interagire con le informazioni associate a ciascun oggetto.
In aggiunta a questa interazione basata sul click, è stata implementata la funzionalità `OrbitControls`, che consente di navigare e esplorare il modello 3D. OrbitControls permette di ruotare, fare zoom e spostarsi all'interno della scena, offrendo così una visione completa e dinamica degli oggetti: con il tasto sinistro del mouse è possibile ruotare la visuale attorno al modello, mentre con il tasto destro del mouse si può spostarsi lungo gli assi, consentendo di esplorare il modello da diverse angolazioni. Infine, la rotella del mouse permette di eseguire lo zoom, avvicinando o allontanando la visuale per focalizzarsi su dettagli specifici.

Per supportare l'utente nell'utilizzo di questi controlli, è stata inclusa una guida visiva nell'interfaccia utente (UI), che illustra in modo chiaro come utilizzare ciascun strumento di navigazione. La guida, mostrata in Figura 2.2, è accessibile direttamente nell'UI e fornisce istruzioni su come interagire con il modello 3D.

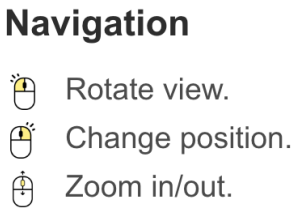

Figura 2.2: Istruzioni OrbitControls

2.4. Modalità di visualizzazione delle informazioni degli oggetti: i Popup

L'obiettivo principale del progetto riguardo all'interazione con il modello 3D consiste nel permettere all'utente di ottenere informazioni dettagliate su qualsiasi oggetto presente nel modello semplicemente cliccando su di esso. Le informazioni visibili devono comprendere: il nome dell'oggetto, le sue dimensioni e un collegamento a documenti PDF contenenti una descrizione approfondita delle pratiche di costruzione, manutenzione e sanificazione relative a ciascun elemento (tende, strutture comuni, etc.). L'intento è quello di offrire un'esperienza fluida e immediata, dove l'utente, esplorando il modello 3D, possa ottenere in tempo reale informazioni che lo aiutino a comprendere meglio la configurazione e la gestione dell'accampamento.
Tuttavia, la struttura del modello 3D fornito ha sollevato una sfida inaspettata: gli oggetti di quest'ultimo non sono stati realizzati come entità uniche, ma piuttosto come un insieme di mesh distinte, ognuna delle quali aveva un nome generico e poco informativo. Prendendo ad esempio una tenda, questa risulta composta da diverse mesh, una per ogni suo elemento compositivo: una per la finestra, una per la porta, una per la parete e così via. Questa struttura crea diversi problemi nella gestione delle informazioni da associare a ciascun oggetto, poiché il sistema, al click su di un elemento del modello 3D, riconosceva ogni volta oggetti distinti, senza riuscire ad identificare correttamente l'intero elemento come un'unica entità, complicando particolarmente l'integrazione con il database.
Per risolvere questa problematica, è stato necessario sviluppare una soluzione che permettesse di raggruppare le singole mesh che componevano un determinato oggetto in un'unica

entità logica. La decisione presa è stata quella di introdurre il concetto di "gruppo", ovvero una struttura che raccoglie tutte le mesh appartenenti alla stessa macroentità, consentendo così di trattarle come un unico oggetto. Ad esempio, nel caso della tenda, è stato creato un gruppo denominato "tenda", all'interno del quale sono state assegnate tutte le mesh che la compongono, come le pareti, il tetto, le finestre e la porta. Una volta completata questa operazione, il sistema riconosce automaticamente che tutte le mesh appartenenti a quel gruppo fanno parte della stessa entità e, in questo modo, quando l'utente clicca su una qualsiasi parte della tenda, il sistema mostra direttamente le informazioni relative all'intero gruppo, invece di identificare la singola mesh selezionata come un oggetto a sé stante.
Un ulteriore vantaggio di questa soluzione risiede nel fatto che non è stato necessario modificare la struttura del modello 3D originale, operazione che sarebbe risultata complessa e poco pratica. Al contrario, il sistema di gruppi è stato implementato a livello software, consentendo una gestione più efficiente e scalabile delle informazioni, garantendo gli stessi risultati indipendentemente dal modello 3D scelto.
Per implementare questa soluzione è stato necessario sviluppare un sistema che consentisse di definire e gestire i gruppi direttamente all'interno dell'applicazione.

Quando un utente clicca su un qualsiasi elemento del modello 3D non ancora presente nel realtime database di Firebase verrà mostrato il popup in Figura 2.3, responsabile dell'assegnazione dell'oggetto al gruppo di appartenenza, con la possibilità di sceglierne uno esistente o crearne uno da zero.

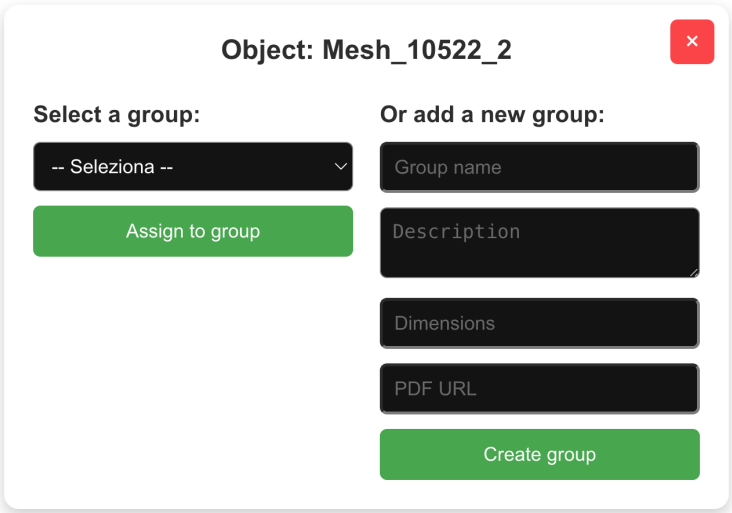

Figura 2.3: Popup per l'assegnazione ad un gruppo

Nella parte alta del popup viene mostrato il nome dell'oggetto, il quale, come precedentemente spiegato, è poco rappresentativo dell'unità selezionata.
La sezione sottostante, suddivisa in due parti principali, permette di assegnare l'oggetto ad un gruppo già creato selezionando uno tra quelli presenti nel database e cliccando il bottone per l'assegnazione. In alternativa, l'utente può decidere di creare un nuovo gruppo, inserendo tutte le informazioni necessarie e, cliccando sul bottone "Create group", dopo un check sui campi inseriti, il sistema crea una nuova istanza gruppo nel database alla quale viene automaticamente associato l'oggetto cliccato.
Nel caso in cui l'utente clicchi su una mesh già presente nel database, il sistema individua il gruppo a cui è associato e mostra le relative informazioni all'interno del popup mostrato in Figura 2.4.

Figura 2.4: Popup delle informazioni del gruppo

Il popup delle informazioni ha una struttura semplice che mostra, in ordine: il nome del gruppo, le dimensioni, una breve descrizione e il bottone che permette l'apertura di un file PDF esterno contenente tutte le informazioni dettagliate per la costruzione, il mantenimento e l'eventuale sanificazione della struttura.
Il bottone "Remove Object from Group", una volta cliccato, permette la rimozione dell'associazione dell'oggetto con il gruppo a cui è legato.
Il bottone in Figura 2.5 viene visualizzato nell'interfaccia solo nel caso in cui sono presenti dei gruppi all'interno del database. Cliccando su di esso si apre un popup che mostra la lista dei gruppi esistenti, consentendo all'utente di modificarne le informazioni associate o, se necessario, di eliminarli.

Figura 2.5: Bottone per la gestione gruppi

Il popup per la gestione gruppi (Figura 2.6) presenta, come anticipato, la lista dei gruppi esistenti, affiancati da due icone che permettono la modifica o l'eliminazione. Cliccando nell'icona del cestino, in rosso, il sistema procede con la rimozione del gruppo e la conseguente eliminazione a cascata delle associazioni di quest'ultimo con le varie mesh. L'icona gialla invece consente all'utente di modificare le informazioni relative ad un gruppo esistente tramite l'apertura di un popup dedicato, mostrato in Figura 2.7.

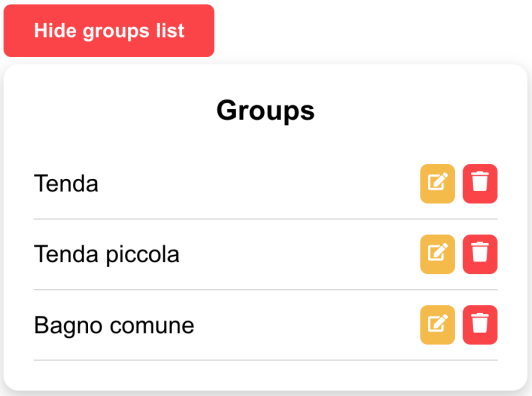

Figura 2.6: Popup per la gestione dei gruppo

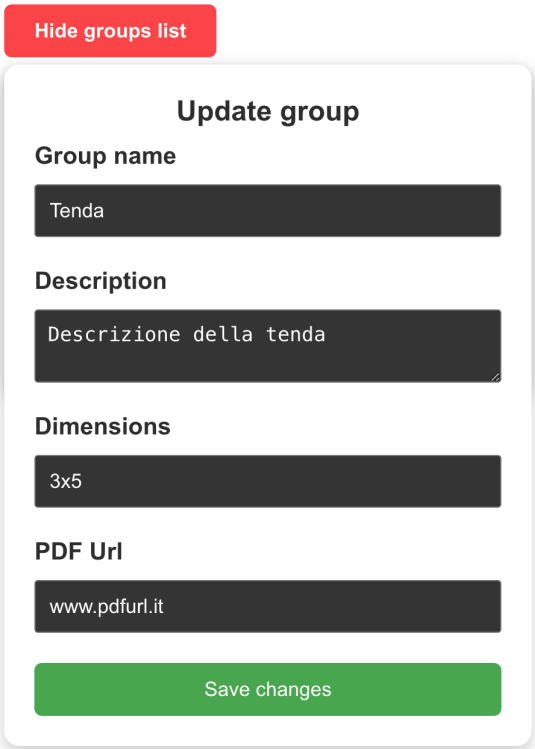

Figura 2.7: Popup per la modifica del gruppo

2.5. Risultati finali

In questo capitolo è stata analizzata l'implementazione dell'applicazione, partendo dalla sua architettura generale fino ai dettagli dei componenti chiave, approfondendo la gestione della visualizzazione 3D attraverso il componente ModelViewer.js, che rappresenta il cuore dell'interazione con il modello. Sono state inoltre descritte le principali funzionalità implementate, come l'integrazione con Firebase per la gestione dei dati, il sistema di Raycasting per l'interazione con gli oggetti del modello 3D oltre all'uso di OrbitControls per la navigazione.
Di seguito vengono illustrate alcune schermate dell'applicazione, accompagnate da una breve descrizione delle funzionalità mostrate.

Figura 2.8: Interfaccia applicazione

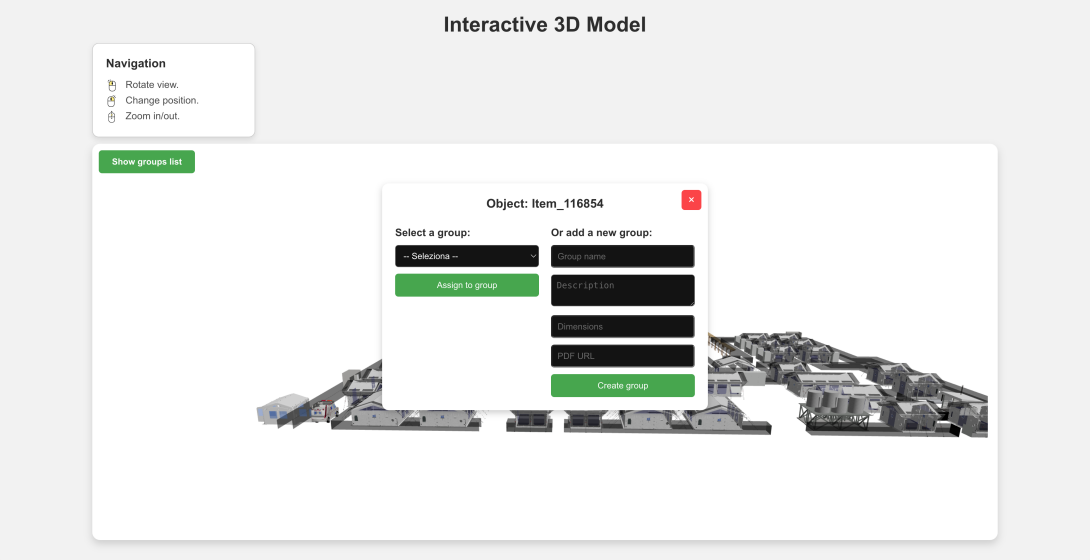

Figura 2.9: Assegnazione dell'oggetto ad un gruppo

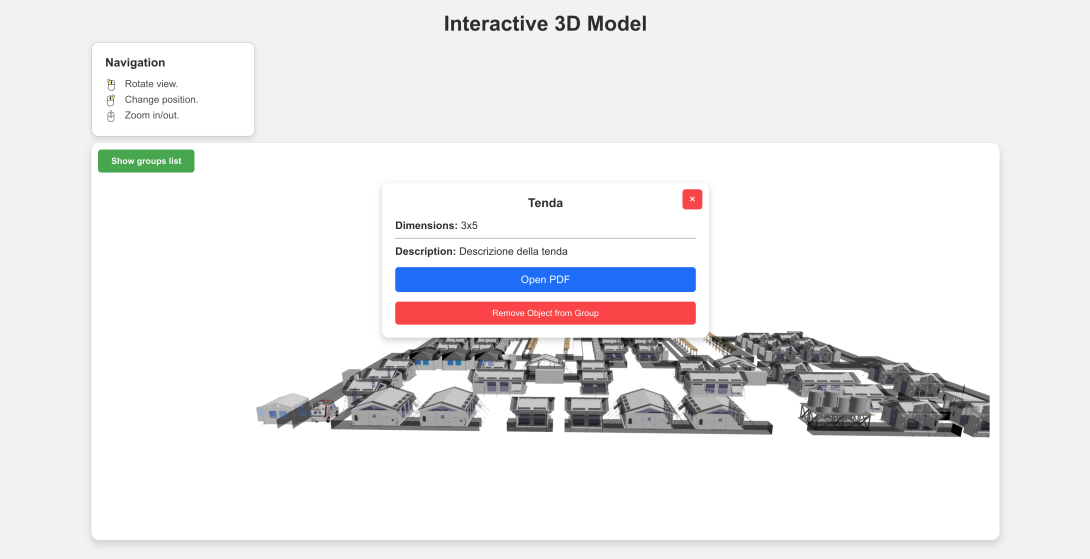

Figura 2.10: Visualizzazione informazioni relative ad un oggetto

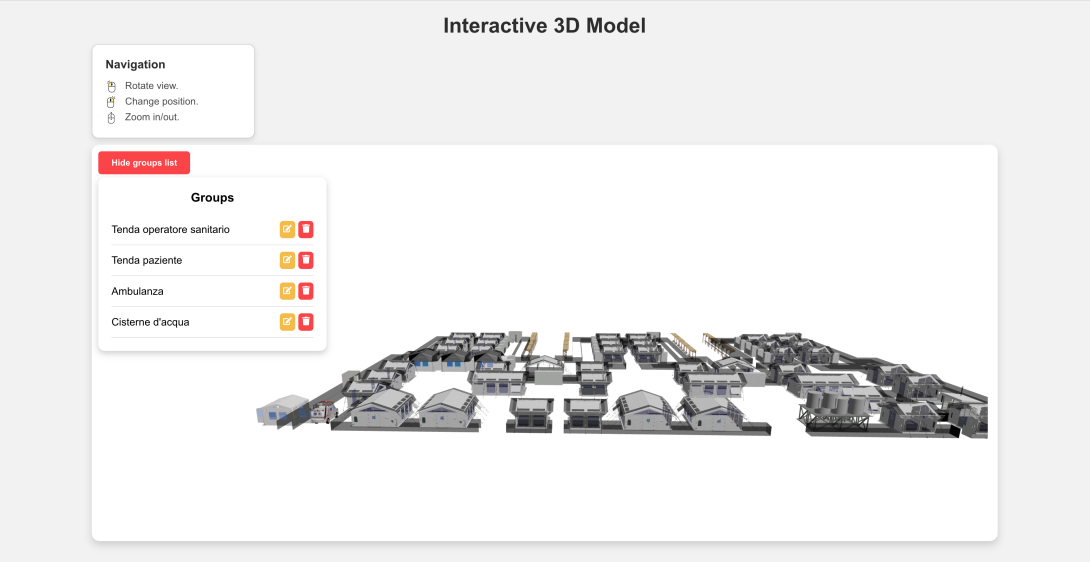

Figura 2.11: Gestione dei gruppi

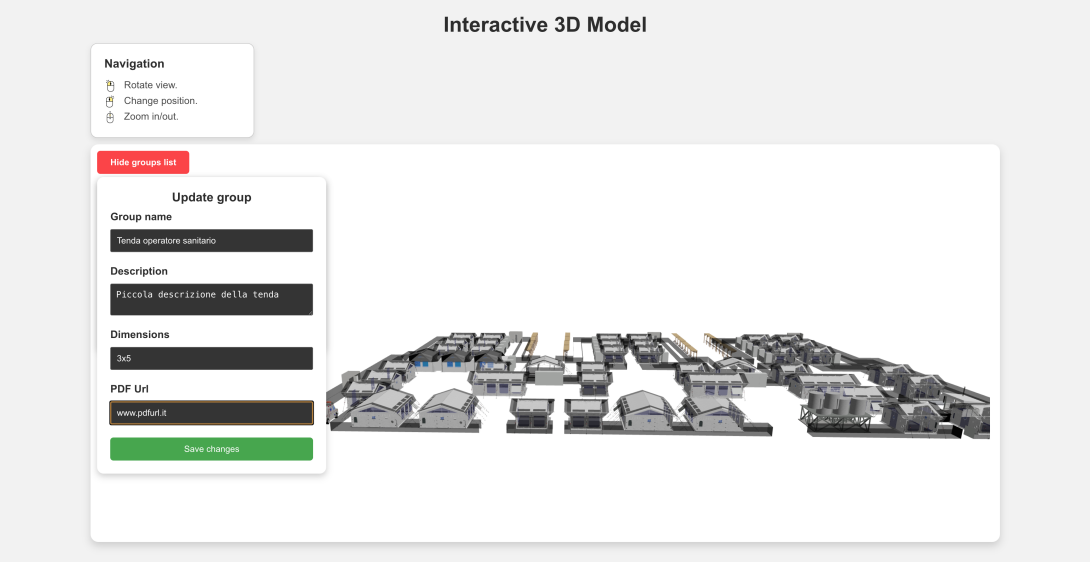

Figura 2.12: Modifica delle informazioni di un gruppo

3 | Conclusione e sviluppi futuri

L'applicazione sviluppata rappresenta una soluzione efficace e interattiva per la visualizzazione e la gestione di un modello 3D relativo a un accampamento sanitario. Grazie all'integrazione di Firebase e all'utilizzo di Raycasting il sistema consente agli utenti di esplorare il modello e di interagire con esso, ottenendo informazioni dettagliate sugli elementi presenti, come tende e strutture comuni.
Le principali funzionalità implementate includono:

- Interazione con il modello 3D: gli utenti possono cliccare su qualsiasi oggetto per visualizzarne le informazioni.
- Gestione dinamica dei gruppi: possibilità di creare, modificare ed eliminare gruppi di mesh direttamente dall'interfaccia utente, senza necessità di alcuna conoscenza nell'ambito della programmazione.
- Integrazione con Firebase: archiviazione e gestione delle informazioni in tempo reale, garantendo aggiornamenti immediati dei dati.

L'obiettivo principale del progetto, ovvero fornire uno strumento utile per istruire gli operatori sul campo riguardo alla costruzione e manutenzione dell'accampamento, è stato raggiunto.

3.1. Sviluppi futuri

Nonostante l'applicazione abbia raggiunto gli obiettivi prefissati, ci sono diverse aree in cui è possibile migliorare e ampliare le funzionalità. Di seguito sono riportati alcuni degli sviluppi futuri che potrebbero essere implementati:

1. Autenticazione: Una delle principali migliorie previste riguarda l'implementazione di un sistema di autenticazione per gli utenti. Questo permetterebbe di distinguere tra coloro che hanno il permesso di apportare modifiche alle informazioni mostrate nel modello 3D e coloro che hanno solo accesso in modalità visualizzazione. L'autenticazione garantirà una gestione più sicura e controllata dei dati, assicurando che solo gli utenti autorizzati possano modificare le informazioni relative agli oggetti, come la creazione di nuovi gruppi, la modifica di quelli esistenti o l'aggiunta di documenti informativi;

2. Evidenziazione degli oggetti selezionati nel modello 3D: Un'altra possibile estensione riguarda il miglioramento dell'interazione con il modello 3D. Quando un utente seleziona un gruppo, come ad esempio una tenda, sarebbe utile evidenziare visivamente tutti gli elementi che fanno parte di quel gruppo (finestra, porta, parete, etc.). Un cambiamento di colore o l'aggiunta di un bordo luminoso potrebbe rendere ancora più chiaro quali componenti appartengano ad un determinato gruppo, migliorando l'esperienza utente e facilitando la comprensione del modello 3D;

3. Miglioramenti estetici dell'interfaccia: Sebbene l'applicazione sia funzionale, c'è ancora spazio per miglioramenti estetici: un'attenzione maggiore alla cura del design e dell'usabilità dell'interfaccia potrebbe rendere l'esperienza più piacevole e intuitiva. Ad esempio, si potrebbero aggiungere animazioni per l'interazione con il modello 3D, migliorare l'aspetto grafico dei pulsanti e delle finestre di dialogo e ottimizzare la disposizione degli elementi per garantire una navigazione più chiara e fluida.

Questi sviluppi futuri non solo migliorerebbero la funzionalità dell'applicazione, ma contribuirebbero anche a rendere l'interazione con il modello 3D ancora più intuitiva e soddisfacente per gli utenti, offrendo un'esperienza più personalizzata e interattiva.

Bibliografia

[1] Firebase. Guida alla configurazione di firebase per il web, 2025. URL https://firebase.google.com/docs/web/setup?hl=it.

[2] Three.js. JavaScript 3D Library, 2025. URL https://threejs.org/.

[3] Three.js Documentation. Orbitcontrols, 2025. URL https://threejs.org/docs/#examples/en/controls/OrbitControls.

[4] Three.js Documentation. Raycaster, 2025. URL https://threejs.org/docs/#api/en/core/Raycaster.

Elenco delle figure

Ringraziamenti

Desidero esprimere la mia sincera gratitudine a tutte le persone che hanno contribuito alla realizzazione di questa tesi e che mi hanno supportato durante il mio percorso accademico.

Innanzitutto, un sentito ringraziamento al mio relatore, il prof. Baresi, per la disponibilità mostrata, per il supporto continuo e per i preziosi consigli forniti nel corso di questo lavoro. La sua guida è stata fondamentale per la buona riuscita di questo progetto.

Un grazie speciale va all'Organizzazione Mondiale della Sanità (OMS), che mi ha dato l'opportunità di lavorare su un progetto di grande valore sociale e sanitario che, spero, possa contribuire a migliorare la gestione delle emergenze sanitarie.

Infine, voglio ringraziare la mia famiglia e le persone a me care per il loro sostegno incondizionato e per la pazienza dimostrata durante tutto il mio percorso. Il loro affetto e incoraggiamento sono stati un motore fondamentale per raggiungere questo traguardo.

A tutti voi, il mio più sentito grazie.