UNIVERSITAT DE BARCELONA

GRAU D'ENGINYERIA INFORMÀTICA

Treball Final de Grau

Customizable and Performant 3D Portfolio

Autor: Joan Badia i Andreu

Director: Dr. Ricardo Marques

Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, June 10, 2024

Acknowledgements

I would like to extend my heartfelt appreciation to the Faculty of Computer Science at the Universitat de Barcelona (UB) for providing the academic framework necessary to carry out this project. I would also like to thank my advisor, Ricardo Marques, for his exceptional supervision and support, as well as his regular and consistent evaluations of my progress, which have been invaluable. Furthermore, I must acknowledge the unconditional support of my family and friends, who have stood by me throughout this academic journey. Their support has been essential to my motivation and the realization of this work.

Abstract

Historically, portfolios have served to present oneself to potential clients by showcasing skills, abilities, and background. This traditional method of self-presentation has evolved with the advent of digital technology. In today's competitive online environment, the need to display one's professional trajectory has never been more critical and more accessible. Recruiters and potential clients delve deep into these digital representations for assessments and reassurance. As such, an online portfolio has become a crucial component of one's professional journey.

This project offers a glimpse into a not-so-distant future in which everyone might have a personal website to present themselves to the world. Currently, the most prevalent forms of such personal representation are work and social network profiles. However, these platforms offer limited customization and control over one's online image. While dedicated website profiling is not yet commonplace, the increasing desire for personalized online presence indicates a clear market for such technologies.

The final application aims to provide an edge over the numerous visually static 2D portfolios available today. It achieves this by developing a 3D immersive experience, allowing users to represent themselves, customize their environment, and visualize relevant information in a dynamic, interactive format. This product is designed for users seeking a more realistic and personalized online presence that leaves a lasting impression, transcending the flat, static presentation of traditional online portfolios.

In the final phase of the project, we will present the results of this application, showcasing a final product that meets the specified requirements. The product's effectiveness will be evaluated through individual user satisfaction tests and performance benchmarks. To ensure its performance, the product will be tested in two distinct environments. This will allow us to determine its limits and complexity across both devices, ensuring a comprehensive analysis of the user experience.

Resum

Històricament, els portafolis han servit per presentar-se a clients potencials exhibint habilitats, capacitats i antecedents. Aquest mètode tradicional d'autopresentació ha evolucionat amb el progrés de la tecnologia digital. En l'entorn competitiu a dia d'avui, la necessitat de mostrar la trajectòria professional mai ha estat tan crítica ni tan accessible. Els reclutadors i els clients potencials s'endinsen profundament en aquestes representacions digitals per avaluacions i garanties. Així doncs, un portafoli en línia s'ha convertit en un component crucial del viatge professional dels treballadors moderns.

Aquest projecte ofereix una visió d'un futur no gaire llunyà en què tothom podria tenir una pàgina web personal per presentar-se al món. Actualment, les formes més prevalents de representació personal són els perfils laborals i de xarxes socials. No obstant això, aquestes plataformes ofereixen una personalització i un control limitats sobre la imatge en línia d'una persona. Tot i que la creació de llocs web dedicats encara no és habitual, la creixent demanda de presència en línia personalitzable indica un clar mercat per a aquestes tecnologies.

L'aplicació final té com a objectiu proporcionar una avantatge sobre els nombrosos portafolis 2D visualment estàtics disponibles avui dia. Ho aconsegueix desenvolupant una experiència immersiva en 3D, permetent als usuaris representar-se a si mateixos, personalitzar el seu entorn i visualitzar informació rellevant en un format dinàmic i interactiu. Aquest producte està dissenyat per a usuaris que busquen una presència en línia més realista i personalitzada que deixi una impressió duradora, transcendint la presentació plana i estàtica dels portafolis en línia tradicionals.

En la fase final del projecte, presentarem els resultats d'aquesta aplicació, mostrant un producte final que compleixi els requisits especificats. L'eficàcia del producte es valorarà mitjançant proves de satisfacció de l'usuari individual i indicadors de rendiment. Per assegurar-ne el bon funcionament, el producte es provarà en dos entorns diferents. Això ens permetrà determinar els seus límits i complexitat en ambdós dispositius, assegurant un anàlisi exhaustiu de l'experiència de l'usuari.

Resumen

Históricamente, los portafolios han servido para presentarse a posibles clientes mostrando las habilidades, capacidades y antecedentes. Este método tradicional de autopresentación ha evolucionado con la llegada de la tecnología digital. En el tan competitivo entorno en línea actual, la necesidad de mostrar la propia trayectoria profesional nunca ha sido más crítica ni más accesible. Los reclutadores y los clientes potenciales profundizan en estas representaciones digitales en busca de valoraciones y garantías. Por ello, un portafolio en línea se ha convertido en un componente crucial de la trayectoria profesional de cada uno.

Este proyecto ofrece una visión de un futuro no tan lejano en el que todo el mundo podría tener un sitio web personal para presentarse al mundo. En la actualidad, las formas más extendidas de este tipo de representación personal son el trabajo y los perfiles en redes sociales. Sin embargo, estas plataformas ofrecen una personalización y un control limitados sobre la propia imagen. Aunque la creación de perfiles web específicos aún no es habitual, el creciente deseo de personalizar la presencia en línea indica la existencia de un mercado claro para este tipo de tecnologías.

La aplicación final pretende ofrecer una ventaja sobre los numerosos portafolios 2D visualmente estáticos disponibles en la actualidad. Lo consigue desarrollando una experiencia inmersiva en 3D, que permite a los usuarios representarse a sí mismos, personalizar su entorno y visualizar información relevante en un formato dinámico e interactivo. Este producto está diseñado para usuarios que buscan una presencia en línea más realista y personalizada que deje una impresión duradera. Trascendiendo así la presentación plana y estática de los portafolios en línea tradicionales.

En la fase final del proyecto, presentaremos los resultados de esta aplicación, mostrando un producto final que cumpla los requisitos especificados. La eficacia del producto se evaluará mediante pruebas individuales de satisfacción de los usuarios y parámetros de rendimiento. Para garantizar su rendimiento, el producto se probará en dos entornos distintos. Esto nos permitirá determinar sus límites y complejidad en ambos dispositivos, garantizando un análisis exhaustivo de la experiencia del usuario.

Contents

1 Introduction

A portfolio is a curated collection of an individual's work, intended to showcase their skills, talents, and accomplishments. It can be used in various fields such as art, design, photography, architecture, education, and software engineering. Historically, the concept of a portfolio has its roots in the art world. Artists, like painters and sculptors, would carry a portfolio of their works to show to potential patrons, clients, or galleries. This allowed them to demonstrate their style, skill, and range of abilities in a tangible way.

In today's digital age, numerous professionals significantly benefit from having an online presence, which may include social media profiles, work portfolios, and personalized showcases of their work. There is an abundance of templates and integrated user portfolios available in various applications, leading to a highly saturated market. In such an environment, individual profiles can easily be overshadowed and may struggle to leave a lasting impression on viewers.

Developing these website portfolios requires both knowledge and vision. As programmers, we often encounter these types of portfolios and are encouraged to create our own. However, there is a distinct knowledge barrier that can deter others who wish to follow the same path.

It would also be very beneficial for users to have an outlet where to plasmate their work, somewhere engaging. Normative webistes are very static in this aspect and the users work has to be relevant by itself since every webpage has the same alure. This limited customization options can be a significant hurdle for many, preventing them from fully expressing their vision.

From this thought process, it was identified as an opportunity to create an application that fills the gap left by the existing platforms. This product would aim to empower individuals from diverse backgrounds to showcase their work in the competitive online landscape.

1.1 Motivation

To adequately introduce the problem at hand, it is also essential to begin with a comprehensive examination of the initial motivations behind addressing this particular issue.

This motivation originates from the desire to bridge the gap between traditional portfolio platforms and a dynamic, interactive environment. By allowing users to interact with their portfolios in a first-person manner, we aim to create a more personal and engaging experience. This approach enables individuals to showcase their work as if it were being experienced in firsthand.

Drawing from acquired experiences as programmers, we understand the challenges faced by users in navigating through existing platforms. This solution looks to facilitate the portfolio creation process, offering intuitive tools and features to enable users to stand out in the competition.

From this deliberation the concept of a real-time customizable 3D scene website presents a unique

opportunity to meet this need. Therefore, it is decided to proceed with the development of an application that enables users to create such product.

1.2 Personal Motivation

From a young age, I was captivated by browser games and their nature as self-contained interactive website applications. This early fascination played a role in guiding my academic and professional pursuits, being one of the factors that led me to study software engineering with a specialization in front-end development. When the time came to select a topic for my Final Degree Project (TFG), I was certain it should align with my professional path and childhood interests.

Currently, I am employed as a developer working on a website that generates customized templates for clients through a back-office interface. This experience inspired me to explore the concept of real-time customization, enabling users to observe changes to parameters instantly. This provides a more precise and interactive experience than traditional page reloading.

With this initial concept in mind, I began to visualize the final application. I decided to develop a real-time, customizable 3D portfolio. This portfolio would not only serve a professional purpose, providing a unique platform to showcase work, but also fulfill a personal initiative.

1.3 Objectives

This section outlines the key objectives of the project, categorized into visual, front-end, back-end, and performance aspects.

Visual Objectives

The customizable 3D application should enable users to interact with a 3D environment through an interface. Users should be able adjust parameters such as camera configurations, shadow properties, and lighting. Users should also be able to add personal information as welll as some other customizable content within the 3D environment.

Furthermore, the application should include advanced functionalities such as environment customization, allowing users to change backgrounds for realistic lighting effects. The system should also incorporate interactive elements like clickable objects and animations to furthermore improve the user's immersiveness.

Front-end Objectives

React should be integrated into the system architecture to provide a layer of abstraction, in order to create a more maintainable and clean code, as a long-term goal. More specifically, by using React's component-based structure, developers create reusable components for different elements within the 3D environment.

With React Three Fiber we can design dynamic user interfaces that interact seamlessly with the 3D environment in real-time. By leveraging React's state management and event handling capabilities, we will create intuitive controls for users to customize the 3D scene on-the-fly. This improves the user experience by providing immediate feedback and enhancing immersion within the virtual environment.

Additionally user authentication should be implemented to enable secure access control and personalized data management. By implementing authentication protocols, we ensure that only authorized users can access the application. This feature also allows for personalized experiences by associating user preferences and customization settings with individual user accounts.

Back-end Objectives

Designing and implementing a back-end infrastructure involves developing efficient mechanisms to manage data transmission for our 3D scene and facilitate user authentication. This includes establishing protocols for storing user data and customized scene information in a database, as well as ensuring retrieval of this data. By leveraging advanced back-end technologies, we aim to create a high-performance system that meets these objectives.

Our focus is on providing quick and efficient data storage and retrieval, ensuring minimal latency in user authentication and data transmission translating to a consistent user experience for our 3D scene application.

Performance Objectives

Conducting a comprehensive performance analysis is essential to thoroughly assess the application's efficiency and responsiveness. This involves examining the customizable scene components and it's influence on the overall scene performance.

The optimization process should focus on enhancing the application's ability to handle real-time 3D graphics without experiencing significant delays or lag. Utilizing performance charting allows us to tailor the user experience to suit individual device requirements, ensuring consistent performance. While automation may not be employed, manual adjustment of performance charts can serve as valuable guidance for users, highlighting what consumes the most resources and enabling them to optimize their experience for their target audience.

1.4 Document Structure

This document's primary focus is on the project itself, with its structure revolving around the code behind the prototype. After reviewing the state of the art, we will analyze the main technologies and their associated benefits. These technologies form the foundation of the project and are consistently referenced throughout.

The core sections of the document include the design, implementation, and results, followed by conclusions. These sections provide a comprehensive overview of the project's lifecycle, from conceptualization to execution and evaluation.

The development and implementation of the web application are divided into front-end and back-end components. This division allows for a clear understanding of each part's role and how they integrate to form the complete application.

The first section details the basic implementation of the 3D scene, which is fully customizable. This section is organized according to the iterations and building progress of the elements within the scene, providing a clear view of the development process.

2 State of the Art

In this section, we will explore the main lines of research and development work similar to the present one, along with the tools used for their resolution. We will provide a detailed explanation of how each of the following projects has influenced the current one. In addition, there will be a detailed technical explanation of each of the fundamental relevant technologies.

2.1 3D Web Technologies

2.1.1 WebGL

WebGL operates as an interface to OpenGL ES. Architecturally, the browser exposes the WebGL API through JavaScript, which includes browser-supplied functions that wrap around OpenGL ES specifications. These specifications were intentionally kept at a low level to encourage re-implementations from the community, such as frameworks and automation. Without these, a significant amount of overhead would be spent on reinventing the wheel.

The graphics drivers provide the implementation of OpenGL and execute your code on the machine hardware. To synthesize the final image, the JavaScript code retrieves the 3D context from the HTML5 canvas elements, then registers a set of shaders, which are written in GLSL (OpenGL Shading Language).

The rest of the process involves the graphics pipeline, a GPU process that renders the 3D scene as seen in Figure 1. It starts with the application sending draw commands, including vertex data and shader programs, to the GPU. The vertex shader processes each vertex individually, transforming 3D positions into screen space and calculating lighting. Vertices are then assembled into primitives and clipped to the viewing frustum. The rasterizer converts primitives into fragments, interpolating vertex attributes across them. The fragment shader computes the final color of each fragment, involving texture mapping, lighting, and color.

Finally, the contents of the framebuffer are displayed on the HTML canvas element, resulting in the rendered image being shown in the browser.

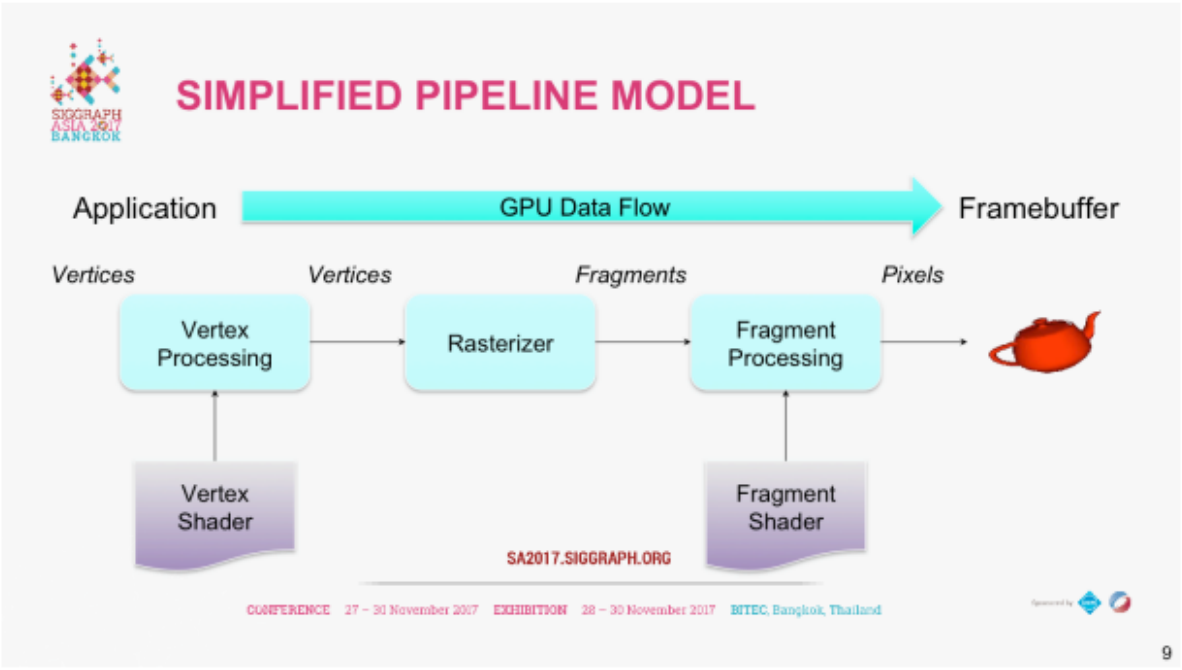


Figure 1: Simplified graphics pipeline model. Image from [1].

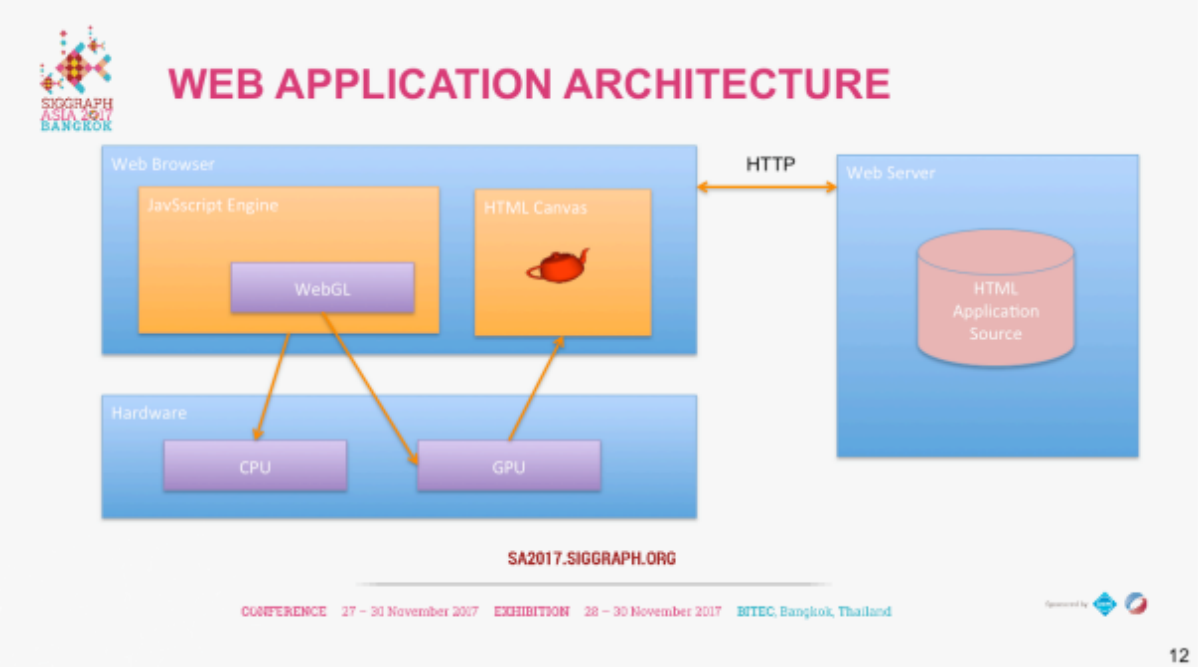


Figure 2: WebGL architecture: From the CPU where the application source code runs commands to WebGL, the rendered image is presented to an HTML canvas element in the web application. Image from [1].

2.1.2 Unity WebGL

Unity WebGL is a powerful technology that allows developers to create and deploy complex 3D and 2D games and interactive experiences directly to modern web browsers. It leverages the WebGL JavaScript API for rendering graphics in a web browser without the need for plugins.

Unity's core engine, is used to create these experiences, which are then exported to WebGL-compatible JavaScript code. This allows developers to take advantage of Unity's robust tools for asset management, physics, lighting, and animation, among others, while still delivering their content to a broad audience via the web.

Unity WebGL operates by running the Unity runtime within the web browser using JavaScript and WebAssembly. This allows the game code, written in C# to be executed directly in the browser, providing a seamless user experience.

One of the key features of Unity WebGL is its ability to handle complex 3D graphics in the browser. It does this by leveraging the power of the user's GPU, allowing for hardware-accelerated graphics and animations. This makes it possible to deliver high-quality, immersive experiences. In addition to its core features, Unity has a rich ecosystem of plugins and libraries that extend its functionality. With a very active community, it is a popular choice among game developers.

2.1.3 Comparison of Unity WebGL and WebGL

In this Subsection, we will delve into the key differences between Unity WebGL and WebGL with its dedicated libraries, focusing on various aspects relevant to application development. By analizing these factors, we aim to provide a thorough comparison to aid in determining the most suitable choice for our specific project requirements.

Initially, we are dealing with different programming languages. As a front-end developer, my main experience lies in working with the React framework and JavaScript. This contrasts with Unity, which primarily supports C# as its scripting language. When building for WebGL, Unity translates your C# scripts into JavaScript (or asm.js). This translation enables the scripts to run in a web browser environment. Therefore, the development experience and the language syntax between these two environments can be significantly different.

Thus, we are comparing a scripting language with a compiled language. For Unity WebGL, we would also require to install its game engine as a development environment. Although Unity offers a more comprehensive development environment, Three.js, being a JavaScript library, affords more control over the development process. In addition, as a JavaScript library, Three.js integrates seamlessly with the web ecosystem.

Regarding community support, Three.js attracts many new developers for web applications. A large and updated community simplifies the development process when encountering obstacles. Unity

also boasts a large community with abundant resources, forums, official tutorials, and documentation. However, its primary focus is game development, and finding specific solutions may be more challenging compared to libraries and frameworks designed explicitly for web development.

Three.js is also open-source and free of licensing costs, contrasting with Unity's privatized model and associated pricing issues. In terms of cross-browser compatibility, WebGL takes the lead, eliminating the need for cross-browser testing or compatibility fixes. Three.js is also mobile-friendly, while Unity WebGL lacks support.

Finally, when exporting the project in Unity, files tend to be significantly larger than in WebGL. Moreover, when programming in WebGL, you can adhere to web programming standards, including accessibility guidelines and SEO requirements (best practices). Unity WebGL might require further optimization to meet these standards.

Considering factors such as file size, added complexity, language experience, cross-code compilation, optimization, cost, and compatibility, choosing Three.js over Unity WebGL appears justified for our specific needs and preferences.

2.1.4 Three.js

Three.js is a cross-browser JavaScript library and Application Programming Interface (API) used to create and display animated 3D computer graphics in a web browser. It uses WebGL to render graphics and provides a multitude of features to create rich, interactive 3D content.

Three.js situates itself right above WebGL and drastically simplifies the process of development. It removes the handling and writing of specific shaders and vertices without removing the accessibility to low-level WebGL from the pipeline.

The downsides of not using WebGL directly are that it comes with the implementation of specific functionalities that might not work for the project, it is also not the most efficient at data storage, and it needs to be downloaded to work with it locally. However, in exchange, it enormously simplifies the development process as we program at a higher level. One example where Three.js facilitates the development process is by providing abstractions and helper functions. Abstractions allow developers to work with higher-level concepts rather than dealing with low-level details, which can make coding more efficient, robust and less error-prone. The following Figure 3 illustrates what the Three.js Library abstracts, if one would decide to avoid this library, it would be imperative to reprogram their own similar to Three.js.

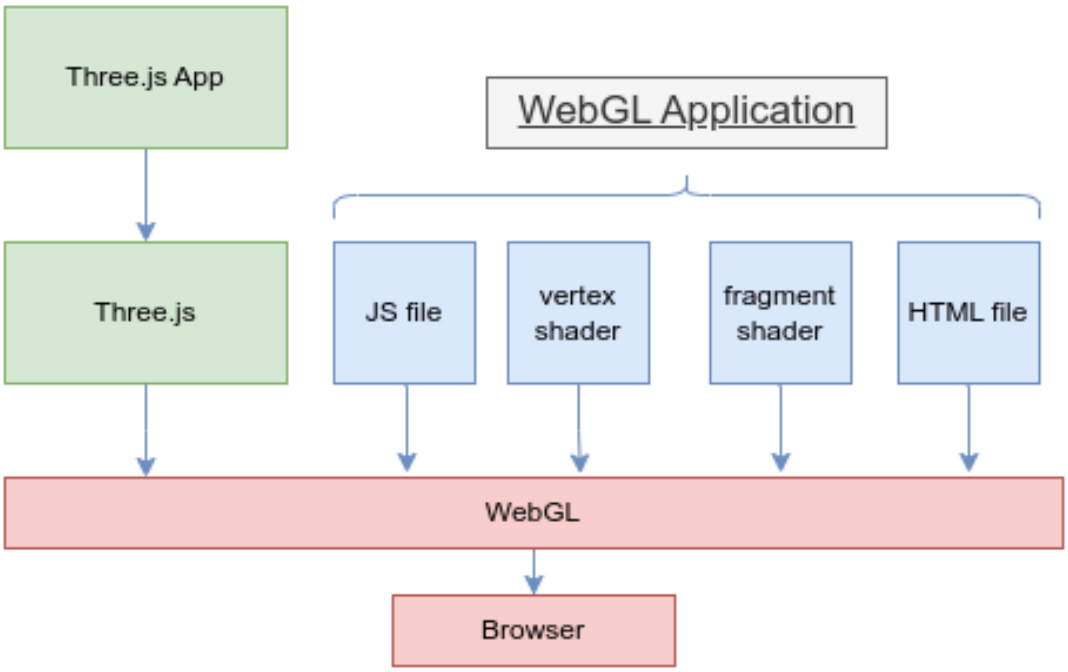


Figure 3: Render flow comparative of WebGL and Three.js. Image from [1].

2.1.5 React

React is a JavaScript library for building user interfaces, primarily for single-page applications. It's used for handling the view layer in web and mobile apps. React allows you to design simple views for each state in your application, and it will efficiently update and render the right components when your data changes.

React components are simple, stateful, and reusable. They can either be JavaScript functions or classes, and they output a component of the user interface. Each of these has its lifecycle separated into three phases: mounting, updating, and unmounting, which you can control and manipulate according to the specific needs of your application.

React also introduces a virtual DOM, which is an abstraction of the actual DOM. It's lightweight and detached from the browser-specific implementation details. One of the biggest benefits of the virtual DOM is optimizing performance by minimizing the amount of direct manipulation of the DOM.

In addition to these core features, React has a rich ecosystem of tools and libraries that complement its functionality. With a rich community it currently stands as the most used front-end framework among Web Developers with more than 40% [2].

2.1.6 React Three Fiber

React Three Fiber (r3f) is a powerful tool that brings together the capabilities of Three.js and the simplicity of React. It provides a way to use Three.js in a React environment, allowing developers to work with 3D graphics in a React methodology.

Because working directly with Three.js can be challenging due to its imperative nature, React Three Fiber comes in to provide a declarative approach. This shift from imperative to declarative, and from object-oriented to functional programming, makes it easier to integrate 3D graphics into a React application. React's primary role is to complement imperative systems like Three.js with these more modern, functional paradigms.

React Three Fiber introduces an additional layer of abstraction on top of Three.js, leveraging the power of React allows developers to predominantly use r3f's abstractions while resorting to imperative Three.js code when necessary.

In conclusion, the combination of React, Three.js, and React Three Fiber provides a robust and efficient framework for creating interactive 3D web applications. It leverages the strengths of each library, providing a high-level, declarative approach to 3D web development without sacrificing the power and flexibility of lower-level WebGL operations.

2.1.7 Drei

Drei is a utility library designed to complement react-three-fiber (r3f). It provides a collection of reusable components, helper functions, and hooks that simplify the process of creating 3D scenes. [3].

It was developed by the Poimandres, a collective of open-source developers who are passionate about creating tools for building rich, interactive user interfaces. They are the team behind several popular libraries in the React ecosystem, including react-three-fiber.

The library offers a variety of components that wrap around Three.js classes, providing a more React-like API. These include primitives, controls, lights, helpers for debugging,components for working with shaders, materials, and asset loading.

Additionally, Drei provides higher-level abstractions for common use-cases, such as mixing Three.js and DOM content in the same scene. It also offers a set of hooks specifically designed for working with Three.js in a React environment.

2.1.8 Additional Relevant Libraries

Leva Controls

Leva is a lightweight, intuitive and fully customizable React library also developed by the Poimandres collective that allows you to create UI controls for your application [4].

Leva allows you to easily add controls such as sliders, color pickers, checkboxes, and more. These controls can be used to manipulate your application's state in real-time, which can be particularly useful in scenarios such as tweaking visual parameters for a Three.js scene.

Context Provider

In React, the Context API is a feature that allows you to share global state across multiple components without having to pass props through intermediate elements [5]. A Context Provider is a component that wraps other components and provides them with access to its state.

The Provider component shares its state with its child components. Any child component can access the shared state using the useContext Hook, regardless of how deep it is in the component tree. In summary, the Context Provider is a powerful tool in React that can help to simplify your codebase and make your state management more efficient.

React Perf

React Perf is a performance measuring tool for React applications. It provides a way to measure the time spent in various aspects of your React application, like rendering components, updating the DOM, and more [6]. This can be useful for identifying performance bottlenecks and optimizing your application.

React Suspense

React Suspense is a feature in React that lets you defer rendering part of your application until some condition is met. Suspense allows you to display some fallback content (like a loading indicator) while waiting for something like data fetch to complete [7].

Google Authentication

Google Authentication is a service that provides authentication features for your application. It allows users to sign in with their existing Google accounts, which can help reduce the registration logic.

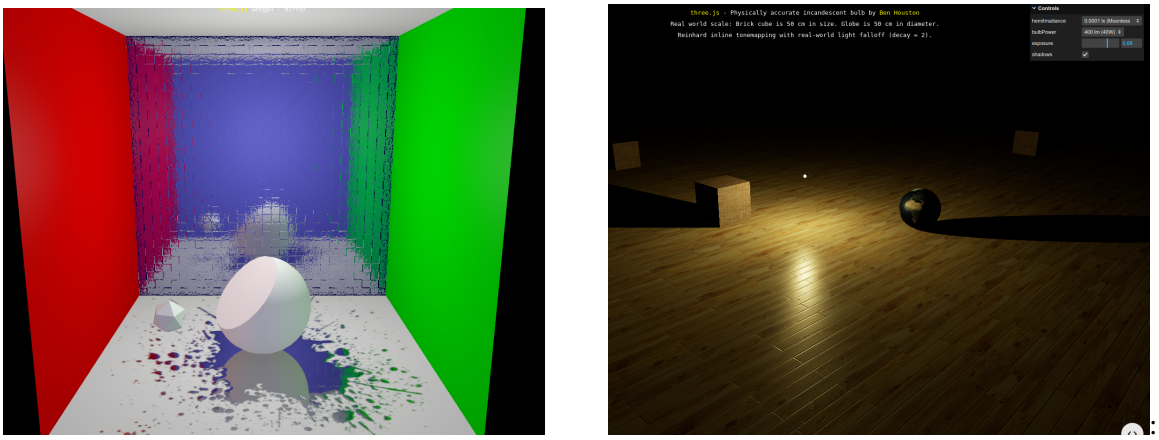
In a React application, you can use libraries such as react-google-login to integrate Google Authentication. This library provides a GoogleLogin component that you can use in your application [8]. When a user clicks on this component, they will be redirected to the Google login page. After successful authentication, Google will redirect the user back to your application with an access token.

2.2 Existing Applications

WebGL with Three.js represents a potent tool integrated into most modern browsers, offering a lot of projects and implementations from which to draw inspiration. In the subsequent section, we shall talk about these projects that have guided my decision-making process towards the development of the specified viable product

2.2.1 Three.js Examples

Firstly, let us examine the foundational examples provided by the official Three.js website visible in Figure 4. In the initial example, we encounter a simplistic room where all objects are geometrically rendered, accompanied by rudimentary physics-based animations [9]. While my primary focus was on scene generation, I looked for real-time user interaction with this technology, an aspect often limited to mere scrolling and clicking, lacking full accessibility for the user. This aspect is shallowly demonstrated in the following example [10].



(a) WebGPU mirror and animation.

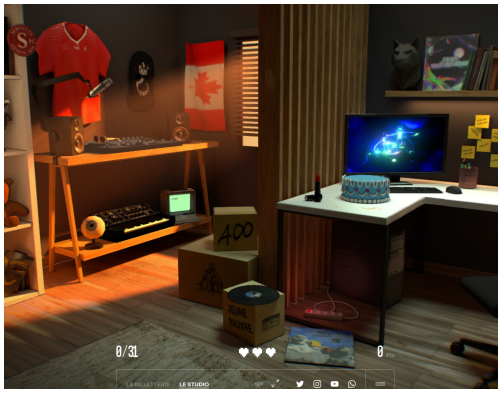
(b) Pysically accurate incandescent bulb.

Figure 4: Three.js Examples projects visualization.

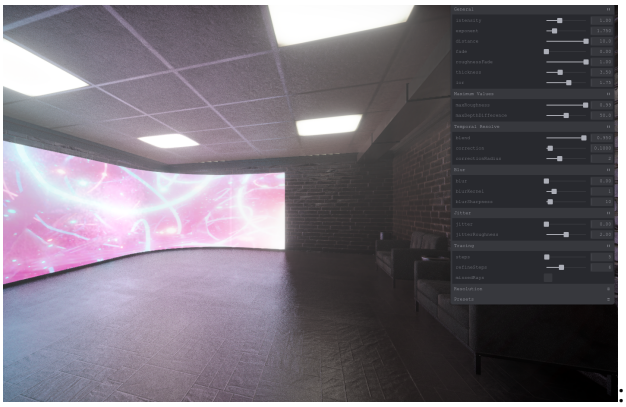
2.2.2 Inspiring Creative Projects

Beyond the official examples, research was conducted with the aim of constructing a fully customizable 3D space for more complex projects like those displayed in Figure 5. Particularly inspiring was Paul Soulhiard's application for the 1863 music tour [11]. Here, we observe a remarkably clean and desirable camera movement, complemented by a room setup akin to that in my project vision. However, it lacks user customization and login/session capabilities.

Furthermore, the project by user @0beqz brings forth impressive graphics to the web, featuring performant Screen Space Reflections in Three.js [12]. This project highlights the extensive customization potential offered by the Three.js UI tool. Nevertheless, it lacks camera movement and data-saving capabilities.



(a) Paul Soulhiard: 1863 Music Tour.



(b) 0beqz: Screen Space Reflections.

Figure 5: Inspiring and creative projects visualizations.

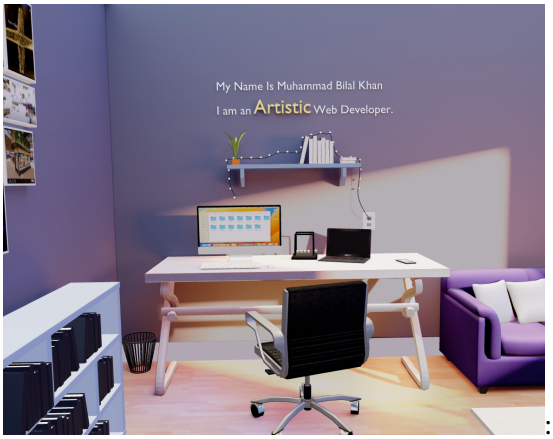
2.2.3 Portfolio Examples

We also conducted a thorough exploration of projects related to ours within the domain of portfolio creation. A selection of two web applications are displayed in Figure 6.

The first exemplar by David Lee4[13] in Figure 6a, demonstrates remarkable prowess with a highly visually impactful experience. It also features scrollable navigation and 'click' interactions. This project introduces innovative ideas such as zooming into paintings and customized guided scroll navigation to showcase the entire scene effectively.

Concluding our analysis of state-of-the-art projects, we present Bilal Khan's portfolio [14] represented in Figure 6b, in which a similar project is implemented, featuring responsive 3D elements that react to user clicks and interactions, seamlessly linking to social media and projects, a vision very similar to mine.

(a) David Lee: Artistic portfolio.



(b) Bilal Khan: Developer portfolio.

Figure 6: Personal portfolio examples visualization.

2.2.4 Comparative Analysis

To visualize these projects side by side, a table was devised with each column representing a key feature while the projects are displayed in the rows, as shown in Table 1. These columns outline notable features, user interaction status, customization levels, potential contributions to our current project, and linked instances for each project.

From this comprehensive analysis, detailed user stories were developed to capture various user needs and scenarios (more details in Section 3.2). Additionally, three core points were identified as essential components for the final product, focusing the development process into these key components (more details in Section 3.3).

Project	**Notable Features**	**User Interaction**	**Customization**	**Contributions**	**Link**
Three.js Official Example	Geometric scene, basic physics animations	Limited	No	Real-time user interaction	1
Interactive Three.js Example	Real-time user interaction demonstration	Yes	Limited	Scene generation	2
1863 Music Tour Visualization	Clean camera movement, room setup	Limited	Limited	Scene setup	3
Screen Space Reflections Implementation	Performant graphics, UI customization	No	Extensive	Graphics enhancement	4
David Lee's Portfolio	Customizable navigation, interactive elements	Yes	Extensive	UI/UX enhancement	5
Bilal Khan's Portfolio	Responsive 3D elements, social media integration	Yes	Limited	3D element integration	6

Table 1: Comparative analysis of the related project examples.

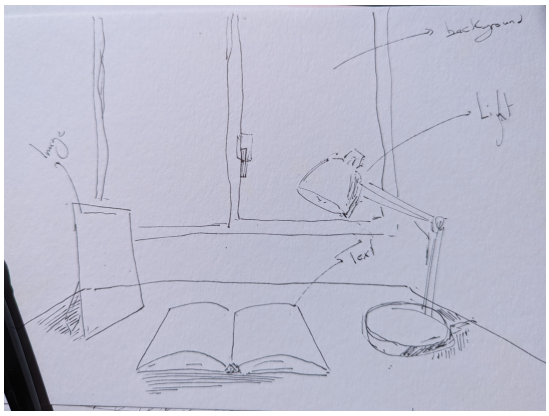
3 Design

In this section, we will conduct a preliminary analysis of the project technologies and establish a stack with the ones we will use, along with the roles of each in the final project. We will also delve into the initial project design, including the various diagrams and concepts that will aid us during the development phase.

3.1 Mock Layout

In compatibility with the previously relevant examples, in the sections "State of the Art" and "Introduction", it was decided on creating a desktop setup with variable objects that represent the user.

The ultimate objective of this project, in broad terms, is to visualize a specific environment. This environment is characterized by the inclusion of at least the following pertinent objects: a laptop, which represents a website; a book, symbolizing user data; and a frame containing an image. These visualizations are depicted in Figure 7. Side by side, we have an initial mock drawing of the conceptual idea, followed by the placement of these items in the real world. This comparative allows for a clear comparison between the initial concept and its real application, providing a more comprehensive understanding of the project's visual goals.



(a) Crude ink desktop design.

(b) Item placement representation.

Figure 7: Design process examples.

3.2 User Stories

The following are a set of comprehensive user stories that encapsulate the primary objectives and needs of our potential users. From facilitating seamless login and data saving for personalized experiences to empowering users to customize their 3D space according to their preferences, these stories lay the groundwork for an engaging and user-focused application.

- **User Story: Login and Data Saving**
  - **As a user, I want to** be able to log in **so that** my customized data can be saved and retrieved in future sessions.
- **User Story: Customizing 3D Space**
  - **As a user, I want to** be able to customize the 3D space **so that** I can create an environment that suits my preferences and needs.
- **User Story: Portfolio Showcase**
  - **As a user, I want to** be able to showcase my portfolio in the 3D space **so that** I can present myself in an engaging way.
- **User Story: WebGL Interaction**
  - **As a user, I want to** be able to interact with the WebGL in the browser **so that** I can experience real-time 3D graphics in this web application.

3.3 Requirements

In the process of designing the project foundations we specified three minimum product requirements. By prioritizing interactivity, personalization, and database management we ensure a final viable project.

These requirements condensed in the following figures are defined as follows: The first one references the user's ability to interact with a 3D space in our application. This is a core functionality of the proposed project since the main aim is to provide a customizable 3D environment. Users should be able to modify various scene properties, including light settings, shadows, camera angles, and geometry materials see in Table 2.

The second requirement allows users to input their personal information and save the customized state of their desired scene. It adds a personal touch to the application, enabling users to create a portfolio that reflects their preferences. The ability to save the customized state enhances user experience by preserving their customization efforts across sessions, visible in Table 3.

The final requirement involves the application storing a database of information about the customized scene. This feature is important for retrieving user-specific information when needed, enhancing

the personalization aspect of the application. Although this is a non-functional requirement, it significantly impacts the overall user experience. We can visualize this specification in Table 4

Number	1
Description	The user will be able to interact with a 3D space
Rationale	This is the Minimum viable product of the proposal, a fully customizable 3D Application
Type	Functional
How to validate	User testing - can the user customize the scene properties such as light, shadows, objects and environment

Table 2: User interaction requirement 1.

Number	2
Description	The user shall be able to input personal information and save the customized state of the desired scene
Rationale	Gives purpose to the client as they create their desired portfolio and facilitates their session efforts
Type	Functional
How to validate	User testing - input personal data and visualize its scene integration

Table 3: User customization requirement 2.

Number	3
Description	The app shall store a database of information about customized scenes
Rationale	This will allow retrieving this information for each user when desired
Type	Non-functional
How to validate	Developer testing - check it has been saved in the specified database and integration test the endpoints.

Table 4: Appclication storage requirement 3.

Tables 2, 3, and 4 show the functional and non-functional minimum product requirements of the system.

3.4 Environment

In this segment, we will outline the software development environment and its key characteristics that play a role in application development.

Blender plays a crucial role in handling the heavy lifting of modeling tasks. Its exclusive use for modeling and exporting objects simplifies this aspect of development significantly. These low-poly objects are exported in GLB format, decompressed to jsx, and further customized within the actual scene using a React component.

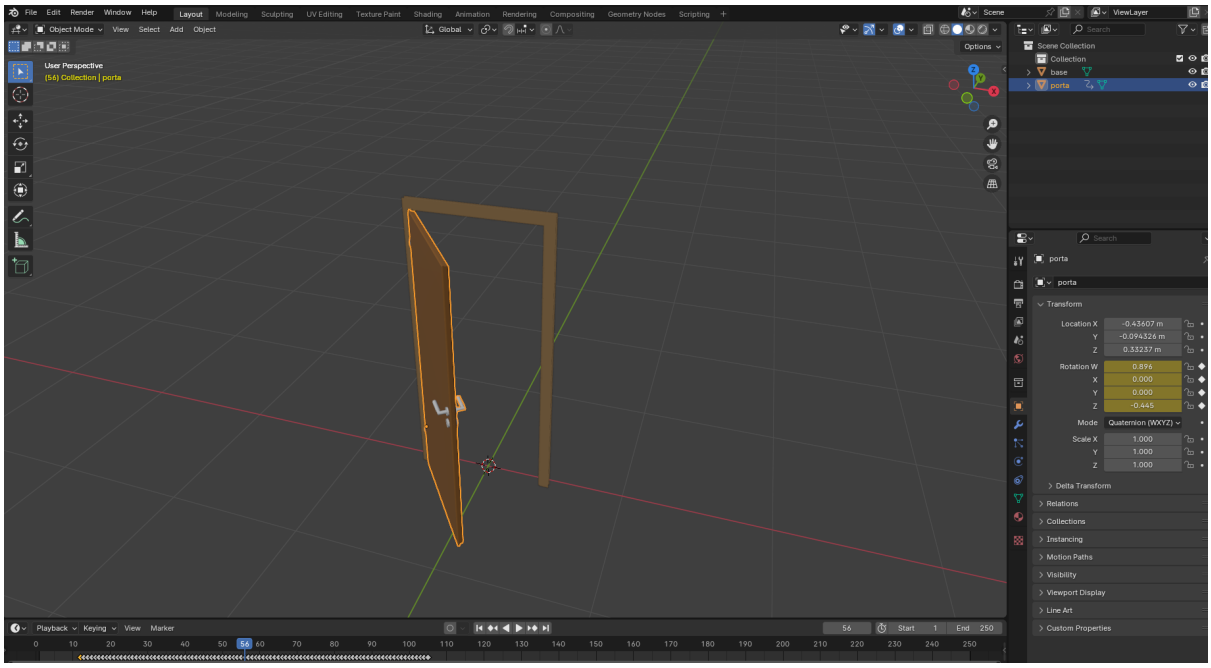


Figure 8: Example visualization of an animated door in Blender [15].

The main operating system utilized for development is the latest version of Ubuntu. Developed with Visual Studio Code (VS Code) as our Integrated Development Environment (IDE). With VS Code integrated with GitHub for version control, project advancements are securely stored in a remote repository, safeguarding against any potential data loss. To launch the project, a simple command is used, which intuitively opens the project in the default web browser, primarly tested for Google Chrome.

```
npm run dev
```

Database management is facilitated using DBeaver, offering a comprehensive solution for handling database operations efficiently [16]. This application has been selected based on previous experience, familiarity, and stable performance in other projects, ensuring a good development experience and productivity throughout the application development lifecycle.

3.5 Technology Stack

This Section will discuss the estimated necessary technologies for the project implementation. Their presence and role will be further explained as necessary. We will also provide a comprehensive overview of the technologies based on their relevance, to give a bird's eye view of what this project encompasses.

The previously specified scene will be visualized on a web page as the main functionality of the project. Therefore, the implementation of a front-end system will be necessary to provide the page with the visualization of these components on the scene, along with rendering lights and shadows. Additionally, the front-end will handle session management and credentials for all users accessing the page.

In line with the functionalities mentioned above, we will have to establish a connection between the front-end and back-end system. The back-end system is responsible for managing users at a lower level, providing basic functionalities to connect front-end data with some form of data persistence mechanism and retrieval.

In Figure 9, we can explore the three primary technology stacks intuitively segregated by their layers in application development. The front-end layer encompasses a progression from the webpage to the graphics rendering, traversing through various technologies, frameworks an libraries and also specifying the most pertinent complementary libraries.

The back-end likewise highlights the principal technologies and their significance, accompanied by a description, followed by the mention of the database management system.

3.6 Front-end

To articulate all the interactions between the various visual elements that together form the overall user experience on a webpage, it is necessary to create a tree-based structure that integrates all these elements into a single common container.

Within this container, various groups of elements that shape the user experience on the page are included, such as the interaction with scene loading, the visualization of scene elements, interaction with login systems, and various web layout elements like sidebars or menus.As shown in Figure 10, these four groups of elements are branches of a single container called App. This container element allows the inclusion of other nested container elements.

Initially, the application is focused only on a single component called "Scene", in which are included all the observable elements from the scene, such as the lights, room and different furniture that will be displayed.

Unfortunately, the previously defined scene cannot be rendered immediately. To provide the user with feedback indicating that the scene is rendering, a message should be displayed on the visual interface. This is implemented using a Fallback component, which is included in a separate container

called Suspense. Another notable element is the provider component which supplies a context for the applications global data so they can be used throughout the project.

After implementing the previous components that display the core functionalities to the user, a menu should be introduced to enhance user experience. This sidebar component should primarily include a form for user input and personal information customization tools. Positioned over the canvas element, this pure HTML component creates depth through layered interaction, thereby adding technical complexity.

The final component is the login element, this is the first element the user will visualized and interacted with. Through Google Authentication, users can gain access to the application's landing page. This component serves as the gateway to the rest of the application.

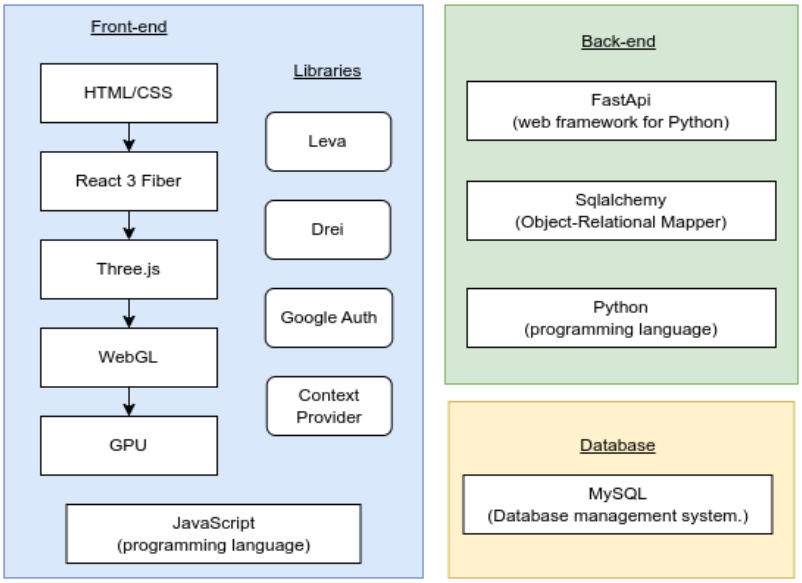


Figure 9: Project technologies full-stack diagram

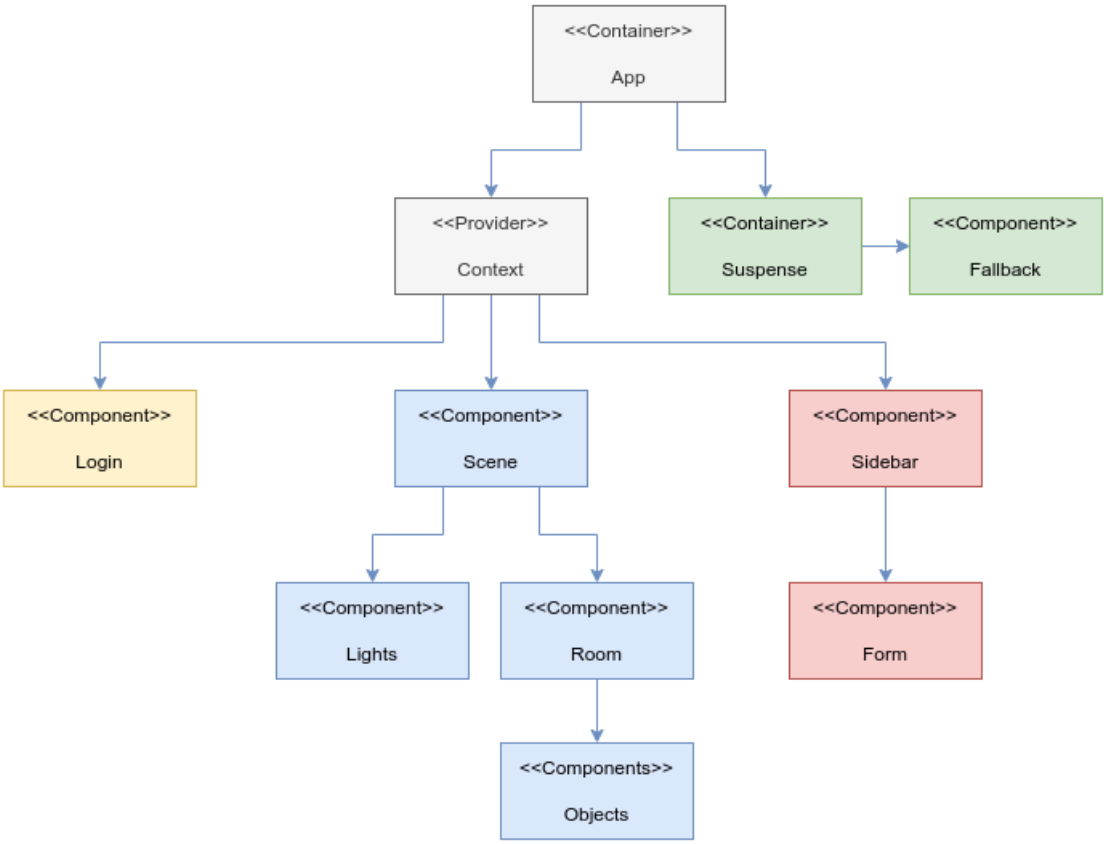


Figure 10: React Three Fiber project component diagram.

3.7 Back-end and Database

For the server side, we will be using FastAPI due to its compatibility with React and its ease of use. FastAPI will provide several endpoints to send and retrieve all the scene data, as well as to handle user authentication. FastAPI's high performance and automatic interactive API documentation (Swagger UI and ReDoc) make it an excellent choice.

In terms of data storage, we will be utilizing SQLite, a lightweight, file-based database. It's ideal for small applications and will be used in this project to store and retrieve scene data and user information for the login system. To interact with SQLite, we will use SQLAlchemy, a SQL toolkit and Object-Relational Mapping (ORM) system for Python. SQLAlchemy allows us to interact with the database like we would with SQL.

The ideal database structure is defined based on an entity relation diagram seen in Figure 11, which was created from an initial study of the functionalities. Each entity (like User, Scene, etc.) in the diagram should correspond to a table in the SQLite database, and the relations between them are represented as foreign keys.

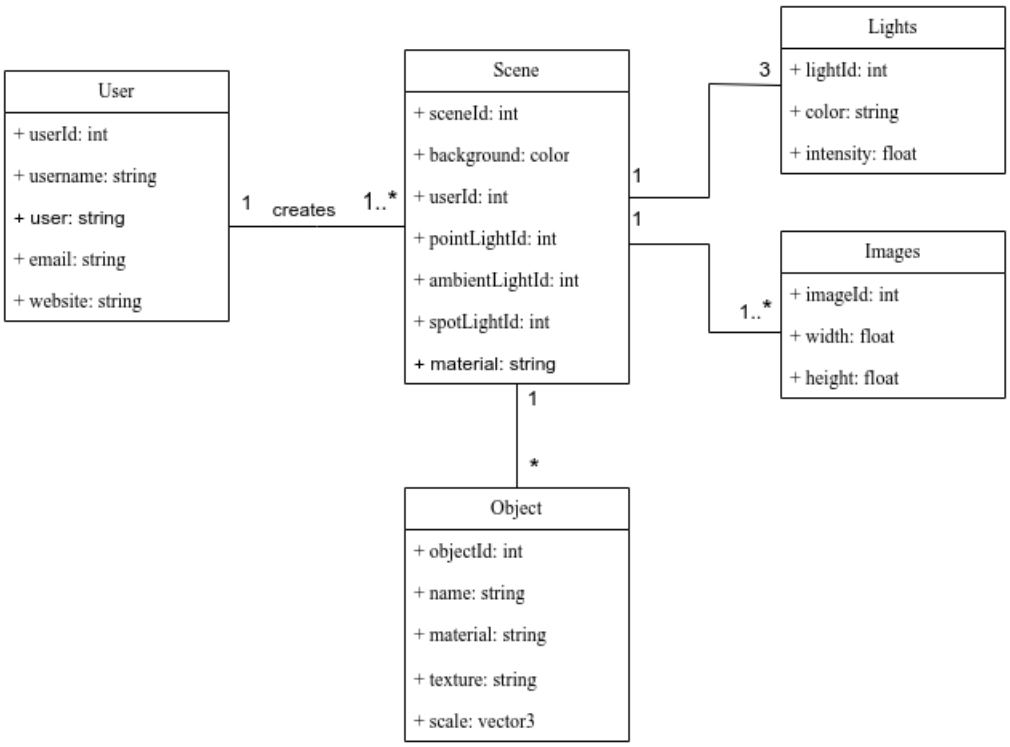


Figure 11: Database entity relationship (ER) diagram.

4 Implementation

The implementation phase encompasses the methodology that serves as a representation of the application's progress. It also includes the rationale and justification for the various libraries that were utilized to meet the product requirements. The bulk of the implementation is focused on the client-side, which will be elaborated upon in the subsequent sections.

We structured the initial project setup in accordance with the r3f documentation. This allowed us to visualize basic geometries and create a testing playground to familiarize ourselves with the framework.

Upon completion of the initial setup, the development phase of the application was based on an iterative methodology. The workflow involved visualizing an object that would complement the scene, importing/modelling it in Blender, exporting it to our scene, adding it to the component tree, configuring its visualization, and adding custom configuration. This basic template facilitated the development of the scene, although there were instances where deviation from the guideline was necessary.

This system, with its emphasis on flexibility and continuous improvement, aligns with Agile principles [17]. To further illustrate this process, Figure 12 depicts the theory complemented by Figure 13 which provides an example of the initial blender visualization and the exported model to the scene.

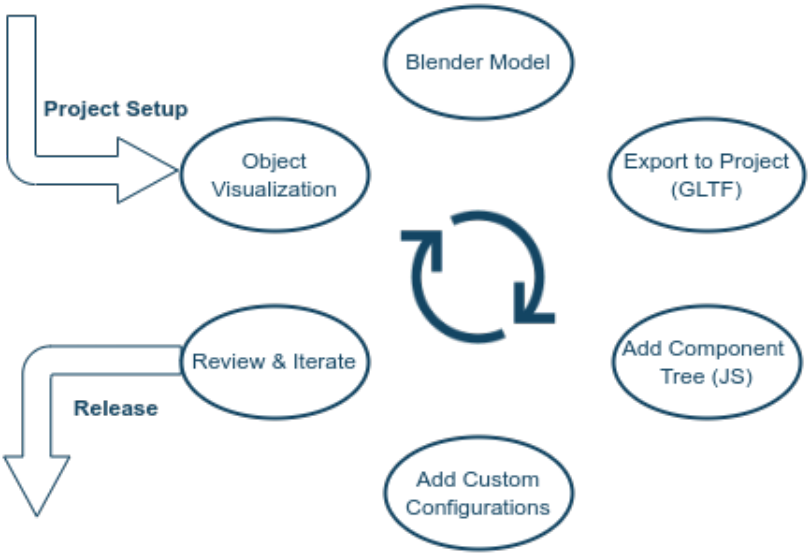


Figure 12: Iterative Agile methodology diagram.

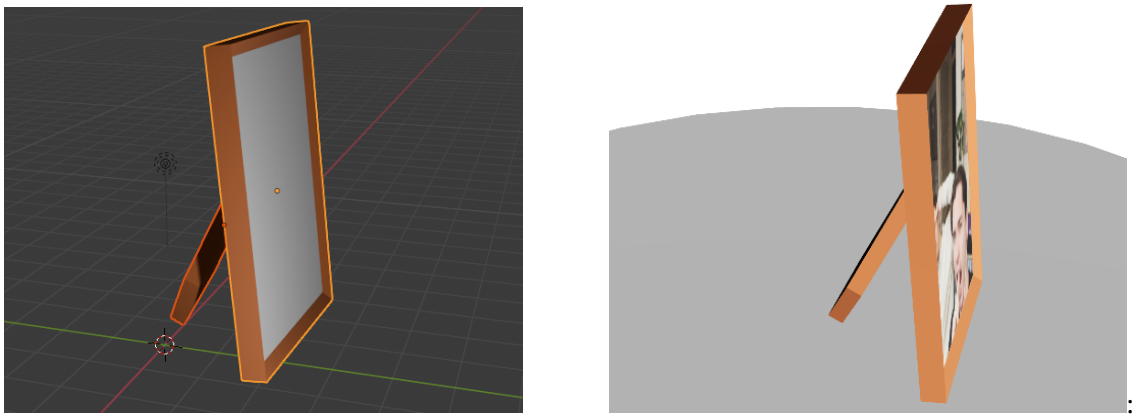


Figure 13: From Blender design to functional website view, a framed image object.

4.1 Front-end Implementation

This section will provide a detailed account of the crucial feature of the application. The client-side implementation is the project's backbone and required the most time to develop. Consequently, this section is the most comprehensive part of the implementation and includes numerous visual representations of the process. It focuses on key components and the decisions made along the way.

4.1.1 3D Scene

Upon establishing a functional base project, we began adding models with the objective of recreating the studio-like scene as visualized in the design mockup of Figure 7. The execution of this segment involved utilizing all the specified technologies to render the desired scene. Given the lack of complex modelling background, we relied on free-use models and modelled the basic geometries ourselves. The challenges of this process arose primarily due to unfamiliarity with the technology. However, the development process, despite a slow start, progressed exponentially.

To accomplish this task, research and familiarization with the Blender Application were crucial. Consequently, resource research from various fields was conducted. The most relevant resources were YouTube tutorials that provided detailed, step-by-step modelling guides [20] [21]. The models were primarily extracted from the Poimandres market [22], and Poly Pizza platform [23].

To simplify the positioning of these models in our 3D environment, we first imported them into Blender. With the acquired basic knowledge of placement, scale, and customization, a preliminary setup was elaborated. The Figure 14 illustrates the design process of a framed image as well as the initial Blender scene layout.

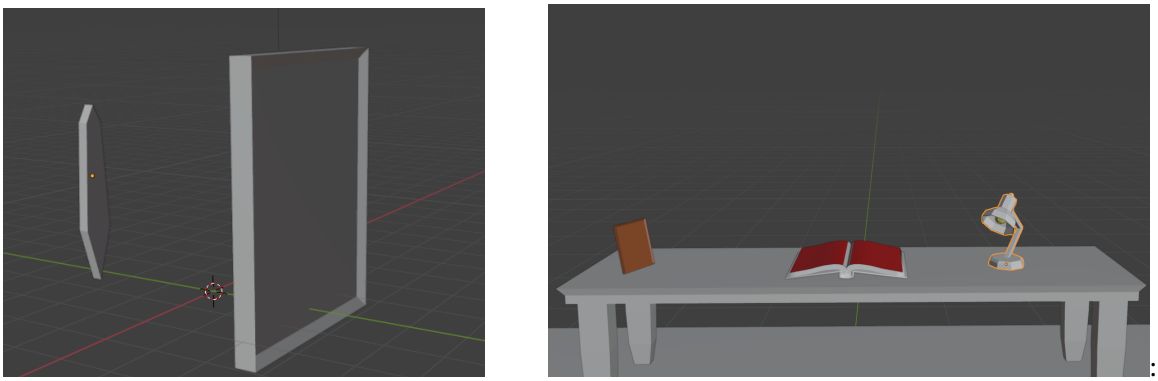
 :

Figure 14: Initial blender framed image model and scene design.

From this initial scene, further improvements and additions were made to match the original design. The same methodology was applied to implement the login, room and all of its containing objects. The placement in Blender exports seamlessly into our application layout, making the fine-tuning of each object less tedious. For this, we also used the Poimandres website [24] that automatically generates the corresponding JavaScript file for our exported Blender in GLTF, a standard format for scenes and 3D models. To complete the process, it was necessary to link our components with their respective models. Both were placed in their standard directories: `public/models` and `src/components`. To do so, we defined the following code using the `useGLTF` hook from the Drei library to load a 3D model from a GLTF file. This mechanic process is the second to last step followed by including our component to the HTML Canvas element.

To visualize this scene in our web application, we added a basic ambient light from the Three.js library. This light illuminates all components of the scene equally, allowing us to view our exported models in the final scene, visible in Figure 15. The following figure shows the website view comparison when adding the first light to our scene. It was also essential to initialize a camera. Although not definitive for our scene, to facilitate roaming and designing the application, we defined one of the default camera settings component `OrbitControls` [25] that implements a similar visualization and dynamic scene movement to that of Blender.

Figure 15: Inclusion of Three.js ambient light in the scene [26].

Lights, Environment and Shadows

The next logical step in the development process was to implement the different types of lighting in the scene. According to the design, there would be three basic types: a directional light representing the sun and its rotation, a point light identifying the desktop lamp, and a customizable ambient light for the user.

The ambient light was simply implemented using Three.js, requiring only the configuration of the intensity. The point light was placed at the same coordinates as the desktop lamp. To clearly visualize its position, we utilized another helper from Drei creating a reference to the Point Light component and attaching a visualization helper the debugging stage represented in Figure 16.

The same process was repeated for the directional light, but with the added decision to mimic the sun to enhance user immersion. For this, the Drei library provides the Sky Helper [27]. This offers a highly customizable environment. Although the sphere representing the sun is visible, it does not equate to a light source. To achieve a more realistic effect, we matched the Sun position prop to our light's position and extracted the logic to an external component.

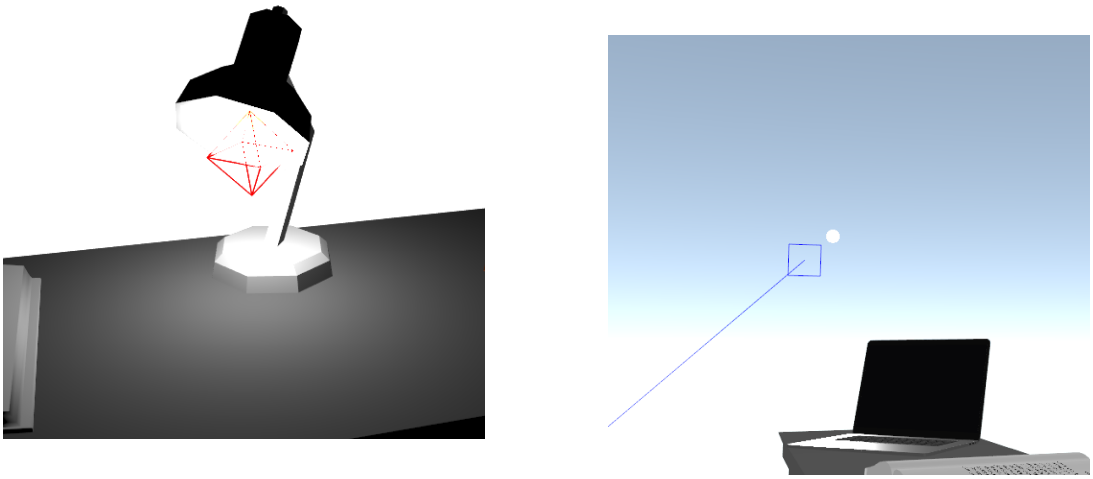


Figure 16: Visualization of Light, Sky, and Helper.

Research led to the inclusion of various HDRI environments with lights. The justification behind this was to contribute to a more immersive experience outside of the room. Such an environment was implemented with the Drei Environment Helper, which sets a global cubemap that affects the scene's environment and background.

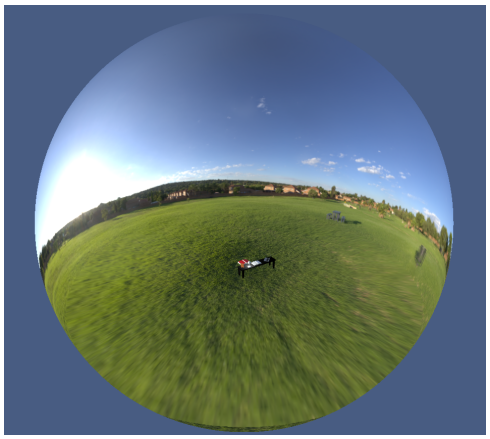
A cubemap is a collection of six square textures that represent the reflections on an environment surrounding an object. They are mapped to the inside faces of a large cube surrounding the object and are used to simulate the environment reflecting off the object's surface.

This component simplifies the setup of such an environment. It comes with a selection of presets as seen in Figure 17a from HDRI Haven[28], a source of high-quality HDRI images. It also allows to Import and use custom environment maps with higher quality as Figure 17b exemplifies.

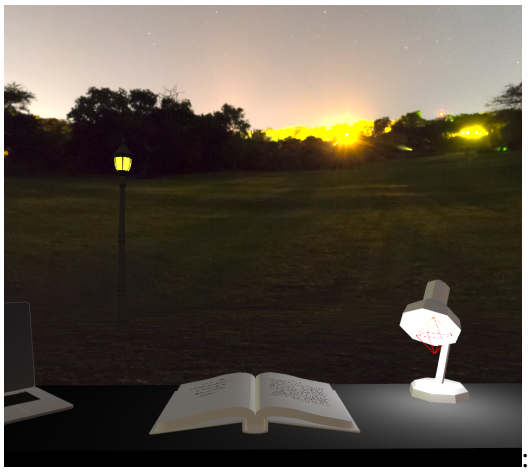
These maps serve as light probes containing detailed information about how light behaves in the captured environment, including intensity, color, and direction. When an HDRI map is applied to a 3D scene, it acts as a source of diffuse lighting. The lighting information from the HDRI is used to uniformly illuminate the scene, providing a base level of light that mimics real-world ambient light conditions, substituting our existing ambient light and Sky for a more realistic alternative.

In react-three-fiber, configuring shadows involves several steps, the first step being to enable shadows in the WebGL renderer. This is done by setting the shadows property of the <Canvas> component to true. Then we need to enable shadows for specific lights in our scene. This is done by setting the castShadow property of the light component. Finally, we need to specify which objects should cast and receive shadows. This is done by setting the `castShadow` and `receiveShadow` properties of the desired mesh component.

In our research we found the `<SoftShadows>` component from the Drei library which provides a more realistic rendering of shadows compared to the standard shadow implementation in Three.js.



(a) Preset HDRI viewed from outside the environment map.



(b) Custom 4k HDRI first person view.

Figure 17: Park preset and custom Environment Maps visualizations.

In real life, shadows are not perfectly sharp. They tend to blur or soften at the edges, and this blurring increases as the distance between the shadow-casting object and the shadow-receiving surface increases. This phenomenon is known as "penumbra".

The standard shadow implementation in Three.js does not account for this penumbra effect. It renders shadows as sharp, which can look unrealistic, especially in scenes where realism and inmersion is important like in our case. This Drei component, on the other hand, simulates this penumbra effect, resulting in softer, more realistic shadows. It does this by taking multiple samples per pixel and averaging the results, which creates a blur effect.

Both of these shadows are visible in the Figure 18, The directional light(blue) has configured soft shadows while de desktop lamp uses the default Three.js shadows.

Staging

This section documents the addition of objects to the scene with the aim of creating a more cohesive and visually appealing environment. It also includes the customization of existing elements to maximize visual impact. We implemented such colors and configured the material of each object with MeshStandardMaterial.

MeshStandardMaterial is a versatile and realistic material type in Three.js that supports physically-based rendering (PBR). It is designed to mimic real-world materials by considering the interaction of light with surface details like roughness and metalness. This allows for more realistic shading and lighting effects.

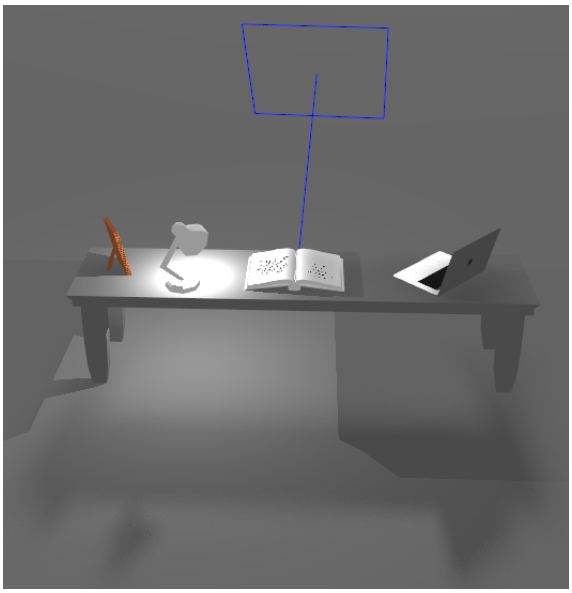


Figure 18: Default Three.js and Drei `SoftShadows` in the same scene.

In conclusion, the careful selection and configuration of materials, combined with the strategic addition and placement of objects, resulted in a visually rich and cohesive 3D scene. The final result is a scene that is not only visually appealing but also immersive, providing a satisfying user experience seen in Figure 19.

Figure 19: Scene setup featuring high intensity ambient lighting, environmental background, additional objects and materials.

Suspense and Performance Tools

During the development of the previous sections, there was a significant increase in application overhead, particularly when loading the initial scene on devices with integrated graphics cards. My research led me to implement the following key technologies to address this issue.

Firstly, I utilized Suspense from React. Implementing one of the default spinners as seen in the left of Figure 20 to appear while the application was idle. The design is a animated white loading circle on a black background. This is particularly useful in our 3D application where assets can be large and take time to load.

Secondly, I implemented the React Developer Tools performance profiler(Perf) to visualize performance in real time. This tool provides a detailed breakdown of the component render times and helps identify bottlenecks in the application. Additionally, it shows the number of geometries, textures, shaders, real-time FPS, and CPU and GPU milliseconds as well as other information, also displayed in Figure 20. We situate such information at the top left of the screen.

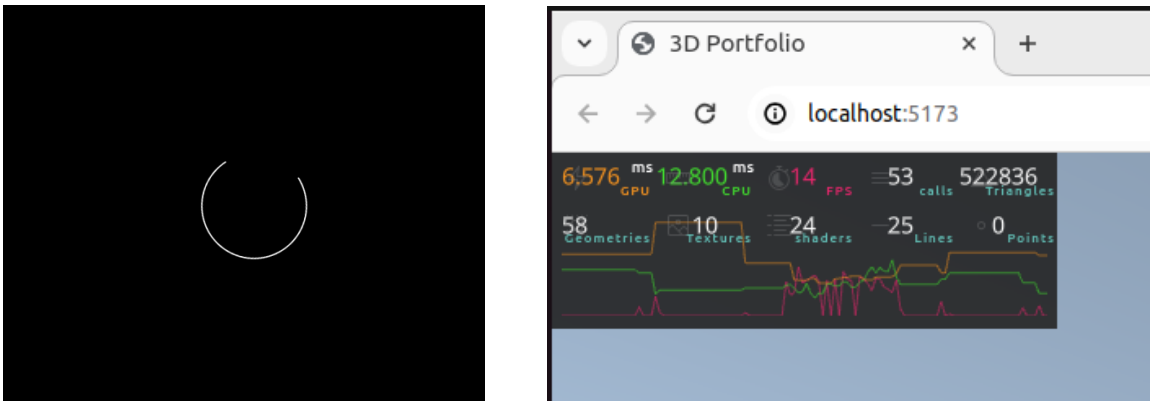


Figure 20: Suspense spinner fallback screen and performance monitor.

4.1.2 User Interface

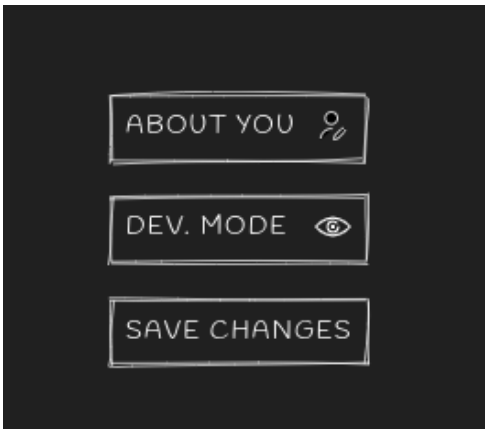
User interaction is a critical aspect of our application. The objective was to implement an HTML-based User Interface (UI) that overlays the entire scene. This design choice was made to avoid direct interference with the Scene, thereby creating a clear distinction between UI customization and the actual scene experience and interaction. This additional layer, positioned above the `Canvas` element—which renders the entire scene, follows a simple, hand-drawn design aesthetic, provided by the Wired Elements component library [18], as illustrated in Figure 21.

The buttons located at the bottom left of the screen serve specific functions. They facilitate the opening of a user form, toggle visualization to allow users to enjoy the scene without the UI and control elements, and include a save button to transmit data to the back-end for future retrieval; present in Figure 21a. The necessary logic and styling for these buttons were incorporated and

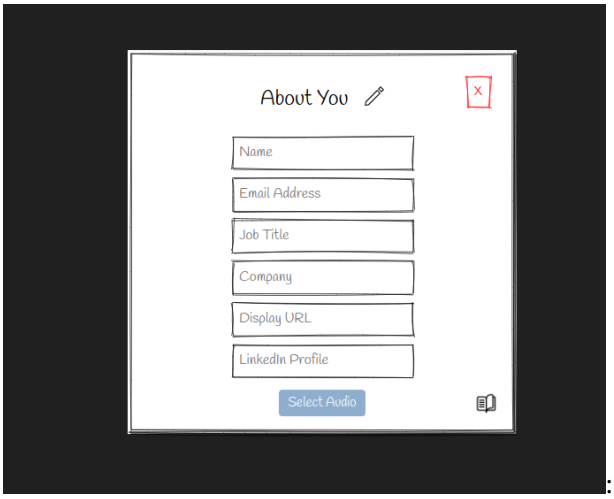
positioned strategically at the lower corner of the screen for easy access and minimal interference.

Form

The Form component is designed to capture personal user data, which is subsequently mapped to the scene elements. As depicted in Figure 21b, these fields gather pertinent information to create a user profile. Additionally, there is a separate page that provides users with an opportunity to share more about themselves in an 'About You' section. This user-centric design approach personalizes the interaction, making the application more engaging and user-friendly.



(a) Menu displayed at the lower left of the landing page.



(b) User form with personal information displayed

Figure 21: Wired Elements UI implementation

Having established that the user interacts with the scene through clicking, research was conducted on existing forms of interaction and discovered various code implementations for a hover state of scene objects. These implementations aligned perfectly with the project vision without overriding the scene properties [29] [30]. This hover state highlights the border of the object with a thin white line, subtly indicating interactivity without disrupting the overall aesthetic of the scene. This feature was implemented across all intended interactive elements within the scene enhancing the user experience by providing clear visual cues for interaction.

Camera

The camera configuration was also designed with immersion as a primary goal. To achieve this, a first-person camera was chosen without a doubt, as it offered the most value in terms of user experience. However, considering the need for the user to interact with the UI, it was not feasible to have the camera follow the mouse position on a one-to-one basis.

This challenge required a solution where the mouse would follow the camera with a slight delay,

remaining stationary most of the time. The solution involved interpolating the mouse coordinates with the appropriate rotation axis of the camera. This approach ensured a smooth and natural camera movement, notably improving the immersive experience while still allowing for user interaction with the UI.

The code snippet provided in Listing 1 demonstrates this implementation. The `useEffect` hook sets up event listeners for mouse movement and window resizing. The `onMouseMove` function updates the mouse coordinates relative to the center of the window.

In the `useFrame` hook, the camera's rotation is updated based on the mouse position, creating a delayed follow effect. The camera's rotation is interpolated towards the target rotation, this translates to a smooth transition. This setup ensures a balance between an immersive first-person camera and the necessity for user interaction with the UI.

```
const { camera } = useThree();
// Create references for the target and mouse positions
const target = useRef(new THREE.Vector2());
const mouse = useRef(new THREE.Vector2());
const windowHalf = useRef(new THREE.Vector2(window.innerWidth / 2, window.
    innerHeight / 2));

useEffect(() => {
  // Update mouse position based on mouse move event
  const onMouseMove = (event) => {
    mouse.current.x = event.clientX - windowHalf.current.x;
    mouse.current.y = event.clientY - windowHalf.current.y;
  };

  // Add event listener for mouse movement
  window.addEventListener('mousemove', onMouseMove, false);

useFrame(() => {
  // Calculate target position based on mouse position
  target.current.x = (1 - mouse.current.x) * 0.0002;
  target.current.y = (1 - mouse.current.y) * 0.0002;

  // Smoothly update camera rotation towards the target position
  camera.rotation.x += 0.05 * (target.current.y - camera.rotation.x);
  camera.rotation.y += 0.05 * (target.current.x - camera.rotation.y);
  camera.rotation.z = 0; // Keep camera's z-rotation fixed at 0
});
}
```

Listing 1: Dynamic camera movement: adjusting rotation with mouse position

In addition to visualizing each element from up close, `onClick` events were implemented for key scene

objects. This allows the user to navigate with the camera and view these objects up close. Examples of such navigation can be seen in Figure 23. This interactive feature provides a sense of freedom and dynamism when navigating through the scene layout.

Login

This component functions as a gateway for users to access the platform and authenticate themselves. Specifically, we initialize this component when users are not logged in. Given the project's theme of being set within a room, I found it both playful and visually appealing to design a door-opening animated login mechanism using Google's React OAuth2.

The user visualizes the door through a customized camera that consistently focuses on the object, yet moves based on the mouse position, thereby providing a sense of interaction. The door includes a hover state, and when clicked, it opens to reveal a black background within which the Google Auth login button fades in. These visual states appear in the Figure 22. Upon a successful login, the camera zooms into the room, simulating the user's entry. After a brief delay, the components change, and the user can explore the full application.

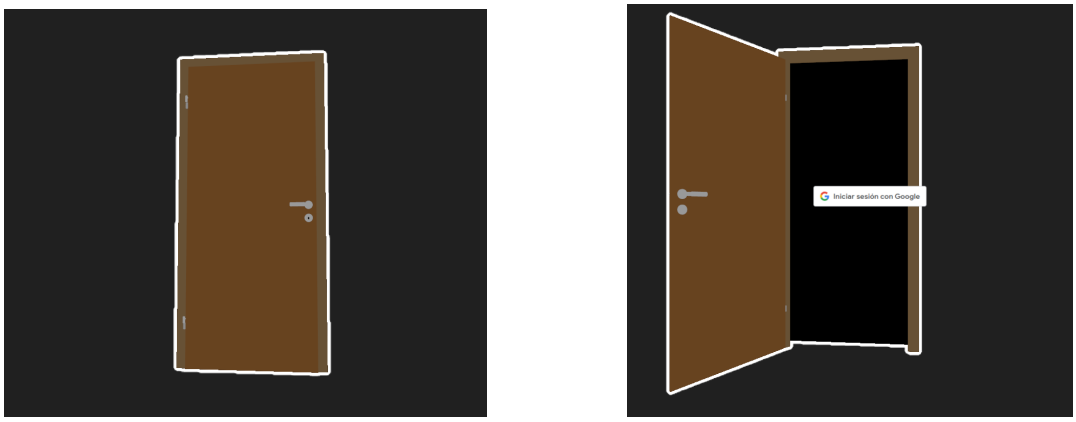


Figure 22: Login hovered interaction visualizations: Before and after the initial click interaction

4.1.3 Interactive Customization

This section implements one of the key functionalities of our application: real-time customization. There is a clear division in these customizations, separated by User Input via HTML and the Leva functionality tab. Both affect the scene in various ways, but the first subsection is highly functional, while the second serves a larger purpose.

Since the beginning of the implementation process, we have added complexity to our scene, which has been reflected in the performance. The objective of Leva as a technology is not just to add these customization values, but also to provide the user with tools to adapt the application's performance to their computer's capabilities.

Input UI

After highlighting the design of this layer interaction previously when defining the input form in Section 4.1.2, we now encounter the challenges of mapping such input to the scene.

The three fields to be modified are the laptop, which needs to display a website from the input form; the book, being the centerpiece of the scene, which needs to display the user's description; and lastly, the functionality to input MP3 audio to play in the background.

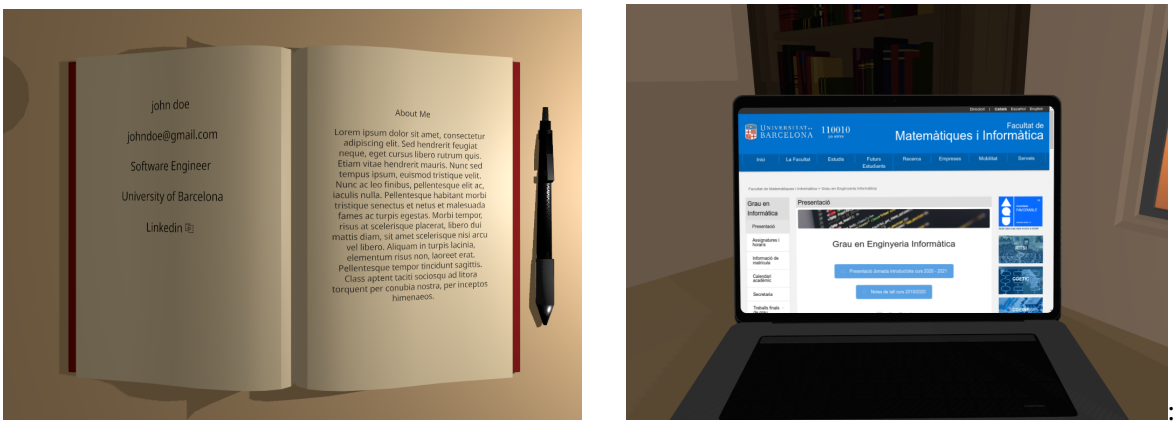
The MP3 audio player utilized the `react-h5-audio-player` library [19], integrating it into our application and styling it to resemble the Spotify player. The component configurations set it to play automatically upon loading and in a loop, which can be used to input a voice narration or desired music adding another layer of ambiance to the scene.

Both the Laptop and Book components reside deep inside our component tree while the HTML form, that contains user data, is required to exist outside the `Canvas` element for it to be positioned absolutely in our viewport. This means that in order to send data down to each child component, a practice known as prop drilling, which is generally discouraged, was not the best solution. The solution found was React's context provider, a method of defining global variables accessible throughout the specified context. With this in mind, and using the HTML Drei component and an Iframe for the website, the mapping of these elements was straightforward. The following figures illustrate this implementation by showcasing both end products with example contents 23.

Leva UI

The most impactful visual configurations reside within the implementation library. Integrating this library is straightforward; it involves adding variable parameters to our components and manually defining the customization of each one. These customizations can range from presets to colors, slider bars, and more.

To provide reliable customization, all scene objects have been equipped with a toggle button. Some



(a) The book component displaying a user-inputed information.

(b) The laptop component displaying a user-inputed website.

Figure 23: Scene components displaying user mock data.

specific objects have additional configuration options. For instance, the properties of the window can be modified and are defined by the roughness, transmission, and the index of refraction (ior) values.

Additionally, the FrameImage component has been designed to allow customization of size and map user input images to the existing layout. This is achieved using the Decal component from the Drei library that adds details to 3D objects without modifying the base texture or geometry. It projects this texture onto an object, following the object's surface details.

Finally, within these customizations, there is a dedicated folder for lights and environment. From here, we can adjust various parameters such as intensities, positions, and colors of different light sources. We have the flexibility to toggle ambient, directional, and point lights as needed. The environment map offers presets along with a custom background option. The Sky simulator allows for modifications to the sun's height and rotation around the scene, providing a dynamic and customizable lighting environment. The most relevant of these customizations can be seen, as the user would, in Figure 24.

4.2 Back-end and Database Implementation

Upon reevaluating the problem, it was found that the necessary data was already configured in the context provider. This led to the creation of two endpoints, a PUT and a GET, both designed to retrieve or update data from the database. These endpoints were initialized upon user login and also in the UI save button. The initial plan for a complex database structure was simplified as the data to be saved was minimal. Axios library was imported for making HTTP requests, and JWT tokens were handled using the jwt-decode library. This approach facilitated secure authentication and efficient data management within the application. Consequently, the database was implemented with a single SQLite table, a decision driven by time constraints and resource savings. However, this setup should be revisited, with a transition to MySQL being a viable option if scalability becomes a requirement.

Figure 24: Leva customization dropdown display.

5 Results

In this section, we will explain the final application conducted experiments and their results, this being a performance test conducted on two different devices.

5.1 Experimental Environment

The first system specifications can be seen in Figure 25. The system runs on Ubuntu 22.04.4 LTS, ensuring a stable and secure environment for development and testing. The browser installed is Chrome version 125.0.6422.112, which provides excellent support for WebGL, enhancing the rendering capabilities of 3D applications. The laptop is equipped with 16 GiB of DDR4 SODIMM memory, which allows for smooth multitasking and efficient handling of the 3D application data. The Intel Core i5-8265U processor, with its 4 cores and 8 threads, offers sufficient computational power to manage the demands of the application, balancing performance and energy efficiency. Finally, the Intel UHD Graphics 620 GPU, while not high-end, is capable of supporting WebGL rendering, ensuring that the 3D application performs adequately even on this relatively modest hardware setup. This system configuration represents the most limited user setup for running the application, making it a suitable benchmark for ensuring broad accessibility and performance.

Category	Specification
Product	Lenovo ThinkPad L490
OS	Ubuntu 22.04.4 LTS
Browser	Chrome 125.0.6422.112
Memory	
Total Size	16 GiB
Type	DDR4 SODIMM
Processor (CPU)	
Model	Intel Core i5-8265U
Cores	4
Threads	8
Graphics (GPU)	
Model	Intel UHD Graphics 620
Resolution	1920x1080

Figure 25: Specifications of Lenovo ThinkPad.

The second device displayed in Figure 26 system runs Windows 10 Pro version 10.0.19045 a user-friendly standard environment that integrates with our hardware. The AMD Ryzen 5 3600 6-Core Processor, with its 6 cores and 12 threads, offers substantial computational power, ensuring smooth rendering and interactivity in the 3D application. The other notable component is he NVIDIA GeForce RTX 3080 GPU, a high-end graphics card known for its exceptional rendering capabilities and performance. With a resolution of 1920x1080, this GPU can deliver stunning visuals and smooth framerates, enhancing the overall experience of the 3D application.

Overall, this custom-built system boasts powerful hardware components, including a robust processor and a high-end GPU, making it well-suited for demanding tasks such as 3D rendering.

These contrasting environments will be used to navigate the application, test the scene load with its different configurations, and evaluate the overall user experience. This approach will allow us to assess the application's performance under various conditions and configurations. Additionally, it will provide insights into how different environments and settings impact the application's functionality and performance.

Category	Specification
Product	Custom Build
OS	Windows 10 Pro 10.0.19045
Browser	Chrome 125.0.6422.112
Memory	
Total Size	16 GB
Type	DDR4 SODIMM
Processor (CPU)	
Model	AMD Ryzen 5 3600 6-Core Processor
Cores	6
Threads	12
Graphics (GPU)	
Model	NVIDIA GeForce RTX 3080
Resolution	1920x1080

Figure 26: Specifications of Custom Built PC.

5.2 Performance Analysis

With the previously mentioned systems, the application will undergo testing with particular focus on frames per second (FPS), which will be monitored using the Performance Controls library. The goal is to maintain an average FPS of around 30, which is generally considered acceptable for smooth performance.

The testing methodology will involve manually toggling the scene elements and observing the impact on the FPS. Both computers will start with the most basic scene configuration possible, containing only essential objects such as a table and a background ambient light. The complexity will be gradually increased while analyzing performance. Interestingly, the system with the dedicated graphics card showed no noticeable impact on performance, maintaining a constant 60fps throughout the tests. Consequently only the Lenovo ThinkPad system was analyzed and the results visible in Figure 27 are generated only with this device.

Lighting can significantly impact performance. While ambient and directional lights can create a more realistic and visually appealing scene, they are more computationally intensive than environment maps. Therefore, using environment maps can be a slightly more efficient way of configuring the scene. Additional objects in the scene can also influence performance. Each additional object increases the complexity of the scene, requiring more computational resources to render. While these individual additions may not be noticeable, the cumulative effect can significantly impact performance.

Materials can also have a significant impact on performance. especially the window glass material, which requires complex calculations for refraction and reflection that reduce performance drastically. To optimize the scene, shadow elements can be 'baked'. Baking shadows refers to the process of pre-calculating the shadows and storing them as textures, which can then be applied to the objects in the scene.

To visualize these scenes, Figure 28 displays the high-performance configurations, and Figure 29 shows the counterpart low-performance configuration. Their partial configuration is visible on the right, while the corresponding frame rate can be seen at the top left of the panel.

In conclusion, most systems should be able to handle the 3D application, the actual performance can be affected by various factors related to the complexity of the 3D scene and the use of its features. That being said, thanks to our implemented configuration, there exists the possibility of navigating the main features of the application even on a low-end computer.

Configuration	FPS
Minimal Scene (Env Map)	36 fps
Minimal Scene (Sky)	33 fps
Minimal Scene (Sky + Point Light)	31 fps
Basic Scene (Simple Objects)	23 fps
Complex Scene (Glass Material)	16 fps

Figure 27: Frame rates across configurations from minimalistic to complex scenes.

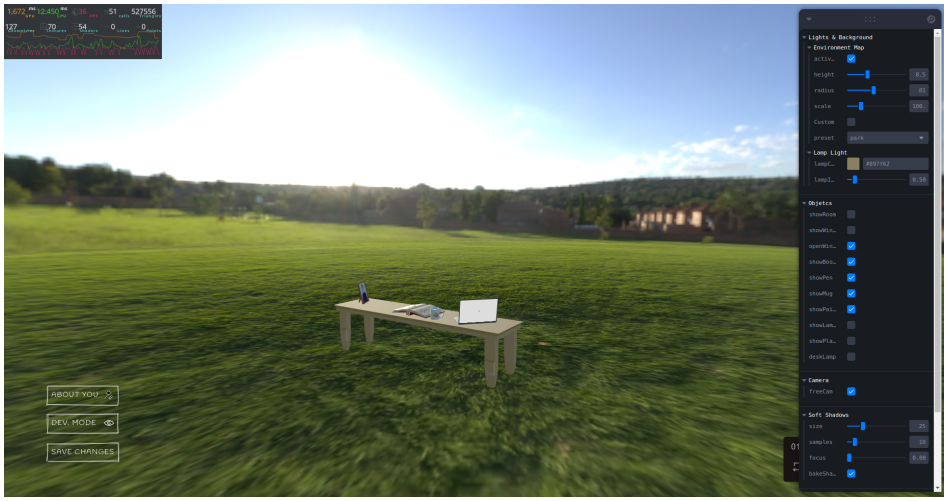


Figure 28: Optimal performance configuration, free camera setup.

Figure 29: Limited performance configuration, defaul camera setup.

5.3 Final Result

The resulting application successfully meets the specified main objectives, from the essential basic configuration to the performance flexibility and the preliminary login implementation although with a limited save state functionality, thus reaching the minimum viable product. From here, we conclude that the foundation for a virtual customizable 3D portfolio has been established, thanks to the comprehensive integration cemented during the implementation process.

The application includes a login page, a performance monitor, a customizable scene panel, and a user interface form, among many other features and details that together result in a visually appealing and highly customizable 3D scene. This crucial groundwork can be indefinitely developed to enhance personal and scene-oriented customization, but that falls outside the scope of this project.

Finally, to conclude the project, we will review the outcome of the application with examples of possible end-use cases. We will visualize three scene configurations:

The first use case, seen in Figure 30, is a cityscape configured with a Barcelona-themed website, light color, and background with a very bright ambient light.

The second display, shown in Figure 31, is a example client configuration of an animal shelter. The user is hovering over the book configuration that details information about their organization. The scene is situated in a park with a calming environment and low light. This scene was chosen due to my personal connection to this cause.

The final scene represents my study room, as viewed in the design process with a highly refractive glass and a sky simulation at dusk. It is customized with the University of Barcelona website, my contact information, and my image. By choice, the developer mode is configured as it better represents myself.

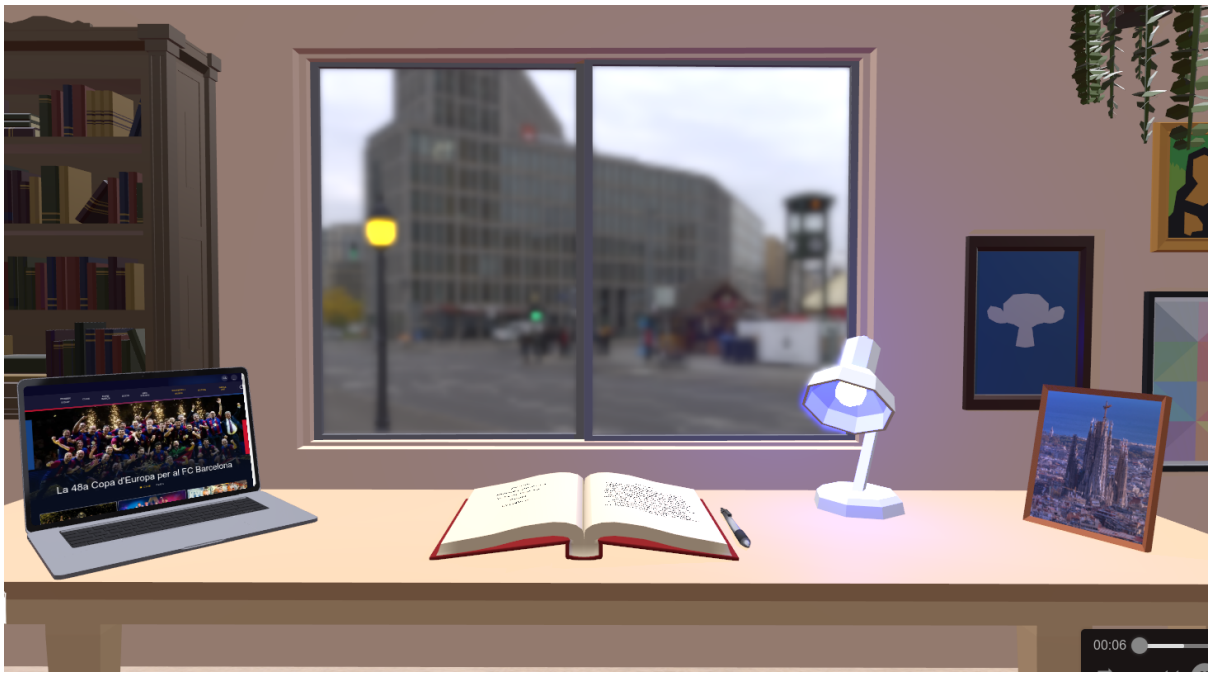


Figure 30: A city setup featuring a Barcelona-themed configuration.

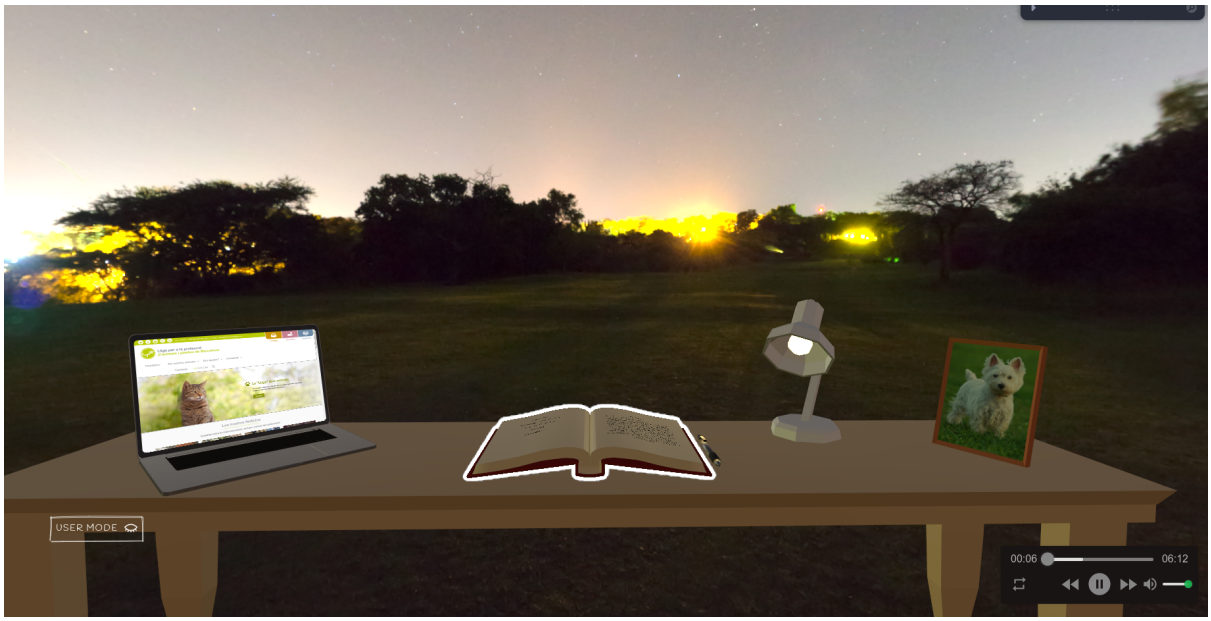


Figure 31: An example client configuration of an animal shelter's portfolio.

Figure 32: Customized 3D Personal Portfolio.

6 Conclusions

During the process of carrying out this project, we identified several issues that need to be addressed. These issues ranged from technical difficulties to resource limitations, all of which required careful analysis and solutions. In addition to these challenges, we also explored potential expansions and enhancements for future scenarios.

6.1 Addressed problems

Upon completion of this project, we can conclude that this application's development process has been non-linear. When faced with new technologies, numerous problems and issues arose. Despite these challenges, the realization of the initial vision has been achieved. Although the project's life-cycle could very well just be its beginning, I am confident in the initial tool that has been developed and its future possibilities, extensions and applications.

Even with thorough problem analysis and documentation, the extent of unknown knowledge was significant and consumed a large part of the implementation process. The experience gained during this process will undoubtedly be beneficial for future projects and enhancements. Moreover, this experience has highlighted the importance of continuous learning and adaptability in the field of software development, especially when dealing with emerging technologies.

6.2 Potential expansions and enhancements

When considering potential future enhancements to the current project, various options emerge. Firstly, there is the possibility of expanding the capabilities of the existing application. This would involve adding new functionalities, objects, and generating various scenes from which to choose.

The second enhancement consists of a key functionality that would drastically improve the quality of life for inexperienced users. Establishing presets for automatic scene configuration would save time, reduce complexity, and lower the knowledge requirements for users who would benefit from such a feature.

Another extension would be the possibility for users to share and explore rooms designed by other clients. To allow this, we would need to build an alternative login system, such as using a room ID.

Finally, a clear incorporation would be a reinvention of a more scalable and fully functional back-end that saves each and every one of the user scene customizations. Such data management would allow users to save their progress in its entirety, encouraging continued use of the application.

References

[1] Ed Angel and Dave Shreiner. WebGL App Development. https://www.cs.unm.edu/~angel/SIGGRAPH_ASIA_17/WebGL%20App%20Development.pdf, Accessed: 2024-02-28.

[2] Stack Overflow. Developer Survey. https://survey.stackoverflow.co/2022#overview, Accessed: 2024-05-06.

[3] Poimandres. Drei Library Repository. https://github.com/pmndrs/drei, Accessed: 2024-06-02.

[4] Poimandres. Leva Controls Library Repository. https://github.com/pmndrs/leva Accessed: 2024-06-01.

[5] React Context Official Documentation. https://legacy.reactjs.org/docs/context.html Accessed: 2024-05-010.

[6] React Performance Tools Official Documentation https://legacy.reactjs.org/docs/perf.html Accessed: 2024-04-15.

[7] React Suspense Official Documentation. https://react.dev/reference/react/Suspense Accessed: 2024-03-08.

[8] NPM. React OAuth2 | Google https://www.npmjs.com/package/@react-oauth/google Accessed: 2024-06-05.

[9] ThreeJS Official Examples. https://threejs.org/examples/#webgpu_mirror Accessed: 2024-02-20.

[10] Interactive ThreeJS Official Example. https://threejs.org/examples/#webgl_lights_physical Accessed: 2024-02-20.

[11] Paul Soulhiard. 1863 Music Tour Visualization. https://tour.1863.fr/ Accessed: 2024-02-22.

[12] 0beqz. Screen Space Reflections Implementation. https://github.com/0beqz/screen-space-reflections Accessed: 2024-02-20.

[13] David Lee's Portfolio. https://www.davidlee.studio/ Accessed: 2024-02-20.

[14] Bilal Khan's Portfolio. https://mbilalkhan.com/ Accessed: 2024-02-20.

[15] Olav3D Tutorials . Blender Tutorial: Adding, Rigging, And Animating Doors. https://www.youtube.com/watch?v=8-p7JPtk6n0&t=186s Accessed: 2024-04-14.

[16] DBeaver. Universal Database Tool. https://dbeaver.io/
Accessed: 2024-02-20.

[17] Atlassian. Agile Methodologies. https://www.atlassian.com/agile
Accessed: 2024-03-13.

[18] Wired Elements Component Library. https://wiredjs.com/
Accessed: 2024-04-01.

[19] NPM. React H5 Audio Player. https://www.npmjs.com/package/react-h5-audio-player
Accessed: 2024-04-03.

[20] 3DMegaverse. Modelling a Basic Room in Blender Tutorial, https://www.youtube.com/watch?v=ogs2b6j01DI Accessed:2024-03-18.

[21] Bruno Simons. Blender Fundamentals. https://www.youtube.com/watch?v=5C5VkPynQwo&list=PL5nApUt6Z8sS9XcXJmjgfFOz38DF8acIv&index=18 Accessed: 2024-03-18.

[22] Poimandres. Model Market https://market.pmnd.rs/,
Accessed: 2024-05-29.

[23] Poly Pizza. Model Market. https://poly.pizza/
Accessed: 2024-05-29.

[24] Poimandres. GLTF to JS Converter. https://gltf.pmnd.rs/
Accessed: 2024-06-05.

[25] Three.js. Orbit Controls. https://threejs.org/docs/#examples/en/controls/OrbitControls Accessed: 2024-01-31.

[26] Three.js. Ambient Light https://threejs.org/docs/#api/en/lights/AmbientLight
Accessed: 2024-01-31.

[27] Poimandres. Drei Sky Helper https://github.com/pmndrs/drei?tab=readme-ov-file#sky
Accessed: 2024-01-31.

[28] HDRI Haven https://hdri-haven.com/
Accessed: 2024-04-08.

[29] CodeSandbox. Outline Example 1 https://codesandbox.io/p/sandbox/selective-outlines-d36mw?file=%2Fsrc%2Findex.js Accessed: 2024-05-05.

[30] CodeSandbox. Outline Example 1 https://codesandbox.io/p/sandbox/r3f-outlines-mq5oy?file=%2Fsrc%2Findex.js Accessed: 2024-05-05.

Appendix

In our project, we have adopted the Agile methodology, specifically implementing it through a series of sprints. Each sprint is a short, time-boxed period when a scrum team works to complete a set amount of work. Our sprints typically last from one week to three weeks. This iterative process allows us to continually improve and respond to changing project requirements.

Sprint 1: February 28, 2024 - March 5, 2024

- Review similar Projects
- Define 3 stages of minimum viable product
- Functional requirements
- Initial Technical requirements
- Technical Progress - learn the basics in threeJS
  - Geometries
  - Resizing
  - Cameras
  - Debug UI
  - Materials
  - Textures

Sprint 2: March 6, 2024 - March 12, 2024

- Document initial structure
- State of the art
- Technical Progress
  - Lights
  - Shadows
  - Implement basic project in react-three-fiber

Sprint 3: March 13, 2024 - April 17, 2024

- App mobile study
- Unity WebGL vs React Three Fiber

- Document introduction and motivation
- Architecture Diagram

Sprint 4: April 17, 2024 - April 26, 2024

- Document Three.js
- Visualization Pipeline
- Document WebGL
- Technical Progress
  - Familiarize with Blender
  - First Blender export to Project

Sprint 5: April 25, 2024 - May 2, 2024

- Add complexity to the project
- Study pipeline development idea of blender exports
- Study logic behind login
- Technical Progress
  - Create Repository
  - Finish first demo export r3f

Sprint 6: May 8, 2024 - May 15, 2024

- Technical Progress
  - Structure Project and update repository
  - Spinner
  - Incorporate HTML to scene
  - Incorporate hover highlight to scene
  - Mouse events change camera
  - Include Animation, adapt and customize
  - Finish Login
  - Optimize Research

Sprint 7: August 15, 2024 - May 22, 2024

- Fix google oauth
- Force performance issues
- Apply textures and shading
- Customize picture size
- Sound Effects
- Customize lights
- Customize shadows/softshadows
- Back-end progress
- Customize Env Map
- Day Light alternative to env maps
- Custom UI user setup
- Environment map justification - outdoors scene (define environment map)
- Application adapts to PC - out of scope

Sprint 8: May 24, 2024 - May 29, 2024

- Play music dynamically
- Spotify styling player
- Customize text on object

Sprint 9: May 29, 2024 - June 3, 2024

- Laptop with website
- Glass window
- Staging
- Room
- Camera Setup for objects
- Functional Back-end