UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería del Software

Desarrollo de una API REST para la optimización de los tiempos de espera de las peticiones de renderizado de una aplicación web

Development of a REST API to optimize the waiting times of rendering requests of a web application

Realizado por
Pablo Sánchez Vecino

Tutorizado por
Rafael Marcos Luque Baena
Iván García Aguilar

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, septiembre de 2023

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Graduado en Ingeniería del Software

Desarrollo de una API REST para la optimización de los tiempos de espera de las peticiones de renderizado de una aplicación web

Development of a REST API to optimize the waiting times of rendering requests of a web application

Realizado por
Pablo Sánchez Vecino

Tutorizado por
Rafael Marcos Luque Baena
Iván García Aguilar

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2023

Fecha defensa:

Abstract

Web technologies constitute the basis of many applications and services, being a good part of these large scale and oriented to service a significant number of customers, whether they are final users or other computer programs.

A common feature of these is the necessity of a scalable system, that is, a system capable of servicing a large number of requests within a reduced span of time, which can also increase as the popularity or usage needs of the application or service grow.

In the context of this work, we depart from an already developed web application, *PhotoSpaces* [55]. The service it provides to its users consists of the rendering of photorealistic images in a server coming from scenes users load into the system from files that are then sent to it. The main goals of this application were based on providing a service that allowed relieving the user's machine from the computational effort the process of rendering requires, and also keeping it as simple, intuitive, and user-friendly as possible, objectives that have already been accomplished.

Nevertheless, the current system is not prepared for servicing multiple users simultaneously, and the waiting times for the requests these may send would become excessively high if the user base were to grow.

The need for the work to do in this project arises from these shortcomings of the application, which attempts to search for a proper solution to these problems and then modify and extend the application to implement it.

Keywords: Web development, REST API, Node.js, Microservices

Resumen

Actualmente, las tecnologías web constituyen la base de multitud de aplicaciones y servicios, siendo gran parte de estos de una gran escala y estando orientados a servir a un elevado número de clientes, ya sean usuarios finales u otros programas informáticos.

Una característica común de estos es la necesidad de un sistema que sea escalable, es decir, que sea capaz de prestar servicio a una gran cantidad de peticiones en un intervalo reducido de tiempo. También se requiere que puedan ir aumentando a medida que crece la popularidad o necesidad de uso de la aplicación o servicio.

En el contexto de este trabajo, se parte de una aplicación web que ya ha sido desarrollada, *PhotoSpaces* [55]. El servicio que esta proporciona a sus usuarios consiste en el renderizado de imágenes fotorrealistas en un servidor a partir de escenas que estos cargan mediante ficheros y que son enviadas a dicha aplicación. Los principales objetivos de esta se basaban en proporcionar un servicio que permitiera aliviar al dispositivo del usuario final de la aplicación del esfuerzo computacional que el proceso de renderizado supone y, además, mantenerla tan simple, intuitiva y fácil de utilizar como fuera posible, los cuales ya se han logrado.

Sin embargo, el sistema actual no está preparado para prestar servicio a múltiples usuarios al mismo tiempo, y los tiempos de espera de las peticiones que estos pueden realizar se volverían excesivamente elevados si comenzara a crecer el número de usuarios que utilizan la aplicación.

A partir de estas carencias de la aplicación, surge la necesidad del trabajo a realizar en este proyecto, que trata de buscar una solución adecuada a estos problemas y modificar y extender la aplicación para implementarla.

Palabras clave: Desarrollo web, API REST, Node.js, Microservicios

Índice

1 Introducción

1.1. Motivación

El trabajo realizado en este TFG consiste en la extensión de *PhotoSpaces*, una aplicación web ya existente que permite a sus usuarios cargar escenas 3D y enviarlas a un servidor de renderizado para obtener las imágenes correspondientes.

La necesidad de este trabajo surge debido a que el sistema actual, que utiliza un enfoque monolítico con un servidor que se encarga de la recepción de la petición, del proceso de renderizado y de la devolución de la imagen, no es capaz de manejar adecuadamente múltiples peticiones en un período de tiempo reducido. Además, estas darían lugar a tiempos de espera excesivamente elevados para los usuarios, puesto que deben esperar a que el único servidor termine el renderizado de todas las peticiones anteriores antes de comenzar con la que acaba de recibir. Todo esto se traduce en una experiencia deficiente.

Estas limitaciones resultan incompatibles con un entorno de despliegue real, donde una sola instancia del servidor en el *backend* de la aplicación no es suficiente. Es decir, el sistema actual no es escalable. La escalabilidad es una propiedad esencial para cualquier aplicación web exitosa, ya que esta debe estar preparada para soportar aumentos en su base de usuarios y en la demanda de procesamiento.

Para abordar este problema, se propone la extensión y modificación del sistema para que este cuente con un *backend* basado en una granja de renderizado distribuida que ofrezca una interfaz en forma de *API REST* con la que se pueda interactuar desde el exterior.

El uso de una granja de renderizado distribuida ofrece una serie de ventajas significativas. Por un lado, permite distribuir la carga de trabajo de forma eficiente entre los distintos servidores registrados, mejorando por ello los tiempos de respuesta del sistema, entregando a los usuarios las imágenes renderizadas de forma más eficiente, mejorando por ello la experiencia de estos. Por otro lado, esta será fácilmente escalable al proporcionar la flexibilidad para agregar o eliminar servidores de forma sencilla según sea necesario, lo que permite al sistema adaptarse a los cambios en la demanda de procesamiento.

Para estos fines, resulta fundamental contar con una interfaz que facilite la administración del sistema. Esta brinda al administrador un control completo sobre este, permitiendo la monitorización y gestión de los servidores y peticiones, así como la solución de problemas. La administración centralizada también permite un seguimiento constante del rendimiento del sistema y la detección temprana de posibles cuellos de botella o problemas técnicos gracias a una serie de estadísticas que ofrecerá.

Resumiendo, los principales cambios consisten en transformar el *backend* de la aplicación en una granja de renderizado y en proporcionar una interfaz web para que los administradores puedan monitorizar y gestionar los servidores de renderizado y las solicitudes. Sobre esto, se incorporan además una serie de optimizaciones que puedan contribuir a reducir más los tiempos de espera.

Los resultados de este trabajo poseen potencial para repercutir de forma positiva a las industrias en cuya actividad la obtención de imágenes renderizadas de forma eficiente resulte un factor fundamental. Estas se beneficiarían del hecho de que el software sea abierto y gratuito, permitiendo a cualquier interesado configurar, desplegar y administrar su propio sistema de forma rápida y eficaz mediante un proceso simple. De esta forma, con que una persona se encargue de estos aspectos, multitud de usuarios que no tienen por qué poseer tantos conocimientos técnicos podrán hacer uso del sistema desde la perspectiva de un usuario final a través de una interfaz web fácil de utilizar, abstrayendo la complejidad del sistema que se encarga de la gestión de sus peticiones.

1.2. Objetivos

1.2.1. Objetivo general

El objetivo principal en este trabajo se basa en permitir que la aplicación *PhotoSpaces* sea capaz de ofrecer a sus usuarios todas las funcionalidades que ya habían sido implementadas con anterioridad en un entorno de despliegue real y de forma satisfactoria, además de facilitar la administración del sistema.

1.2.2. Subobjetivos específicos

Para lograr este objetivo, resulta conveniente dividirlo en una serie de subobjetivos más concretos, los cuales se han ido alcanzado en las diversas etapas de desarrollo. A continuación, estos se presentan con una breve descripción general.

- **Añadir soporte para un mayor volumen de peticiones que además pueden ser**

concurrentes: Modificando la arquitectura del sistema para que permita contar con múltiples servidores de renderizado y desarrollando una *API REST* que se encargue de realizar las comprobaciones y las acciones necesarias para que todo funcione correctamente.

- **Minimizar los tiempos de espera de las peticiones que los usuarios pueden enviar:** Realizando un estudio de técnicas y mecanismos que puedan permitir reducir estos tiempos y comprobando sus efectos en estos tras su implementación en el sistema.
- **Permitir la administración y monitorización del sistema desplegado:** A través del desarrollo de un panel de administración que permita a un usuario con el rol de administrador realizar cambios en el sistema y consultar diversas estadísticas.

1.3. Tecnologías usadas

En esta sección, se presentan las tecnologías utilizadas a lo largo del desarrollo. La mayoría de estas coinciden con las que se utilizaron para el desarrollo de la aplicación original, puesto que no solo no son incompatibles con los objetivos del trabajo, sino que además constituyen las mejores opciones para alcanzarlos. Además, facilitará y agilizará la modificación y extensión de la aplicación.

Figura 1: Tecnologías utilizadas a lo largo del desarrollo

- **HTML5:**

HTML5 (*HyperText Markup Language*, versión 5) [15] es el lenguaje de marcado estándar para proporcionar estructura, semántica y contenido a una página web, por lo que resulta esencial en el desarrollo del nuevo cliente de administración.

- **CSS3 / Bootstrap:**

  CSS3 (*Cascading Style Sheets*, versión 3) [14] es el lenguaje estándar para definir las propiedades gráficas de los elementos definidos en el *HTML* de una página web.

  Más concretamente, permite proporcionar un estilo al contenido *HTML* del cliente de administración que aporte a la simplicidad y accesibilidad de la interfaz de usuario del administrador del sistema.

  Para simplificar el desarrollo, se recurre a la biblioteca *Bootstrap* [6], la cual proporciona una serie de clases *CSS* con propiedades predefinidas, las cuales se han reutilizado.

- **JavaScript:**

  JavaScript [17] es un lenguaje de programación muy versátil que se puede utilizar en multitud de ámbitos, siendo el principal el desarrollo web.

  Este lenguaje resulta adecuado para este desarrollo, puesto que constituye el estándar en los navegadores. Además, posee un gran apoyo por parte de la comunidad con la que cuenta, traduciéndose todo ello en una gran variedad de módulos que se aprovechan para la implementación de distintas funcionalidades. Una de las ventajas adicionales que posee, es que dispone de una abundante y detallada documentación.

  Su utilización conjunta con *Node.js* [18] resulta adecuada tanto para el desarrollo de los servidores de renderizado y los microservicios como para el de los clientes.

- **Node.js**

  Node.js es un entorno de ejecución de código *JavaScript*, el cual permite el desarrollo con este lenguaje para diferentes fines que no tienen por qué estar contenidos en un navegador web. Principalmente se utiliza en la capa del servidor de aplicaciones web.

  Este fue creado con el objetivo de facilitar el desarrollo de aplicaciones de red altamente escalables, ajustándose adecuadamente a los objetivos de este trabajo, principalmente debido a características de este entorno tales como el soporte multiplataforma, una arquitectura orientada a eventos, la asincronía en la entrada/salida de datos y su eficiencia y ligereza, entre otras.

  Se utiliza *Node.js* tanto en los servidores de renderizado y los microservicios como en los clientes.

- **Express.js:**

  Express.js [37] es el módulo de *Node.js* más utilizado para simplificar el desarrollo de *APIs* que sirvan peticiones *HTTP* sobre este entorno, permitiendo desarrollar una *API* funcional con muy pocas líneas de código.

  La aplicación original ya la utiliza tanto para servir tanto los *endpoints* de la *API* en el servidor como el contenido del cliente.

  El minimalismo, la rapidez y la flexibilidad de esta herramienta resultan adecuados para cumplir con los objetivos establecidos. Se utiliza para servir los *endpoints* de los distintos microservicios y de los servidores de renderizado.

- **Three.js:**

  Three.js [4] es una biblioteca de *JavaScript* que permite mostrar escenas y animaciones tridimensionales en el navegador. Esta se utiliza en el visor del cliente de los usuarios finales de la aplicación, para la carga de las escenas 3D desde fichero y para la exportación de la escena una vez el usuario genera una petición, funcionalidades ya implementadas en la aplicación original.

- **Draco:**

  Draco [1] es una biblioteca de compresión 3D desarrollada por *Google* que ayuda a reducir el tamaño de los modelos 3D en las aplicaciones que hacen uso de estos para que se carguen y muestren más rápida y eficientemente.

  Esta se utiliza para implementar una optimización de carácter opcional, con el propósito de disminuir los tiempos de transferencia de archivos en el sistema.

- **Visual Studio Code:**

  A lo largo del desarrollo se hace uso de *Visual Studio Code* [48], un *IDE* ligero de *Microsoft*, útil para muchos tipos de proyectos y tecnologías gracias a la posibilidad de instalar extensiones que permitan facilitar y agilizar el proceso de desarrollo.

- **Git / GitHub / GitHub Desktop:**

  Git [53] es una herramienta que posibilita mantener un control de versiones durante el desarrollo. Se hace uso de *GitHub* [30], una plataforma *SaaS* que permite tener una copia del repositorio en la nube, y *GitHub Desktop* [31], la aplicación de escritorio de *GitHub*, para acceder a *Git* a través de una interfaz gráfica en lugar de mediante comandos, agilizando así el proceso.

- **Blender:**

Blender [10] es una herramienta de código abierto que se utiliza para trabajar con gráficos tridimensionales. Aunque ofrece otras funcionalidades como el modelado, la iluminación y la animación de escenas 3D, la aplicación hace uso de la de renderizado en los servidores encargados de este proceso.

El programa ofrece *BPY* [11], una *API* que permite trabajar con este de forma automatizada mediante *scripts* escritos en *Python* sin tener que utilizar su interfaz gráfica. Esto constituye la base del funcionamiento del procesamiento en los servidores.

- **MongoDB / Mongoose:**

  Por cómo fue desarrollada la aplicación original, esta no requiere de persistencia. Sin embargo, para lograr los objetivos de este trabajo, es necesario que los distintos componentes tengan acceso a determinada información del sistema.

  Se hace uso de *MongoDB* [33], una base de datos no relacional muy eficiente, flexible, escalable y fácil de usar, motivos por los que se ha convertido en una de las principales opciones para proyectos basados en desarrollo web. Esta funciona mediante colecciones formadas por documentos, las cuales se utilizan tanto para guardar la información de cada servidor de renderizado registrado en el sistema como para cada petición de renderizado que realice un usuario final de la aplicación. Cada documento se corresponderá a una instancia de estos.

  También se utiliza *Mongoose* [43], una biblioteca *ODM* (*Object Document Mapper*) que permite trabajar fácilmente con *MongoDB* desde *JavaScript*.

1.4. Metodología de trabajo

A lo largo del desarrollo, se aplica la metodología *Scrum* [56]. Este marco de trabajo se engloba dentro de las *metodologías ágiles*, las cuales han demostrado ser útiles para la mayoría de proyectos de desarrollo de software, siendo actualmente estas las más extendidas en dicho ámbito.

Este paradigma sigue un enfoque iterativo y se fundamenta en la división del trabajo a realizar en distintos ciclos de desarrollo o *sprints*, cada uno con una serie de objetivos que se establecen al principio de estos.

Aunque esta forma de trabajar está más orientada a equipos de desarrollo, resulta sencillo adaptarla al contexto de este proyecto. Entrando más en detalle, posibilita enfocar las primeras iteraciones en desarrollar de forma incremental una primera versión del sistema con todas las funcionalidades requeridas, y una vez se disponga de esta, centrar las posteriores en ir mejorándolo con diversas optimizaciones, además de corregir errores que hayan surgido en

el desarrollo. Otra ventaja importante viene dada por la posibilidad de alterar aspectos como el diseño y los requisitos del sistema una vez se haya comenzado con la implementación del mismo.

1.5. Estructura del documento

En esta sección, se describe de forma breve y resumida los contenidos de cada capítulo en el que se ha estructurado esta memoria.

2. Estado del arte: Este capítulo describe el contexto dentro del ámbito en el cual se enmarca la aplicación, aportando una breve introducción al concepto de *granja de renderizado*, las posibles aplicaciones de estas, qué alternativas hay actualmente al sistema planteado, qué aspectos lo diferencian del resto, etc.

3. Análisis de los requisitos del sistema: En este capítulo se lleva a cabo un análisis de los nuevos requisitos que debe satisfacer el sistema. Adicionalmente, se detallan los casos de uso relacionados con el nuevo actor que surge con las nuevas funcionalidades que ofrece la aplicación, la figura del usuario administrador del sistema.

4. Modelado y diseño del sistema: Este capítulo incluye los distintas modelos, diagramas y decisiones que se han ido tomando en el planteamiento del sistema.

5. Estudio de posibles técnicas y mecanismos para la optimización de los tiempos de espera: Este capítulo consiste en la realización de un estudio en base a las diferentes formas de plantear algunos aspectos del sistema y de técnicas y mecanismos que se permitan aplicar, para así tratar de reducir los tiempos de espera de las peticiones de renderizado que vaya recibiendo.

6. Desarrollo iterativo: Este capítulo abarca todo lo relacionado con las distintas iteraciones en las que se ha dividido la implementación del sistema, tratando con detalle los objetivos de cada una, qué se ha ido implementando, problemas y dificultades que hayan ido surgiendo, etc.

7. Despliegue y pruebas: Este capítulo trata las distintas pruebas que se realizan sobre el sistema completo desplegado, así como los problemas que se descubran gracias a estas y las configuraciones de despliegue utilizadas.

8. Conclusiones y líneas futuras: Por último, una vez finalizado el trabajo, en este capítulo se incluye una retrospectiva sobre este. Por ello, se analiza si se han alcanzado los objetivos, los problemas y dificultades que hayan surgido y qué se podría haber hecho para evitarlos, además de posibles mejoras o funcionalidades que se le podrían añadir a la aplicación para líneas futuras.

2

Estado del arte

A lo largo de este capítulo, se proporciona una visión general del estado del arte en el área de la aplicación presentada. Dado que ya se realizó un estudio previo para el desarrollo de la base que conforma la aplicación, este estudio se realizará acorde con las nuevas características aportadas en este trabajo.

2.1. Contexto

El renderizado es un proceso que consiste en la generación de imágenes a partir de información, como pueden ser modelos tridimensionales, cámaras, texturas y fuentes de luz, mediante el uso de técnicas y algoritmos encargados de calcular el color de los píxeles que formarían la imagen resultante utilizando diversos enfoques.

Actualmente, existen multitud de técnicas de renderizado. Algunas, como la rasterización [58], resultan menos costosas y permiten la generación de imágenes en tiempo real a cambio de una menor fidelidad con respecto a lo que ocurre en la realidad. Algunos ejemplos podrían ser los gráficos de la mayoría de videojuegos, los visores de escenas 3D como el que hace uso de *Three.js* en la aplicación desarrollada en este trabajo, o el propio motor *Eevee* [13] de *Blender*. Estas también se denominan *técnicas de renderizado online [22]*.

Por otro lado, otras técnicas más realistas como las basadas en el trazado de rayos o *ray tracing* [57], intentan simular aspectos de la realidad como los rebotes de los rayos de luz o las interacciones de esta con distintos tipos de materiales, lo que implica ingentes cantidades de cálculos complejos, dando lugar a procesos computacionalmente costosos. Por ejemplo, en la aplicación se hace uso del motor *Cycles* [12] de *Blender*, el cual utiliza *Path Tracing* [5]. Estas también reciben el nombre de *técnicas de renderizado offline*.

Cabe destacar que, debido a la enorme velocidad de desarrollo de las técnicas de renderizado que existe actualmente (optimizaciones en los algoritmos, compromisos que no tienen un gran impacto en la calidad de la imagen pero que abaratan enormemente el coste computacional, paralelización, uso de inteligencia artificial...) y la disponibilidad de hardware cada vez

más potente, barato y optimizado para estas tareas, la línea de separación de esta clasificación es cada vez más difusa.

Cuando surge la necesidad de trabajar con *renderizado offline* de forma recurrente, la cantidad de tiempo que supondría obtener las imágenes renderizadas utilizando un equipo convencional resultaría excesiva, haciendo inviable esta opción.

Algunos campos donde la utilización de estas técnicas resulta esencial serían la animación, el desarrollo de efectos visuales, la arquitectura, el diseño de interiores, la medicina, la simulación computacional, el diseño de productos, etc.

La creciente demanda de renderización por parte de estas industrias propicia el surgimiento de las granjas de renderizado a finales de los años 80 y principios de los 90 como solución a este problema.

2.2. Granjas de renderizado

El término *granja de renderizado* o *render farm* hace referencia a un sistema consistente en una serie de equipos conectados entre sí (nodos) que trabajan juntos aportando su potencia de procesamiento con el objetivo de renderizar imágenes a partir de escenas 3D. Estos sistemas cuentan con software capaz de distribuir el trabajo entre los distintos nodos.

De esta forma, se aprovecha la paralelización que estas permiten para ahorrar grandes cantidades de tiempo con respecto a los que obtendríamos si utilizáramos un solo equipo convencional para la misma tarea.

La aplicación presentada en este trabajo se podría considerar un sistema de este tipo, ya que las principales funcionalidades añadidas son la posibilidad de utilizar múltiples servidores encargados de renderizar y de que el sistema sea capaz de distribuir el trabajo que le va llegando entre estos.

Se podrían distinguir los siguientes tipos de granja de renderizado, aunque también pueden existir sistemas híbridos con características de más de un tipo:

2.2.1. Basadas en la nube

Estas suelen ser ofrecidas de forma comercial como un servicio online en la nube, donde el usuario paga en función de sus necesidades. En este caso, los nodos se corresponden con servidores aportados por el proveedor que ofrece el servicio en su propia infraestructura remota desde el punto de vista del usuario, el cual accede a los servicios a través de internet.

2.2.2. *On-premises*

Los recursos de procesamiento se encuentran en las instalaciones de los usuarios de la granja. Debido a la gran inversión, tanto en recursos hardware, como en una infraestructura acondicionada, como en personal cualificado que se encargue de mantenerla, este tipo de granjas requieren una gran inversión, por lo que son más comunes en empresas que disponen de recursos suficientes para hacer frente a estos gastos.

2.2.3. Colaborativas

Estas granjas se basan en el concepto de *computación grid o en malla [54]*. En estas, la capacidad de procesamiento es proporcionada por cualquier usuario interesado (ya sea por altruismo o por recibir algún tipo de recompensa a cambio), que instala en su equipo un software que se encarga de hacer posible que este actúe como un nodo más de la granja.

Debido a su naturaleza, se caracterizan por contar con nodos más heterogéneos y una menor fiabilidad, ya que se tiene menor control sobre estos.

2.2.4. *Self-built*

Englobarían a las granjas locales de menor tamaño montadas de forma *ad-hoc* para un propósito concreto. Estas suelen ser más pequeñas debido a la gran inversión que requieren. Pueden ser una buena alternativa cuando no se precisa de una capacidad de procesamiento excesiva.

2.3. Principales opciones actuales

2.3.1. Granjas de renderizado basadas en la nube

Debido a su mayor seguridad, menor coste y facilidad de uso, los servicios en la nube han estado experimentando un enorme crecimiento en los últimos años. Por este motivo, no es de extrañar que las granjas basadas en la nube sean la alternativa más popular actualmente y que existan multitud de servicios y proveedores.

Las más conocidas son *RebusFarm* [59] y *Fox Renderfarm* [52]. Ambas ofrecen precios competitivos con distintos modelos de suscripción, además de una modalidad *pay-as-you-use* [65]. También cuentan con una gran infraestructura, son compatibles con la mayoría del software utilizado para el 3D (*Blender*, *Maya*, *Arnold*, *Cinema 4D* o *3ds Max*, entre otros) y permiten

escoger entre la utilización de *CPUs* o *GPUs* para el renderizado y la monitorización de los procesos que se ejecutan.

Atendiendo a los dos principales proveedores de servicios en la nube, *Amazon* y *Microsoft*, estos también ofrecen servicios relacionados.

- *Amazon* adquirió la compañía *Thinkbox Software* en 2017, integrándola bajo el paraguas de *Amazon Web Services* con el objetivo de fortalecer su presencia en el sector de los gráficos por computadora y ampliar su oferta de servicios en la nube para la industria creativa. Actualmente, *AWS Thinkbox* ofrece una colección de soluciones de software diseñadas para la administración de renderizado y la gestión de recursos tanto en la nube como *on-premises*.

  Se podría destacar *AWS Thinkbox Deadline* [25], un software de pago el cual ofrece funcionalidades como la administración y distribución de tareas de renderizado, el escalado automático y el seguimiento y recepción de notificaciones sobre el progreso de los trabajos y la disponibilidad de recursos.

- Por su parte, *Microsoft* ofrece a través de su plataforma *Azure*, por un lado, *Remote Rendering* [46], un servicio muy similar al de *PhotoSpaces*, el cual también se basa en el renderizado remoto para aliviar al dispositivo cliente del esfuerzo computacional, que obtiene mediante *streaming* las imágenes resultantes, permitiendo integrar este concepto en aplicaciones que hagan uso de gráficos 3D.

  Por otra parte, también ofrece una serie de tutoriales para aprovechar las instancias de máquinas virtuales en *Azure* tanto para convertir una granja *on-premises* ya existente en una híbrida, así como para montar de cero una granja basada en la nube incluyendo también algún software de gestión como podría ser la herramienta *AWS Thinkbox Deadline* mencionada anteriormente.

2.3.2. Granjas de renderizado colaborativas

Atendiendo a las granjas colaborativas, la más conocida corresponde con *SheepIt Render Farm* [8]. Esta se centra en el renderizado con *Blender* y permite a cualquier usuario interesado descargar una instancia del cliente que ofrece de forma gratuita para poder donar la capacidad de procesamiento de su equipo. Se encarga de instalar *Blender* en el caso de que aún no se encuentre en el sistema y permite configurar algunos parámetros (Véase por ello la Figura 2).

La herramienta permite añadir de forma gratuita un número indefinido de proyectos. Un proyecto consiste en un conjunto de escenas a renderizar, normalmente correspondientes a

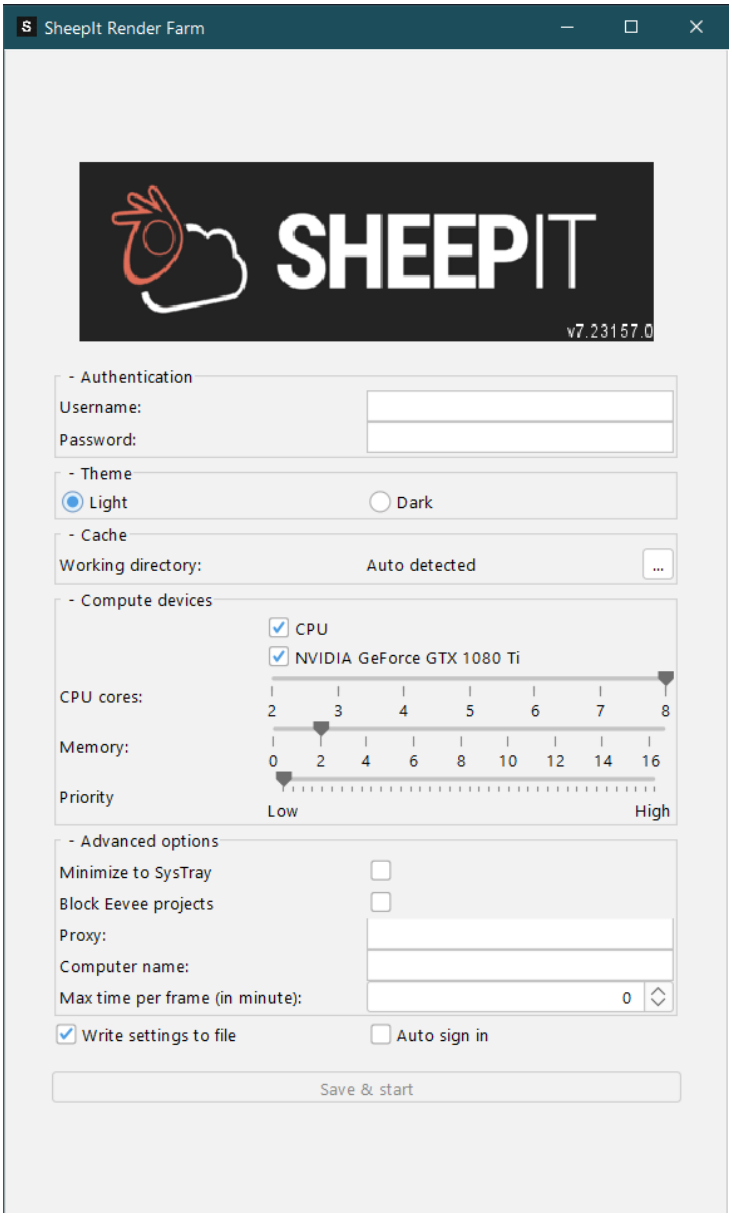

Figura 2: Parámetros configurables en el cliente de *SheepIt Render Farm*

una animación. Sin embargo, como máximo, solo es posible tener dos de estos renderizando al mismo tiempo.

Además, utiliza un sistema basado en puntos en el que la puntuación de un usuario se irá incrementando conforme este vaya renderizando los proyectos de otros usuarios, y se irán consumiendo a medida que otros usuarios rendericen los suyos. Si un usuario no dispone de puntos, sus proyectos no podrán ser renderizados por el resto de usuarios. En la Figura 3, es posible visualizar una sesión activa.

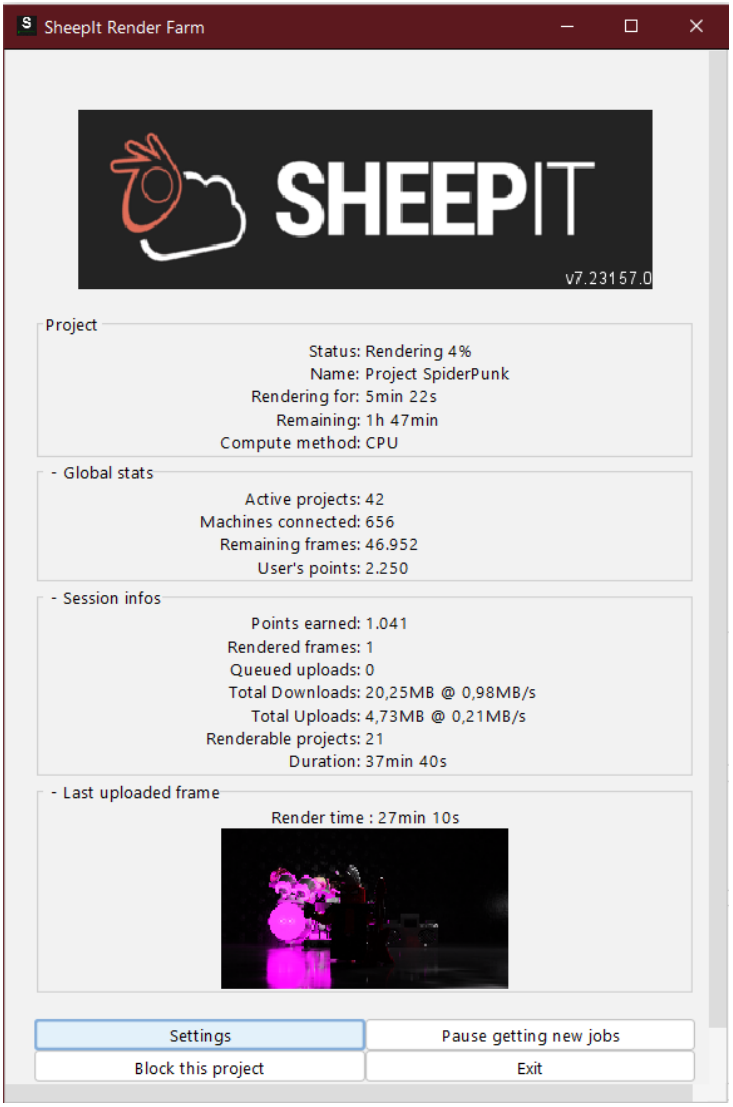

Figura 3: Ejemplo de sesión de renderizado en *SheepIt Render Farm*

2.4. Qué aporta *PhotoSpaces* en este contexto

El resultado tras el desarrollo realizado en este trabajo no va a repercutir considerablemente al lugar que ocupa *PhotoSpaces* de cara a cualquier perfil de usuario que pudiera estar interesado en el servicio que ofrece la aplicación, ya que los objetivos no se centran en las funcionalidades ofrecidas a estos, las cuales, salvo por algunos cambios y adiciones menores, van a ser las mismas.

Los atractivos de la aplicación seguirán siendo su simplicidad y facilidad de uso frente a otras herramientas, así como la versatilidad que proporciona su visor 3D y la posibilidad de editar la escena, todo esto desde la comodidad de una interfaz web. Una de las características más notorias que se verá afectada será la experiencia de estos, al encontrarse con un sistema más rápido y con tiempos de espera menores.

Sin embargo, desde el punto de vista del administrador, puede resultar bastante interesante analizar el sistema teniendo en cuenta los distintas granjas anteriormente citadas en los apartados anteriores.

Centrándonos en el aspecto de administración de servidores, el sistema ofrece una facilidad de uso elevada. Para añadir un nodo, el usuario solo tiene que indicar la dirección *IP* del equipo que quiere añadir como nodo a la granja, teniendo en cuenta que en este equipo debe haberse instalado con anterioridad el software necesario para que el sistema permita añadirlo.

Esta forma de abordar el registro de nodos, junto al hecho de que las comunicaciones utilicen *HTTP* y de que el sistema sea modular gracias a la utilización de microservicios, aportan a la aplicación uno de sus principales atractivos, la flexibilidad que ofrece pese a su simplicidad. Esta libertad de acción hace posible la construcción de distintos tipos de granjas que actúen como *backend* de *PhotoSpaces*, siempre teniendo en cuenta que la configuración de red de los distintos nodos debe ser correcta.

- Por un lado, es posible decantarse hacia el concepto de granja basada en la nube contratando instancias con algún proveedor *cloud*, instalándoles el software necesario y registrándolas en el sistema.

- Por otro lado, también sería posible optar por un enfoque *on-premises* o *self-built* y utilizar los equipos de los que se disponga físicamente.

- También se podría seguir el paradigma colaborativo. Cualquier persona interesada podría preparar su equipo para actuar como un nodo más y proporcionar su dirección *IP* al administrador para que este procediera a registrarlo en el sistema.

Relacionado con esta posibilidad, resulta interesante comparar el paradigma de las granjas que son colaborativas por diseño, en las que suele ser el usuario interesado en aportar capacidad de procesamiento el que solicita registrar su equipo en la granja mediante la instalación del cliente correspondiente, con el de *PhotoSpaces*, en el cual el sistema es el encargado de contactar con los nodos para registrarlos.

- También se podría optar por montar una granja híbrida combinando estas posibilidades.

3

Análisis de los requisitos del sistema

3.1. Requisitos del sistema

En esta sección se enumeran y detallan los nuevos requisitos que debe cumplir el sistema de forma tan clara como sea posible y tratando de evitar ambigüedades, pero sin entrar en cómo se va a abordar su implementación.

3.1.1. Requisitos funcionales iniciales

En primer lugar, se enuncian los requisitos fundamentales establecidos desde el inicio del desarrollo, los cuales corresponden a las funcionalidades esenciales que se han decidido incluir en el sistema desde el principio. Estos se corresponderían con los siguientes:

- **RF01. Manejo de peticiones** El sistema deberá ser capaz de manejar todas las peticiones de renderizado que le lleguen de sus usuarios, lo que implica que:
  - El sistema deberá poder determinar si una petición tiene que ser encolada o si puede ser enviada directamente a un servidor de renderizado.
  - Cuando una petición pueda ser enviada a un servidor, el sistema deberá ser capaz de determinar cuál es el servidor idóneo en ese momento.
  - El sistema deberá asegurarse de que siempre que haya servidores disponibles se les envíe una petición de la cola si esta no está vacía, es decir, tratará de que no se llegue a una situación en la que haya peticiones encoladas y servidores ociosos.
- **RF02. Consulta de información del sistema actualizada** El sistema permitirá consultar información sobre los servidores y las peticiones de renderizado registrados. Los

datos que incluirá se detallan a continuación:

  - Por cada servidor registrado en el sistema, su estado actual, su nombre, su dirección *IPv4* y el sistema operativo, procesador, tarjeta gráfica y versión de *Blender* con los que cuenta.
  - Por cada petición registrada en el sistema, su estado actual, su identificador y la dirección *IPv4* del cliente que la ha enviado.
  - Por cada petición que esté siendo procesada en el momento de consulta de la información, el nombre del servidor que se le ha asignado, el tiempo que lleva siendo procesada en el servidor y una estimación del tiempo de espera restante hasta que finalice el proceso de renderizado.
  - Por cada petición que se encuentre encolada en el momento de consulta de la información, su posición en la cola y el tiempo que lleva esperando en esta.
  - Por cada petición que ya haya sido servida en el momento de consulta de la información, el servidor donde ha sido procesada, el tiempo total que ha estado esperando en cola y el tiempo total que ha pasado siendo procesada.

- **RF03. Añadir un nuevo servidor al sistema** El sistema permitirá a un usuario administrador registrar un nuevo servidor en el sistema mediante la entrada vía formulario web de su dirección *IPv4* y un nombre a su elección para poder identificarlo fácilmente, siempre que se cumplan las siguientes condiciones:

  - La dirección *IPv4* es válida.
  - El servidor correspondiente a la dirección *IPv4* demuestra que es capaz de llevar a cabo el proceso de renderizado.
  - No existe un servidor con el mismo nombre o con la misma dirección *IPv4* ya registrado en el sistema.

- **RF04. Eliminar un servidor del sistema** El sistema permitirá a un usuario administrador desvincular de este un servidor previamente registrado siempre que este no se encuentre procesando una petición en ese momento, lo cual implicará la eliminación de la información correspondiente a este y la imposibilidad de enviarle peticiones sin pasar por el proceso de registro de nuevo.

- **RF05. Deshabilitar un servidor registrado en el sistema** El sistema permitirá a un usuario administrador deshabilitar un servidor previamente registrado siempre que este no se encuentre procesando una petición, un servidor en este estado seguirá siendo visible por el sistema, pero no será tenido en cuenta para el envío de peticiones hasta que vuelva a ser habilitado.

- **RF06. Habilitar un servidor previamente deshabilitado registrado en el sistema** El sistema permitirá a un usuario administrador volver a habilitar un servidor previamente deshabilitado, el cual volverá a ser tenido en cuenta para el envío peticiones.

- **RF07. Abortar el procesamiento en curso de una petición de renderizado en un servidor concreto** El sistema permitirá a un usuario administrador abortar el procesamiento en curso de una petición en un servidor y deberá asegurarse de manejar adecuadamente la petición abortada, que pasará al final de la cola.

- **RF08. Eliminar una petición del sistema** El sistema permitirá a un usuario administrador eliminar cualquier petición del sistema independientemente del estado en el que se encuentre esta en ese momento.

    - Si la petición se encuentra encolada o ya ha sido satisfecha, simplemente se eliminará su información del sistema.
    - En cambio, si estaba siendo procesada por un servidor de renderizado, será necesario abortar antes el procesamiento en dicho servidor.

3.1.2. Requisitos funcionales emergentes

A lo largo del desarrollo, han surgido una serie de requisitos funcionales adicionales que complementan a los previamente detallados, además de la necesidad de realizar modificaciones en algunos de estos. Estos nuevos requisitos y cambios se detallan a continuación:

- **Modificación RF01. Manejo de peticiones** Este requisito se mantendrá inalterado, excepto en el aspecto de asignación de servidores, debido a la optimización relacionada con la paralelización de las transferencias de ficheros detallada en la sección 5.2.3.

  El sistema ya no deberá ser capaz de asignar el servidor cuando una petición va a ser extraída de la cola, sino en el momento en el que esta es recibida, de forma que sea posible comenzar con la transferencia, dando lugar al hecho de que se distinga una cola diferente por cada servidor de renderizado.

- **Modificación RF02. Consulta de información del sistema actualizada** Debido en parte a las optimizaciones incorporadas, el sistema permitirá a un usuario administrador consultar información adicional a la especificada anteriormente, más concretamente datos relacionados con el estado de las transferencias de ficheros, el rendimiento de los servidores y los parámetros especificados por el usuario para cada renderizado.

La información concreta que se ha acabado añadiendo para satisfacer este requisito se puede consultar en las secciones B.3.1 y B.3.2 del anexo correspondiente al manual de usuario de la aplicación final.

- **RF09. Selección de la resolución a la que se llevará a cabo el renderizado** El sistema permitirá a un usuario final seleccionar a qué resolución se llevará a cabo el proceso de renderizado antes de generar una petición, que será la que tenga la imagen resultante que obtendrá.

  Este podrá elegir entre las siguientes resoluciones del formato 16:9: *480p*, *720p*, *1080p*, *1440p* y *2160p*.

- **RF10. Descarga de imágenes renderizadas desde el panel de administración** El sistema permitirá a un usuario administrador descargar las imágenes renderizadas asociadas a las peticiones generadas previamente por los usuarios finales de la aplicación. Estas imágenes estarán disponibles para descarga una vez que hayan sido procesadas.

- **RF10. Generación de peticiones de renderizado mediante interfaz de línea de comandos** Además de mediante el uso de la interfaz web que ya ofrecía, el sistema permitirá a un usuario final generar una petición de renderizado mediante una interfaz de línea de comandos adicional equivalente. Los parámetros necesarios para la petición no sufrirán cambios, pero deberán ser introducidos en formato textual.

3.1.3. Requisitos no funcionales

- **Facilidad de uso** De acuerdo con los requisitos originales de la aplicación, esta se tratará de mantener simple, intuitiva y fácil de utilizar para el usuario.

- **Facilidad de despliegue** El despliegue del sistema no deberá resultar excesivamente complejo, de forma que cualquier usuario con acceso a la guía de instalación sea capaz de ponerlo en funcionamiento sin problemas.

- **Robustez** El sistema deberá ser robusto, es decir, se tratará de evitar que un pequeño error en alguna parte del sistema o una entrada no esperada provoquen un fallo del sistema completo.

- **Errores descriptivos** En caso de que algo vaya mal tratando de satisfacer una petición de cualquier usuario, este deberá ser informado del error exacto que se ha producido.

- **Seguridad** Se tendrá en cuenta la seguridad del sistema durante su desarrollo, tratando de evitar posibles vulnerabilidades y restringiendo lo máximo posible las comunicacio-

nes de forma que cada componente sepa de qué direcciones *IP* pueden llegarle peticiones *HTTP* y rechace cualquiera proveniente de otra dirección.

3.2. Casos de uso del administrador del sistema

La documentación de casos de uso es un proceso fundamental en el desarrollo de software, ya que desempeña un papel crucial en la captura y comunicación de los requisitos funcionales del sistema desde la perspectiva del usuario.

Por este motivo, a continuación se documentan los principales casos de uso relacionados con el nuevo actor del sistema: el usuario administrador que interactúa con este a través del panel de administración.

Además, el resultado podrá ser utilizado como referencia durante todo el ciclo de desarrollo para asegurar que lo que se está desarrollando se ajusta a lo que se ha especificado inicialmente.

Por cada caso de uso, se incluye un identificador, un título, su actores, una descripción, sus precondiciones y postcondiciones, el flujo que sigue el escenario de éxito principal, flujos de escenarios alternativos o de excepción en caso de haberlos y, finalmente, el diagrama *UML* de casos de uso correspondiente.

3.2.1. CU01. Añadir un nuevo servidor al sistema

Actores: Usuario administrador (actor principal), servidor de renderizado (actor secundario).

Descripción: Registrar un nuevo servidor de renderizado en el sistema indicando un nombre para identificarlo y su dirección *IP* para que sea utilizado por este para procesar peticiones de renderizado.

Precondiciones: El usuario administrador se encuentra en el panel de administración.

Flujo del escenario de éxito principal:

1. El usuario administrador proporciona un nombre y dirección *IP* para el servidor.

2. El sistema valida los valores recibidos.

3. El sistema contacta con el servidor de renderizado especificado.

4. El servidor de renderizado realiza una prueba de renderizado.

5. El servidor de renderizado proporciona sus especificaciones.

6. El sistema guarda la información recibida del servidor.

7. El sistema informa al usuario administrador sobre el éxito de la operación.

8. Fin del caso de uso.

Flujo de escenario de excepción. Parámetros no válidos

2E.1. El sistema detecta valores no válidos.

2E.2. El sistema informa al usuario administrador del problema. Ir al paso 8.

Flujo de escenario de excepción. Error relacionado con el servidor de renderizado

(3-6)E.1. Se produce un error en el servidor o en la comunicación con este.

(3-6)E.2. El sistema informa al usuario administrador del problema. Ir al paso 8.

Flujo de escenario de excepción. Error relacionado con la base de datos

7E.1. Se produce un error en la base de datos o en la comunicación con esta.

7E.2. El sistema informa al usuario administrador del problema. Ir al paso 8.

Postcondiciones: Al final del escenario de éxito, el servidor indicado habrá quedado vinculado al sistema y su información correspondiente persistida y visible desde el panel de administración.

Diagrama: Figura 4

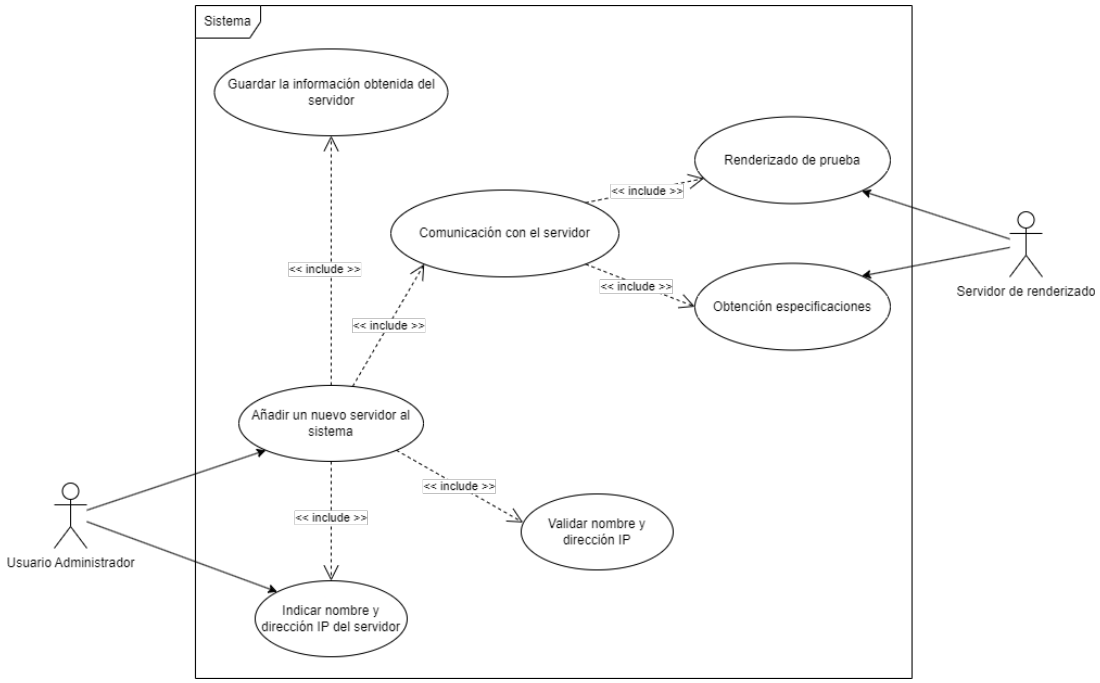

Figura 4: Diagrama para el caso de uso *Añadir un nuevo servidor al sistema*

3.2.2. CU02. Eliminar un servidor del sistema

Actores: Usuario administrador (actor principal), servidor de renderizado (actor secundario).

Descripción: Desvincular un servidor de renderizado previamente registrado del sistema, eliminando su información correspondiente y dejando este de ser tenido en cuenta a la hora de reenviar peticiones de renderizado.

Precondiciones: El usuario administrador se encuentra en el panel de administración y el servidor que quiere eliminar se muestra en este, es decir, ha sido registrado previamente en el sistema.

Flujo del escenario de éxito principal:

1. El usuario administrador indica el servidor que desea eliminar.

2. El sistema solicita la confirmación del usuario para proceder con la eliminación.

3. El usuario administrador confirma su intención de continuar.

4. El sistema inicia el proceso de desvinculación del servidor de renderizado.

5. El servidor de renderizado queda desvinculado correctamente.

6. El sistema elimina la información relacionada con el servidor.

7. El sistema notifica al usuario administrador sobre el éxito de la operación.

8. Fin del caso de uso

Flujo de escenario de excepción. Error relacionado con el servidor de renderizado

(4-5)E.1. Se produce un error en el servidor de renderizado o en la comunicación con este. Ir al paso 6.

Flujo de escenario de excepción. Error relacionado con la base de datos

6E.1. Se produce un error en la base de datos o en la comunicación con esta.

6E.2. El sistema informa al usuario administrador del problema. Ir al paso 8.

Postcondiciones: Al final del principal escenario de éxito, el servidor indicado habrá quedado desvinculado del sistema y su información correspondiente eliminada.

Diagrama: Figura 5

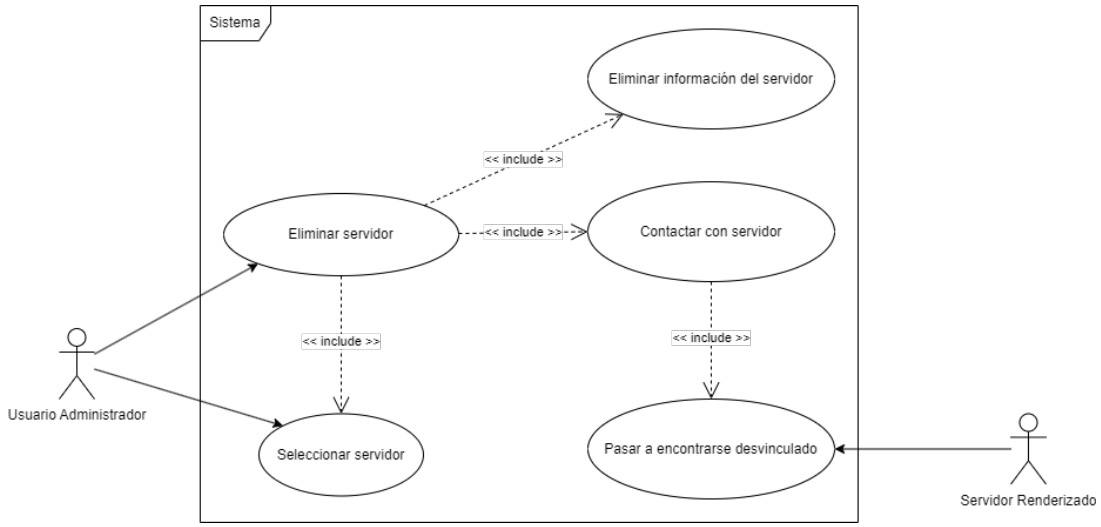

Figura 5: Diagrama para el caso de uso *Eliminar un servidor del sistema*

3.2.3. CU03. Deshabilitar un servidor registrado en el sistema

Actores: Usuario administrador (actor principal), servidor de renderizado (actor secundario).

Descripción: Deshabilitar un servidor disponible previamente registrado para que el sistema deje de ser tenido en cuenta a la hora de reenviar peticiones.

Precondiciones: El usuario administrador se encuentra en el panel de administración y el servidor que quiere deshabilitar se muestra en este, figurando como disponible.

Flujo del escenario de éxito principal:

1. El usuario administrador selecciona el servidor que desea deshabilitar.

2. El sistema solicita una confirmación adicional para proceder.

3. El usuario administrador confirma su decisión.

4. El sistema inicia el proceso de deshabilitación del servidor de renderizado.

5. El servidor de renderizado es deshabilitado correctamente.

6. El sistema actualiza la información relacionada con el servidor.

7. El sistema notifica al usuario administrador sobre el éxito de la operación.

8. Fin del caso de uso

Flujo de escenario de excepción. Error relacionado con el servidor de renderizado

(4-5)E.1. Se produce un error en el servidor o en la comunicación con este.

(4-5)E.2. El sistema informa al usuario administrador del problema. Ir al paso 8.

Flujo de escenario de excepción. Error relacionado con la base de datos

6E.1. Se produce un error en la base de datos o en la comunicación con esta.

6E.2. El sistema informa al usuario administrador del problema. Ir al paso 8.

Postcondiciones: Al final del principal escenario de éxito, el servidor indicado habrá quedado deshabilitado del sistema y su información correspondiente actualizada reflejando el cambio.

Diagrama: Figura 6

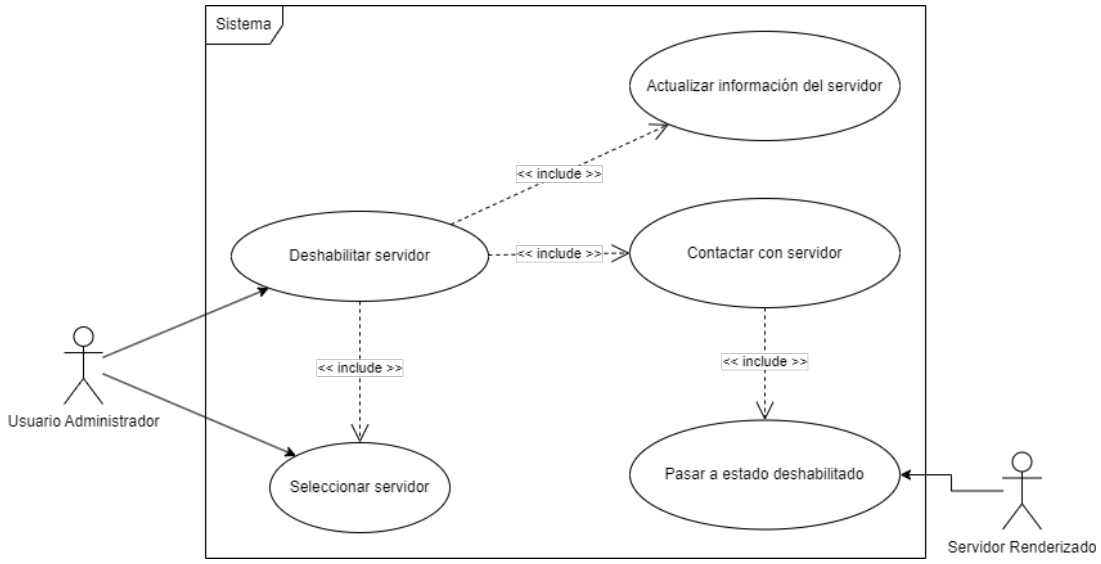

Figura 6: Diagrama para el caso de uso *Deshabilitar un servidor registrado en el sistema*

3.2.4. CU04. Habilitar un servidor previamente deshabilitado registrado en el sistema

Actores: Usuario administrador (actor principal), servidor de renderizado (actor secundario).

Descripción: Habilitar un servidor previamente registrado en el sistema y deshabilitado para que vuelva a ser tenido en cuenta a la hora de reenviar peticiones.

Precondiciones: El usuario administrador se encuentra en el panel de administración y el servidor que quiere habilitar se muestra en este, figurando como deshabilitado.

Flujo del escenario de éxito principal:

1. El usuario administrador selecciona el servidor que desea habilitar.

2. El sistema solicita una confirmación adicional para proceder.

3. El usuario administrador confirma su decisión.

4. El sistema inicia el proceso de habilitación del servidor de renderizado.

5. El servidor de renderizado es habilitado correctamente.

6. El sistema actualiza la información relacionada con el servidor.

7. El sistema notifica al usuario administrador sobre el éxito de la operación.

8. Fin del caso de uso

Flujo de escenario de excepción. Error relacionado con el servidor de renderizado

(4-5)E.1. Se produce un error en el servidor o en la comunicación con este.

(4-5)E.2. El sistema informa al usuario administrador del problema. Ir al paso 8.

Flujo de escenario de excepción. Error relacionado con la base de datos

6E.1. Se produce un error en la base de datos o en la comunicación con esta.

6E.2. El sistema informa al usuario administrador del problema. Ir al paso 8.

Postcondiciones: Al final del principal escenario de éxito, el servidor indicado volverá a encontrarse habilitado y su información correspondiente actualizada reflejando el cambio.

Diagrama: Figura 7

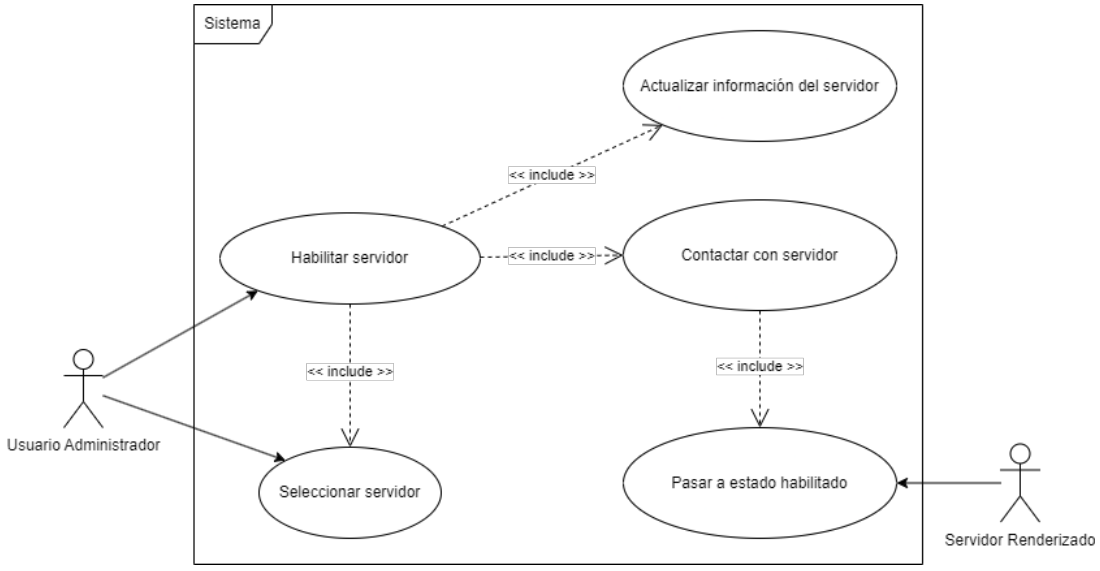

Figura 7: Diagrama para el caso de uso *Habilitar un servidor previamente deshabilitado registrado en el sistema*

3.2.5. CU05. Abortar el procesamiento en curso de una petición de renderizado en un servidor concreto

Actores: Usuario administrador (actor principal), servidor de renderizado (actor secundario).

Descripción: Interrumpir el procesamiento de una petición en un servidor de renderizado cuando este se encuentra ocupado atendiéndola.

Precondiciones: El usuario administrador se encuentra en el panel de administración, el servidor donde se va a abortar el procesamiento se encuentra atendiendo una petición y figura en el panel de administración como ocupado.

Flujo del escenario de éxito principal:

1. El usuario administrador selecciona el servidor en el que desea abortar el procesamiento.

2. El sistema solicita una confirmación adicional al usuario.

3. El usuario administrador confirma su decisión.

4. El sistema se comunica con el servidor de renderizado.

5. El servidor interrumpe el proceso de renderizado.

6. El sistema actualiza la información relacionada con el servidor.

7. El sistema encola de nuevo la petición que ha sido interrumpida.

8. El sistema notifica al usuario administrador sobre el éxito de la operación.

9. Fin del caso de uso

Flujo de escenario de excepción. Error relacionado con el servidor de renderizado

(4-5)E.1. Se produce un error en el servidor o en la comunicación con este.

(4-5)E.2. El sistema informa al usuario administrador del problema. Ir al paso 9.

Flujo de escenario de excepción. Error relacionado con la base de datos

(6-7)E.1. Se produce un error en la base de datos o en la comunicación con esta.

(6-7)E.2. El sistema informa al usuario administrador del problema. Ir al paso 9.

Postcondiciones: Al finalizar el escenario de éxito, la petición que estaba siendo procesada por el servidor indicado habrá sido interrumpida y encolada de nuevo. La respuesta del servidor a esta interrupción dependerá de la situación del sistema, siendo los posibles casos:

- El servidor podrá pasar a encontrarse disponible para procesar nuevas peticiones.

- El servidor podrá comenzar a procesar una petición distinta que se encontrara encolada y asignada este.
- El servidor podrá reiniciar el procesamiento de la misma petición desde el principio al ser esta reenviada de nuevo.

Diagrama: Figura 8

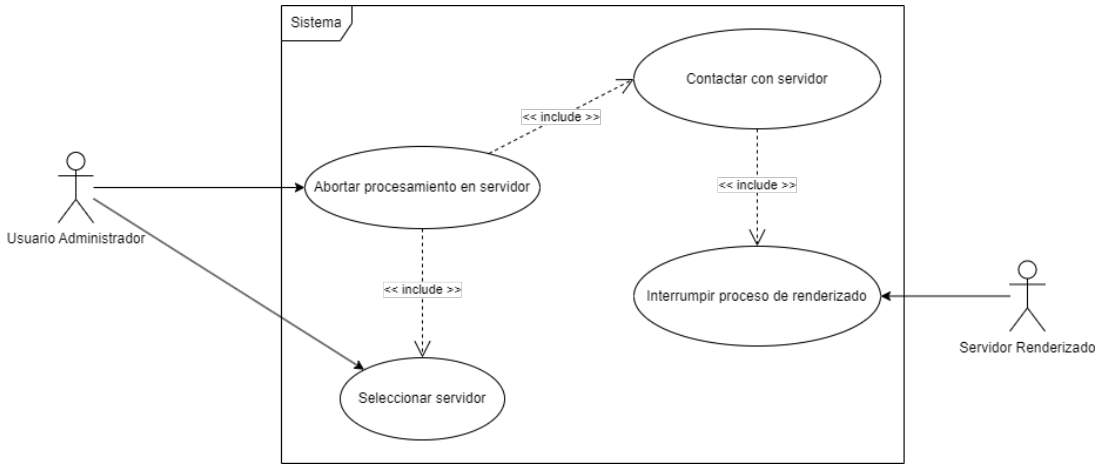

Figura 8: Diagrama para el caso de uso *Abortar el procesamiento en curso de una petición de renderizado en un servidor concreto*

3.2.6. CU06. Eliminar una petición del sistema

Actores: Usuario administrador (actor principal), servidor de renderizado (actor secundario, solo en escenario alternativo).

Descripción: Eliminar una petición de renderizado del sistema, borrando la información correspondiente a esta y evitando su procesamiento si esta no ha sido procesada aún.

Precondiciones: La petición ha sido generada con anterioridad por un usuario y por lo tanto se muestra en el panel de administración.

Flujo del escenario de éxito principal. Petición finalizada o encolada

1. El usuario administrador selecciona la petición que desea eliminar.

2. El sistema solicita una confirmación adicional al usuario.

3. El usuario administrador confirma su decisión.

4. El sistema comprueba que la petición no está siendo procesada en un servidor en ese momento.

5. El sistema elimina la información correspondiente a la petición.

6. El sistema notifica al usuario administrador sobre el éxito de la operación.

7. Fin del caso de uso

Flujo de escenario de éxito alternativo. Petición en proceso

4A.1. El sistema comprueba que la petición está siendo procesada en un servidor en ese momento.

4A.2. El sistema se comunica con el servidor de renderizado.

4A.3. El servidor interrumpe el proceso de renderizado.

4A.4. El sistema actualiza la información relacionada con el servidor. Ir al paso 5.

Flujo de escenario de excepción. Error relacionado con el servidor de renderizado

(4A.2-4A.3)E.1. Se produce un error en el servidor o en la comunicación con este.

(4A.2-4A.3)E.2. El sistema informa al usuario administrador del problema. Ir al paso 7.

Flujo de escenario de excepción. Error relacionado con la base de datos

5E.1. Se produce un error en la base de datos o en la comunicación con esta.

5E.2. El sistema informa al usuario administrador del problema. Ir al paso 7.

Postcondiciones: Al final de cualquiera de los escenarios de éxito, se habrá eliminado la información relacionada con la petición del sistema. En el caso del escenario de éxito alternativo en el que la petición está siendo procesada, el procesamiento habrá sido abortado en el servidor correspondiente.

Diagrama: Figura 9

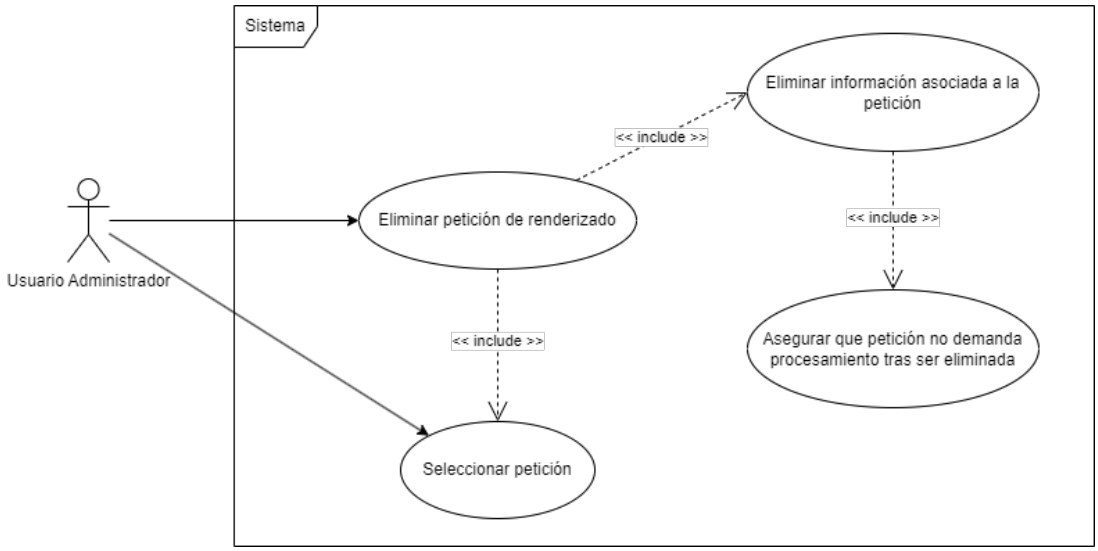

Figura 9: Diagrama para el caso de uso *Eliminar una petición del sistema*

3.2.7. CU07. Descargar la imagen renderizada asociada a una petición

Actores: Usuario administrador (actor principal).

Descripción: Descargar la imagen renderizada resultante del procesamiento de una petición ya finalizada.

Precondiciones: La petición ha sido generada con anterioridad por un usuario, esta ya ha sido procesada por un servidor de renderizado y, por lo tanto, se muestra en el panel de administración figurando como finalizada.

Flujo del escenario de éxito principal:

1. El usuario administrador selecciona la petición cuya imagen desea descargar.

2. El sistema localiza el fichero solicitado.

3. El sistema inicia la transferencia del fichero.

4. La transferencia finaliza correctamente.

5. Fin del caso de uso.

Flujo de escenario de excepción. Error en la localización del fichero

2E.1. El sistema no logra encontrar el fichero solicitado.

2E.2. El sistema informa al usuario administrador del problema. Ir al paso 5.

Flujo de escenario de excepción. Error en la transferencia del fichero

4E.1. Se produce un error durante la transferencia del fichero.

4E.2. El sistema informa al usuario administrador del problema. Ir al paso 5.

Postcondiciones: Al finalizar el escenario de éxito, el usuario administrador dispondrá de una copia en local de la imagen renderizada que ha solicitado.

Diagrama: Figura 10

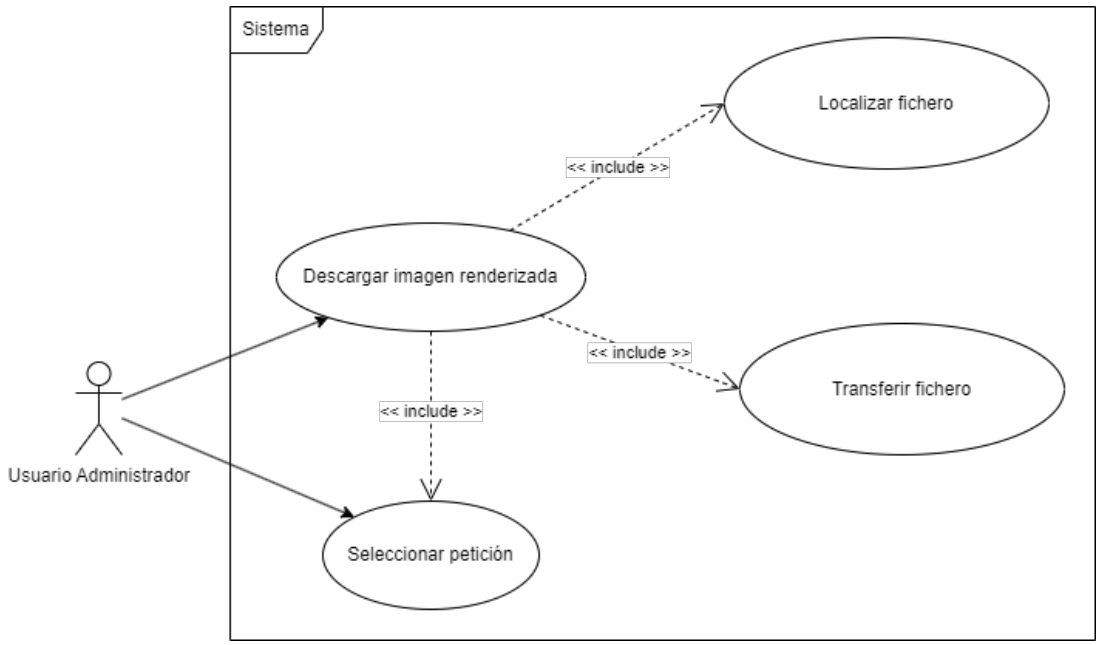

Figura 10: Diagrama para el caso de uso *Descargar la imagen renderizada asociada a una petición*

4

Modelado y diseño del sistema

Este capítulo tiene como objetivo proporcionar una visión general tanto del problema abordado, así como del sistema desarrollado y las distintas decisiones que se han ido tomando a la hora de plantear el mismo.

Desde una perspectiva de un alto nivel de abstracción, la solución desarrollada se basa en la existencia de una *API REST* que funcione como intermediaria entre el cliente actual y el nuevo panel de administración de la aplicación y los distintos servidores de renderizado. Esta *API* facilitará la comunicación y la transferencia de información entre ambos componentes del sistema mediante el protocolo *HTTP*.

Con esta idea, se plantean una serie de incógnitas relacionadas con cómo abordar la implementación, las cuales se irán desarrollando en los siguientes apartados.

4.1. Dominio de la aplicación

Aunque a lo largo de la implementación no se han utilizado ni las funcionalidades de programación orientada a objetos que ofrece *JavaScript* ni el concepto de clase como tal, resulta esencial tener una idea clara del problema que se está tratando de solucionar antes de intentar abordarlo.

Por este motivo, es buena práctica elaborar un modelo del dominio de la aplicación que, además de ayudar a entender sus diferentes elementos, sus relaciones y la lógica que los rige, servirá como referencia durante la implementación. Un claro ejemplo podría ser utilizarlo para determinar qué información interesa manejar en el sistema.

Para elaborar el modelo, se ha utilizado un diagrama de clases *UML* realizado con la herramienta *USE*, el cual es posible visualizar en la Figura 11.

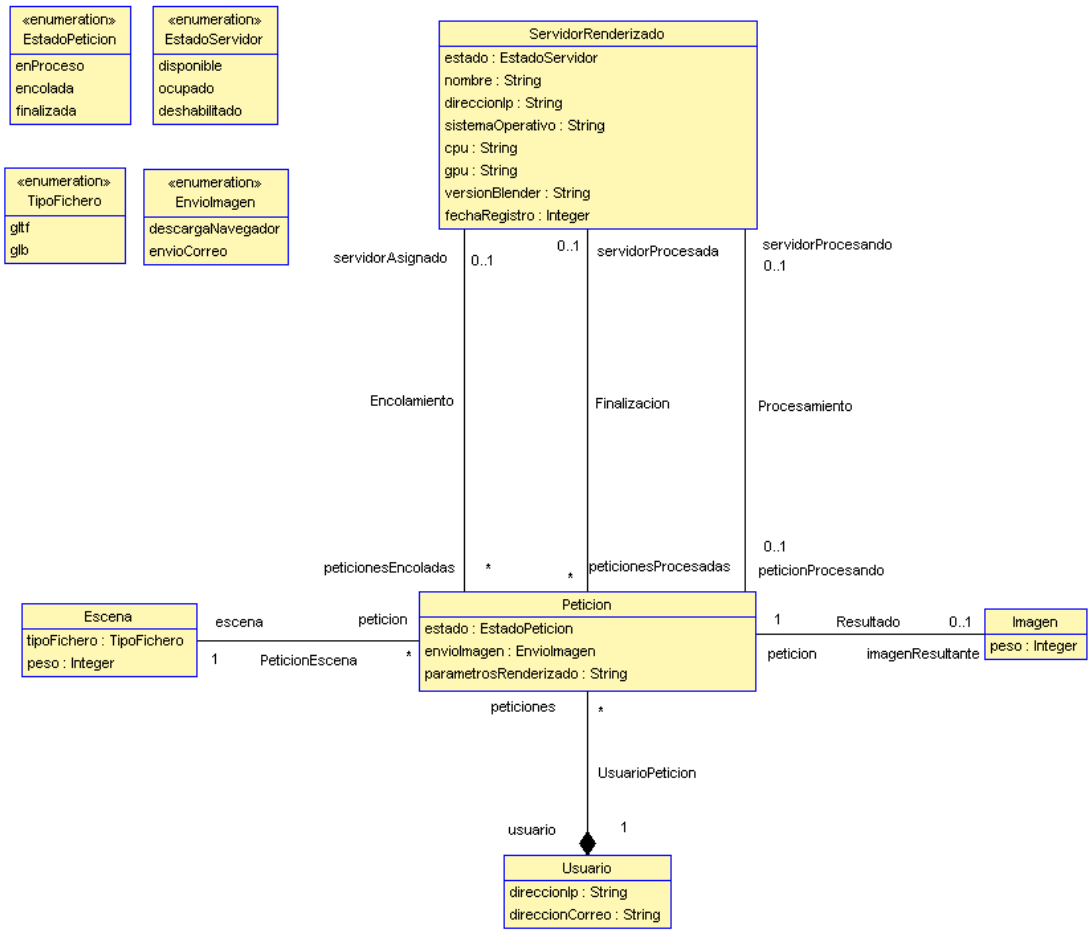

Figura 11: Diagrama de clases que incluye los distintos elementos del dominio de la aplicación y las relaciones entre estos

4.2. Arquitectura de la aplicación

Otro aspecto esencial se corresponde con la arquitectura que va a seguir el sistema.

4.2.1. Arquitectura de microservicios

En este caso, se ha optado por seguir una arquitectura basada en microservicios [49], la cual consiste en dividir el *backend* de la aplicación en una serie de servicios independientes que operan de manera autónoma y se comunican mediante *APIs*.

Frente al enfoque monolítico, en el cual se desarrollan las funcionalidades y componentes del sistema de una forma fuertemente acoplada para que todos estos se ejecuten en un solo proceso, este enfoque resulta más adecuado para el alcance de los objetivos planteados. De

todas las ventajas que ofrece, es posible beneficiarse principalmente de las siguientes:

- **Modularidad:** Esta se ajustaría a la idea de contar con servidores independientes, además de permitir una variedad mucho más amplia de configuraciones para el despliegue al tener la opción de seleccionar dónde se desea desplegar cada componente.

  También ofrece la posibilidad de utilizar distintas tecnologías en los microservicios. Esto permite tener separadas las dependencias que requiera cada componente, sus variables de entorno y sus despliegues.

- **Escalabilidad y alta disponibilidad:** Permitiendo el escalado horizontal independiente de cada componente desplegando más instancias de estos.

- **Mayor facilidad de mantenimiento:** El alcance más limitado de los microservicios y el hecho de que la modificación en uno no afecta al resto en la mayoría de los casos hará más sencillo el desarrollo de nuevas funcionalidades y la introducción de cambios.

Sin embargo, es necesario tener en cuenta que la adopción de este enfoque no solo aporta beneficios, sino que también supone una serie de inconvenientes a los que surge la necesidad de hacer frente. En el contexto de la aplicación presentada en este trabajo, destacan los siguientes:

- **Mayor número de comunicaciones en el sistema:** Es posible generar una sobrecarga de comunicaciones si estos necesitan comunicarse frecuentemente.

  Esto no supondrá un gran problema, ya que en principio solo se requerirán dos comunicaciones *HTTP* (de gran simplicidad y cuyo fallo no tiene consecuencias importantes, detallado en las especificaciones de las *API*s *REST*) entre los dos microservicios, ya que ambos obtienen toda la información que necesitan de los clientes, de la base de datos y de los servidores de renderizado, no del otro microservicio.

- **Mayor dificultad de monitorización del sistema durante el desarrollo:** Este proceso puede resultar más laborioso, al tener que estar pendientes de todos los componentes por separado.

4.2.2. Componentes del sistema

Se ha tratado de mantener una cantidad reducida de microservicios. Aunque se podrían separar aún más las responsabilidades, se ha considerado una buena opción limitarse a la implementación de dos microservicios diferenciados, donde cada uno de ellos se encargará de dar soporte a las peticiones provenientes de un cliente concreto de la aplicación.

De acuerdo con esta idea, se obtiene un sistema con los siguientes componentes, a los cuales se asignarán nombres para identificarlos fácilmente a lo largo de esta memoria:

- **Cliente estándar:** Se encarga de poner a disposición de los usuarios que acceden a la aplicación el contenido del cliente ya desarrollado del que se parte, proporcionándoles acceso a la carga de escenas para su visualización 3D y su edición, así como el envío de las peticiones de renderizado al sistema.

- **Cliente de administración:** Servidor encargado de ofrecer a cualquier usuario administrador el contenido del panel de administración de la aplicación, desde el que es posible monitorizar y administrar el sistema.

- **Microservicio de administración:** Se encarga de atender las peticiones que puede generar el *cliente de administración*, llevando a cabo la comunicación con los *servidores de renderizado* y con la base de datos para realizar las acciones necesarias en el sistema de acuerdo con lo que se solicite en estas.

- **Microservicio de gestión de peticiones:** Su responsabilidad consiste en el manejo de las peticiones de renderizado que vayan llegando. Entre otros aspectos, esta incluirá el manejo de la cola, la recepción, almacenamiento local o en la base de datos y envío a los *servidores de renderizado* de la información necesaria, el envío de las imágenes renderizadas de vuelta al cliente mediante descarga a través del navegador o por correo electrónico y la provisión del tiempo de espera estimado al *cliente estándar*.

- **Servidor de renderizado:** Sus responsabilidades incluyen llevar a cabo el proceso de renderizado a partir de los parámetros y la escena que reciba en las peticiones de renderizado, así como demostrar que es capaz de llevar a cabo este proceso y proporcionar información sobre sí mismo cuando se trate de añadir al sistema.

  Este no tendrá conocimiento alguno del sistema, simplemente se limitará a atender las peticiones que vaya recibiendo y seguirá funcionando de forma *headless*. Únicamente mantendrá un estado interno con los objetivos de evitar conflictos debidos a situaciones como podría ser el hecho de que este reciba una petición de deshabilitación o de realización de otro renderizado en un instante en el que ya se encuentre procesando una petición y de facilitar el proceso de depuración durante el desarrollo de la implementación.

- **Base de datos:** Los componentes encargados de leer y escribir sobre el estado del sistema serán los dos microservicios, los cuales necesitarán realizar estas acciones en un medio de persistencia común.

El estado del sistema se podría resumir en un conjunto de servidores, otro de peticiones, los datos asociados a cada uno y sus relaciones entre sí. Esta será la información que necesita persistir. La base de datos utilizada corresponde con *MongoDB*. Esta hace uso de una colección para los servidores de renderizado y otra para las peticiones.

Aunque una alternativa factible sería montar los propios servidores con esta base de datos, se ha optado por hacer uso de la plataforma *MongoDB Atlas* [34], la cual proporciona toda la infraestructura necesaria para contar con un *cluster* de base de datos en la nube con tres réplicas en el nivel gratuito.

La utilización de estos servicios permite la abstracción de aspectos tales como la seguridad, el despliegue, la alta disponibilidad y la posibilidad de escalar tanto horizontal como verticalmente (refiriéndose al propio componente de la base de datos, no al sistema al completo).

Con la excepción de la base de datos, todos estos componentes son implementados como servidores *Express.js* sobre el entorno de ejecución *Node.js* y hacen uso de módulos de *npm* que puedan resultar útiles, los cuales se describen en detalle en el capítulo 6, dedicado al desarrollo iterativo.

4.2.3. Sistema antes y después de la extensión

En las Figuras 12 y 13 es posible observar la comparación entre el sistema del que se parte y el obtenido tras el desarrollo de la extensión. Las principales diferencias corresponden con:

- División del componente *backend* en el *microservicio de gestión de peticiones* y los *servidores de renderizado*.
- División de la *API REST* que comunicaba los componentes originales *frontend* y *backend* en dos distintas (*API principal* y *API Servidor de Renderizado*).
- Creación de los nuevos componentes relacionados con la administración del sistema: el *cliente de administración* y el *microservicio de administración*.
- Adición de persistencia mediante la inclusión de una base de datos como un nuevo componente de la aplicación.

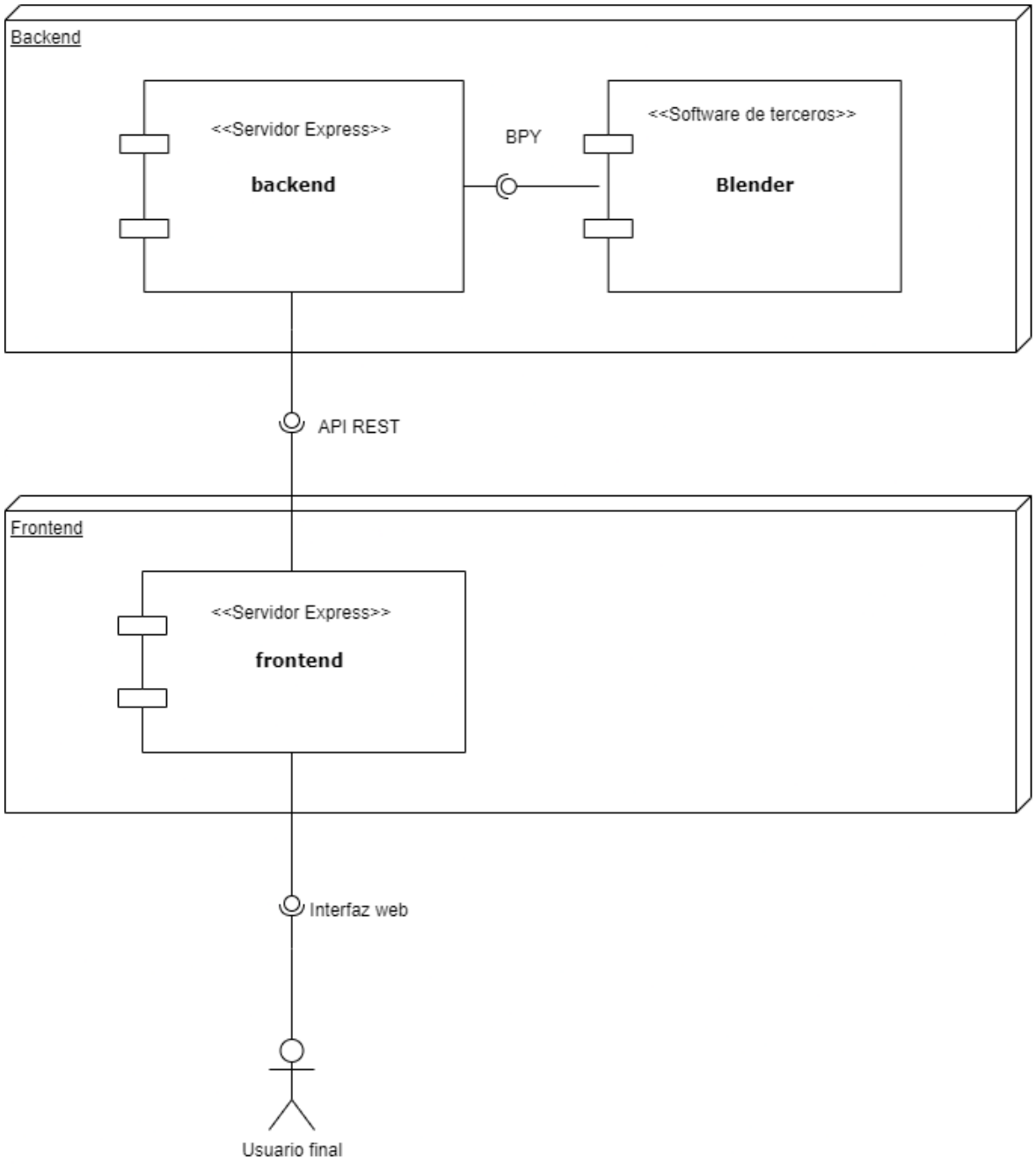

Figura 12: Diagrama de componentes del sistema antes de la extensión

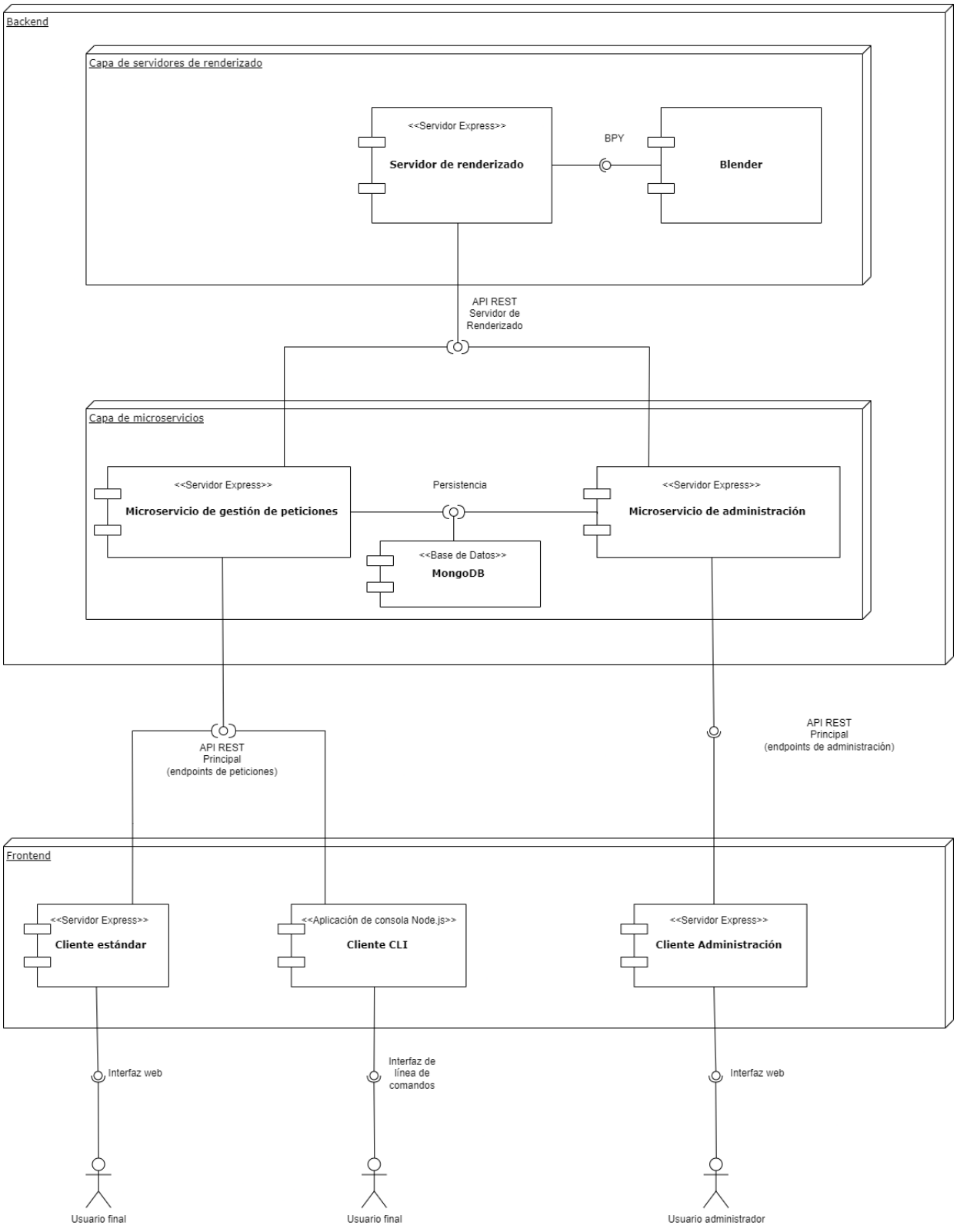

Figura 13: Diagrama de componentes del sistema tras la extensión

4.3. Especificaciones de las *APIs REST*

Debido a los objetivos que se han alcanzado y a los cambios en la arquitectura de la aplicación, la API que comunicaba el *frontend* con el *backend* de la que hacía uso la aplicación original ya no es necesaria. El nuevo sistema cuenta con dos *APIs REST*, cuyas detalles y especificaciones se tratan a continuación.

Para documentarlas, se ha utilizado la herramienta *Swagger Editor* [61], la cual permite generar una documentación clara y legible de estas mediante la edición de un archivo *.yaml*. En las Figuras 14 y 15 es posible ver los distintos *endpoints* que implementan. Para más detalle, estos archivos pueden ser consultados en el código fuente del proyecto[1]. La propia herramienta web de *Swagger Editor*[2] permite renderizar la documentación a partir del contenido de estos.

4.3.1. *API* principal

Es la encargada de ofrecer todas las funcionalidades del sistema accesibles a través de los clientes. Esta es implementada tanto por el *microservicio de administración* como por el *microservicio de gestión de peticiones* y actúa como interfaz entre la capa de microservicios y la de los dos clientes, existiendo también algunos *endpoints* auxiliares utilizados para las comunicaciones entre los dos microservicios.

[1] https://github.com/pablosanchezvecino/PhotoSpaces/tree/main/swagger
[2] https://editor.swagger.io/

Figura 14: Documentación elaborada para la *API* principal

4.3.2. *API* de los servidores de renderizado

Se trata de una *API* más simple y reducida que actúa como interfaz entre los microservicios y un servidor de renderizado concreto. Mediante esta, los servidores de renderizado exponen las funcionalidades que ofrecen para que sean invocadas por los microservicios, los cuales las utilizan para satisfacer las peticiones que estos últimos reciben a través de la *API* principal por parte de los clientes.

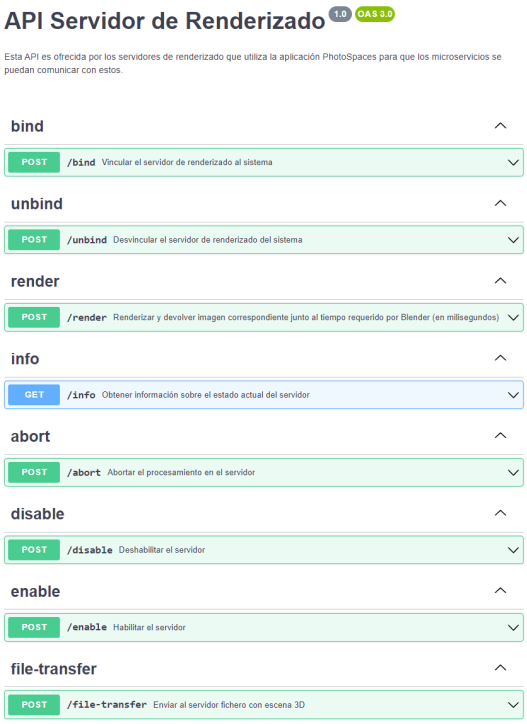

Figura 15: Documentación elaborada para la *API* de los servidores de renderizado

4.4. Prototipado de la interfaz de usuario del panel de administración

Al igual que la interfaz del cliente utilizado por los usuarios finales de la aplicación, el panel de administración también se ha intentado mantener lo más simple e intuitivo posible, lo que ha llevado a la decisión de plantearlo también como una *Single Page Application* desde la que el administrador tenga acceso directo a toda la información relevante de todos los servidores y peticiones de renderizado del sistema, así como a todas las acciones que puede realizar sobre estos.

Antes de comenzar a desarrollar la implementación del panel de administración, resulta conveniente tener una idea clara de cómo se va a plantear el aspecto visual de este. Por ello, se ha recurrido al desarrollo de un prototipo de la interfaz de usuario.

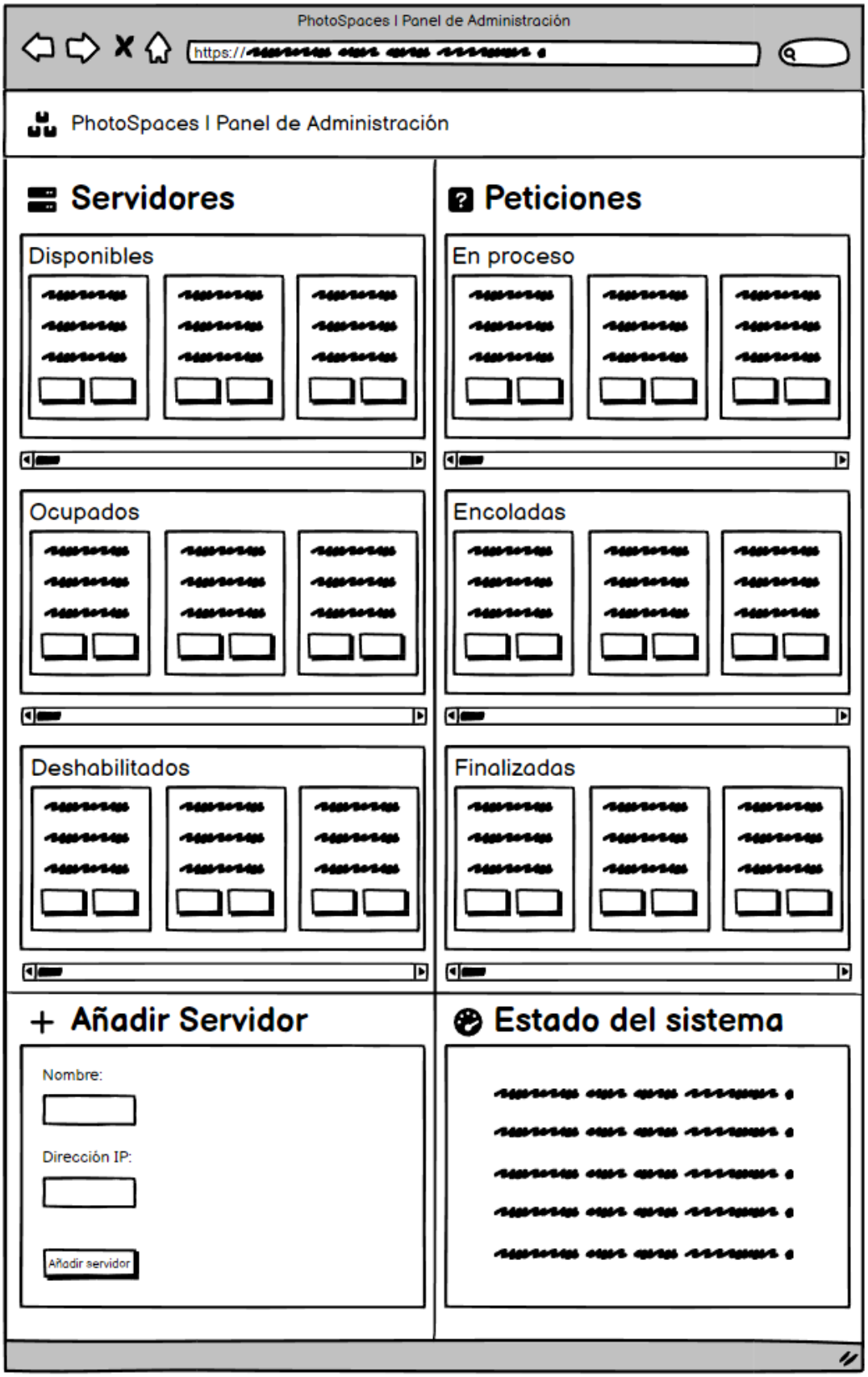

Figura 16: Prototipo de la interfaz de usuario del panel de administración

El prototipado se puede enfocar de diversas maneras, ya que existen distintos tipos, técnicas y herramientas para elaborarlos [3]. Debido a que la navegabilidad de la interfaz en el trabajo presentado es muy simple, se ha optado por utilizar prototipos de baja fidelidad, ya que permitirán un desarrollo en poco tiempo e introducir cambios con facilidad si fuese necesario. Inicialmente, se han realizado algunos bocetos previos. Una vez el diseño estaba totalmente claro, se ha recurrido a la herramienta *Balsamiq Wireframes* [44] en su versión web. En la Figura 16, es posible observar el prototipo desarrollado.

4.5. Seguridad

Aunque la aplicación no se trata de un sistema crítico, y los únicos datos personales del usuario que almacena son su dirección *IP* pública y su dirección de correo electrónico (si la ha indicado), es conveniente tener en cuenta la seguridad desde el principio del desarrollo, ya que una petición inesperada puede suponer problemas importantes para el sistema desarrollado. A continuación, se analizan las necesidades de cada componente de la aplicación:

- **Servidor de renderizado:** Solo instancias de los microservicios podrán comunicarse con los servidores, ya que los primeros son los encargados de mantener el sistema consistente, cualquier llamada externa a un *endpoint* de un servidor podría provocar una inconsistencia entre el estado local del servidor y la base de datos del sistema. Las direcciones *IP* de los microservicios se le proporcionarán a las instancias mediante variables de entorno.

- **Microservicio de administración y cliente de administración:** Debido a la gran repercusión en el sistema de las operaciones que se pueden realizar desde el panel de administración, es conveniente tanto que este sea inaccesible por cualquier usuario no administrador, como que el microservicio sea capaz de ignorar cualquier petición que no provenga de un administrador. La solución pasará por indicar también mediante variables de entorno a a ambos componentes las distintas direcciones *IP* correspondientes a administradores que queramos permitir en el sistema, de esta forma el cliente solo servirá el contenido del panel de administración a los administradores y el microservicio solo atenderá a sus peticiones correspondientes.

- **Microservicio de gestión de peticiones y cliente estándar:** En este caso, es conveniente que los endpoints sean accesibles por cualquier dirección *IP*, ya que la idea es que cualquier usuario pueda enviar desde su navegador una petición de renderizado.

  Sin embargo, hay que tener en cuenta que, al poder recibir peticiones desde cualquier dirección *IP*, no es posible asumir que estas vayan a proceder siempre de un navegador

que esté utilizando el cliente de la aplicación. Por ejemplo, sería posible recibir peticiones malintencionadas construidas con herramientas como *Postman* que omitan parámetros necesarios, incluyan algunos no esperados o que sí se esperaban pero no son válidos. Por este motivo, el microservicio deberá ser capaz de validar los datos que recibe antes de realizar acciones que afecten al sistema.

5

Estudio de posibles técnicas y mecanismos para la optimización de los tiempos de espera

En este capítulo, se realiza un análisis de los factores que afectan a los tiempos de espera de las peticiones que puede recibir el sistema. Entre otros que resultarían más marginales y que se podrían despreciar, estos se ven afectados principalmente por tres factores principales:

- Tiempo empleado en el proceso de renderizado.
- Tiempo empleado en las transmisiones de los archivos con las escenas 3D.
- Tiempo de espera en la cola.

A continuación, se exploran algunas ideas que podrían permitir optimizar estos tiempos.

5.1. Criterio de selección del servidor de renderizado al que se envía una petición

Debido a la heterogeneidad que caracteriza el sistema presentado, los tiempos empleados en los procesos de renderizado van a estar condicionados por los equipos que se tengan registrados como servidores de renderizado en un momento dado, por lo cual esta vía no es altamente explotable para conseguir optimizar el sistema.

Sin embargo, algo sobre lo que sí se posee el control corresponde con el criterio de selección que utiliza el sistema para decidir a qué servidor debe ser reenviada una petición. En un momento dado, contando con dos servidores disponibles donde el primero tardaría la mitad de tiempo que el segundo en llevar a cabo el proceso de renderizado, el tiempo de espera total será menor si esta es enviada al primero. Por este motivo, surge la necesidad de dotar al sistema de la capacidad de determinar cuál es el servidor disponible más rápido.

Aunque las especificaciones del equipo son un buen indicador para determinar si el renderizado va a tardar más o menos en un servidor, no son perfectas (por ejemplo, podría existir un cuello de botella que ralentice a un equipo que a priori pareciera más potente que otro y que tarde más). Esto, junto al hecho de que no existe una forma homogénea de comparar las especificaciones de cualquier equipo, ha llevado a optar por un criterio más empírico, el uso de *benchmarks*.

Inicialmente, se ha considerado utilizar *benchmarks* ya existentes, sin embargo, debido a que no existen mediciones para cualquier equipo que se pueda registrar en el sistema, se ha optado por realizar uno propio más específico, que además proporcionará una medida más fiable.

De esta forma, se va a aprovechar el renderizado de prueba que se utiliza para asegurar que un equipo es capaz de asumir el papel de servidor de renderizado y a utilizarlo al mismo tiempo como *benchmark*, midiendo el tiempo que tarda este en completar el proceso. Una vez registrado el servidor de forma satisfactoria, el sistema tendrá acceso a este tiempo medido y lo utilizará para saber a qué servidor de los disponibles enviar la petición.

Este criterio es el que se ha utilizado en las primera versiones del sistema. Sin embargo, como se detalla posteriormente, tras la aplicación del resto de optimizaciones, el criterio principal pasará a ser un nuevo valor generado a partir de los tamaños de los ficheros procesados, los tiempos de sus transferencias, los tiempos de renderizado y la resolución solicitada por el usuario. Aun así, este seguirá resultando útil como criterio secundario cuando no se disponga del nuevo valor, ya que este último requiere de un conocimiento previo del que no se dispone cuando el servidor acaba de ser registrado en el sistema.

5.2. Almacenamiento y transmisión de ficheros

Las características del sistema presentado implican la necesidad de transmitir archivos para lograr satisfacer las peticiones de renderizado. Estos envíos y recepciones ya constituían una necesidad antes de comenzar con la extensión de la aplicación, pero cobran incluso más importancia al tratar de satisfacer el requisito relacionado con la utilización de múltiples servidores de renderizado, ya que se pasa de contar con dos componentes para estas comunicaciones

(*frontend* y *backend* originales) a tres (*cliente estándar*, *microservicio de gestión de peticiones* y *servidor de renderizado*).

Estas transmisiones serían las siguientes:

- Envío de la escena 3D del *cliente estándar* al *microservicio de gestión de peticiones*.
- Reenvío de la escena 3D del *microservicio de gestión de peticiones* al servidor de renderizado seleccionado por este.
- Envío de la imagen renderizada resultante del servidor de renderizado al *microservicio de gestión de peticiones*.
- Reenvío de la imagen renderizada del *microservicio de gestión de peticiones* al *cliente estándar* que la solicitó.

En cuanto a las transmisiones de las imágenes renderizadas, estas utilizan archivos *PNG* que no ocupan demasiado espacio, por lo que no suponen un problema. Sin embargo, centrándonos en las dos primeras, debido principalmente a la gran cantidad de información requerida para su representación de forma precisa, los archivos utilizados para almacenar escenas y modelos tridimensionales pueden alcanzar un tamaño considerable, aunque depende en gran medida de la complejidad de estos.

Cuando se dan estas circunstancias, las comunicaciones correspondientes suponen un cuello de botella para el rendimiento de la aplicación. Al tratarse de un sistema distribuido y seguir el paradigma cliente-servidor y de microservicios, existe una dependencia total de la transmisiones de estos archivos de un componente a otro.

El gran volumen que pueden ocupar estos archivos implica periodos de espera más largos por parte de los usuarios al requerir una mayor cantidad de tiempo la transferencia de todos los datos. En algunas situaciones no resultará demasiado apreciable, por ejemplo si la petición va a estar encolada de todas formas una vez recibida, pero sí que afectará de forma indirecta a toda la parte del sistema encargada de la gestión de las peticiones.

Estos aspectos del sistema son precisamente donde existe mayor margen para incorporar optimizaciones, las cuales se centrarán en limitar estas transmisiones a las estrictamente necesarias, en paralelizarlas y en reducir el tamaño de los archivos transmitidos.

5.2.1. Archivos *glTF* y *GLB*

PhotoSpaces actualmente trabaja con los recursos 3D mediante la utilización de archivos *glTF* y *GLB*. Estos dos formatos se enmarcan dentro de la especificación *glTF* [40], la cual ha si-

do desarrollada por el *Khronos Group* [38], un consorcio sin ánimo de lucro en el que participan más de 170 organizaciones tecnológicas, entre las que se encuentran las más conocidas. Algunos ejemplos serían *AMD*, *NVIDIA*, *Microsoft*, *Arm*, *Intel*, *Qualcomm*, *Samsung*, *Apple*, *Google*, *Valve*, *Sony*, *Epic Games*...

Esta asociación se encarga del desarrollo y promoción de estándares abiertos relacionados con campos como el renderizado de gráficos, la realidad virtual, la realidad aumentada, la visión por computador y la computación paralela con el objetivo de facilitar el desarrollo de aplicaciones que utilicen este tipo de tecnologías que resulten más eficientes. Algunos estándares abiertos bastante conocidos desarrolladas y/o gestionados por este grupo a destacar serían *OpenGL* [64] y *WebGL* [41]. Ambas se corresponden con *APIs* orientadas a gráficos, siendo esta segunda sobre la que se apoya *Three.js*.

Las distintas implementaciones concretas de acuerdo a estas especificaciones son normalmente desarrolladas y ofrecidas por los distintos fabricantes de hardware.

Centrándonos en la especificación *glTF*, esta ha sido desarrollada con la intención de optimizar la transmisión y carga de recursos 3D mediante la minimización del espacio que ocupan y la simplificación del procesamiento necesario para cargarlas a partir del fichero correspondiente, dando lugar a un proceso más eficiente que pueden aprovechar las distintas aplicaciones que operan con este tipo de recursos.

El contenido de los archivos *glTF* está basado en *JSON*, mientras que el de los archivos *GLB* utiliza binario, motivo por el cual un fichero *.glb* suele tener un menor tamaño comparado con su versión *.gltf* equivalente. Los archivos *.gltf* ofrecen la posibilidad de incluir toda la información en el propio fichero, o también la de incluir referencias a otros archivos en este, un ejemplo sería referenciar un archivo que contiene los datos correspondientes a una textura utilizada por algún modelo. En cambio, los archivos *.glb* son autocontenidos, es decir, toda la información de la que hacen uso se encuentra contenida en el propio archivo, lo que facilita su portabilidad al no depender este de archivos externos.

En el sistema desarrollado, estos ficheros se utilizan en la carga de escenas en el visor 3D del *cliente estándar* a través de *Three.js*, y como parámetro que se le proporciona a *Blender* a la hora de llevar a cabo el proceso de renderizado en un *servidor de renderizado*.

También resulta importante tener claro que el tamaño del fichero que se carga en el *cliente estándar* no tiene por qué ser el mismo que el del fichero asociado a la petición que va a gestionar el sistema. Esto se debe a que, al enviar la escena, esta información se genera a partir de la exportación que realiza *Three.js*, no del fichero cargado por el usuario.

5.2.2. Compresión con *Draco*

Draco es una librería de compresión de gráficos 3D de código abierto desarrollada por *Google*. Esta surge como respuesta al rápido crecimiento en el número de aplicaciones que demandan este tipo de recursos. Trabaja con mallas poligonales y nubes de puntos, aplicando distintas técnicas de compresión con el objetivo de reducir el tamaño de los archivos para lograr un almacenamiento y una transmisión más eficientes.

La especificación *glTF* ofrece la posibilidad de incluir extensiones, siendo una de estas la compresión con *Draco*, lo que significa que esta tecnología es compatible con los archivos con los que se trabajan en el sistema desarrollado. Además, *Blender* es capaz de encargarse automáticamente de descomprimir los datos durante el proceso de importación para llevar a cabo el renderizado, por lo que resulta factible añadir esta funcionalidad a la aplicación.

Figura 17: Imágenes renderizadas utilizando los mismos parámetros, pero unas sin comprimir el archivo con los modelos (izquierda) y otras comprimiéndolo utilizando el nivel 10 de *Draco* (derecha)

Sin embargo, es necesario tener en cuenta que para alcanzar reducciones de tamaño significativas, la compresión con *Draco* pierde algo de información, la cual no es posible recuperar al llevar a cabo el proceso de descompresión (compresión con pérdida o *lossy* [2]). Esto se traduce en una calidad ligeramente inferior en las escenas y en sus imágenes renderizadas correspondientes. Esta circunstancia iría en contra del objetivo relacionado con el fotorrealismo, por lo que aplicar la compresión se ofrece como una nueva funcionalidad opcional.

Por ejemplo, en la Figura 17, es posible observar la comparativa entre las imágenes resul-

tantes de una misma escena cargada a partir de un archivo especialmente pesado de 248 *MiB*, una sin aplicar compresión y otra utilizando el máximo nivel de compresión (que va del 0 al 10). Se puede observar como en la que se ha aplicado compresión la calidad es menor, pero a cambio el tamaño del archivo comprimido es de 64 *MiB*, lográndose así un porcentaje de compresión de aproximadamente un 74.19 %.

No obstante, es necesario tener en cuenta que esto dependerá mucho de una escena a otra, ya que se llevan a cabo distintas compresiones en los distintos elementos de estas. En el caso de la escena anterior, una gran parte de la información se corresponde con las texturas, que cuentan con una resolución muy elevada.

En la Figura 18 se puede observar los tamaños de dicha escena en formato *.gltf*, *.glb* y comprimida con *Draco* utilizando los niveles del 0 al 10, se puede apreciar como el archivo *.gltf* ocupa más que su equivalente en formato *.glb* y que, a medida que se aumenta el nivel de compresión, mayor es la reducción de tamaño obtenida.

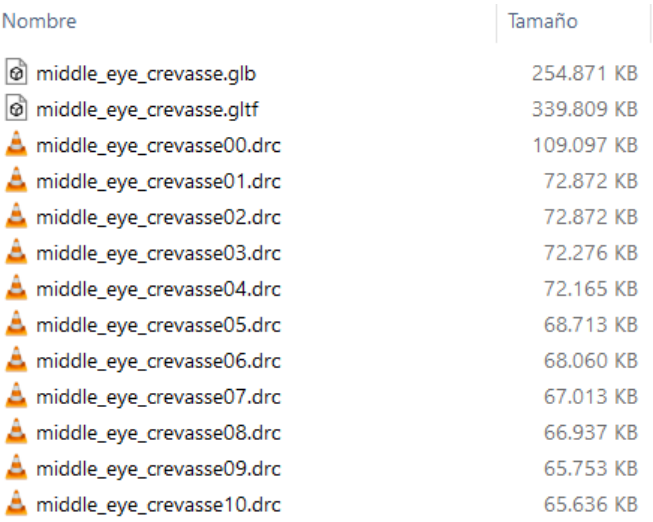

Figura 18: Distintas variantes de fichero para una misma escena y sus tamaños

5.2.3. Paralelización de las transferencias de ficheros

Hasta ahora, el sistema contaba con una única cola y, hasta que una petición no era desencolada de esta, no se comenzaba a transferir el fichero correspondiente.

Otra optimización a implementar consiste en reemplazar este comportamiento por uno que trate de paralelizar las transferencias de los ficheros del *microservicio de gestión de peticiones* al *servidor de renderizado* asignado a la petición. De esta forma, los ficheros se irían enviando al servidor mientras la petición se encuentra encolada y el servidor renderizando otra distinta y, cuando la primera fuera desencolada y enviada al servidor, este ya contaría con el fichero,

pasando a un reenvío casi instantáneo, ya que ahora la petición */render* solo contendría el *string* con los parámetros de renderizado, cuyo tiempo de transferencia se podría considerar despreciable.

Ante este nuevo enfoque, surge la necesidad de establecer una cola para cada servidor, ya que el sistema debe conocer a cuál de estos tiene que ir transfiriendo el fichero al encolarse una petición, así como qué peticiones son las asignadas al servidor cuando este finaliza un renderizado y es necesario ir enviándole la siguiente.

También es necesario establecer un nuevo criterio para la selección del servidor, ya que el seleccionado anteriormente (tiempo empleado en el renderizado de prueba), no tiene en cuenta las transferencias de fichero.

El nuevo criterio originalmente se ha planteado como el cociente entre el total de información procesada por el servidor (la suma de los tamaños de todos los ficheros correspondientes en *bytes*) y el tiempo que le ha tomado al sistema procesarla (teniendo en cuenta tanto las transferencias de los ficheros como el proceso de renderizado), mediante el cual obtendríamos el número de *bytes* que ha sido capaz de procesar el sistema utilizando un servidor por cada milisegundo de tiempo que ha necesitado.

Aunque se trata de un criterio prometedor y que tendría sentido a la hora de determinar el servidor a seleccionar, se plantean una serie de problemas:

- Cuando un servidor acaba de ser registrado en el sistema, este no cuenta con la información necesaria para aplicar el criterio.

  Frente a esta circunstancia, se ha planteado un algoritmo que trata de proporcionarles una oportunidad a estos servidores, dándoles prioridad a la hora de asignar las peticiones para obtener así esta información y compararlos con el resto de servidores en el futuro.

  Además, es posible reutilizar el criterio del tiempo empleado en el renderizado de prueba para realizar la selección cuando exista más de un servidor que cumpla estas condiciones.

- Para un mismo tamaño de fichero, el renderizado con *Eevee* requiere mucho menos tiempo que uno que utilice *Cycles*, por lo que se verían beneficiados los servidores que reciban una mayor cantidad de peticiones que soliciten el primer motor.

  La solución frente a este obstáculo ha consistido en separar este criterio según el motor que se solicite. De esta forma, se mantienen dos medidas como la descrita anteriormente, pero independientes y que se irán actualizando cada una solo con las peticiones de su mismo tipo de motor. Cuando llegue el momento de asignar un servidor, se consultará el valor correspondiente al motor que indique la petición.

- El último inconveniente surge al añadir la funcionalidad correspondiente a que el usuario sea capaz de indicar la resolución a la que quiere llevar a cabo el renderizado. De nuevo, para un mismo tamaño de fichero y un mismo motor, el renderizado tardará más si la resolución es mayor.

  Para abordar esta dificultad, se plantean dos posibles soluciones:

  - Actualizar y utilizar como criterio distintos valores por cada combinación de motor y resolución de la misma forma que se hecho separando los motores. Esta probablemente sea la más justa, sin embargo, debido a que el problema que trata de solucionar no es crítico, se ha considerado que no merecería la pena, ya que añadiría demasiada complejidad e información para mostrar en el panel de administración.
  - Incluir de alguna forma en la fórmula utilizada como criterio la cantidad de píxeles que se han renderizado. Aquí se parte de dos ideas:
    - Modificar la fórmula del cociente para que en lugar de $\frac{bytesTotales}{milisegundosTotales}$ sea $\frac{bytesTotales+pixelesTotales}{milisegundosTotales}$.
      De esta forma, se considera que un píxel de la resolución añade la misma complejidad al renderizado que un *byte* de la escena.
    - Cada vez que se actualice el cociente al terminar de procesar una petición, multiplicar los *bytes* de esta por un peso proporcional a la resolución del renderizado. Por ejemplo, se podría considerar que la resolución estándar fuera *1080p* y que si se renderiza a *720p*, se multiplicara por $\frac{1280*720}{1920*1080} = 0{,}4444$. Si se renderiza a *2k*, se multiplicara por $\frac{2560*1440}{1920*1080} = 1{,}77777$, por 4 si fuera a *4k*...
      Por ello, se considera que, si se mantiene el tamaño de la escena constante, la complejidad del renderizado sería directamente proporcional a los píxeles totales de la resolución.

  Inicialmente se ha optado por utilizar la primera idea, principalmente porque se ha considerado lo suficientemente adecuada y menos compleja. Sin embargo, se ha terminado seleccionando la segunda.

  Esta decisión se debe a que, tras implementar la primera opción, en determinados casos la suma no resultaba muy adecuada (por ejemplo cuando con un fichero muy pequeño y simple de renderizar se solicitaba una resolución muy alta, la resolución tenía más peso y afectaba al resultado de la fórmula más de lo que debería). Por este motivo, se ha concluido que la segunda se adapta mejor a distintos tipos de situaciones y que va a contribuir a generar un valor más acorde a la realidad.

  De todas formas, es importante tener claro que ambas son imperfectas, al existir más factores que afectan a la complejidad del renderizado y al tiempo que va a

necesitar.

Teniendo todo esto en cuenta, los siguientes diagramas de flujo ilustran los cambios realizados en la lógica que rige el manejo de peticiones y servidores por parte del sistema.

Los dos primeros (Figuras 19 y 20) se corresponden con los pasos a seguir y las comprobaciones a realizar ante la llegada y la finalización del procesamiento de una petición, respectivamente, antes de aplicar los cambios, mientras que las dos últimas (Figuras 21 y 22) los muestran tras aplicarlos.

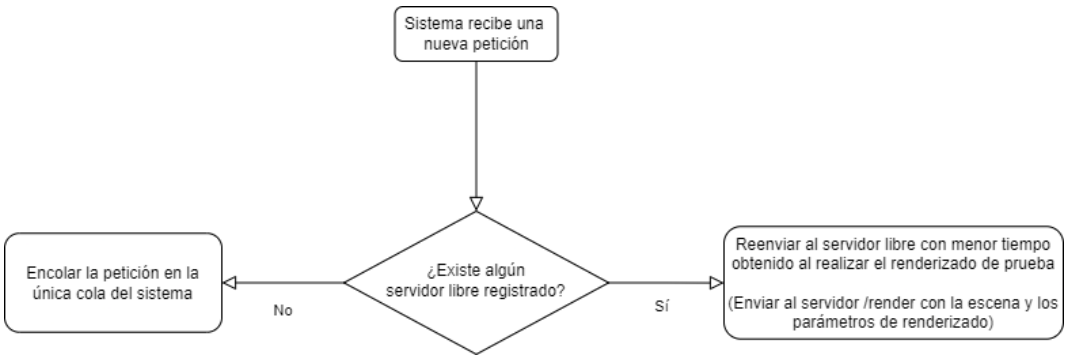

Figura 19: Lógica que sigue el sistema ante la llegada de una nueva petición antes de aplicar la paralelización de las transferencias de ficheros

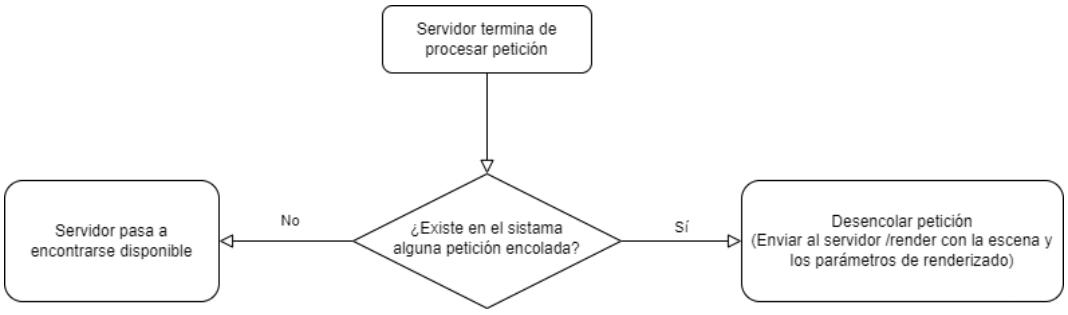

Figura 20: Lógica que sigue el sistema ante la finalización del procesamiento de una petición antes de aplicar la paralelización de las transferencias de ficheros

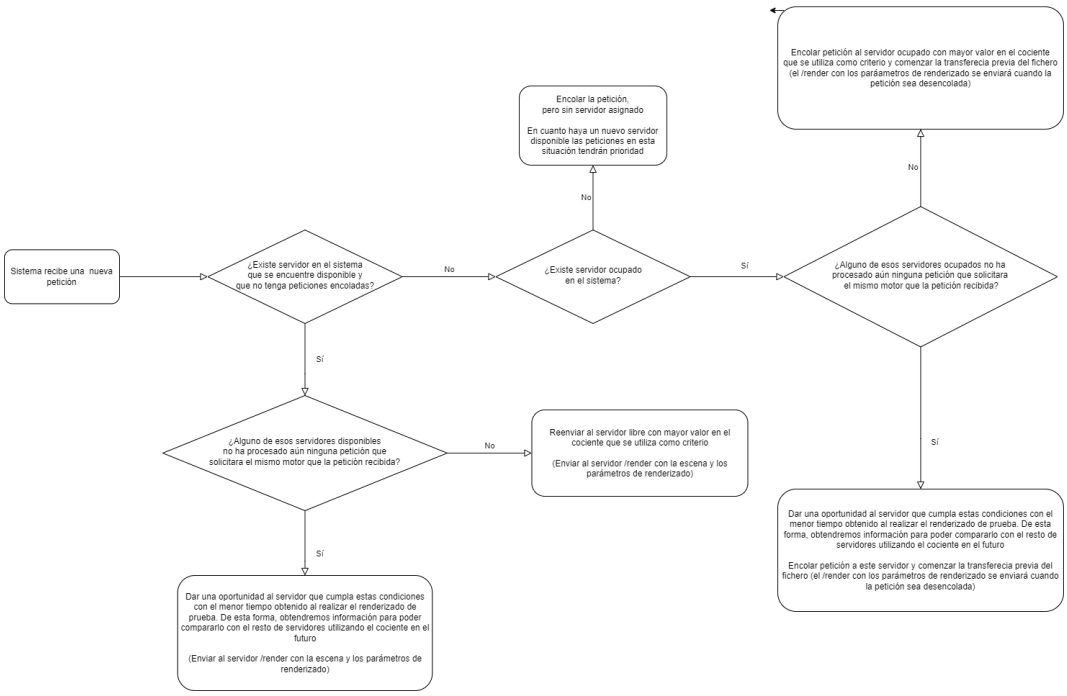

Figura 21: Lógica que sigue el sistema ante la llegada de una nueva petición tras aplicar la paralelización de las transferencias de ficheros

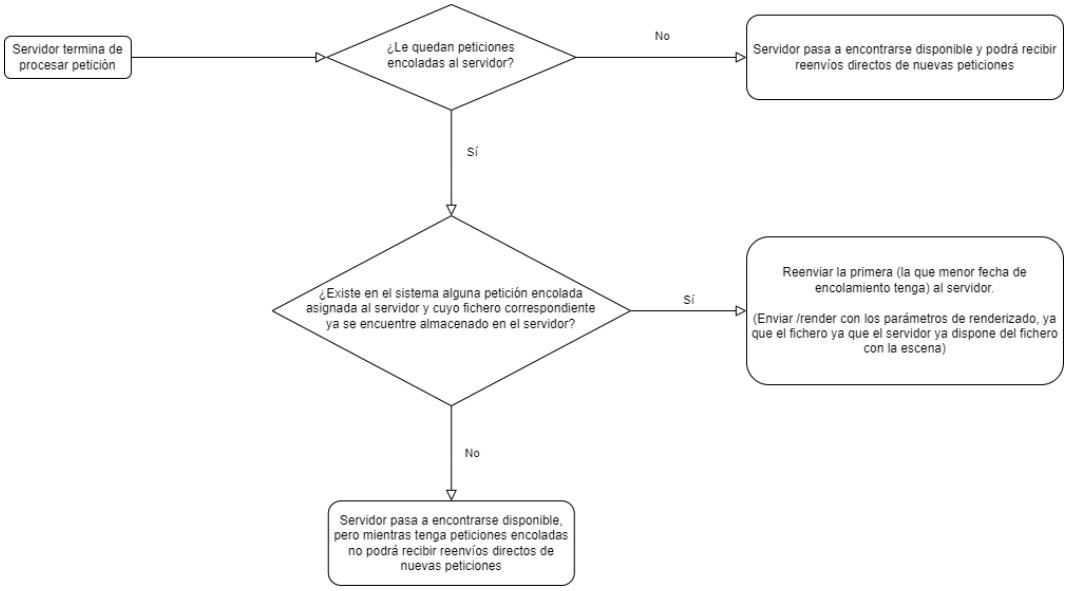

Figura 22: Lógica que sigue el sistema ante la finalización del procesamiento de una petición tras aplicar la paralelización de las transferencias de ficheros

5.2.4. Optimización adicional de carácter experimental

Como se ha mencionado previamente, uno de los aspectos del sistema que ofrece un mayor margen para reducir los tiempos de espera es la transferencia de las escenas 3D.

De acuerdo con esta idea, se ha estudiado la viabilidad de una nueva propuesta consistente en la sustitución de los archivos *glTF* y *GLB* utilizados para la transmisión de información por archivos de texto plano (*.txt*). Estos archivos tienen un tamaño significativamente menor en comparación con las escenas 3D, lo que conlleva un ahorro de tiempo en las transferencias entre los distintos componentes de la aplicación.

Estos llegarían al *servidor de renderizado*, el cual se encargaría de generar la escena en formato *glTF* que requiere *Blender* para llevar a cabo el renderizado. El proceso de generación de la escena implica que el servidor realice solicitudes para obtener una serie de recursos específicos indicados en el archivo *.txt*, los cuales se almacenan en un servidor remoto en la nube.

A pesar del ahorro de tiempo en las transferencias entre componentes en comparación con el funcionamiento original, resulta necesario evaluar si las solicitudes adicionales que el servidor debe realizar y la espera adicional asociada a la generación del fichero *.gltf* requieren una cantidad de tiempo lo suficientemente reducida como para que el tiempo de espera total mejore.

Con el objetivo de tratar de dar respuesta a la incógnita planteada, este enfoque se ha integrado en el sistema a partir de un código funcional suministrado una vez finalizado el desarrollo de las demás funcionalidades, proceso descrito detalladamente en la sección 6.3.9 del capítulo dedicado al proceso de implementación de los distintos aspectos del sistema.

Además, se ha llevado a cabo un análisis comparativo de los tiempos obtenidos con y sin la aplicación de esta técnica, obteniéndose unos resultados favorables de cara a seguir explorando esta idea. Estos se detallan en la sección 7.2.6 del capítulo dedicado a los despliegues del sistema y las pruebas realizadas.

Es importante destacar que este último desarrollo no constituye una funcionalidad adicional del sistema, sino que se ha incorporado con el propósito de evaluar si merece la pena o no considerar este enfoque para optimizar los tiempos de espera durante las transmisiones de archivos. Esta forma de enviar la escena solo funciona con una escena concreta, que es la que se ha utilizado para comparar los tiempos obtenidos.

6

Desarrollo iterativo

Este capítulo se centra en las distintas iteraciones del proceso de desarrollo planteado.

A continuación, se proporciona un análisis detallado de los objetivos, el proceso de implementación, los desafíos enfrentados y los resultados obtenidos en cada una de estas.

6.1. Primera Iteración

6.1.1. Objetivos

Los objetivos de la primera iteración consistían en desarrollar una primera versión funcional del panel de administración de la aplicación (*cliente de administración*).

El primer paso ha consistido en realizar un *fork* del repositorio en *GitHub* donde se encuentra el código del que se parte[3]. Los cambios se han ido realizando en el repositorio correspondiente a esta bifurcación[4].

Debido a la naturaleza de las funcionalidades a implementar, ha sido necesario comenzar a desarrollar la mayoría de componentes de la aplicación. A continuación, se detallan cada una de las implementaciones.

6.1.2. *Microservicio de administración*

Para implementar las funcionalidades del panel de administración, ha sido necesario empezar a desarrollar el *microservicio de administración*, el cual actúa como intermediario entre el *cliente de administración* y los *servidores de renderizado*, manteniendo al cliente actualizado con el estado del sistema y realizando las comprobaciones y acciones necesarias para que se puedan satisfacer las peticiones de este.

De esta forma, cuando el usuario indica mediante la interfaz gráfica del panel de admi-

[3] https://github.com/JoseSFOc/PhotoSpaces
[4] https://github.com/pablosanchezvecino/PhotoSpaces

nistración que quiere realizar una acción determinada, el cliente realiza una petición *HTTP* al *microservicio de administración*, que a su vez puede comunicarse con la base de datos para consultar el estado del sistema y con los servidores de renderizado realizando peticiones a una pequeña *API* que cada uno de estos implementa.

6.1.3. *Servidor de renderizado*

Cada uno de los servidores de renderizado que se pueden añadir al sistema debe encontrarse a la espera de peticiones y cuenta con cuatro estados posibles descritos por el diagrama de máquina de estados de la Figura 23.

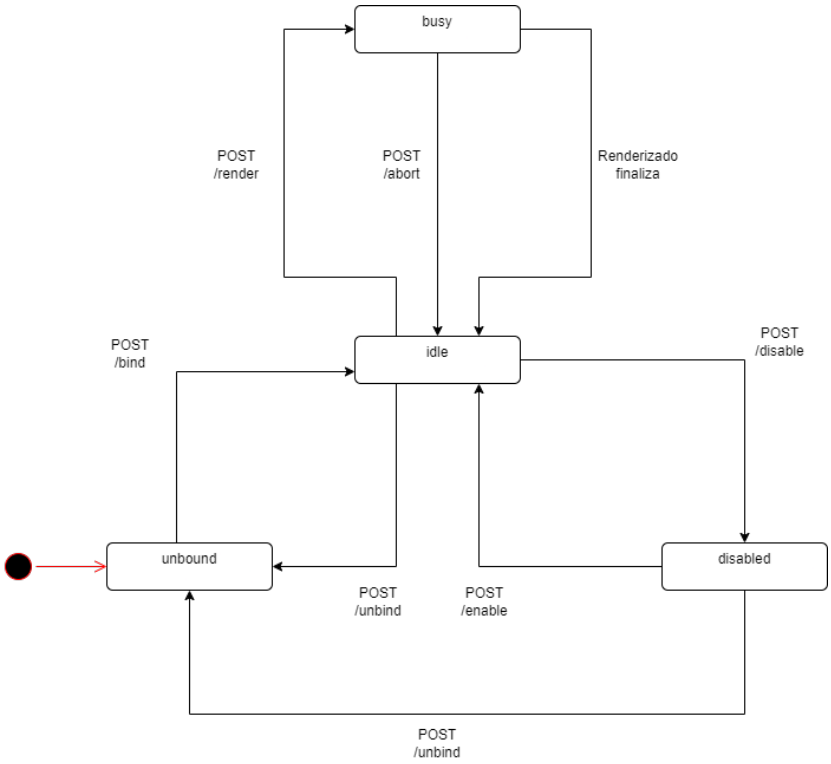

Figura 23: Diagrama de máquina de estados con los posibles estados de un servidor de renderizado

Para que el *servidor de renderizado* pueda formar parte del sistema, el *microservicio de administración* debe asegurarse de que el servidor está preparado para procesar las peticiones de renderizado que puedan ir llegándole, por lo que el primero cuenta con un *endpoint /bind* que el segundo puede consultar antes de registrarlo en el sistema para asegurarse de que el equipo es capaz de llevar a cabo el proceso de renderizado y, además, conocer con qué sistema operativo, con qué procesador, con qué tarjeta gráfica y con qué versión de *Blender* cuenta este. Para obtener esta información se ha recurrido al módulo *systeminformation* [24].

También se han implementado los *endpoints /enable* y */disable* para pasar su estado de *idle* a *disabled* y de *disabled* a *idle* respectivamente.

6.1.4. Persistencia

Una vez comprobado un servidor, su información es persistida por el *microservicio de administración* en la base de datos, de esta forma, cuando se refresque la interfaz gráfica este aparecerá junto al resto. En la Figura 24 se visualiza un ejemplo de la información del servidor que se almacena:


```
_id: ObjectId('63f60762a88cab89e4d1edb6')
name: "Portátil"
ip: "192.168.1.40"
status: "idle"
os: "Microsoft Windows 11 Pro"
cpu: "Intel® Core™ i7-9750H"
gpu: "NVIDIA GeForce GTX 1650"
blenderVersion: "3.4.1"
registrationDate: 2023-02-22T12:15:30.046+00:00
__v: 0
```


Figura 24: Ejemplo de documento almacenado en MongoDB

6.1.5. *Cliente de administración*

El *cliente de administración* consiste en una interfaz gráfica que actúa como panel de control desde el que un usuario administrador del sistema pueda monitorizar los *servidores de renderizado* registrados en el sistema y el estado de las peticiones que se van realizando, además de realizar acciones sobre ellos.

Uno de los objetivos era mantenerla simple y fácil de utilizar, por lo que se ha planteado como una *Single Page Application* desde la cual el administrador tiene acceso a toda la información actualizada y a todas las acciones realizables en ese momento.

Para lograr esto, el cliente realiza peticiones al *microservicio de administración* para obtener en formato *JSON* la información correspondiente al estado del sistema, y mediante código *JavaScript* modifica el contenido de la página para que coincida con el contenido recibido.

6.1.6. Problemas encontrados y resultados de la iteración

La principal complicación ha sido proporcionar el estilo con *Bootstrap* a la interfaz de usuario para que se ajustara al prototipo establecido inicialmente, lo cual ha llevado bastante tiempo en comparación con el resto de funcionalidades.

Otro aspecto a destacar es que, como se ha optado por utilizar *JavaScript* en el lado del cliente sin ninguna herramienta que facilite dicha tarea (*JavaScript* puro o *Vanilla JavaScript*) por mantener la consistencia con la aplicación base, el código resultante tiene más líneas y puede resultar más confuso que el que se habría obtenido con ayuda de alguna biblioteca o *framework* como podrían ser *Angular*, *React* o *Vue*.

Al final de esta iteración, se cuenta con los dos componentes *backend* y *frontend* de la aplicación original inalterados y, por otra parte, la base de los nuevos componentes sobre los que se seguirá desarrollando en las siguientes iteraciones: el *servidor de renderizado*, el *microservicio de administración* y el *cliente de administración*.

6.2. Segunda Iteración

6.2.1. Objetivos

El primer objetivo de esta iteración ha sido realizar una refactorización y modularizar el código de la aplicación, ya que la anterior concluyó con una organización del código bastante mejorable.

Atendiendo a nuevas funcionalidades, la idea era añadir todo lo necesario para tener la aplicación funcionando al completo, por lo que los objetivos han sido los siguientes:

- Trasladar la lógica de renderizado de la aplicación original al nuevo sistema.
- Implementar el manejo de peticiones de renderizado en el *microservicio de gestión de peticiones*.
- Añadir la posibilidad de escoger entre que la imagen renderizada resultante se descargue en el navegador o se envíe por correo electrónico.
- Permitir consultar, cuando sea posible, el tiempo de espera estimado para que la petición sea procesada si esta ya ha sido reenviada a un servidor o, si se encuentra encolada, el tiempo de espera estimado para que ascienda una posición en la cola.
- Mostrar en el *cliente de administración* las peticiones y su información correspondiente.

- Implementar las comunicaciones necesarias entre los distintos componentes involucrados.

Aunque estos eran los objetivos principales, también se han trabajado algunos aspectos adicionales, los cuales se detallan a continuación.

6.2.2. Refactorización

La refactorización ha consistido, por un lado, en dividir el código en módulos organizados en distintas carpetas en todos los proyectos para que este resulte más fácil de mantener.

Además, se ha pasado de la sintaxis con *require* y *module.exports* de *CommonJS* a la de *import/export* de *ECMAScript Modules*, principalmente por consistencia. Se han instalado en los proyectos *ESLint* [23], un *linter* (herramienta de análisis estático de código), configurándolo para que avise de errores y ayude a mantener un estilo de código consistente en todos los componentes del sistema.

6.2.3. *Servidor de renderizado*

Implementar el proceso de renderizado en el nuevo sistema no ha resultado excesivamente complicado, ya que se contaba previamente con la lógica en la aplicación original, requiriendo por ello simplemente trasladarla al código del *servidor de renderizado* y realizar algunas pequeñas modificaciones.

Una vez se ha tenido funcionando el proceso de renderizado, ya ha sido posible implementar el renderizado de prueba que sirve tanto para comprobar que el equipo que se va a registrar en el sistema es capaz de llevar a cabo el proceso, como para que este actúe de *benchmark* y se mida el tiempo que tarda el equipo en el procesamiento.

Por otro lado, también se ha implementado el *endpoint /abort* , el cual se encarga de finalizar el proceso en ejecución de *Blender* si recibe una petición a esta ruta mientras está llevando a cabo un renderizado.

Para esta funcionalidad ha sido necesario distinguir entre equipos *Windows* y *Unix-like*, ya que los comandos necesarios para acabar con un proceso difieren. En este caso, se han utilizado los dos siguientes:

- **Windows:** `taskkill /im blender.exe /f` [47]
- **Unix-like:** `pkill blender` [7]

6.2.4. Gestión de peticiones

Para desarrollar este aspecto de la aplicación, ha sido necesario comenzar con el desarrollo del *microservicio de gestión de peticiones*. Este hace de intermediario entre el *cliente estándar* (desde donde se envían las peticiones de renderizado) y los *servidores de renderizado* encargados de procesarlas.

En este microservicio se ha implementado la lógica que sigue el manejo de las peticiones, lo que incluiría la recepción de las escenas 3D, el envío de estas al servidor de renderizado correspondiente, la recepción de la imagen renderizada devuelta por el servidor, su envío al cliente, el mecanismo de *polling* que se utiliza para conocer el tiempo de espera restante aproximado en los dos clientes y la posibilidad del envío por correo electrónico de la imagen renderizada.

Con respecto a esta última funcionalidad, se ha creado una cuenta *Gmail* que actúe como remitente de los correos (photospacesapp@gmail.com) y se ha configurado para que permita la automatización del envío de correos, la cual se ha implementado en el código haciendo uso del módulo *node-mailer* [51].

También se ha implementado un reintento del envío del correo, el cual se ejecutará en el caso de que ocurra algún error en este.

Su funcionamiento consiste en persistir el contenido de la imagen y la dirección de correo del usuario que la solicitó en la base de datos cuando se detecte el error. El sistema se encargará de comprobar periódicamente si existe algún documento correspondiente a un envío fallido e intentará reenviar el correo electrónico a partir de esa información. Una vez enviado correctamente, esta información será eliminada. En cambio, si volviera a producirse un fallo, la información seguiría almacenada y se llevaría a cabo otro reintento en la siguiente comprobación.

También ha sido necesario realizar algunas modificaciones en el *cliente estándar*, como la inclusión de botones de radio que permitan al usuario indicar si quiere obtener la imagen renderizada por correo o por descarga desde el navegador, así como un campo de texto para que indique la dirección de correo electrónico a la que quiere que se envíe esta, la cual es validada tanto por el cliente antes de ser enviada al *microservicio de gestión de peticiones* como por este último al recibirla (Figuras 25 y 26).

En este también se han añadido algunos *tooltips* en las opciones que puedan resultar más confusas al usuario con el objetivo de que ayuden a aclarar con qué se corresponde cada parámetro.

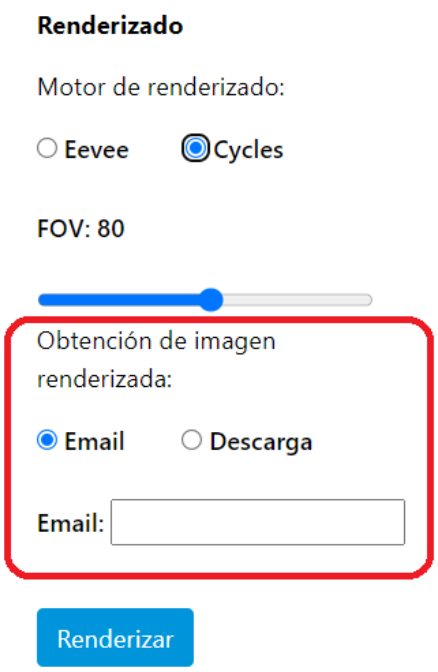

Figura 25: Opciones para la obtención de la imagen en el *cliente estándar*

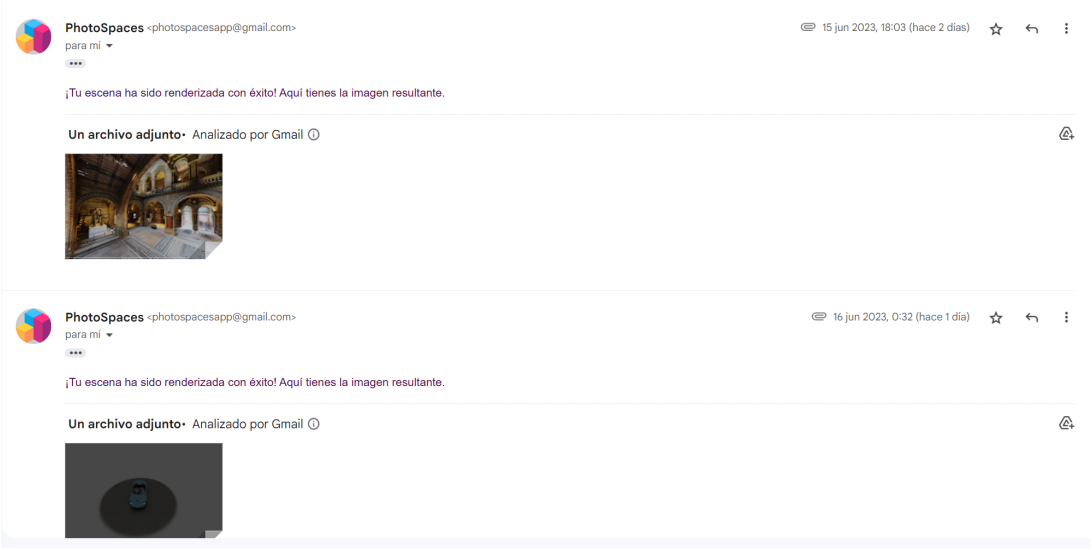

Figura 26: Ejemplos de correos recibidos con las imágenes renderizadas adjuntas

6.2.5. *Cliente de administración*

En el *cliente de administración* se ha añadido la posibilidad de ver las peticiones registradas (Figura 27) en el sistema utilizando el mismo enfoque que en la consulta de servidores de la

primera iteración. Estas son persistidas en la base de datos y el cliente las consulta de forma periódica al *microservicio de administración*, que a su vez realiza la consulta a la base de datos y las devuelve en formato *JSON* para que el cliente pueda modificar el *DOM* de acuerdo a los contenidos de este.

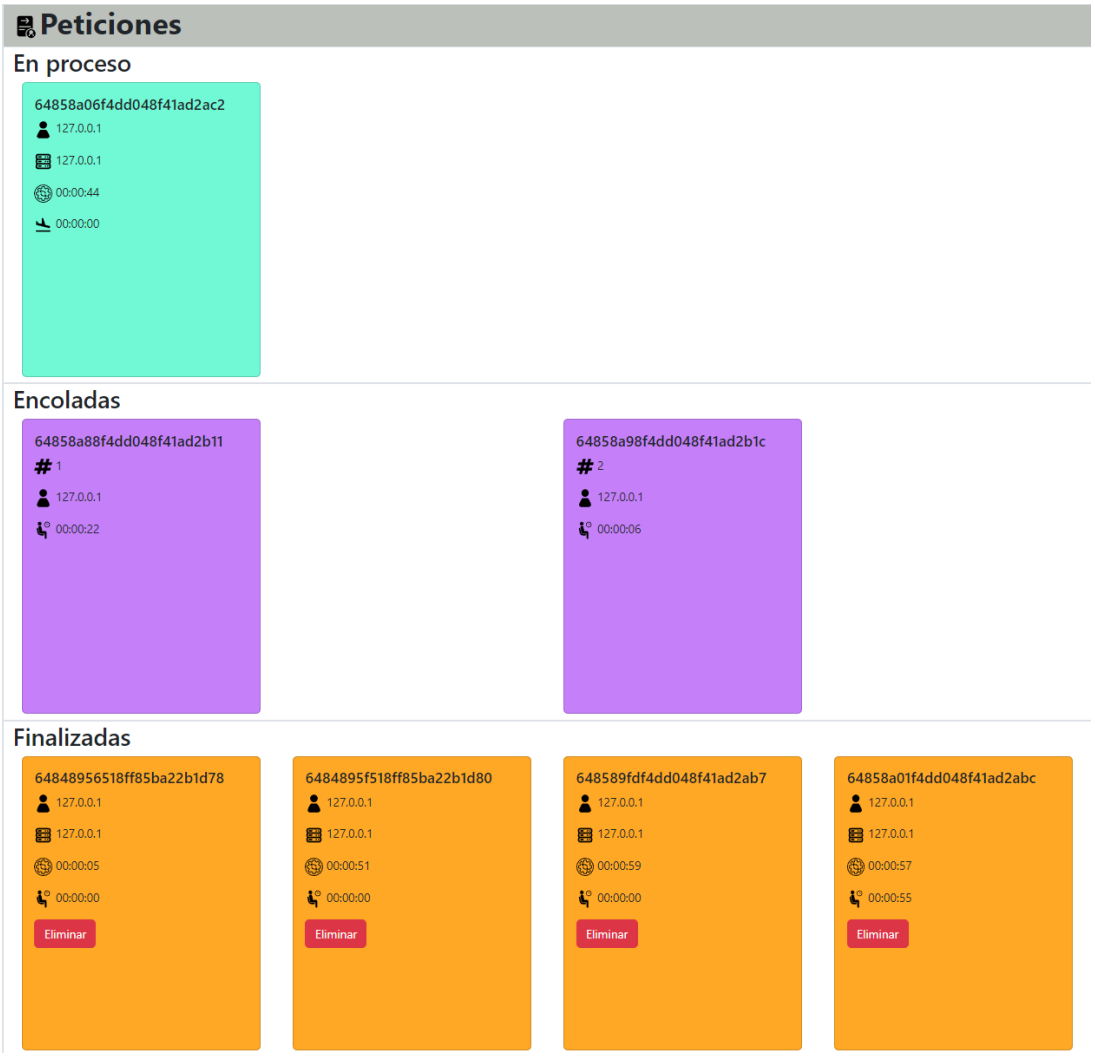

Figura 27: Visualización de las peticiones de renderizado desde el *cliente de administración*

Se requiere medir tanto el tiempo que pasan en la cola las peticiones como el que tardan en ser procesadas. En la base de datos también se almacena el instante de tiempo en el que es encolada (si es que pasa por la cola), el instante de tiempo en el que es enviada al servidor de renderizado y el instante de tiempo en el que es obtenida la imagen resultante en el *cliente de administración de gestión de peticiones*.

A partir de estos, mediante conversiones y restas sencillas, es posible obtener los tiempos correspondientes y utilizarlos para mostrarlos en las tarjetas de las peticiones y en el cálculo de las medias en la parte de estadísticas (Figura 28).

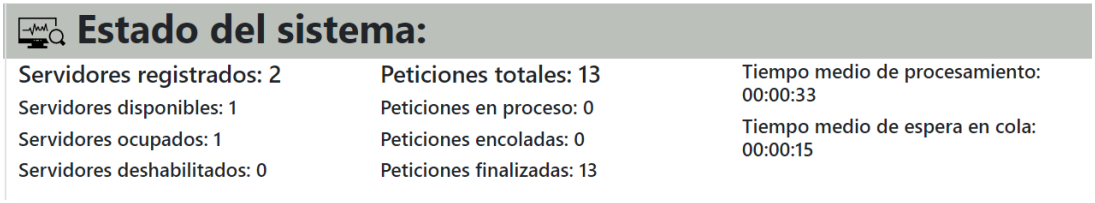

Figura 28: Estadísticas mostradas en el panel de administración

También se ha añadido la posibilidad de eliminar del sistema cualquier petición, teniendo en cuenta que, si esta se encuentra siendo procesada en ese momento, será necesario contactar con el servidor correspondiente para que aborte su procesamiento.

Es necesario distinguir entre dos casos. En el primero, el administrador tiene la intención de eliminar una petición, mientras que en el segundo, solo requiere abortar el procesamiento en un servidor sin eliminar la petición que estaba procesando.

Para el primer caso, se muestra un botón eliminar en cada tarjeta correspondiente a una petición en el panel de administración, independientemente de su estado actual. Para el segundo, cada tarjeta mostrada correspondiente a un servidor de renderizado que se encuentre ocupado incluye un botón que permita abortar el procesamiento sin eliminar la petición, enviándola en su lugar al final de la cola. Ambos se pueden apreciar en la Figura 29.

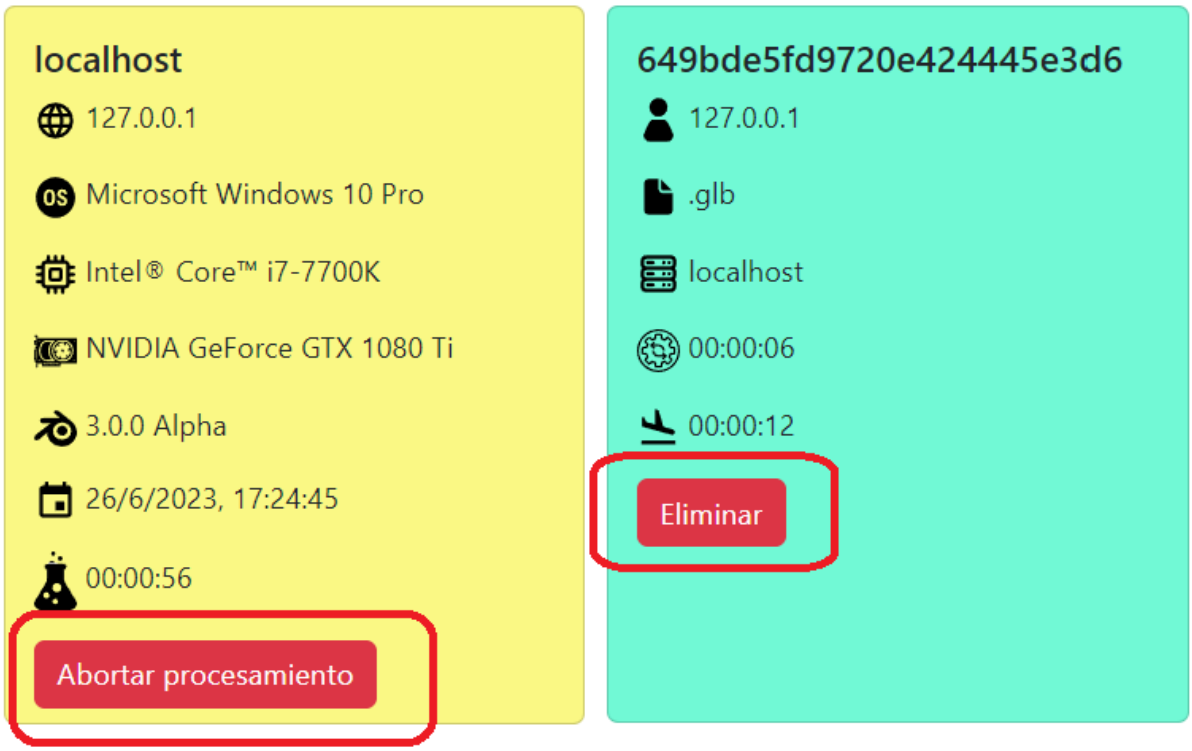

Figura 29: Botón para abortar el procesamiento en un servidor y enviar la petición que estaba procesando al final de la cola (izquierda) y botón para eliminar una petición que se está procesando en ese mismo momento (derecha)

Por último, también se han incluido modales en la interfaz de usuario que se encarguen de solicitar una confirmación antes de proceder con cualquier instrucción con el objetivo de mejorar la usabilidad de la aplicación (Figura 30).

Figura 30: Ejemplo de modal encargado de solicitar una confirmación al usuario

6.2.6. Seguridad, validaciones y manejo de errores

De acuerdo con la sección 4.5 del capítulo dedicado al diseño, en esta iteración se han comenzado a implementar algunas medidas de seguridad. Para el filtrado de direcciones *IP*, se ha definido un *middleware* (Figura 31) encargado de comprobar que la dirección *IP* de la que proviene cada petición coincide con alguna de las permitidas que se hayan definido. Si no es así, devolverá un código de respuesta *403 Forbidden*. Este *middleware* es usado por el *cliente de administración*, por el *microservicio de administración* y por los *servidores de renderizado*.

En el caso del *microservicio de gestión de peticiones*, como pueden llegar peticiones desde cualquier dirección *IP*, es necesario un esfuerzo adicional para que el sistema resista en el caso en el que llegue alguna defectuosa o malintencionada. Este consiste en una serie de validaciones que se asegurarán de que los datos que llegan en las peticiones de renderizado son válidos y permiten llevar a cabo este proceso. A continuación, se detalla cómo es validado cada parámetro de los que pueden llegar al servidor:

- **Archivo con escena 3D:** Se utiliza el módulo *gltf-validator* [39] para comprobar que la escena es válida.
- **Email:** Este parámetro es opcional, pero si se indica alguna dirección se comprueba que sea válida con el módulo *validator* [50].
- **Parámetros de renderizado:** El esquema definido para trabajar con las peticiones de renderizado utilizando *mongoose* lanzará un error controlado en el intento de persistir la petición en la base de datos al no coincidir los valores con la especificación del esquema.

También ha sido necesario atender a los aspectos relacionados con *CORS* [16]. *CORS* (*Cross Origin Resource Sharing o Intercambio de recursos de origen cruzado*) es una medida que implementan los navegadores para evitar problemas de seguridad y que restringirá las peticiones realizadas por el *cliente estándar* y el *cliente de administración*, ya que una vez servidos los contenidos de estos por sus respectivos servidores *Express*, las peticiones que realizan no van dirigidas al mismo origen y serán bloqueadas por el navegador si el servidor al que se realiza la petición no indica que permite peticiones de ese origen.

Esto hace necesaria una modificación adicional en los componentes *microservicio de administración* y *microservicio de gestión de peticiones*, la cual se ha implementado utilizando el módulo *cors* [21] de *npm*, el cual actúa como un *middleware* más de *Express* y permite especificar una configuración de forma sencilla.

Aunque se podrían permitir cualquier origen en ambos, el enfoque más seguro y adecuado consiste en indicar explícitamente los orígenes de los que se quieren permitir peticiones, en

este caso coincidirán con las *URLs* de las instancias de los componentes *cliente estándar* y *cliente de administración* desplegadas que sean accesibles introduciendo estas en el navegador. Estas se especificarán mediante la variable de entorno *CORS_ALLOWED_ORIGINS*.

Por otro lado, también se ha tenido en cuenta la robustez del sistema durante el desarrollo de esta iteración. Se ha intentado en la medida de lo posible que el manejo de errores evite que los componentes se paren por un error no controlado utilizando bloques *try...catch* siempre que se llame a una operación de base de datos, de manejo de archivos o de comunicaciones *HTTP*. Esto ha provocado que el código resultante sea más largo y un poco menos legible, pero resulta necesario para satisfacer los requisitos relacionados con la robustez y los errores descriptivos.

También se han definido una serie de ejecuciones periódicas en el *microservicio de gestión de peticiones* que ayuden al sistema a recuperarse de algunas situaciones inesperadas. En este caso, uno de limpieza de archivos temporales innecesarios y otro que compruebe la cola y los servidores disponibles por si la lógica de gestión de peticiones fallase por algún motivo.

```
import dotenv from "dotenv";

dotenv.config();

let allowedIps = process.env.ALLOWED_IPS.split(",");

const ipCheckMiddleware = (req, res, next) => {

  let requestIp = req.ip;

  // Si se recibe una dirección IPv4 embebida en una dirección IPv6
  if (req.ip.toString().startsWith("::ffff:")) {
    // Extraer IPv4
    requestIp = requestIp.toString().slice(7);
  }

  console.log(requestIp);

  if (!allowedIps.includes(requestIp)) {
    // Envía una respuesta de acceso denegado si la dirección IP no coincide con ninguna de las permitidas
    return res.status(403).send("Acceso denegado");
  }

  next();
};

export { ipCheckMiddleware };
```

Figura 31: *Middleware* definido para filtrar las peticiones por su dirección *IP*

6.2.7. *Docker* y *scripts*

También se ha aprovechado esta iteración para ir comenzando a trabajar con *Docker*, pues uno de los objetivos se corresponde con que el sistema pueda seguir ofreciendo la posibilidad de utilizarlo en forma de contenedores. Concretamente, se han utilizado los archivos *Dockerfile* de la aplicación original de referencia, sobre los que se han introducido una serie de modificaciones para los distintos componentes del sistema. También se ha modificado el archivo *docker-compose.yaml* para que despliegue el sistema al completo. Como se ha trabajado anteriormente con *Docker*, no han surgido dificultades y se ha podido realizar esta tarea de forma eficiente. Sin embargo, será necesario realizar modificaciones en iteraciones posteriores a

medida que el sistema evolucione.

Para agilizar los despliegues, se ha creado en el proyecto una carpeta */scripts* que, como su propio nombre indica, contiene *scripts* para el despliegue de cada componente tanto en *Windows* (ficheros *.bat*), como en sistemas operativos *Unix-like* (ficheros *.sh*).

6.2.8. *Ubuntu*

En esta iteración también se ha montado un entorno *Linux* para comprobar que lo realizado hasta este punto funciona también en equipos con este tipo de sistema operativo, ya que el desarrollo se está llevando a cabo principalmente en un equipo con *Windows* y la idea es que el sistema pueda funcionar correctamente aunque se utilicen equipos heterogéneos. En este caso, se ha montado en una partición de este mismo equipo la distribución *Ubuntu 22.04.2 LTS Jammy Jellyfish* [45], instalado el software y realizado los cambios necesarios en la implementación hasta que se ha podido comprobar que todo funcionaba correctamente.

6.2.9. Problemas encontrados y resultados de la iteración

Aunque no se ha perdido un tiempo considerable en la resolución de problemas, ha resultado complejo trabajar con la asincronía de *JavaScript* y con la transmisión de archivos en las comunicaciones *HTTP*, principalmente por falta de experiencia. Sin embargo, se ha conseguido que todo funcione tal y como se había planificado previamente.

Relacionado con la transferencia de archivos, otro cambio que se ha realizado con respecto a la aplicación original ha sido utilizar el módulo *multer* [63] en lugar de *express-fileupload* para las transferencias de los archivos con las escenas 3D. En principio no supone ningún beneficio usar un módulo u otro, el cambio se debe a que, al probar el sistema en *Ubuntu*, al recibir el archivo con la escena 3D, el *microservicio de gestión de peticiones* funcionaba de una forma excesivamente lenta hasta el punto de que el programa daba un error debido a que se había superado el límite de memoria reservado para *Node.js*.

Pensando que las transferencias de archivos podrían ser el problema, se realizó un cambio en la implementación para que usara *multer*. Sin embargo, el error se debía a que faltaba por definir una variable de entorno en el archivo *.env* en *Ubuntu*, provocando que dentro del programa esta tuviera el valor *undefined*, por lo que el intervalo de comprobación de la base de datos se ejecutaba continuamente. Esta inconveniencia ha supuesto una pérdida de tiempo importante en las comprobaciones de que todo funcionara en *Ubuntu*.

Otro problema importante ha surgido al intentar utilizar variables de entorno en los clientes para que los distintos parámetros como las direcciones *IP* y los puertos de los microservicios

a los que realizan las peticiones no fueran fijas y se pudieran configurar fácilmente con estas al igual que en el resto del sistema. Al tratarse de código *JavaScript* que se sirve mediante *Express* al cliente para que este sea ejecutado en el navegador, no es posible utilizar las variables de entorno.

{"administrationMicroserviceIp":"127.0.0.1","administrationMicroservicePort":"9000","refreshPeriodMs":"1000","maxCardsPerContainer":"100"}

Figura 32: Contenido del archivo *JSON* utilizado por el *cliente de administración*

Tras dedicar bastante tiempo a buscar una solución que permitiera cargar en estos *scripts* las direcciones *IP* y los puertos antes de servir los contenidos de la carpeta *public*, al final se ha optado por añadir en esta carpeta un archivo *JSON* que incluyera esta información. De esta forma, antes de servir la carpeta *public*, el código que se ejecuta en *Node.js* que pone en marcha el servidor y la sirve se encarga de leer las variables de entorno y escribirlas en el archivo *JSON* para que este pueda ser leído posteriormente por el código ejecutado en el navegador del cliente. En las Figuras 32, 33 y 34 es posible observar dicha implementación.

```
1  import { ipCheckMiddleware } from "./middleware/ipCheckMiddleware.js";
2  import { printAsciiArt } from "./public/js/logic/asciiArtLogic.js";
3  import { writeFileSync } from "fs";
4  import express from "express";
5  import dotenv from "dotenv";
6  import path from "path";
7  import "colors";
8
9  printAsciiArt();
10
11 dotenv.config();
12
13 const app = express();
14 const PORT = process.env.PORT || 8080;
15
16 // Generar archivo parameters.json con las dirección IP y puerto a los que debe dirigirse el navegador
17 try {
18   const parameters = {
19     administrationMicroserviceIp: process.env.ADMINISTRATION_MICROSERVICE_IP,
20     administrationMicroservicePort: process.env.ADMINISTRATION_MICROSERVICE_PORT,
21     refreshPeriodMs: process.env.REFRESH_PERIOD_MS,
22     maxCardsPerContainer: process.env.MAX_CARDS_PER_CONTAINER
23   };
24   writeFileSync("./public/parameters.json", JSON.stringify(parameters));
25   console.log("Archivo parameters.json creado y contenido escrito correctamente".bold.magenta);
26 } catch (error) {
27   console.error(`Error en la escritura del archivo parameters.json. ${error}`.red);
28 }
29
30 // Utilizar comprobación de direcciones IP
31 app.use(ipCheckMiddleware);
32
33 // Servir el directorio /public
34 app.use(express.static(path.join(path.resolve(), "public")));
35
36 app.listen(PORT, () ⇒
37   console.log(
38     `Cliente de administración desplegado en el puerto ${PORT}`.bold.magenta
39   )
40 );
```

Figura 33: Código del servidor *express* que escribe el archivo *JSON* antes de servir el directorio *public*

Como resultado de esta iteración, se cuenta con una primera versión funcional del sistema al completo que permite utilizarlo tanto desde la perspectiva de un administrador del sistema como desde la de un usuario final, la cual servirá como base para la siguiente iteración.

```
/* Obtener los distintos parámetros configurables, que habrán sido establecidos por el
servidor express a través de las variables de entorno antes de servir la carpeta public */

let administrationMicroserviceIp = null;
let administrationMicroservicePort = null;
let refreshPeriodMs = null;
let maxCardsPerContainer = null;

try {
  const response = await fetch("./parameters.json");
  const parameters = await response.json();

  administrationMicroserviceIp = parameters.administrationMicroserviceIp;
  administrationMicroservicePort = parameters.administrationMicroservicePort;
  refreshPeriodMs = parameters.refreshPeriodMs;
  maxCardsPerContainer = parameters.maxCardsPerContainer;
} catch (error) {
  console.error(`Error al intentar leer los parámetros del archivo parameters.json. ${error}`.red);
}

export {
  administrationMicroserviceIp,
  administrationMicroservicePort,
  refreshPeriodMs,
  maxCardsPerContainer
};
```

Figura 34: Código que lee del archivo *JSON* en el navegador

6.3. Tercera Iteración

6.3.1. Objetivos

El objetivo de esta iteración ha consistido en incorporar una serie de optimizaciones y mejoras al sistema, así como añadir detalles y corregir errores que se hayan podido pasar por alto en iteraciones anteriores.

6.3.2. Exportación de la escena de *Three.js*

Independientemente del tipo de fichero que cargara el usuario en el cliente estándar (*.gltf* o *.glb*), al generar la petición de renderizado, la exportación que realizaba *Three.js* se realizaba a *glTF*, es decir, utilizaba *JSON*. Esto se traducía en que todos los archivos temporales que manejaba el *microservicio de gestión de peticiones* y que llegaban a los servidores eran *.gltf*.

Sin embargo, *Three.js* permite especificar si se desea utilizar el formato binario para la exportación, los cuales ocupan menos espacio, pasando a manejar archivos *.glb* en el sistema.

Por consiguiente, la primera optimización implementada ha consistido en que el *cliente estándar* utilice por defecto esta exportación. Esta no tendrá un gran impacto en el sistema,

pero mejorará tanto los tiempos de espera del usuario mientras espera a que se exporte su escena antes de enviarse, como los tiempos de espera de las transferencias de archivos al ser estos ligeramente más pequeños.

Se podría forzar a que siempre se utilice la exportación binaria, pero se ha optado por ofrecerlo como un parámetro configurable debido a las siguientes razones:

- Se quiere seguir permitiendo que desde otros medios que no sean la interfaz web del *cliente estándar* (*Postman* [36], *JMeter* [9], *curl* [62], otros clientes desarrollados, etc.) se puedan seguir generando peticiones. Estas escenas no son generadas por la exportación de *Three.js*, así que resulta interesante permitir la subida de ambos tipos de ficheros (*.gltf* y *.glb*).

- Ya se contaba con una implementación del sistema que distingue y soporta sin problemas los dos tipos de fichero.

- No es necesario ningún cambio en el servidor de renderizado y *Blender* funciona correctamente con los dos formatos.

En las Figuras 35, 36 y 37 es posible observar los principales cambios introducidos.

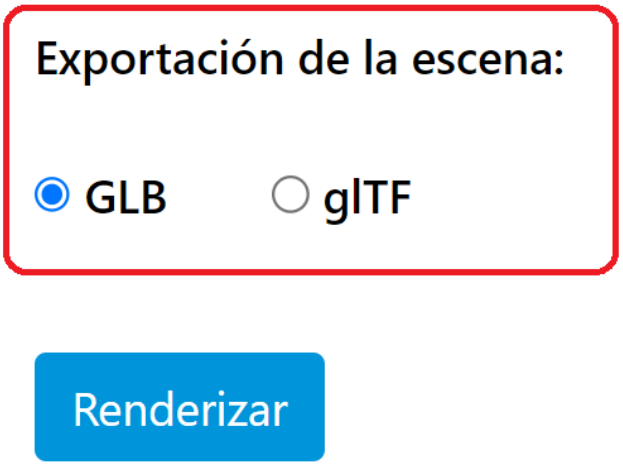

Figura 35: Botones de radio en la interfaz web para seleccionar el formato de exportación

```
// Exportamos la escena y la convertimos en un Blob para enviarla
exporter.parse(scene, async (scene) ⇒ {

  if (glbExport.checked) {
    // Exportación GLB
    formData.append("model", new Blob([scene], { type: "model/gltf-binary" }));
  } else {
    // Exportación glTF
    formData.append("model", new Blob([JSON.stringify(scene, null)], { type: "model/gltf+json" }));
  }
```

Figura 36: Distinto contenido de la petición dependiendo de la exportación seleccionada

```
{ binary: glbExport.checked }
```

Figura 37: Opción *binary* que se le indica al método *parse* de *Three.js*

6.3.3. Distintas opciones de resolución para el renderizado

Hasta ahora el sistema renderizaba por defecto a una resolución de *1080p* (1920×1080 píxeles). Una funcionalidad muy común ofrecida por las granjas de renderizado consiste en la selección de la resolución a la que este proceso se llevará a cabo, por lo que se ha añadido la posibilidad de seleccionar una resolución concreta de entre las más utilizadas con relación de aspecto 16:9, las cuales serían:

- ***480p*** (854×480 píxeles)
- ***720p*** (1280×720 píxeles)
- ***1080p*** (1920×1080 píxeles)
- ***1440p*** (2560×1440 píxeles)
- ***2160p*** (3840×2160 píxeles)

Para implementar este cambio solo ha sido necesario añadir un selector en el *cliente estándar* y propagar el parámetro junto al resto hasta el servidor de renderizado, donde el *script Python* establece la resolución de acuerdo a este.

6.3.4. Nuevas estadísticas

Se han incluido nuevas estadísticas en el panel de administración. En las tarjetas correspondientes a las peticiones de renderizado, estas han sido las siguientes:

- **Motor de renderizado solicitado:** Indica si la petición solicita el motor *Cycles* o el motor *Eevee*.
- **Resolución solicitada:** Muestra la resolución solicitada en la petición.
- **Tipo del fichero correspondiente y su peso:** Informa de la extensión del archivo que maneja el sistema (*.gltf*, *.glb* o *.drc*) y su tamaño.
- **Tiempo empleado por Blender para procesar la petición:** Se corresponde con el valor que imprime *Blender* en su salida estándar cuando termina el renderizado, para obtenerlo, se ha modificado la respuesta del servidor que incluye la imagen renderizada para que también devuelva el tiempo de renderizado.
- **Estado de la transferencia del fichero:** Informa de si el fichero ha sido enviado directamente junto al */render* de la petición o si ha sido enviado mientras la petición se encontraba encolada. Si se ha realizado el envío de forma previa, indicará o bien que todavía no se ha completado la transferencia, o bien que ya ha finalizado junto con el tiempo que ha requerido.

En el caso de las de los servidores de renderizado, la nueva información sería la siguiente:

- **Número de peticiones procesadas:** Total de peticiones renderizadas por el servidor hasta el momento.
- **Total de información procesada:** Sumatorio de los tamaños de los ficheros asociados a todas las peticiones procesadas por el servidor hasta el momento.
- **Total de tiempo que ha necesitado el sistema para procesar todas las peticiones asignadas al servidor:** Sumatorio del tiempo que ha requerido el sistema en su conjunto para procesar todas las peticiones recibidas hasta el momento, teniendo en cuenta el tiempo de renderizado y el de transferencia de archivos, incluso cuando se realizan de forma previa.
- **Valor utilizado como criterio de selección de servidor:** Este es el valor obtenido al aplicar la fórmula cuya selección ha sido detallada detallada en la sección 5.2.3 en el capítulo de estudio de técnicas y optimizaciones.
- **Tiempo total de renderizado en Blender:** Sumatorio de los tiempos de renderizado Blender de todas las peticiones procesadas por el servidor hasta el momento.

- **Cociente entre el total de información procesada por el servidor y el tiempo total requerido por la instancia de *Blender* del servidor para renderizar todas las imágenes de las peticiones recibidas:** Indica la media de *bytes* por milisegundo que ha renderizado el servidor hasta el momento. Esta se incluyó antes de añadir la posibilidad de modificar la resolución del renderizado para comparar la capacidad de renderizado de cada servidor sin tener en cuenta transferencias de ficheros. Sin embargo, debido al hecho de que la resolución afecta también a los tiempos, tras añadir esta nueva funcionalidad resulta menos representativa.

Es preciso destacar que, salvo el número de peticiones totales, toda la demás información se evalúa para los dos motores de renderizado por separado.

6.3.5. Cliente basado en interfaz de línea de comandos (*cliente CLI*)

Durante esta iteración también se ha implementado un nuevo componente del sistema que no estaba planeado desde el principio. Se trata de un nuevo cliente que permite a los usuarios generar peticiones y enviarlas al sistema través de una interfaz de línea de comandos.

Para ello, se ha desarrollado una aplicación de consola con *Node.js* con una lógica muy similar a la del *cliente estándar*, ya que las funcionalidades que ofrece son las mismas, pero adaptadas a la nueva interfaz textual.

6.3.6. Implementación de la paralelización de las transferencias de archivos

De acuerdo con las ideas expresadas en la sección 5.2.3 del capítulo correspondiente al estudio de técnicas y optimizaciones, se ha llevado a cabo la implementación de la nueva lógica que permite la paralelización de las transferencias de ficheros en el sistema.

Para lograrlo, ha sido necesario implementar un nuevo *endpoint /file-transfer* en el *servidor de renderizado*, el cual recibe el archivo con *multer*, pero en este caso únicamente lo almacena. El renderizado no se realizará hasta que el servidor reciba un */render*, que tras los cambios ahora podrá llegar con el fichero incluido (si se trata de una petición que ha podido ser reenviada en cuanto recibida por el *microservicio de gestión de peticiones*), o solo con los parámetros de renderizado (si ya se ha transferido el fichero previamente mientras se encontraba la petición encolada). El panel de administración muestra si se da un caso u otro.

Los cambios en la lógica se han visto reflejados en el código principalmente en la modificación de los esquemas y las consultas de *mongoose*, los cuales reflejan ahora los nuevos algoritmos. Al existir ahora más situaciones posibles, la complejidad de estos ha aumentado con respecto a la anterior implementación.

También ha sido necesario modificar la lógica que se había implementado para la limpieza de archivos en el caso del *servidor de renderizado*. Dado que la lógica anterior solo contemplaba la presencia de los archivos correspondientes a una única petición mientras esta se estaba procesando, cada vez que se terminaba con una, se limpiaba el directorio de ficheros temporales entero para tratar de evitar la acumulación de ficheros debido a errores. Con la nueva lógica, se contempla la presencia de archivos correspondientes a distintas peticiones en el servidor, por lo que ya no es posible borrar todo. Tras los cambios, cuando se termine de procesar una petición, solo se eliminarán los archivos con la imagen renderizada y con la escena 3D correspondientes a la petición finalizada, y solamente el directorio entero cuando se elimine al servidor del sistema.

6.3.7. Mejoras usabilidad

Ha surgido la necesidad de corregir problemas bastante graves de usabilidad y de diseño *responsive* que se han detectado al haber aumentado el número de tarjetas mostradas.

El primero consistía en que al volverse más ancho el diseño de las tarjetas correspondientes a los servidores debido a la mayor cantidad de información que deben mostrar, en tamaños de ventana más pequeños como podría ser la pantalla de un móvil, estas se superponían unas con otras. Se ha solucionado estableciendo un ancho mínimo para estas.

Los otros dos estaban relacionados con el refresco periódico, ya que cada vez que este se llevaba a cabo, restablecía tanto el *scroll* vertical de la página, como los horizontales de cada contenedor de tarjetas.

Para estos últimos, ha bastado con guardar en una variable el estado de desplazamiento previo del *scroll* y restablecerlo una vez se ha generado el contenido actualizado.

Para el vertical, esta idea no ha sido suficiente, pero se ha logrado solucionar modificando el orden en la lógica de refresco. El desplazamiento se generaba porque primero se vaciaban los contenedores de tarjetas y luego se rellenaban, variando la altura de estos cuando se quedaban sin tarjetas. En cambio, si primero se añaden las nuevas tarjetas y luego se eliminan las antiguas, se mantiene la altura constante, evitando este problema.

Por otra parte, se han añadidos algunos iconos y *tooltips* en la sección de opciones del cliente estándar con el objetivo de ayudar y guiar al usuario.

6.3.8. Descarga de imágenes renderizadas desde el panel de administración

También se ha añadido la posibilidad de que un administrador pueda descargar las imágenes renderizadas correspondientes a las peticiones finalizadas del sistema para comprobar la

calidad de estas.

Para la implementación de esta funcionalidad, el principal problema ha consistido en el hecho de que no se dispone del archivo *PNG* generado por el *servidor de renderizado* en el *microservicio de administración*, ya que este es devuelto al *microservicio de gestión de peticiones* para que lo envíe por correo o lo devuelva al cliente directamente.

Inicialmente, se pensó en incluir un nuevo campo en la base de datos que almacenara la imagen, de forma que el *microservicio de gestión de peticiones* lo persistiera al recibirlo y que cuando esta se solicitara indicando el *id* de la petición desde el *cliente de administración*, el microservicio solo tuviera que consultarla en la base de datos y devolverla. Sin embargo, tras implementar esta idea se terminó descartando, ya que las imágenes tienen un tamaño considerable y afectan al rendimiento de la base de datos, además de requerir más trasiego de información que la opción que se detalla a continuación.

En su lugar, se optó por añadir un nuevo *endpoint* auxiliar *POST /requests/:id/rendered-image* mediante el cual el *microservicio de gestión de peticiones* envía la imagen renderizada al *microservicio de administración* una vez recibida esta del *servidor de renderizado*. De esta forma, ya se dispondrá del archivo *PNG* en el sistema de ficheros local del microservicio si lo solicita un administrador, el cual permanecerá almacenado hasta que se elimine la petición correspondiente del sistema.

La opción de descarga se ha añadido en el panel de administración mediante un nuevo botón de descarga que incluirán las tarjetas correspondientes a peticiones finalizadas.

6.3.9. Optimización adicional

La fase final del proceso de desarrollo ha consistido en poner a prueba la posible optimización mencionada en la sección 5.2.4 del capítulo dedicado al estudio de estas.

Para implementarla, se ha partido de un código funcional previamente suministrado con el objetivo de adaptarlo e integrarlo en el sistema ya desarrollado. Sin embargo, esta tarea ha presentado numerosas dificultades, ya que se estaba tratando de utilizar un código desarrollado para ejecutarse en navegadores, el cual hacía uso de módulos diseñados específicamente para su uso en estos entornos, en un entorno *Node.js*. A pesar de que existen módulos *npm* que pueden emular algunos de los elementos propios de *JavaScript* en el navegador, como *jsdom* y *canvas*, su utilización no ha resultado suficiente para conseguir que todo funcionara correctamente debido a problemas relacionados con el uso de *WebGL* que realiza *Three.js*, entre otros.

Por estas razones, y considerando que el único objetivo de esta ejecución de código es

obtener el archivo *.gltf* resultante, que sirve como base para el renderizado posterior, se ha acabado recurriendo a una solución menos ideal, pero efectiva para lograr los objetivos de este experimento.

Esta ha consistido en la utilización de *puppeteer* [32], una librería de *npm* desarrollada por *Google* que permite controlar un navegador desde el código. De esta forma, se ha configurado el *servidor de renderizado* para que se sirva a sí mismo el contenido necesario para la ejecución del código en una nueva carpeta *public* creada para este propósito, de forma que desde el propio servidor se pueda invocar con *puppeteer* un navegador y que este se encargue de ejecutar el código en un entorno adecuado dentro del mismo *host*, ya que de esta ejecución solo interesa el fichero generado, el cual es generado en el directorio */temp* que utiliza el servidor para obtener las escenas.

También ha sido necesaria la introducción de cambios menores y bastante simples en otros componentes de la aplicación para manejar correctamente los nuevos ficheros *.txt*.

6.3.10. Problemas encontrados y resultados de la iteración

Además de las dificultades mencionadas en los apartados anteriores, los principales obstáculos de esta iteración han estado relacionados con la complejidad de los cambios realizados. Esta, junto a la gran cantidad de pruebas que ha sido necesario realizar para comprobar el correcto funcionamiento de todo, ha provocado que el desarrollo tomara bastante tiempo.

Al final de esta iteración, se podría dar por concluido el desarrollo realizado en este trabajo, puesto que todas las funcionalidades y optimizaciones requeridas han sido implementadas, y la incorporación parcial del código relacionado con la optimización experimental ha permitido llevar a cabo el estudio que se trataba de realizar.

7

Despliegue y pruebas

En este capítulo se tratan los aspectos relacionados con los despliegues que se han configurado y las pruebas que se han ido realizando a medida que ha ido avanzando el desarrollo.

7.1. Entornos de despliegue

7.1.1. Entorno de despliegue en red local

El entorno donde se han realizado las pruebas a medida que avanzaba la implementación del sistema ha estado constituido por dos equipos, uno de sobremesa y otro portátil, cuyas especificaciones se pueden consultar en las Figuras 38 y 39. Estos se comunican entre sí a través de una red local privada doméstica.

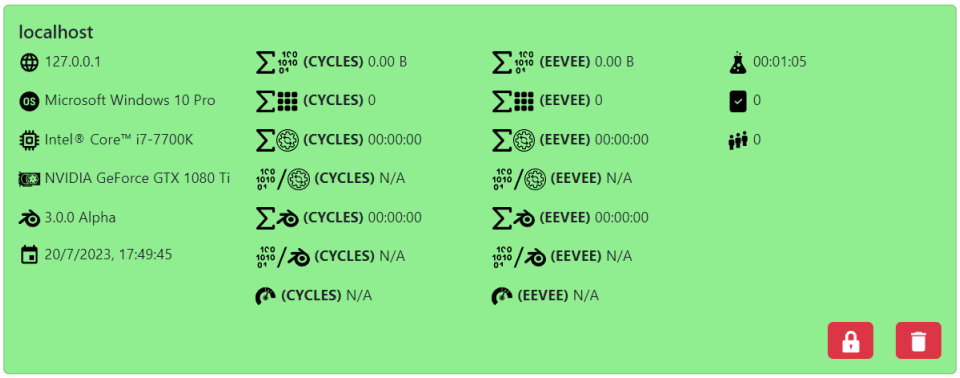

Figura 38: Tarjeta del panel de administración con la información correspondiente al ordenador de sobremesa que se ha utilizado durante las pruebas

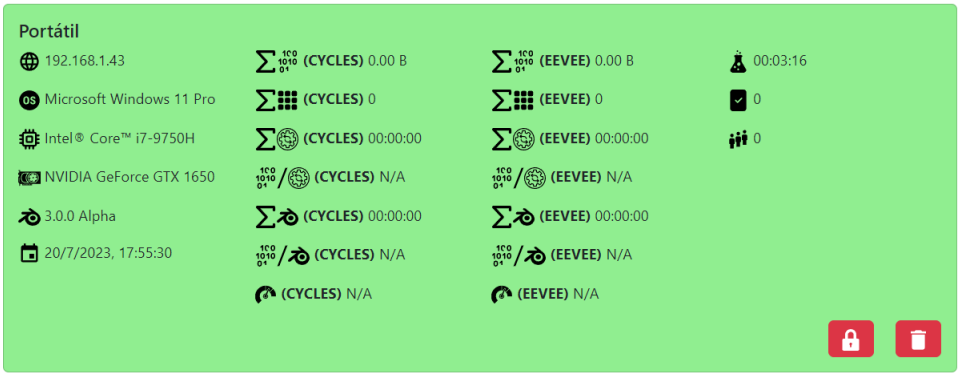

Figura 39: Tarjeta del panel de administración con la información correspondiente al ordenador portátil que se ha utilizado durante las pruebas

Se ha tratado de comprobar que todo funcionaba correctamente con una serie de configuraciones variadas. Ambos equipos actuando como servidor de renderizado, todos los microservicios desplegados en uno de estos equipos, unos microservicios ejecutándose sobre uno y otros sobre el otro, etc. Estas pruebas se han llevado a cabo desplegando los componentes directamente sobre el *host*.

Con respecto al despliegue con contenedores *Docker*, se ha probado las principales modalidades de despliegue que se ofrecerán al usuario. Estas serían:

- Despliegue de forma conjunta con *Docker Compose* [27].
- Despliegue de cada componente de forma independiente utilizando directamente los archivos *Dockerfile* [28] correspondientes.
- Despliegue constituido por componentes desplegados sobre contenedores y componentes que se están ejecutando directamente en el *host*.

La Figura 40 muestra un diagrama de despliegue que ilustra una posible configuración utilizando este entorno.

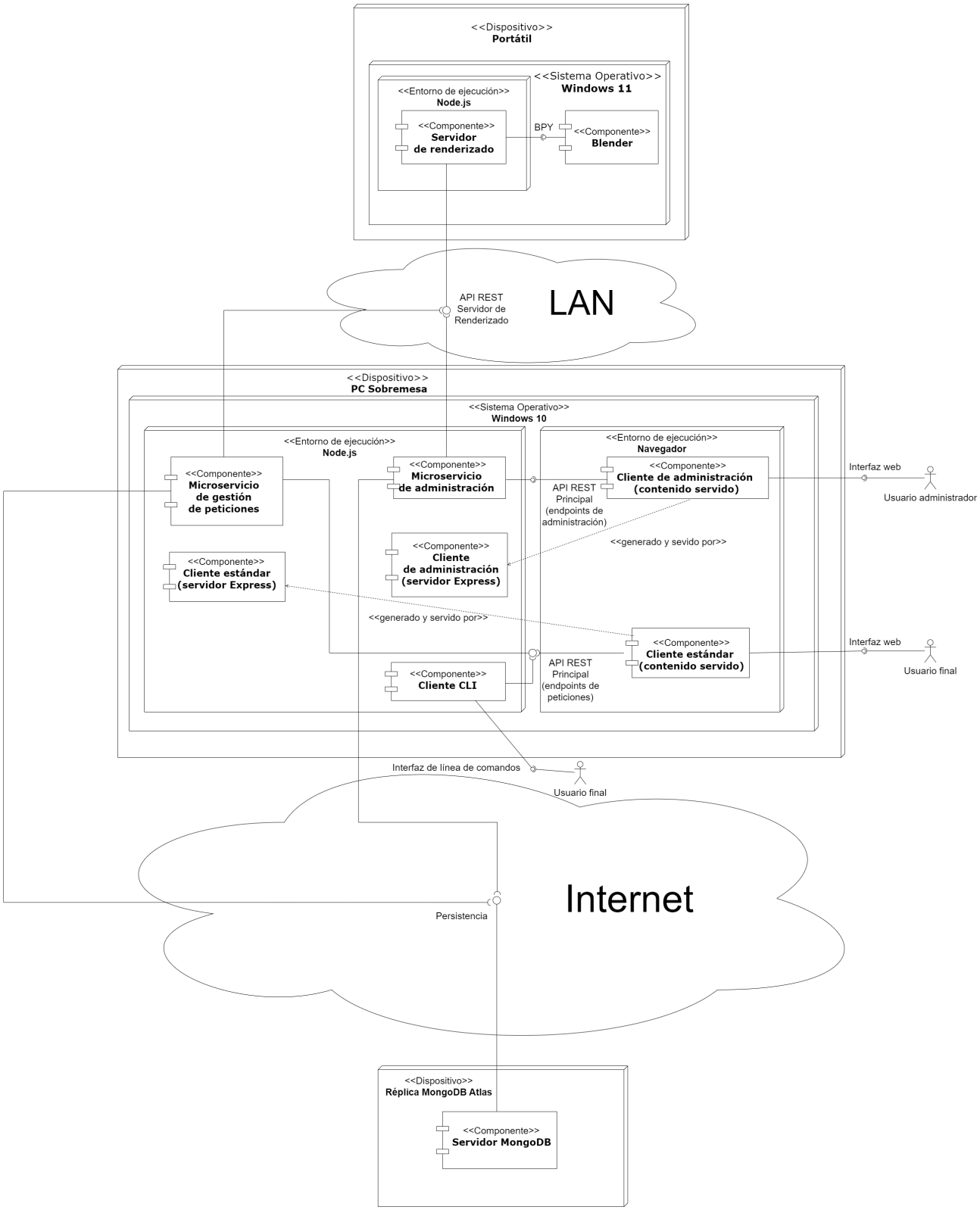

Figura 40: Diagrama de despliegue de una posible configuración utilizando el entorno de despliegue local

El flujo de trabajo ha consistido en ir implementando funcionalidades y realizando pruebas una vez incorporados los cambios, tratando de corregir los errores que han ido surgiendo a medida que estos iban siendo descubiertos.

Tras ir realizando pruebas y solucionando errores utilizando este entorno, se ha llegado a una serie de conclusiones:

- El hecho de utilizar una red local doméstica supone un inconveniente para el despliegue, ya que las direcciones *IP* son normalmente dinámicas en este tipo de entornos, cuando lo que interesa en el contexto de la aplicación es que estas sean fijas.

- Aunque se puede hacer funcionar sin problemas, el despliegue con *Docker* puede resultar más complejo que el que se realiza directamente sobre los *hosts* debido a la configuración de red que es necesario realizar a través de variables de entorno. Es posible que sea necesario documentarse un poco sobre conceptos básicos de cómo funcionan las redes en *Docker*.

- Debido al funcionamiento de las redes de tipo *bridge* [26] de *Docker*, ya no será posible filtrar por dirección *IP* directamente en los componentes, ya que estos siempre recibirán la misma, que sería la correspondiente a la del *host* sobre el que se ejecuta *Docker*. Para filtrar por *IP*, será necesario un nuevo enfoque, como por ejemplo la configuración de un *proxy* inverso.

7.1.2. Uso de *ngrok*

Frente a los inconvenientes que supone utilizar la red local para las pruebas, se ha recurrido a *ngrok* [35], una herramienta gratuita que permite exponer servidores locales para que sean accesibles desde cualquier parte a través de internet generando una *URL* pública temporal que conecta con servicios locales a través de un túnel seguro.

En la Figura 41 es posible visualizar un ejemplo de este proceso para los componentes *cliente de administración* (escuchando en el puerto 8080) y *microservicio de administración* (escuchando en el puerto 9000), para los cuales se han generado *URL*s accesibles desde cualquier parte de internet.

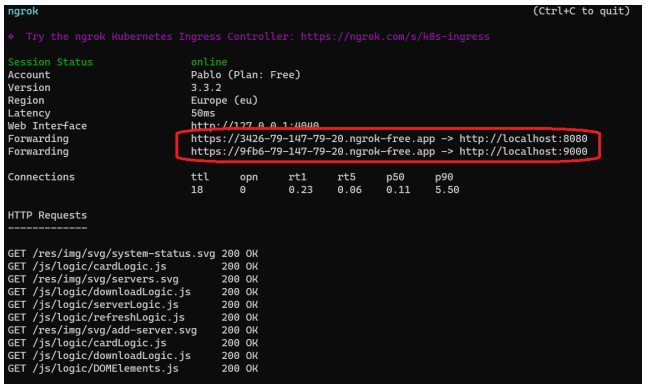

Figura 41: Interfaz de línea de comandos de *ngrok* tras crear dos túneles a partir de una archivo de configuración en formato *YAML*

Esta configuración no sería la más adecuada para un entorno de producción, pero resulta muy conveniente a la hora de realizar pruebas en el sistema cuando se dispone de un despliegue local, ya que las alternativas serían abrir los puertos en la red local, lo cual puede resultar más complejo, o configurar un despliegue con *hosts* remotos y direcciones *IP* públicas utilizando algún servicio en la nube, lo cual, sobretodo debido al uso intensivo de *GPU* que requiere el componente *servidor de renderizado*, puede traducirse en facturas elevadas a tener en cuenta si las pruebas a realizar suponen una gran demanda de renderizado.

Al llevar a cabo las modificaciones necesarias para que funcionara todo a la hora de realizar pruebas con esta herramienta, ha surgido la necesidad de cambiar el diseño de las variables de entorno, ya que hasta ahora se indicaban la dirección *IP* del *host* y el puerto del servicio por separado, los cuales se utilizaban para construir las *URL*s para las comunicaciones *HTTP*.

Al usar un nombre de dominio no hay que indicar el puerto, ya que va implícito en este, por lo que el planteamiento anterior no se adaptaba correctamente para permitir las dos opciones. Esto ha llevado a la sustitución en todos los componentes de las variables de entorno correspondientes a *hosts* y puertos (excepto el puerto del *servidor de renderizado,* cuya dirección *IP* es especificada por el usuario), por variables para la base de la *URL*, las cuales permitirán especificar el protocolo *HTTP* (*http://*) o *HTTPS* (*https://*), distinguir entre el uso de dirección *IP* y puerto o nombre de dominio y simplificar el código al construir los *endpoints* a los que se realizan peticiones de una forma más sencilla.

Por ejemplo, para realizar una comunicación con el *microservicio de administración*, se pasaría a indicar su *URL* de la forma *ADMINISTRATION_MICROSERVICE_HOST=<dirección IP>, ADMINISTRATION_MICROSERVICE_PORT=<puerto>*, a especificarla de la forma *ADMINISTRATION_MICROSERVICE_URL=<protocolo>://<dirección IP>:<puerto>* en el caso de que se utilice dirección *IP* y puerto, o *ADMINISTRATION_MICROSERVICE_URL=<protocolo>://<nombre*

de dominio> en el caso de que solo se hagan uso de nombres de dominio.

7.1.3. Entorno de despliegue remoto

Una vez se ha contado con un desarrollo más avanzado y probado con la configuración detallada anteriormente, se ha utilizado un entorno de despliegue más real para seguir realizando pruebas.

En este caso, se cuenta con tres equipos:

- ***renderizado-master*** Se trata de una máquina virtual remota con el sistema operativo *Ubuntu*. Sobre esta se encuentran desplegados todos los componentes necesarios para el funcionamiento de la aplicación, excepto el *servidor de renderizado*. Esta es la máquina encargada de la administración y gestión del sistema.

  Para comprobar que el sistema también funciona correctamente con un despliegue local de *MongoDB* en lugar de la opción en la nube que se había utilizado hasta el momento, se ha configurado en esta máquina también un servidor local de esta base de datos.

- ***renderizado-server1*** Este equipo consiste en otra máquina virtual remota con *Ubuntu*, la cual alberga un servidor de renderizado. Es importante destacar que esta no dispone de *GPU* y que se encuentra en la misma red que *renderizado-master*.

- ***PC Sobremesa*** Este se correspondería con el *PC* de sobremesa detallado en el entorno de despliegue local, el cual será local en el caso presentado, pero remoto desde el punto de vista del sistema, ya que los componentes se encuentran desplegados en remoto.

  Este actuará como cliente desde el que se generarán las peticiones de administración y de renderizado, las cuales serán satisfechas por *renderizado-master*. Además, adoptará también el papel de un servidor de renderizado adicional del sistema. Para esta conexión se ha utilizado la dirección *IP* pública de la red doméstica de la que forma parte tras haber abierto el puerto correspondiente.

Esta nueva configuración permitirá probar el sistema en unas condiciones más realistas, principalmente por las siguientes razones:

- Ahora el sistema utiliza direcciones *IP* estáticas y públicas, permitiendo que un usuario se puede conectar desde cualquier dispositivo.

- Los componentes del sistema y los clientes servidos a los usuarios se encuentran en redes distintas, lo que implica que el tráfico tiene que atravesar internet.

- El sistema se encuentra desplegado sobre un sistema operativo *Linux*, el cual es muy común en entornos de producción.

En la Figura 42 se muestra de nuevo un diagrama de despliegue, en este caso correspondiente a la nueva configuración. Aunque se ha tratado como cliente al propio *PC* de sobremesa, cualquier dispositivo con acceso a internet y un navegador podría actuar como cliente.

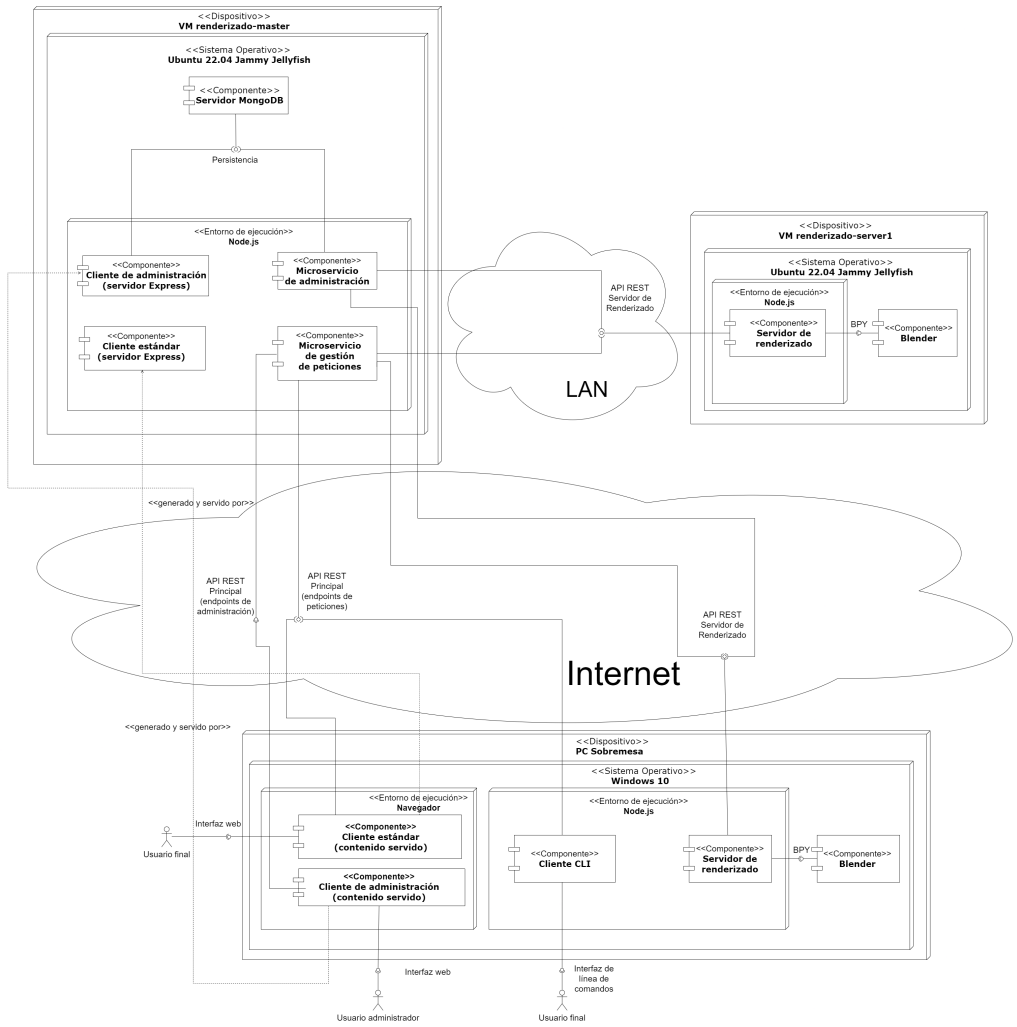

Figura 42: Diagrama de despliegue de la configuración utilizada en el entorno de despliegue remoto

7.2. Pruebas realizadas

7.2.1. Pruebas de los *endpoints* con *Postman*

Para comprobar el correcto funcionamiento de los *endpoints* que se han ido desarrollando y que la correspondiente validación de los datos que estos pueden recibir se comportara de

forma adecuada, se ha utilizado *Postman*.

Postman es una herramienta muy utilizada cuando surge la necesidad de trabajar con *API*s. Esta ofrece una interfaz gráfica que facilita el proceso de desarrollo y prueba de los distintos *endpoints*, permitiendo interactuar con servicios web de forma sencilla. Aunque ofrece numerosas funcionalidades, se han hecho uso principalmente de sus colecciones, las cuales permiten definir plantillas para las peticiones a los distintos componentes de la aplicación fácilmente configurables y reutilizarlas, agilizando el proceso de prueba.

La Figura 43 muestra las colecciones que se han utilizado para los *endpoints* de la *API*. Estos se encuentran organizados por el componente de la aplicación que se encarga de implementarlos.

Figura 43: Colecciones utilizadas en *Postman*

7.2.2. Pruebas manuales de las interfaces de usuario de los distintos clientes

Con respecto a las tres interfaces de usuario que ofrece la aplicación (correspondientes a los componentes *cliente estándar*, *cliente de administración* y *cliente CLI*), las pruebas se han realizado de forma manual debido a la sencillez de estos.

Tras algunas mejoras y correcciones en aspectos menores sin demasiada importancia, gracias a estas se han identificado problemas de usabilidad en al panel de administración debido al refresco periódico que se realiza en este. Las medidas que se han tomado para solucionarlos se detallan en la sección 6.3.7 en el capítulo dedicado a la implementación.

7.2.3. Pruebas de carga y de estrés con *JMeter*

Los objetivos de estas pruebas consistían en, por un lado, comprobar que las distintas versiones de la lógica relacionada con el manejo de peticiones y servidores en el sistema funcionaban correctamente y, por otro, asegurar que el sistema sea capaz de soportar peticiones concurrentes y un mayor número de estas.

Para lograrlos, se ha utilizado *JMeter*, una herramienta de código abierto desarrollada por la *Apache Software Foundation* que permite crear escenarios de prueba personalizados que simulan múltiples usuarios enviando solicitudes a un servidor.

La alta configurabilidad que ofrece este programa ha permitido simular distintas situaciones en el sistema, como situaciones de estrés en el que llegan numerosas peticiones en muy poco tiempo o escenarios más tranquilos en las que llegan peticiones de varios equipos distintos de forma indefinida con periodos aleatorios entre una petición y la siguiente.

La Figura 44 muestra algunos detalles de la configuración para este último. A la izquierda, es posible observar cómo se ha definido un bucle que realiza una petición *HTTP* y lleva a cabo una espera aleatoria antes de ejecutarse una nueva iteración. A la derecha, se puede observar la configuración de la petición (en esta no se puede apreciar el fichero que se va a enviar junto a esta, el cual se especifica en la pestaña *Files Upload*).

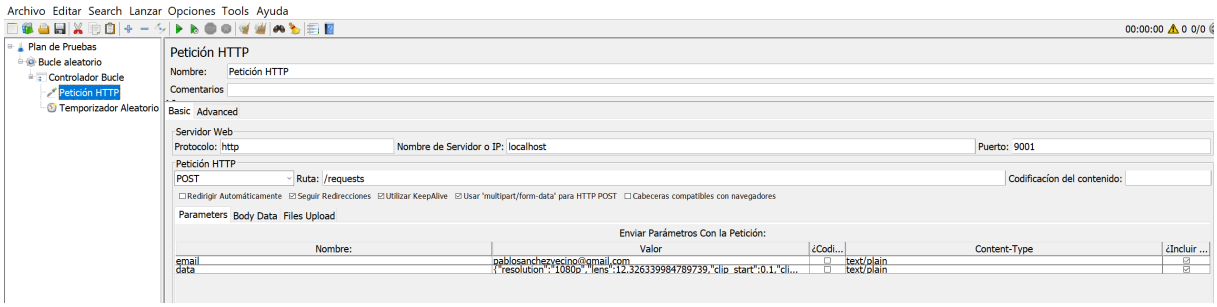

Figura 44: Configuración de una petición *HTTP* correspondiente a una petición de renderizado con *JMeter*

7.2.4. Análisis de tiempos sobre una configuración de despliegue concreta

Una vez configurado el despliegue en remoto detallado anteriormente, se ha realizado un análisis de tiempos para obtener información sobre cómo se comporta el sistema en un entorno más realista.

A la hora de realizar las pruebas, se han utilizado ficheros de distintos tamaños para com-

probar cómo afecta a los tiempos de espera el incremento del tamaño de las escenas enviadas. Es importante tener en cuenta que el tamaño manejado por el sistema no es el que muestra el fichero cargado en el visor de la aplicación, sino el de la escena que exporta *Three.js*, que es la que se envía este.

Los archivos utilizados son:

- ***suzanne.glb*:** Como ejemplo de fichero de tamaño reducido se ha utilizado una escena muy simple con *Suzanne*, la mascota de *Blender*. La escena exportada no ocupa apenas espacio (36,53 *KiB*), siendo el principal modelo de la escena un recurso predeterminado y reconocido por *Blender* (Figura 45).

  Los valores obtenidos para esta escena se pueden consultar en las Tablas 1 y 2.

- ***bmw.glb* :** Esta escena de 5 *MiB* con un coche es algo más compleja y se ha utilizado como punto medio para el tamaño de la escena (Figura 46).

  Los valores obtenidos para esta escena se pueden consultar en las Tablas 3 y 4.

- ***bike.glb* :** Para las pruebas con un fichero de tamaño elevado, se ha recurrido a esta escena de 35,02 *MiB*, la cual contiene una moto con una geometría más compleja que la anterior (Figura 47).

  Los valores obtenidos para esta escena se pueden consultar en las Tablas 5 y 6.

Con cada escena, se ha generado la petición cinco veces de manera que fuera asignada a los dos servidores disponibles (cinco veces al *PC* de sobremesa y cinco veces a la máquina virtual remota).

Para todas ellas, se han utilizado la misma configuración para el renderizado, el cual se ha llevado a cabo a una resolución de *1080p* y con el motor de renderizado *Cycles*. Para los renderizados de una misma escena, se han utilizado las mismas fuentes de luz y los mismos parámetros en la cámara, tratando de que las escenas exportadas y las imágenes generadas fueran iguales.

Los valores medidos son los siguientes:

- **Tiempo en completar transferencia del cliente al microservicio de gestión de peticiones:** Tiempo requerido para realizar la primera transferencia desde el *cliente estándar* servido al usuario y el *microservicio de gestión de peticiones*.

- **Tiempo en completar transferencia del microservicio al servidor de renderizado:** Tiempo requerido para enviar la escena desde el *microservicio de gestión de peticiones* al *servidor de renderizado* asignado.

- **Tiempo de renderizado en Blender:** Tiempo requerido por la instancia de *Blender* del servidor para llevar a cabo el proceso de renderizado.
- **Tiempo total requerido por el sistema:** Tiempo transcurrido desde la generación de la petición en el sistema hasta su finalización. Se correspondería con la suma de los dos valores anteriores más una pequeña cantidad de *overhead* adicional del sistema (validaciones, comunicaciones con la base de datos, manejo interno de ficheros, etc).

A continuación, se muestran los valores obtenidos en cada prueba. Debajo de la Figura correspondiente a cada escena, se muestra una Tabla por cada servidor de renderizado, la cual incluye los tiempos obtenidos en cada prueba y el valor medio en la última fila.

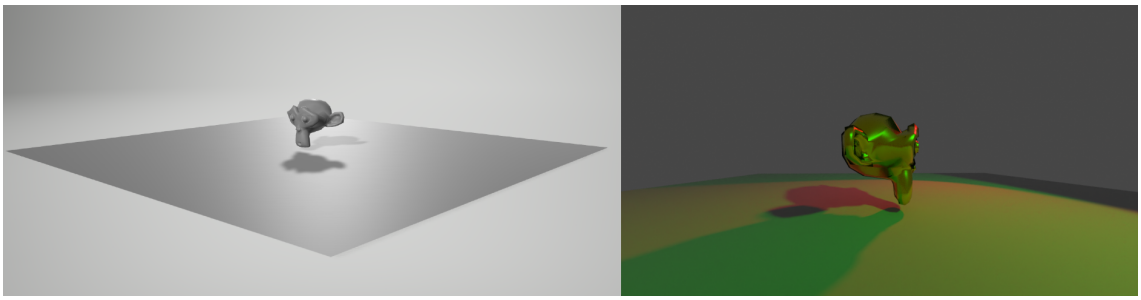

Figura 45: Visualización (izquierda) e imagen renderizada (derecha) de las escena *suzanne.glb*

	Escena: suzanne.glb (36,53 KiB) Servidor: *PC* sobremesa			
	Tiempo en completar transferencia del cliente al microservicio de gestión de peticiones	**Tiempo en completar transferencia del microservicio al servidor de renderizado**	**Tiempo de renderizado en Blender**	**Tiempo total requerido por el sistema**
1	573ms	189ms	49s	52s
2	269ms	171ms	51s	52s
3	235ms	272ms	51s	52s
4	249ms	191ms	48s	49s
5	150ms	170ms	47s	48s
Media	**295ms**	**199ms**	**49s**	**51s**

Cuadro 1: Tiempos obtenidos para la escena *suzanne.glb* utilizando como servidor de renderizado el *PC* de sobremesa.

	Escena: suzanne.glb (36,53 KiB) Servidor: Máquina virtual			
	Tiempo en completar transferencia del cliente al microservicio de gestión de peticiones	**Tiempo en completar transferencia del microservicio al servidor de renderizado**	**Tiempo de renderizado en Blender**	**Tiempo total requerido por el sistema**
1	488ms	9ms	1min 51s	1min 53s
2	329ms	7ms	1min 52s	1min 53s
3	236ms	7ms	1min 55s	1min 55s
4	329ms	7ms	1min 53s	1min 54s
5	275ms	8ms	1min 54s	1min 55s
Media	**331ms**	**8ms**	**1min 53s**	**1min 54s**

Cuadro 2: Tiempos obtenidos para la escena *suzanne.glb* utilizando como servidor de renderizado la máquina virtual.

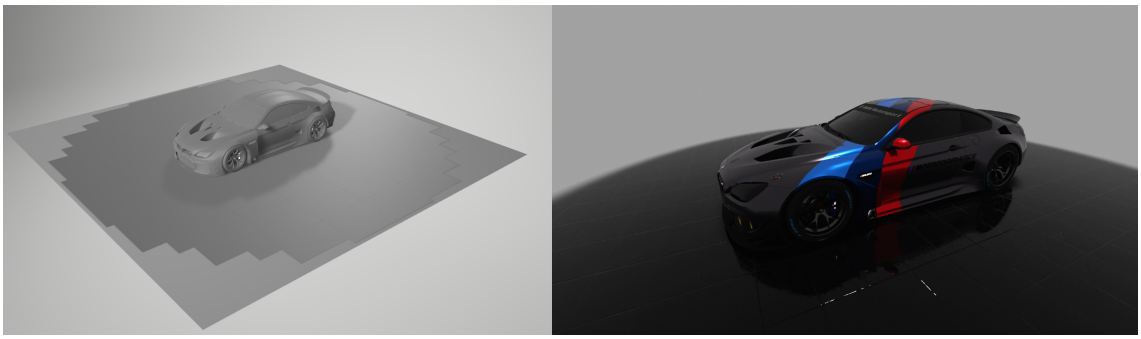

Figura 46: Visualización (izquierda) e imagen renderizada (derecha) de las escena *bmw.glb*

	Escena: bmw.glb (5 MiB) **Servidor: *PC* sobremesa**			
	Tiempo en completar transferencia del cliente al microservicio de gestión de peticiones	**Tiempo en completar transferencia del microservicio al servidor de renderizado**	**Tiempo de renderizado en Blender**	**Tiempo total requerido por el sistema**
1	10s	11s	49s	1min 1s
2	10s	10s	50s	1min 2s
3	10s	10s	48s	1min 1s
4	10s	10s	48s	1min
5	9s	10s	48s	1min 1s
Media	**10s**	**10s**	**49s**	**1min 1s**

Cuadro 3: Tiempos obtenidos para la escena *bmw.glb* utilizando como servidor de renderizado el *PC* de sobremesa.

	Escena: bmw.glb (5 MiB) **Servidor: Máquina virtual**			
	Tiempo en completar transferencia del cliente al microservicio de gestión de peticiones	**Tiempo en completar transferencia del microservicio al servidor de renderizado**	**Tiempo de renderizado en Blender**	**Tiempo total requerido por el sistema**
1	11s	62ms	2min 32s	2min 33s
2	13s	45ms	2min 17s	2min 18s
3	10s	98ms	2min 16s	2min 17s
4	10s	50ms	2min 11s	2min 12s
5	14s	70ms	2min 13s	2min 14s
Media	**12s**	**65ms**	**2min 18s**	**2min 19s**

Cuadro 4: Tiempos obtenidos para la escena *bmw.glb* utilizando como servidor de renderizado la máquina virtual.

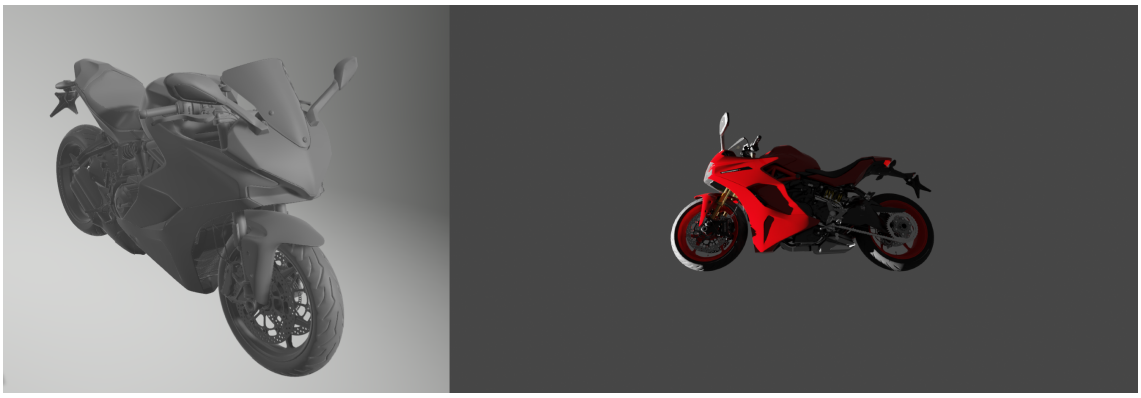

Figura 47: Visualización (izquierda) e imagen renderizada (derecha) de las escena *bike.glb*

	Escena: bike.glb (35,02 MiB) Servidor: *PC* sobremesa			
	Tiempo en completar transferencia del cliente al microservicio de gestión de peticiones	**Tiempo en completar transferencia del microservicio al servidor de renderizado**	**Tiempo de renderizado en Blender**	**Tiempo total requerido por el sistema**
1	41s	43s	50s	1min 35s
2	40s	41s	50s	1min 33s
3	41s	41s	49s	1min 33s
4	43s	40s	49s	1min 32s
5	42s	40s	49s	1min 32s
Media	**41s**	**41s**	**49s**	**1min 33s**

Cuadro 5: Tiempos obtenidos para la escena *bike.glb* utilizando como servidor de renderizado el *PC* de sobremesa.

	Escena: bike.glb (35,02 MiB) Servidor: Máquina virtual			
	Tiempo en completar transferencia del cliente al microservicio de gestión de peticiones	**Tiempo en completar transferencia del microservicio al servidor de renderizado**	**Tiempo de renderizado en Blender**	**Tiempo total requerido por el sistema**
1	40s	391ms	2min 42s	2min 47s
2	44s	412ms	2min 44s	2min 49s
3	41s	304ms	2min 42s	2min 48s
4	41s	395ms	2min 46s	2min 51s
5	40s	290ms	2min 44s	2min 49s
Media	**41s**	**358ms**	**2min 44s**	**2min 48s**

Cuadro 6: Tiempos obtenidos para la escena *bike.glb* utilizando como servidor de renderizado la máquina virtual.

Los valores obtenidos proporcionan la información necesaria para llegar a una serie de conclusiones:

- Aunque el sistema no lo monitorice debido a que aún no tiene constancia de la petición en ese momento, es necesario tener en cuenta el tiempo requerido para la transferencia de la escena desde el cliente hasta el *microservicio de gestión de peticiones*, ya que con ficheros grandes estos tiempos son considerables.

- Un factor que influye en los tiempos de renderizado es que el *PC* de sobremesa cuenta con una *GPU* y especificaciones superiores a las de la máquina virtual, que solo puede renderizar utilizando la *CPU*. Como resultado, se obtienen tiempos de renderizado más reducidos en el primero.

- Es posible observar que los tiempos de transferencia de la escena al servidor de renderizado son mucho menores cuando se utiliza la máquina virtual, lo cual se debe a que ambas máquinas están en la misma red, lo que favorece la transferencia.

  Frente a esta situación, en el caso del *PC* de sobremesa, este se encuentra en una red remota más inestable y, aunque la red no es especialmente lenta, la situación del equipo con respecto al *router* a través del cual se realizan las conexiones con el exterior de la red contribuye a que su velocidad se vea decrementada de forma considerable, dando lugar a tiempos de transferencia más elevados.

- Atendiendo a las Tablas 1, 3 y 5, correspondientes al *PC* de sobremesa, veremos que los tiempos de transferencia de la escena desde el cliente hasta microservicio son muy parecidos a los de las transferencias del microservicio al servidor de renderizado, esto se debe a que en este caso el *PC* de sobremesa actúa tanto de cliente como de servidor de renderizado, por lo que ambas transferencias serían similares, solo intercambiándose el papel de cliente y servidor de las máquinas.

- Aunque en un principio puede parecer que la potencia del servidor de renderizado es el principal factor a tener en cuenta para reducir los tiempos de espera, se puede observar que una buena configuración de red que permita llevar a cabo las transferencias de forma veloz resulta esencial también. Esto debería ser tenido en cuenta a la hora de desplegar el sistema y añadirle servidores de renderizado.

  Sin embargo, con respecto a la transferencia desde el cliente hasta el sistema, esta no dependerá tanto de la configuración de despliegue seleccionada, ya que los tiempos se verán afectados principalmente por el entorno del usuario.

7.2.5. Pruebas con balanceador de carga *NGINX*

La propuesta principal para lograr un sistema escalable consistía en añadir la posibilidad de añadir servidores de renderizado en función del aumento en la demanda de procesamiento. Sin embargo, incluso tras el desarrollo de la extensión, este todavía cuenta con un punto crítico que impactará negativamente en situaciones de alta carga.

Este punto débil se encuentra en el *microservicio de gestión de peticiones*. Este es responsable de realizar una serie de acciones por cada petición que llegue al sistema. Aunque estas no son

excesivamente costosas, el aumento significativo de peticiones podría llevar a un eventual deterioro del rendimiento o incluso al fallo del componente. Por ejemplo, podrían presentarse situaciones en las que se necesite enviar simultáneamente a múltiples servidores un conjunto considerable de escenas, agotando el ancho de banda, o cuando se requiera monitorizar el estado de numerosas peticiones y transmitir las imágenes renderizadas a numerosos clientes al mismo tiempo.

La solución a este problema pasa por el escalado horizontal de este componente, permitiendo la distribución de la carga entre múltiples instancias del microservicio. Como se ha detallado en la sección 4.2.1 dedicada a la arquitectura de la aplicación, esta necesidad se ha tenido en cuenta durante la selección del enfoque basado en microservicios y a la hora de implementar el componente. En consecuencia, este último es compatible con el despliegue de distintas instancias y el uso de un balanceador de carga para distribuir el tráfico entre sus distintas réplicas.

El objetivo de esta prueba consiste en verificar que el sistema sigue funcionando correctamente utilizando más de una instancia del *microservicio de gestión de peticiones*. Para su realización , se ha recurrido a *NGINX* [29], un servidor web y proxy inverso de código abierto muy conocido y utilizado que con frecuencia es empleado como balanceador de carga. En la Figura 48 se muestra el contenido del fichero *nginx.conf* que se encarga de configurar *NGINX* como balanceador de carga en la prueba realizada, cuyos contenidos se detallan más adelante.

```
18  http {
19      upstream microservicio-gestion-peticiones {
20          ip_hash;
21          server 164.92.169.220:9001;
22          server 127.0.0.1:9001;
23      }
24
25      server {
26          listen 10001;
27          location /requests {
28              proxy_pass http://microservicio-gestion-peticiones;
29          }
30      }
31  }
```

Figura 48: Contenido del fichero *nginx.conf* utilizado para configurar *NGINX* para que actúe como balanceador de carga

Al replicar el microservicio, surge un desafío en el proceso de devolución de imágenes renderizadas al cliente. Cuando un servidor de renderizado finaliza un renderizado y envía la

imagen resultante de vuelta al microservicio, es preciso asegurarse de que el cliente solicita la imagen a la misma instancia del microservicio, ya que si se realiza a otra, esta no contará con el fichero solicitado.

Para hacer frente a este problema, se ha aprovechado *ip_hash*, uno de los modos que permite especificar *NGINX* para la selección del nodo al que reenviar el tráfico, el cual garantiza que las solicitudes provenientes de una misma dirección *IP* sean siempre redirigidas al mismo nodo, solucionando de esta forma esta limitación.

Atendiendo a la configuración realizada en el fichero, en el apartado *upstream* se indican los *hosts* a los que se redirigirá el tráfico con la palabra reservada *server*. En este caso se han configurado el *PC* de sobremesa (que también alberga la instancia de *NGINX*) y la máquina virtual *renderizado-master*, los cuales cuentan cada uno con una instancia del microservicio escuchando en el puerto 9001.

Antes de estos se especifica el criterio de selección del nodo. Por los motivos detallados anteriormente, se utiliza *ip_hash*, pero también es posible indicar una asignación *round robin* o asignar pesos a cada nodo, entre otras opciones.

Más abajo, en el apartado *server*, con *listen* se especifica el puerto de escucha de *NGINX*. Dado que el *proxy inverso* se encuentra desplegado también en el *PC* de sobremesa, surge la necesidad de utilizar un puerto diferente para evitar conflictos, por lo que se ha utilizado el 10001. Este es el puerto al que tendrá que dirigirse el cliente cuando necesite contactar con el microservicio, *NGINX* se encargará de redirigir el tráfico de forma transparente a una de las réplicas.

A continuación, se especifica en *location* la ruta que se tendrá que recibir para llevar a cabo la redirección, la cual se correspondería con */requests*.

Finalmente, con *proxy_pass* se especifica el *upstream* definido correspondiente, al cual se le ha proporcionado el nombre *microservicio-gestion-peticiones*.

En relación al resto de componentes de la aplicación, su escalado horizontal no constituiría una necesidad tan imperativa. Aún así, se ha contemplado la posibilidad de replicarlos también:

- Por un lado, replicar los clientes no presentaría problemas, ya que solo se encargan de servir contenido, habiéndose comprobado su correcto funcionamiento incluso atendiendo a la misma petición más de una instancia de forma que los recursos recibidos en un mismo cliente provengan unos de una instancia y otros de otra.
- Sin embargo, se plantea un problema en el caso del *microservicio de administración*. Este está relacionado con la trasferencia de la imagen renderizada que se realiza desde el *microservicio de gestion de peticiones* a este cuando esta es devuelta por el *servidor de*

renderizado para que el administrador pueda descargarla desde el panel de administración. Surgiría la necesidad de determinar en qué instancia se almacena la imagen, o de almacenarla en todas.

Aunque la introducción de estas modificaciones es factible, tratando de no incrementar la complejidad de las transferencias de archivos, se ha optado por no modificar este aspecto y evitar replicar este microservicio, ya que las funciones de administración están concebidas para un número reducido de usuarios administradores, por lo que una única instancia debería ser suficiente para atender las peticiones.

7.2.6. Análisis de tiempos tras incorporación de optimización adicional

En el contexto del estudio de la viabilidad de la optimización adicional detallada en la sección 5.2.4, se han realizado pruebas para comparar los tiempos obtenidos tanto haciendo uso como prescindiendo de esta técnica.

Para estas se ha utilizado el mismo entorno de despliegue detallado en la sección 7.1.3, con el *PC* de sobremesa actuando como cliente y como *servidor de renderizado* al mismo tiempo, y el resto de componentes del sistema desplegados en remoto.

La escena utilizada ha sido *bedroom.glb* (Figura 49) debido a que el código y los recursos que permiten llevar a cabo estas pruebas han sido preparados para funcionar con esta.

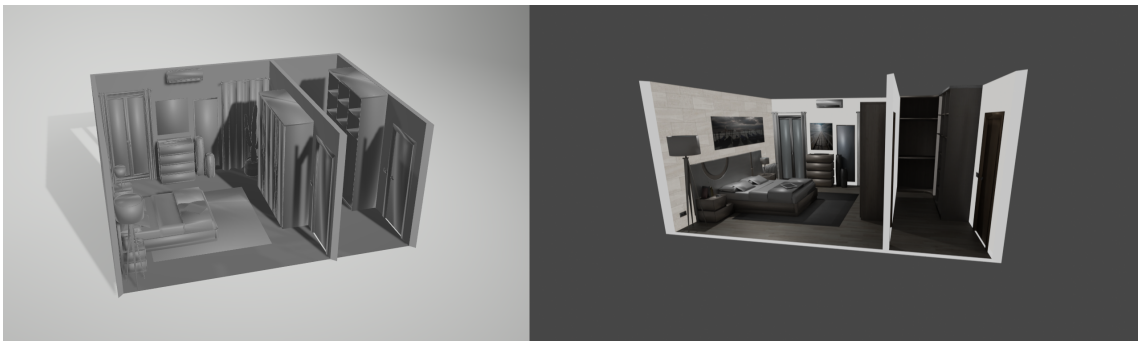

Figura 49: Visualización (izquierda) e imagen renderizada (derecha) de las escena *bedroom.glb*

Tanto los tiempos medidos como la forma de medirlos coinciden con la metodología detallada en la sección 7.2.4, las únicas diferencias serían algunas columnas adicionales en las tablas. Estas serían:

- Una nueva que indica el tiempo precisado para la generación de la escena *glTF* en el *servidor de renderizado* cuando se utiliza la optimización. Esta solo tiene sentido en la

Tabla con los tiempos obtenidos utilizando la optimización (Tabla 7), por lo que solo se incluye en esta.

- Otra adicional que detalla el tiempo de espera total percibido por el usuario, el cual se corresponde con la suma del tiempo de trasferencia de ficheros de la máquina de este al *microservicio de gestión de peticiones* y el tiempo total requerido por el sistema para satisfacer la petición. En este caso se incluye en ambas Tablas (Tabla 7 y Tabla 8).

A continuación, se detallan los tiempos medidos. La Tabla 7 se corresponde con los tiempos obtenidos prescindiendo de la optimización, mientras que la Tabla 8 se corresponde con los valores resultantes de hacer uso de esta.

	Tiempos prescindiendo de la optimización Fichero manejado: bedroom.glb (24,66 MiB)				
	Tiempo en completar transferencia del cliente al microservicio de gestión de peticiones	**Tiempo en completar transferencia del microservicio al servidor de renderizado**	**Tiempo de renderizado en Blender**	**Tiempo total requerido por el sistema**	**Tiempo de espera del usuario**
1	24s	22s	50s	1min 17s	1min 41s
2	24s	23s	51s	1min 19s	1min 43s
3	24s	22s	51s	1min 18s	1min 42s
4	24s	23s	54s	1min 22s	1min 46s
5	24s	24s	50s	1min 21s	1min 45s
Media	**24s**	**23s**	**51s**	**1min 21s**	**1min 43s**

Cuadro 7: Tiempos obtenidos prescindiendo de la optimización. El fichero participante en las transferencias se corresponde con *bedroom.glb*, con un tamaño de 24,66 *MiB*.

	Tiempos haciendo uso de la optimización Fichero manejado: scene.txt (30,64 KiB)					
	Tiempo en completar transferencia del cliente al microservicio de gestión de peticiones	**Tiempo en completar transferencia del microservicio al servidor de renderizado**	**Tiempo requerido en la generación de la escena glTF**	**Tiempo de renderizado en Blender**	**Tiempo total requerido por el sistema**	**Tiempo de espera total del usuario**
1	251ms	203ms	13s	54s	1min 14s	1min 14s
2	181ms	201ms	8s	57s	1min 9s	1min 9s
3	181ms	183ms	11s	52s	1min 7s	1min 7s
4	201ms	214ms	9s	49s	1min 2s	1min 2s
5	190ms	207ms	8s	50s	1min 2s	1min 2s
Media	**201ms**	**202ms**	**10s**	**52s**	**1min 7s**	**1min 7s**

Cuadro 8: Tiempos obtenidos haciendo uso de la optimización. El fichero participante en las transferencias se corresponde con *scene.txt*, con un tamaño de 30,63 *KiB*.

Analizando ambas Tablas, para que mejoren los tiempos, resulta esencial que la suma de las dos columnas correspondientes a los tiempos de transferencia entre componentes y del proceso de generación del fichero *glTF* de la Tabla 8 sean menores que la suma de los dos tiempos de transferencia entre componentes de la Tabla 7, ya que el tiempo de procesamiento en *Blender* y el *overhead* del sistema no se verían alterados.

Observando los tiempos medidos, es posible apreciar que esta condición se cumple:

- Las transferencias entre componentes pasan de requerir entre 22 y 24 segundos cada una (entre 46 y 48 segundos en total), a realizarse de manera casi instantánea en pocos

milisegundos. Esto se debe a que el fichero transferido experimenta una reducción de tamaño de 24,66 *MiB* a 30,64 *KiB*.

- Por su parte, el tiempo adicional requerido para la obtención de recursos y la generación del fichero *.gltf* en el servidor de renderizado al recurrir a la optimización ha tardado entre 8 y 13 segundos.
- Si se compara esta cantidad de tiempo adicional con el ahorro que suponen las nuevas transferencias entre componentes, la primera es mucho menor, por lo que se podría afirmar que, con esta escena y en este entorno de despliegue concretos, los tiempos de espera del sistema se ven reducidos de forma significativa al utilizar la optimización

Todo esto se puede apreciar de forma más clara comparando los tiempos indicados por la última columna de cada Tabla, la cual detalla el tiempo total de espera desde el punto de vista del usuario de la aplicación. La comparación de estos valores arroja un intervalo de entre 27 segundos y 44 segundos de mejora.

Estos resultados brindan un incentivo claro para continuar explorando esta técnica de optimización.

8

Conclusiones y Líneas Futuras

En este capítulo, se exponen las conclusiones alcanzadas una vez realizado este trabajo. Para lograr este objetivo, se lleva a cabo un análisis retrospectivo de los objetivos establecidos, el proceso de desarrollo y los resultados obtenidos.

Asimismo, se presentan una serie de propuestas e ideas que podrían servir como punto de partida para futuras líneas de trabajo relacionadas.

8.1. Conclusiones

Volviendo a los objetivos planteados inicialmente, se puede afirmar que estos se han cumplido. Si se analiza el resultado obtenido, una vez finalizado el proceso de desarrollo, contamos con un sistema ampliado a partir de su versión original, que conserva todas sus funcionalidades previas mientras se le agregan capacidades administrativas y se le dota de escalabilidad y modularidad, además de diversas funcionalidades de menor tamaño y mayor concreción.

Atendiendo al transcurso del desarrollo, se ha llegado a las siguientes conclusiones:

- La elección de tecnologías ha sido acertada. El uso de estas ha permitido una implementación sin limitaciones significativas y el acceso a módulos existentes y recursos elaborados por la comunidad ha agilizado enormemente el desarrollo.
- Contar con el desarrollo original como referencia ha supuesto un ahorro de tiempo considerable, ya que algunos aspectos como podrían ser las conversiones de tipos de los datos de los distintos ficheros involucrados y sus transferencias habrían requerido una cantidad de tiempo mucho más elevada.
- La adopción de un enfoque de desarrollo iterativo ha resultado apropiada. Como se había previsto, ha sido necesaria la incorporación de nuevas funcionalidades, ajustes y cambios, los cuales han podido ser introducidos de manera fluida y sin mayores dificultades.

- La adopción de una arquitectura basada en microservicios ha demostrado ser enormemente beneficiosa. Esta elección ha permitido cumplir con los objetivos establecidos de manera altamente satisfactoria, ya que esta paradigma se adapta muy bien a las características deseadas de modularidad y escalabilidad para el sistema.

- Sin embargo, un área importante de mejora se encuentra en la parte relacionada con la realización de pruebas. Estas se han convertido en un obstáculo para la implementación de los cambios mencionados. Aunque se han utilizado herramientas útiles que han facilitado bastante el trabajo, la naturaleza del sistema y el tiempo de espera requerido para algunas transferencias de ficheros y para el proceso de renderizado en el servidor han supuesto un desafío.

  Aunque estas pruebas son necesarias, no es conveniente probar el sistema al completo (pruebas *end to end*) cada vez que se modifica algo. Una posible solución habría podido pasar por el planteamiento de pruebas unitarias y de integración mediante el uso de *mocking* [42], lo cual podría haber evitado los tiempos de espera y habría permitido probar los componentes de forma aislada. Aunque habría requerido un tiempo considerable aprender cómo funcionan las pruebas al trabajar las tecnologías utilizadas, plantear las pruebas e implementarlas, hubiese sido óptimo de cara al futuro.

  La situación expuesta resalta de manera clara lo esencial que resulta en el ámbito del desarrollo de software la necesidad de dedicar tiempo y recursos al diseño de una estrategia de pruebas efectiva y a la mejora de la automatización con el objetivo de lograr un proceso de desarrollo más eficiente y un software resultante de mayor calidad.

8.2. Líneas Futuras

Aunque el desarrollo realizado ha resultado satisfactorio y se han cumplido los objetivos planteados, el sistema resultante sigue pudiendo mejorarse de diversas maneras:

- En primer lugar, una posible mejora del sistema estaría relacionada con la detección de errores. Actualmente, el sistema no es capaz de identificar fallos en los servidores de renderizado a menos que estos sean reportados o se detecte una interrupción en la comunicación. Una posible solución podría pasar por implementar estrategias para mejorar este aspecto. Por ejemplo, podría incorporarse un algoritmo que permitiera al sistema monitorizar de forma periódica el estado de los servidores de renderizado. Esto le permitiría detectar anomalías y tomar medidas apropiadas de manera automática. Esta mejora se traduciría en un sistema más inteligente capaz de anticipar y prevenir

problemas, aliviando al administrador de la necesidad de intervenir manualmente en algunos casos.

- Por otro lado, dado el hecho de que se partiría de componentes que permiten ser desplegados en *Docker*, podría ser una oportunidad valiosa explorar la posibilidad de integrar la aplicación con una herramienta de orquestación de contenedores como podría ser *Kubernetes* [19]. Estas herramientas son ampliamente utilizadas en el ámbito empresarial y podrían proporcionar mejoras significativas en la administración y escalabilidad de la aplicación. Podría merecer la pena realizar un estudio de las ventajas que supondría la utilización de estas frente a los inconvenientes y a la complejidad adicional que pueda suponer la configuración y el despliegue necesarios.

Atendiendo a funcionalidades adicionales, se podrían añadir algunas relacionadas con distintos aspectos de la aplicación:

- En el visor 3D, además de la inclusión de fuentes de luz, se podría intentar ampliar las opciones de edición disponibles, lo cual dotaría a la aplicación de una versatilidad aún mayor, permitiendo generar una escena completa desde cero o a base de modificar una existente cargada previamente.

  Esta extensión requeriría trabajar en la parte del sistema que utiliza *Three.js*. Esta biblioteca cuenta con una amplia gama de funciones que facilitarían el desarrollo las nuevas funcionalidades de edición.

- Aunque son los formatos más adecuados para las trasferencias, se podría explorar la posibilidad de ampliar el sistema para que fuera compatible con formatos de fichero adicionales a *glTF* y *GLB*.

  Se podría distinguir entre la opción de realizar la conversión de otros formatos a uno de estos, o manejarlos inalterados en el sistema, ya que *Blender* y *Three.js* permiten importar las escenas a partir de otros formatos como podrían ser *.obj*, *.3ds*, *.fbx*...

- Por último, durante el proceso de desarrollo, se ha optado por utilizar solicitudes y respuestas *HTTP* para gestionar las comunicaciones entre los diferentes componentes. Una idea que podría resultar valiosa e interesante consistiría en explorar la viabilidad de rediseñar este aspecto del sistema para incorporar en estas la tecnología *WebSockets* [20].

  Esta consiste en un protocolo relativamente reciente que permite establecer una conexión bidireccional persistente entre dos partes, el cual fue ideado con el propósito de superar las restricciones de *HTTP* en lo que respecta a la necesidad de efectuar múltiples peticiones para acceder a actualizaciones en tiempo real (esta limitación se ha traducido

en la dependencia de mecanismos de *polling* a lo largo de la implementación). Utilizando *WebSockets*, una vez que se establece la conexión, tanto el cliente como el servidor tienen la capacidad de intercambiar mensajes en cualquier instante sin la necesidad de una solicitud explícita.

Estas ventajas se adaptan muy bien a la naturaleza y características de tiempo real del sistema. Además, las tecnologías utilizadas hasta este punto seguirían siendo compatibles con esta posible modificación, existiendo diversos módulos *npm* que permiten trabajar con este protocolo en entornos *Node.js*. Entre estos destaca *socket.io* [60], el cual puede ser utilizado en conjunto con *Express* sin problemas.

Referencias

[1] The Draco Authors. *Draco 3D Graphics Compression*. 2023. URL: https://google.github.io/draco/.

[2] R. Awati. *What are Lossless and Lossy Compression?* 2021. URL: https://www.techtarget.com/whatis/definition/lossless-and-lossy-compression.

[3] J. Ballesteros. *Interacción persona-ordenador. Ingeniería IU + Prototipado*. Transparencias de la asignatura Interfaces de Usuario del Grado en Ingeniería del Software de la Universidad de Málaga durante el curso académico 2021/2022. 2022.

[4] R. Cabello et al. *three.js manual*. s.f. URL: https://threejs.org/manual/.

[5] B. Caulfield. *¿Qué es el Path Tracing? | Blog de NVIDIA*. 10-05-2022. URL: https://la.blogs.nvidia.com/2022/05/10/que-es-el-path-tracing/.

[6] Bootstrap Community. *Introduction · Bootstrap v4.6*. 2021. URL: https://getbootstrap.com/docs/4.6.

[7] Linux Community. *pkill(1) - Linux man page*. s.f. URL: https://linux.die.net/man/1/pkill.

[8] SheepIt Render Farm. *SheepIt Render Farm*. 2023. URL: https://www.sheepit-renderfarm.com/home.

[9] Apache Software Foundation. *Apache JMeter - Apache JMeter™*. 2023. URL: https://jmeter.apache.org/.

[10] Blender Foundation. *About — blender.org*. 2023. URL: https://www.blender.org/about/.

[11] Blender Foundation. *Blender 3.6 Python API Documentation — Blender Python API*. 27-06-2023. URL: https://docs.blender.org/api/current/index.html.

[12] Blender Foundation. *Cycles — Blender Manual*. 7-01-2023. URL: https://docs.blender.org/manual/en/latest/render/cycles/index.html.

[13] Blender Foundation. *Eevee — Blender Manual*. 7/01/2023. URL: https://docs.blender.org/manual/en/latest/render/eevee/index.html.

[14] Mozilla Foundation. *CSS | MDN*. 2023. URL: https://developer.mozilla.org/es/docs/Web/CSS.

[15] Mozilla Foundation. *HTML: Lenguaje de etiquetas de hipertexto | MDN*. 2023. URL: https://developer.mozilla.org/es/docs/Web/HTML.

[16] Mozilla Foundation. *Intercambio de recursos de origen cruzado (CORS) - HTTP | MDN*. 2023. URL: https://developer.mozilla.org/es/docs/Web/HTTP/CORS.

[17] Mozilla Foundation. *JavaScript | MDN*. 2023. URL: https://developer.mozilla.org/es/docs/Web/JavaScript.

[18] OpenJS Foundation y Colaboradores de NodeJS. *About | Node.js*. s.f. URL: https://nodejs.org/en/about.

[19] The Linux Foundation. *¿Qué es Kubernetes? | Kubernetes*. 2023. URL: https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/.

[20] Mozilla Foundtion. *WebSockets - Referencia de la API Web | MDN*. 2023. URL: https://developer.mozilla.org/es/docs/Web/API/WebSockets_API.

[21] T. Goode. *cors - npm*. 2018. URL: https://www.npmjs.com/package/cors.

[22] LS Group. *Real-time and offline 3D rendering: what are the differences?* 20-03-2021. URL: https://www.xrsuite.fr/post/real-time-and-offline-3d-rendering-what-are-the-differences.

[23] M. Hernández. *¿Qué es Linting y ESLint? ¿Cómo empezar?* 26-01-2021. URL: https://www.freecodecamp.org/espanol/news/que-es-linting-y-eslint/.

[24] S. Hildebrandt. *systeminformation*. 2023. URL: https://systeminformation.io/.

[25] Amazon Web Services Inc. *Render Farm Manager – AWS Thinkbox Deadline – Amazon Web Services*. 2023. URL: https://aws.amazon.com/thinkbox-deadline/.

[26] Docker Inc. *Bridge network driver | Docker Documentation*. 2023. URL: https://docs.docker.com/network/drivers/bridge/.

[27] Docker Inc. *Compose file version 3 reference | Docker Documentation*. 2023. URL: https://docs.docker.com/compose/compose-file/compose-file-v3/.

[28] Docker Inc. *Dockerfile reference | Docker Documentation*. 2023. URL: https://docs.docker.com/engine/reference/builder/.

[29] F5 Inc. *What Is NGINX? - NGINX*. 2023. URL: https://www.nginx.com/resources/glossary/nginx/.

[30] GitHub Inc. *About*. 2023. URL: https://github.com/about.

[31] GitHub Inc. *GitHub Desktop | Simple collaboration from your desktop*. 2023. URL: https://desktop.github.com/.

[32] Google Inc. *Puppeteer | Puppeteer*. 2023. URL: https://pptr.dev/.

[33] MongoDB Inc. *MongoDB Documentation*. 2023. URL: https://www.mongodb.com/docs/.

[34] MongoDB Inc. *What is MongoDB Atlas? — MongoDB Atlas*. 2023. URL: https://www.mongodb.com/docs/atlas/.

[35] ngrok Inc. *Overview | ngrok documentation*. 2023. URL: https://ngrok.com/docs/.

[36] Postman Inc. *What is Postman? Postman API Platform*. 2023. URL: https://www.postman.com/product/what-is-postman/.

[37] StrongLoop Inc. y Colaboradores de expressjs.com. *Express 4.x - Referencia de API*. s.f. URL: https://expressjs.com/es/4x/api.html.

[38] The Khronos Group Inc. *About The Khronos Group*. 2023. URL: https://www.khronos.org/about/.

[39] The Khronos Group Inc. *gltf-validator - npm*. 2022. URL: https://www.npmjs.com/package/gltf-validator.

[40] The Khronos Group Inc. *KhronosGroup/glTF: glTF – Runtime 3D Asset Delivery*. 2023. URL: https://github.com/KhronosGroup/glTF.

[41] The Khronos Group Inc. *WebGL Overview - The Khronos Group Inc*. 2023. URL: https://www.khronos.org/webgl/.

[42] J. Jung. *How to test software: mocking, stubbing, and contract testing*. 2019. URL: https://circleci.com/blog/how-to-test-software-part-i-mocking-stubbing-and-contract-testing/.

[43] V. Karpov. *Mongoose v7.3.1: Mongoose*. 2023. URL: https://mongoosejs.com/docs/api/mongoose.html.

[44] Balsamiq Studios LLC. *Balsamiq Wireframes - Industry Standard Low-Fidelity Wireframing Software | Balsamiq*. 2023. URL: https://balsamiq.com/wireframes/.

[45] Canonical Ltd. *Ubuntu PC operating system | Ubuntu*. 2023. URL: https://ubuntu.com/desktop.

[46] Microsoft. *Azure Remote Rendering | Microsoft Azure*. 2023. URL: https://azure.microsoft.com/es-es/products/remote-rendering.

[47] Microsoft. *taskkill*. 12/04/2023. URL: https://learn.microsoft.com/es-es/windows-server/administration/windows-commands/taskkill.

[48] Microsoft. *Visual Studio Code - Code Editing. Redefined*. 2023. URL: https://code.visualstudio.com/.

[49] S. Newman. *Building Microservices, 2nd Edition*. O'Reilly Media, 2021. ISBN: 9781492034021.

[50] C. O'Hara y Contribuidores de validator. *validator - npm*. 2023. URL: https://www.npmjs.com/package/validator.

[51] A. Reinman. *Nodemailer :: Nodemailer*. s.f. URL: https://nodemailer.com/about/.

[52] Fox Renderfarm. *Cloud Render Farm Online | Fox Render Farm*. 2023. URL: https://www.foxrenderfarm.com/.

[53] Chacon S., Long J. y Git Community. *About - Git*. 2023. URL: https://git-scm.com/about.

[54] IONOS Cloud S.L.U. *Grid Computing: cómo funcionan los superordenadores virtuales - IONOS*. 26-01-2022. URL: https://www.ionos.es/digitalguide/servidores/know-how/grid-computing/.

[55] J. M. Sánchez Fernández y R. M. Luque Baena. "Desarrollo de una aplicación web para la generación de imágenes fotorrealistas a partir de escenas modeladas en 3D". Tesis de mtría. Universidad de Málaga, 2021. URL: https://riuma.uma.es/xmlui/bitstream/handle/10630/23389/S%C3%A1nchez%20Fern%C3%A1ndez%20Jos%C3%A9%20Mar%C3%ADa%20Memoria.pdf?sequence=1&isAllowed=y.

[56] K. Schwaber y J. Sutherland. *The Scrum Guide*. Nov. de 2020. URL: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf.

[57] Scratchapixel. *Introduction to Raytracing: A Simple Method for Creating 3D Images*. s.f. URL: https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/how-does-it-work.html.

[58] Scratchapixel. *Rasterization: a Practical Implementation*. s.f. URL: https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/overview-rasterization-algorithm.html.

[59] RebusFarm Render Service. *RebusFarm: Granja de render en la nube en línea*. 2023. URL: https://rebusfarm.net/es/.

[60] Socket.IO. *Introduction | Socket.IO*. 2023. URL: https://socket.io/docs/v4/.

[61] SmartBear Software. *API Editor - Download or Try it in the Cloud*. 2023. URL: https://swagger.io/tools/swagger-editor/.

[62] D. Stenberg y curl Community. *curl*. 2023. URL: https://curl.se/.

[63] E. Vaati. *Subir archivos con Multer en Node.js y Express*. 25-05-2022. URL: https://code.tutsplus.com/es/file-upload-with-multer-in-node--cms-32088t.

[64] J. de Vries. *LearnOpenGL - OpenGL*. s.f. URL: https://learnopengl.com/Getting-started/OpenGL.

[65] Wikipedia. *Pay-as-you-use - Wikipedia*. 15-01-2023. URL: https://en.wikipedia.org/wiki/Pay-as-you-use.

Apéndice A
Manual de Instalación

Este documento pretende guiar al usuario durante el proceso de instalación de todos los componentes de la aplicación *PhotoSpaces*. En las diferentes secciones, se muestran los pasos a seguir para lograr disponer de un despliegue del sistema funcional.

Las distintas acciones necesarias se realizan principalmente sobre un equipo que cuenta con *Windows 10*, pero la aplicación es compatible con otros sistemas operativos.

Los aspectos que difieren lo suficiente de un sistema operativo a otro como para requerir una explicación adicional en sistemas operativos *Linux* se detallan a lo largo del manual, utilizando en este caso una distribución *Ubuntu 22.04 Jammy Jellyfish*.

A.1. Componentes de la aplicación

Es importante tener claro desde el principio que *PhotoSpaces* es una aplicación modular que admite distintas configuraciones según dónde se desee desplegar cada componente.

A continuación, se detallan qué responsabilidades tiene cada uno y qué configuraciones de las que se explican más adelante será necesario realizar si se pretende realizar un despliegue directamente sobre el *host*.

- ***Cliente estándar*:** Sirve a los usuarios el cliente que permite la carga y edición de las escenas 3D y la generación y envío de las peticiones de renderizado.

  Configuraciones necesarias: *Node.js*

- ***Cliente CLI*:** Aplicación de consola que actúa como cliente y que ofrece una interfaz de línea de comandos. Permite el envío de peticiones de renderizado al sistema, pero no ofrece las funcionalidades de visualización y edición a partir de la carga de ficheros del *cliente estándar*.

  Configuraciones necesarias: *Node.js*

- ***Cliente de administración*:** Sirve a los administradores del sistema el panel de administración de la aplicación.

Configuraciones necesarias: *Node.js*

- ***Microservicio de administración*:** Se encarga de manejar las peticiones generadas desde el *cliente de administración*.

  Configuraciones necesarias: *Node.js*, base de datos *MongoDB*

- ***Microservicio de gestión de peticiones*:** Se encarga de manejar las peticiones generadas desde el *cliente estándar*.

  Configuraciones necesarias: *Node.js*, base de datos *MongoDB*, dirección de correo electrónico que actúe como remitente

- ***Servidor de renderizado*:** Sus responsabilidades se limitan a llevar a cabo el proceso de renderizado y responder a las peticiones que puede recibir de los microservicios.

  Configuraciones necesarias: *Node.js*, *Blender*

A.2. *Node.js*

Será necesario instalar *Node.js* en el sistema, ya que todos los componentes de la aplicación se ejecutan sobre este entorno.

A.2.1. *Windows*

Para la descarga en *Windows*, será necesario acceder a la página de descargas de *Node.js* (Figura A.1) y seleccionar la versión correspondiente al sistema. Se recomienda la versión *LTS* más reciente.

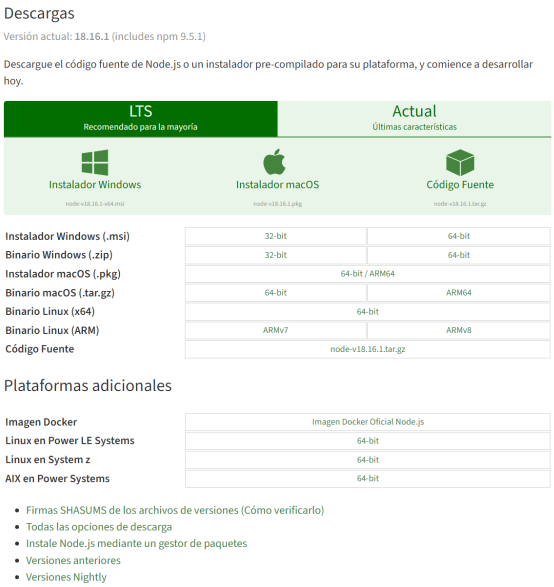

Figura A.1: Página de descargas de *Node.js*

En este caso, se descargará el instalador de *Windows*. Cuando se ejecute el fichero descargado, un asistente de instalación actuará de guía a lo largo de este proceso.

Tras hacer *click* en el botón *Next* en el mensaje de bienvenida del instalador, en primer lugar, será necesario aceptar los términos y condiciones de la licencia para poder continuar, se marcará la casilla correspondiente y se pulsará el botón *Next* (Figura A.2).

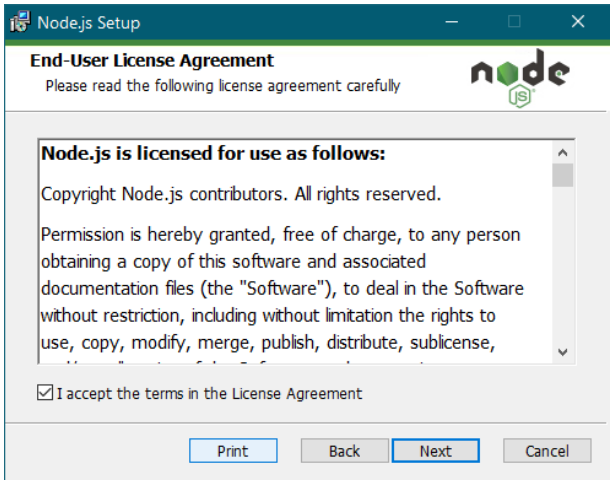

Figura A.2: Licencia en el instalador de *Node.js*

A continuación, se solicitará una ruta para la instalación, indicar el directorio donde se desee instalar el software y hacer *click* en el botón *Next* (Figura A.3).

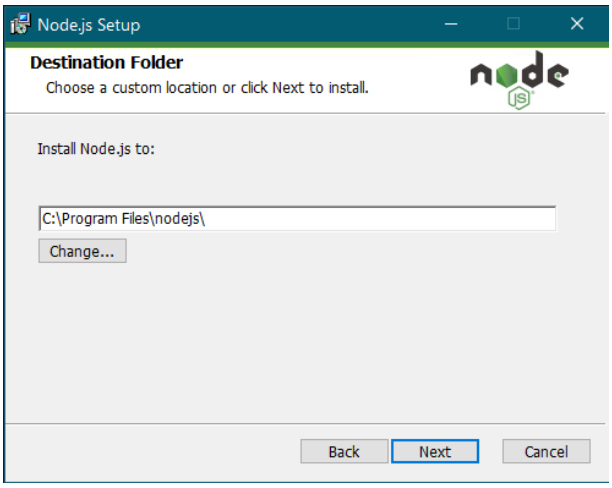

Figura A.3: Ruta de instalación en el instalador de *Node.js*

Tras esto, se requerirá especificar qué componentes instalar, aquí no se modificará nada para que se instale todo, ya que así el asistente se encargará de instalar también *npm* y de modificar la variable *Path* del sistema. Ambas acciones son requeridas para tener el sistema funcionando, por lo que se evitará tener que realizarlas manualmente más adelante (Figura A.4).

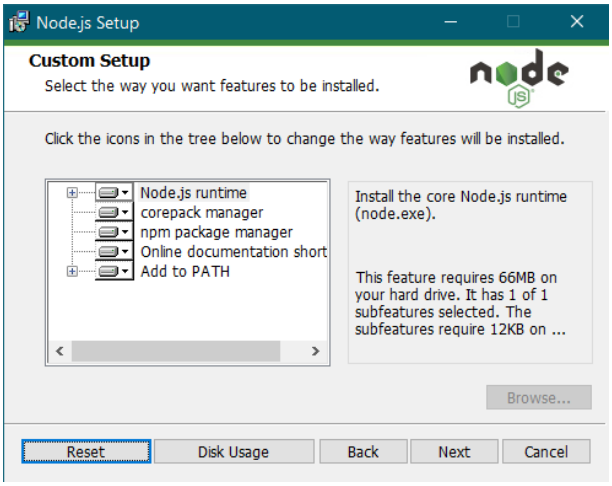

Figura A.4: Selección de componentes en el instalador de *Node.js*

Por último habrá que indicar si se desea instalar también las herramientas necesarias para compilar módulos nativos, se indicará que sí marcando la casilla correspondiente y se hará *click* el botón *Next* (Figura A.5).

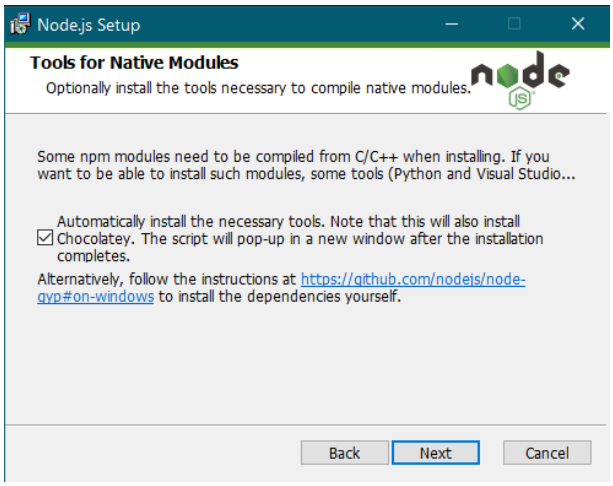

Figura A.5: Instalación de herramientas para la compilación de módulos nativos en el instalador de *Node.js*

El asistente comenzará con el proceso de instalación y notificará una vez haya finalizado. Se pulsará el botón *Finish*. Tras esto, ya se debería disponer de *Node.js* en el equipo (Figura A.6).

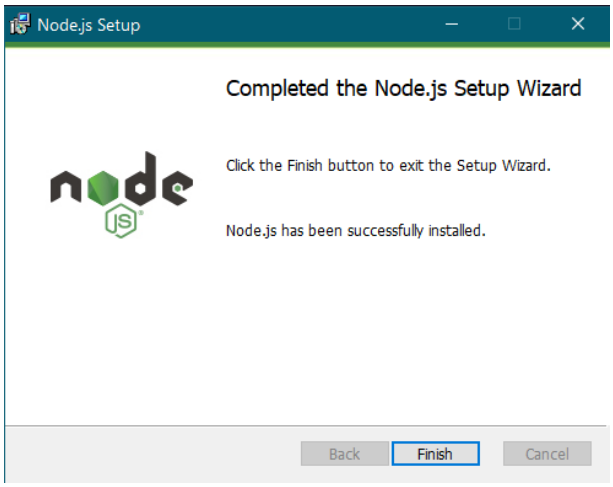

Figura A.6: Proceso de instalación de *Node.js* finalizado

A modo de verificación, es posible abrir cualquier terminal e introducir los comandos `node -v` y `npm -v` para comprobar que tanto *Node.js* como *npm* se han instalado correctamente y se encuentran en la variable *Path* del sistema (Figura A.7).

```
PS C:\Users\Pablo\Desktop> node -v
v18.16.1
PS C:\Users\Pablo\Desktop> npm -v
9.7.2
```

Figura A.7: Verificación de instalación correcta de *Node.js* y *npm*

A.2.2. *Linux*

Para instalar *Node.js* en *Linux* se recurrirá a *nvm*, una herramienta que permite descargar, instalar y manejar múltiples versiones de este software de manera sencilla y versátil.

Para instalar *nvm*, se ejecutará el comando `curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.4/install.sh | bash`, el cual descargará y ejecutará un *script* que se encargará de clonar el repositorio de *nvm* y realizar las configuraciones necesarias (Figura A.8).

```
pablo@DESKTOP-P7Q0KC0:~$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.4/install.sh | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 15916  100 15916    0     0  56439      0 --:--:-- --:--:-- --:--:-- 56240
=> Downloading nvm from git to '/home/pablo/.nvm'
=> Cloning into '/home/pablo/.nvm'...
remote: Enumerating objects: 360, done.
remote: Counting objects: 100% (360/360), done.
remote: Compressing objects: 100% (306/306), done.
remote: Total 360 (delta 40), reused 168 (delta 28), pack-reused 0
Receiving objects: 100% (360/360), 220.04 KiB | 869.00 KiB/s, done.
Resolving deltas: 100% (40/40), done.
* (HEAD detached at FETCH_HEAD)
  master
=> Compressing and cleaning up git repository

=> Appending nvm source string to /home/pablo/.bashrc
=> Appending bash_completion source string to /home/pablo/.bashrc
=> You currently have modules installed globally with `npm`. These will no
=> longer be linked to the active version of Node when you install a new node
=> with `nvm`; and they may (depending on how you construct your `$PATH`)
=> override the binaries of modules installed with `nvm`:
```

Figura A.8: Descarga y ejecución del *script* de instalación de *nvm*

Una vez finalizada la ejecución del *script*, será posible comprobar que la instalación ha ido bien ejecutando el comando `command -v nvm` y comprobando que se obtiene la salida *"nvm"* (Figura A.9). Es importante tener en cuenta que antes de realizar la comprobación será necesario reiniciar el *shell* para que sean cargadas las modificaciones realizadas por el *script*.

```
pablo@DESKTOP-P7Q0KC0:~$ command -v nvm
nvm
```

Figura A.9: Comprobación de instalación correcta de *nvm*

Una vez instalado *nvm* correctamente, se utilizará para instalar la versión de *Node.js* que se requiera, esto se hará con el comando `nvm install <versión de Node.js>`. En este caso, se utilizará la versión *LTS* más reciente en el momento de elaboración de esta guía, (*v18.16.1*), por lo que se introducirá el comando `nvm install 18.16.1`. La versión correspondiente de *Node.js* comenzará a descargarse e instalarse (Figura A.10).

```
pablo@DESKTOP-P7Q0KC0:~$ nvm install 18.16.1
Downloading and installing node v18.16.1...
Downloading https://nodejs.org/dist/v18.16.1/node-v18.16.1-linux-x64.
tar.xz...
######################################################################
######################################################################## 100.0
%Computing checksum with sha256sum
Checksums matched!
Now using node v18.16.1 (npm v9.5.1)
Creating default alias: default -> 18.16.1 (-> v18.16.1)
```

Figura A.10: Descarga e instalación de una versión de *Node.js*

Una vez finalizado el proceso, es posible comprobar que todo ha ido bien ejecutando los comandos `node -v` y `npm -v` (Figura A.11).

```
pablo@DESKTOP-P7Q0KC0:~$ node -v
v18.16.1
pablo@DESKTOP-P7Q0KC0:~$ npm -v
9.5.1
```

Figura A.11: Comprobación de instalación correcta de *Node.js* y *npm*

A.3. *Blender*

Solo será necesario instalar *Blender* en los equipos que se vayan a utilizar como servidores de renderizado.

A.3.1. *Windows*

Para instalar *Blender* en *Windows*, será necesario acceder a la página de descargas de *Blender* (Figura A.12) y descargar la versión estable más reciente.

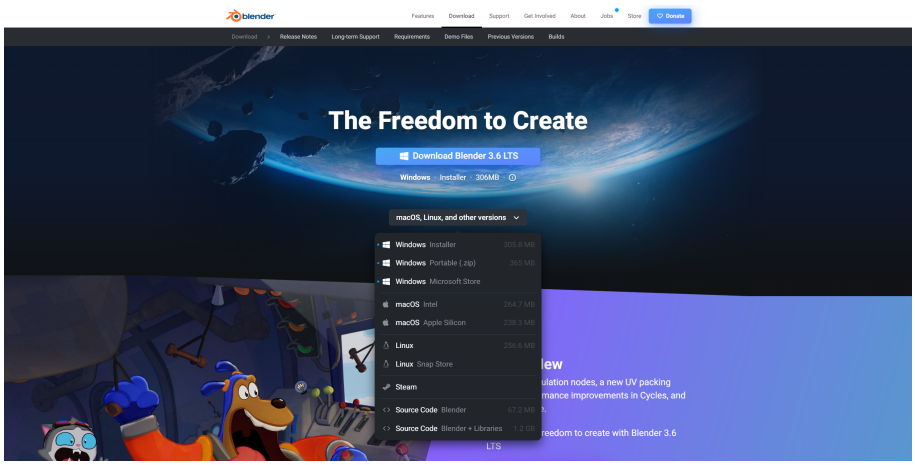

Figura A.12: Página de descargas de *Blender*

En este caso, se descargará el instalador para *Windows*. Al ejecutarlo, se iniciará el asistente de instalación, el cual es muy similar al de *Node.js*, pero incluso más sencillo. De nuevo, tras el mensaje de bienvenida, se hará *click* en *Next* y se aceptarán los términos y condiciones de la licencia marcando la casilla correspondiente y volviendo a hacer *click* en *Next* (Figura A.13).

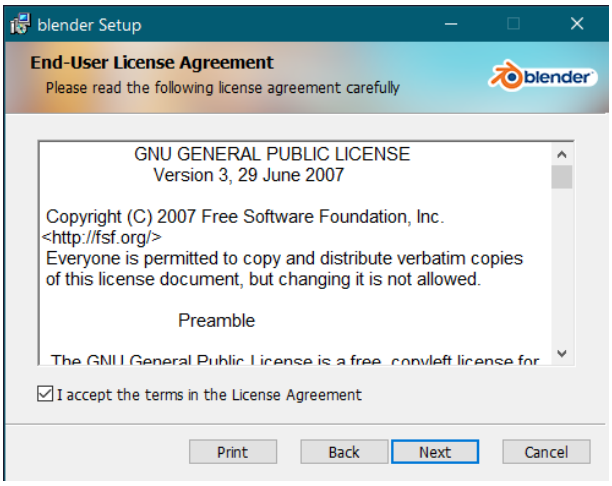

Figura A.13: Licencia en el instalador de *Blender*

A continuación, será necesario indicar los componentes que se desean instalar y la ruta de instalación. En la primera parte no se modificará nada, en la ruta de instalación será posible especificar otro directorio si se desea. Se hará *click* en *Next* (Figura A.14).

Aquí es posible apreciar que, a diferencia del instalador de *Node.js*, el de *Blender* no añade a la variable *Path* del sistema, por lo que será necesario hacerlo manualmente.

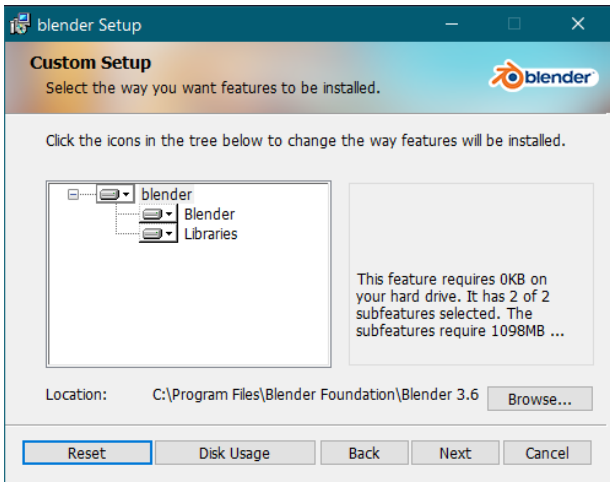

Figura A.14: Selección de componentes y ruta de instalación en el instalador de *Blender*

El instalador comenzará con el proceso de instalación y notificará una vez finalice (Figura A.15). *Blender* ya estaría instalado, pero faltaría añadir la ruta de instalación a la variable *Path* del sistema, lo cual se detalla más adelante en la sección A.4.

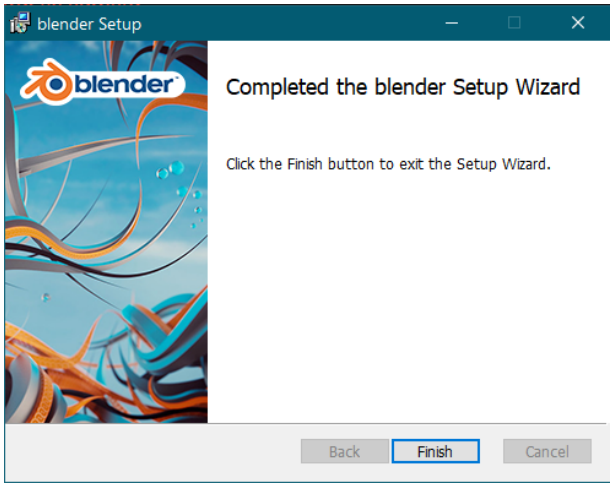

Figura A.15: Proceso de instalación de *Blender* finalizado

Una vez añadida, para comprobar que todo funciona correctamente es posible utilizar el comando `blender -v` (Figura A.16).

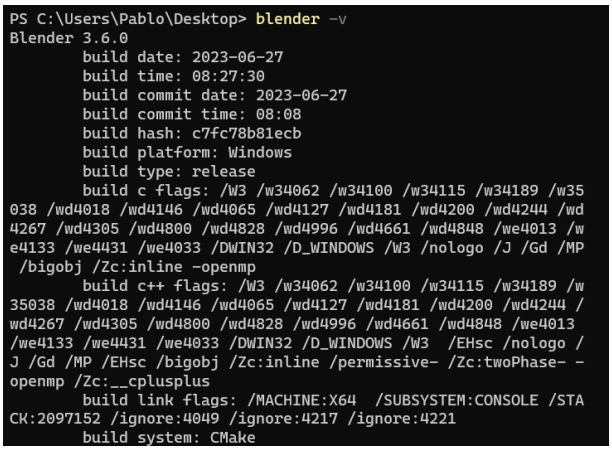

Figura A.16: Verificación de instalación correcta de *Blender*

A.3.2. *Linux*

La instalación en *Linux* es bastante distinta, así que se detalla a continuación.

En este caso, se realizará la instalación en el directorio `/home` (Figura A.17), junto al repositorio del proyecto, pero es posible realizarla en la ruta que se desee, tan solo será necesario modificar la ruta `/home` de los comandos con la del directorio que se vaya a utilizar.

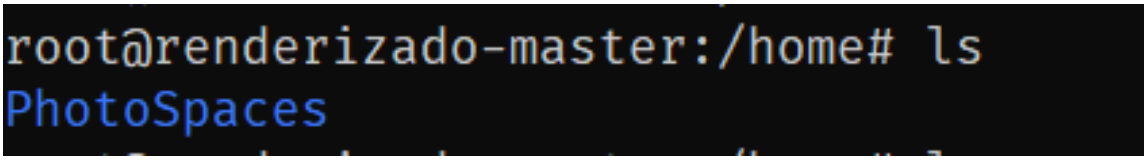

Figura A.17: Directorio `/home`

En primer lugar se instalarán las dependencias necesarias con el comando `apt-get install -y curl libxi6 libxrender1 xvfb xz-utils unzip python3-pip`

Tras esto, se descargará el fichero comprimido con *Blender* con el comando `curl https://www.dropbox.com/s/7ce0lcj1qjmh6pp/blender.zip?dl=1 -o /home/blender.zip -J -L`. Se podrá apreciar que ha aparecido un nuevo fichero *blender.zip* en el directorio especificado (Figura A.18).

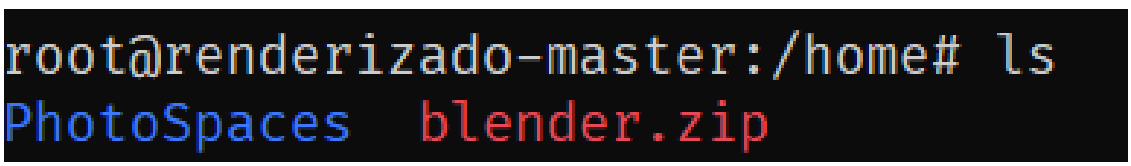

Figura A.18: Fichero *blender.zip* descargado

A continuación se extraerá el archivo *ZIP* con el comando `unzip /home/blender.zip -d /home`. Una vez realizada la extracción, será posible eliminar el fichero comprimido con el comando `rm /home/blender.zip` (Figura A.19).

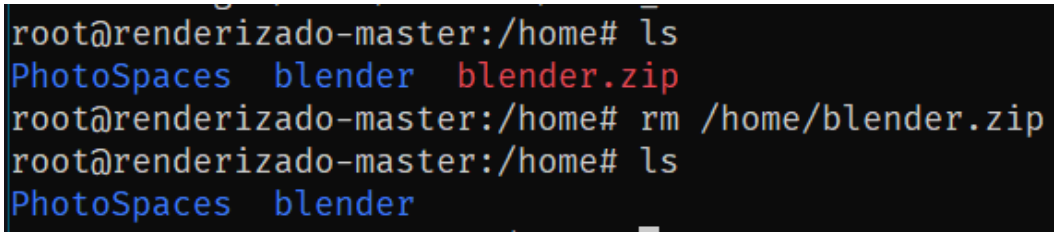

Figura A.19: Extracción y eliminación del fichero comprimido

Ahora será necesario proporcionar permisos de ejecución al ejecutable de *Blender*. Para ello se introducirá el comando `chmod +x /home/blender/blender` (Figura A.20).

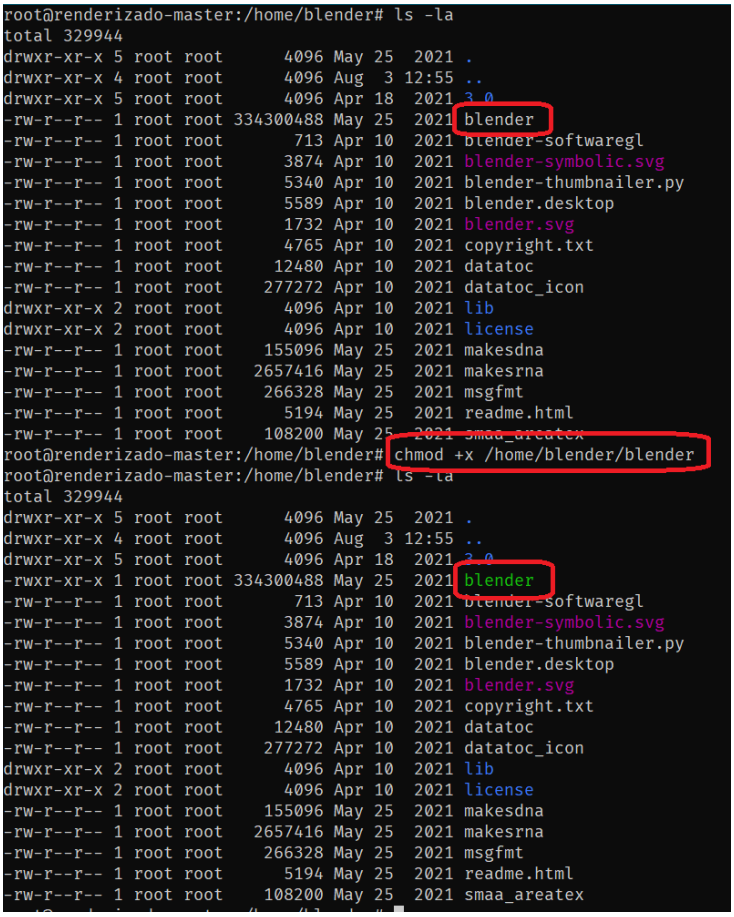

Figura A.20: Proporcionar permisos de ejecución al ejecutable de *Blender*

También será necesario instalar *pyvirtualdisplay* utilizando *PIP* para que funcione el *script Python*, para ello se utilizará el comando `pip3 install pyvirtualdisplay`.

Por último, solo quedaría añadir la ruta de instalación de *Blender* a la variable *Path* del sistema y añadir dos nuevas variables de entorno:

- ***PYTHONPATH:*** *Python* utiliza esta variable para buscar las librerías, se le asignará como valor la cadena correspondiente a la ruta del directorio `/usr/local/lib/python3.10/dist-packages`.

- ***BLENDER_MAJOR:*** Esta variable es utilizada por el *script Python* para saber si tiene que importar *pyvirtualdisplay*. Con que cuente con un valor definido será suficiente. En este caso, como se está utilizando la versión *3.0.0 Alpha* de *Blender*, se indicará el valor *3*.

La modificación de la variable *PATH* se detalla más adelante en la sección A.4. Para añadir las nuevas variable de entorno se hará lo mismo que se especifica en dicha sección, pero utilizando las siguiente líneas al final del archivo: `export BLENDER_MAJOR="3"` y `export PYTHONPATH="/usr/local/lib/python3.10/dist-packages"`.

Al igual que en *Windows*, será posible comprobar que se ha llevado a cabo la instalación y las configuraciones correctamente introduciendo el comando `blender -v` y verificando que funciona desde cualquier parte del sistema (Figura A.21). El comando `blender` provocará un error, lo cual es normal.

```
root@renderizado-master:/# blender -v
Blender 3.0.0 Alpha
```

Figura A.21: Verificación de instalación de *Blender* correcta

A.4. Añadir ruta a la variable *Path* del sistema

A.4.1. *Windows*

Para modificar esta variable, primero será necesario acceder a la sección *Editar las variables de entorno del sistema*. Para llegar a esta existen diferentes métodos, el más sencillo consiste en realizar una búsqueda en la barra de tareas de *Windows*. Simplemente introduciendo la cadena *"env"* como en la Figura A.22 debería ser suficiente, aunque también es posible buscarla por el nombre completo.

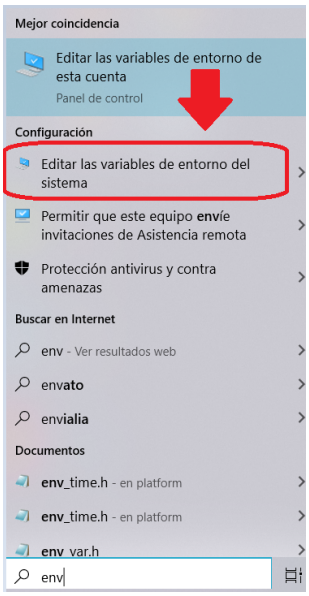

Figura A.22: Acceder a *Editar las variables de entorno del sistema* a través de la barra de tareas de *Windows*

Se abrirá una nueva ventana, donde se hará *click* en el botón *Variables de entorno...* (Figura A.23).

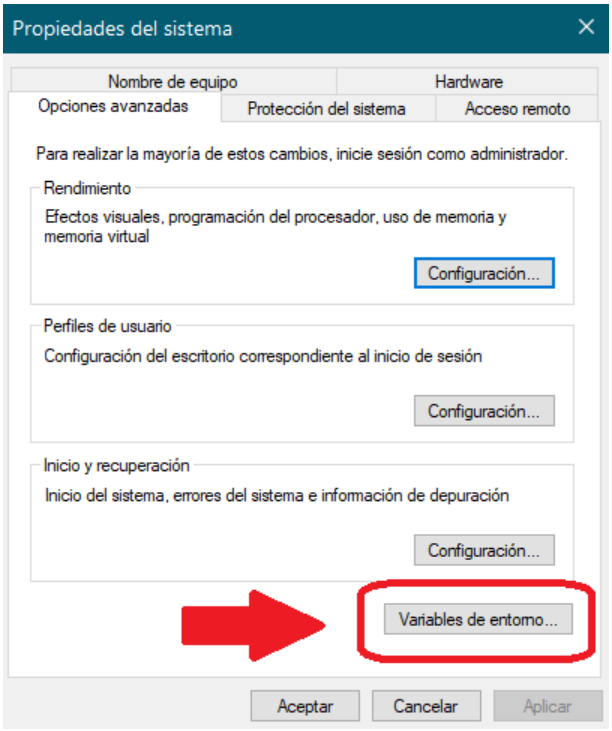

Figura A.23: Acceder a las variables de entorno del sistema

Se generará otra ventana que mostrará las distintas variables de entorno definidas. Se bus-

cará y seleccionará la que tiene el nombre *"Path"* y se hará *click* en el botón *Editar* (Figura A.24).

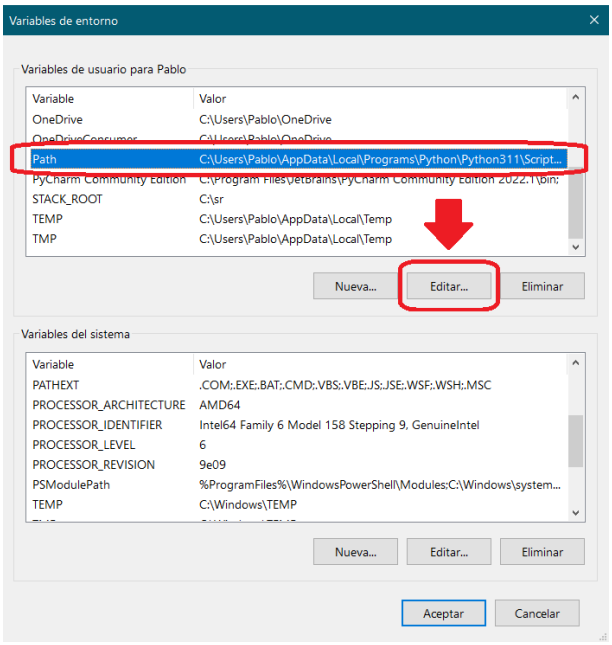

Figura A.24: Modificar la variable *Path*

Aparecerá una nueva ventana que mostrará todas las rutas incluidas en la variable *Path*. Se hará *click* en el botón *Nuevo*, se rellenará el campo de texto con la ruta que se desee añadir y se hará *click* en el botón *Aceptar* (Figura A.25).

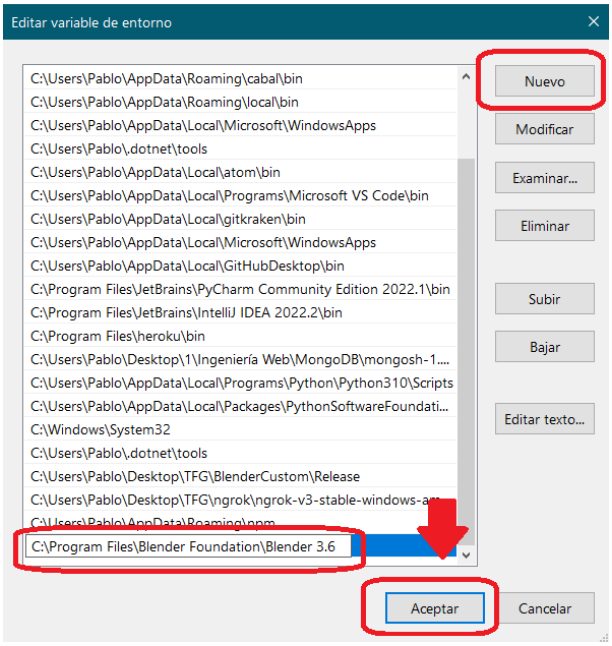

Figura A.25: Añadir nueva ruta a la variable *Path*

Se cerrará la ventana y se volverá a la anterior, donde será necesario hacer *click* en el botón *Aceptar*, ya que si se sale directamente no se guardarán los cambios realizados (Figura A.26).

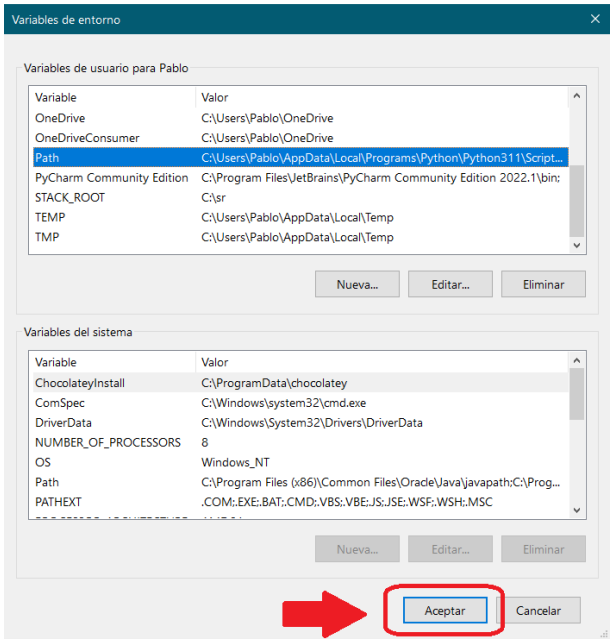

Figura A.26: Guardar los cambios realizados en las variables de entorno

Tras esto, el sistema será capaz de encontrar el ejecutable cuya ruta se ha añadido desde cualquier parte del sistema.

A.4.2. *Linux*

Para modificar la variable *PATH* en *Linux*, se modificará el fichero de configuración del *shell*. En este caso, como *Ubuntu* utiliza *bash*, se editará el fichero *~/.bashrc*. Para modificar los contenidos del archivo, se utiliza el editor de texto *Vim*, aunque cualquier otro serviría también. Se introducirá el comando `vim \textasciitilde/.bashrc`.

Una vez dentro del fichero, se buscará una línea que configure la variable de entorno *PATH* haciendo un *export*. Si se logra encontrar, se modificará el valor que se le da a la variable para que incluya también la ruta que se desea añadir, separándola con dos puntos. Si no se encuentra en el archivo, se añadirá al final del fichero la siguiente línea: `export PATH="$PATH:<nueva ruta que se quiera añadir>"` (Figura A.27).

```
export PATH="$PATH:/home/blender"
```

Figura A.27: Línea añadida al fichero *~/.bashrc* para añadir la ruta de instalación de *Blender*

Tras guardar las modificaciones realizadas en el fichero, será posible introducir el comando `echo $PATH` para consultar el valor de la variable (Figura A.28). Sin embargo, antes será necesario cerrar y abrir el *shell* para que sea cargada la nueva configuración. Si se hace antes, no se verán reflejados los cambios aún.

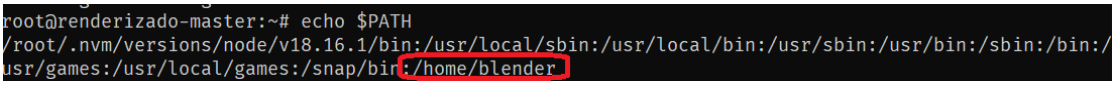

Figura A.28: Consultar nuevo valor de la variable de entorno *PATH*

A.5. Base de datos *MongoDB*

Para la base de datos, mientras se disponga de una instancia de *MongoDB* a la que conectar los microservicios de la aplicación (*microservicio de administración* y *microservicio de gestión de peticiones*), será suficiente.

A.5.1. Despliegue en *MongoDB Atlas*

A continuación se explica una de las posibles alternativas, consistente en el despliegue de la base de datos en un *cluster* en la nube manejado por el servicio *MongoDB Atlas*.

En primer lugar, será necesario acceder a la página web de *MongoDB Atlas* y rellenar el formulario de registro (el campo *Company* no es obligatorio). También es posible iniciar sesión con una cuenta de *Google* (Figura A.29).

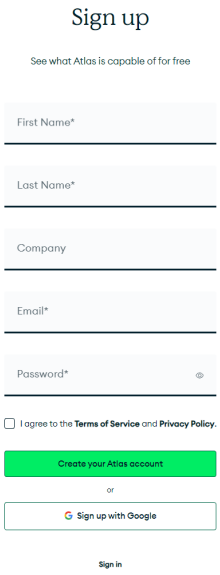

Figura A.29: Formulario de registro de *MongoDB Atlas*

Una vez solicitado el registro, será necesario verificar el correo electrónico especificado, esto se logrará haciendo *click* en el botón incluido en el correo recibido en la dirección indicada (Figuras A.30 y A.31).

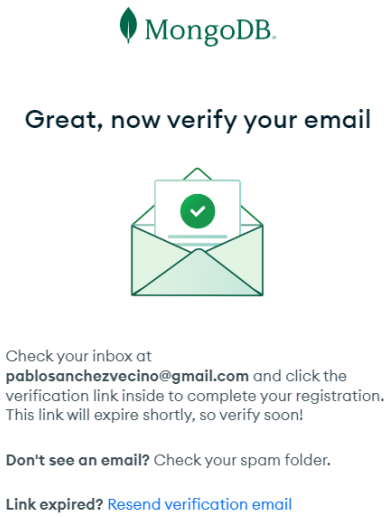

Figura A.30: Solicitud de verificación de correo electrónico de *MongoDB Atlas*

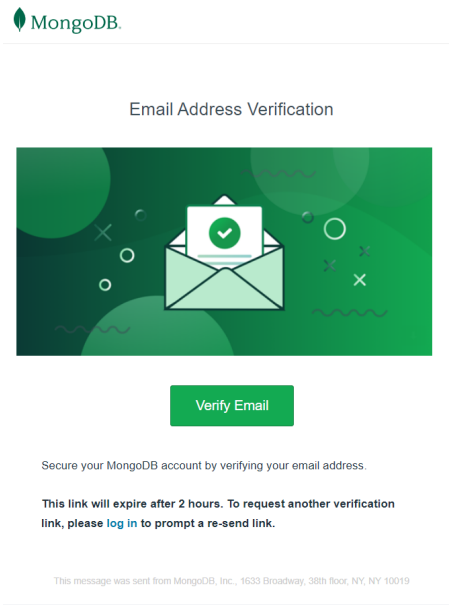

Figura A.31: Contenido del correo de verificación enviado por *MongoDB Atlas*

Tras verificar el correo, será necesario responder a una serie de preguntas con fines estadísticos. Aquí se rellenarán los campos de forma libre y se hará *click* en *Finish* (Figura A.32).

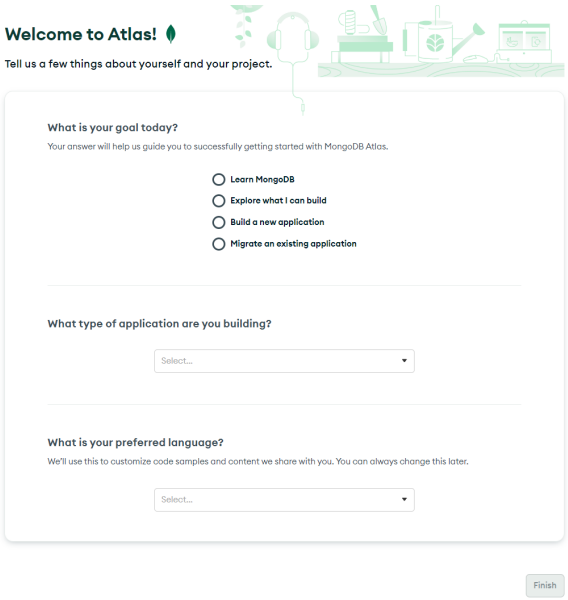

Figura A.32: Formulario para estadísticas de *MongoDB Atlas*

A continuación, se indicarán las preferencias para el despliegue de la base de datos. Se seleccionará la opción gratuita (si se quedara pequeña, existe la opción de actualizarla a una de pago) y como proveedor *cloud* será posible seleccionar cualquiera de las tres opciones. Para

la región, se seleccionará la más cercana a la localización desde donde se vayan a realizar las consultas. En el campo del nombre, se indicará el que se prefiera. Las etiquetas no serán necesarias. Cuando se haya terminado, se hará *click* en *Create* (Figura A.33).

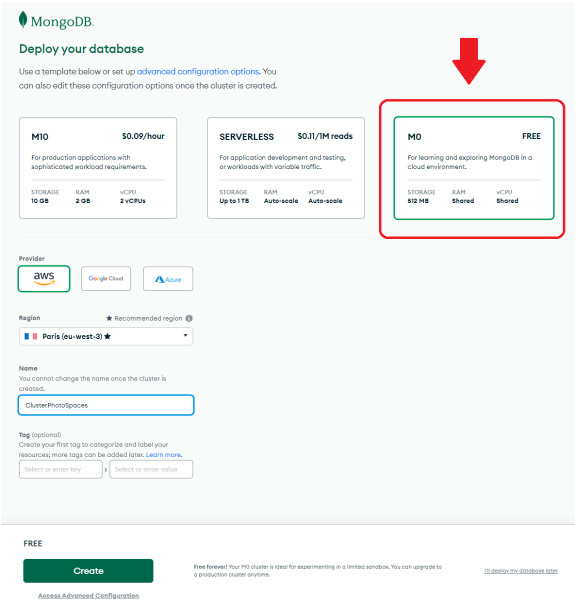

Figura A.33: Configuración de despliegue del *cluster* en *MongoDB Atlas*

Tras esto, *MongoDB Atlas* obligará a realizar unas configuraciones de seguridad iniciales. En primer lugar, será necesario crear un usuario, se seleccionará la opción *Username and Password* y se generará un usuario con las credenciales deseadas. Tras esto, se seleccionará la opción *My Local Environment* y se indicarán las direcciones *IP* a las que se desee limitar el acceso al *cluster*. Si no se desea restringir por dirección *IP*, es posible indicar la dirección *0.0.0.0*. Una vez realizadas las configuraciones, se hará *click* en *Finish and Close* (Figura A.34).

Más tarde, será posible configurar los usuarios y permisos. Por ejemplo, durante el desarrollo, se ha optado por definir un usuario por cada uno de los microservicios que solo tienen permiso de lectura y escritura, y otro aparte para consultas manuales desde *MongoDB Compass*.

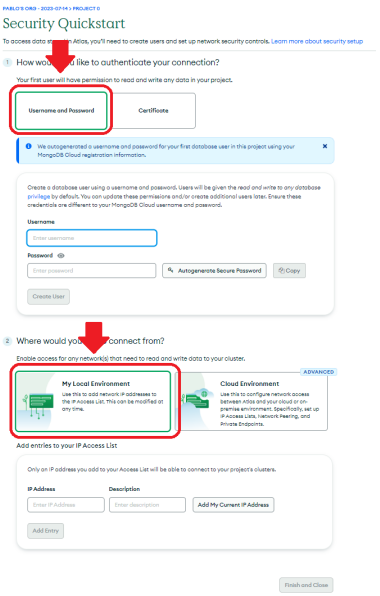

Figura A.34: Ajustes de seguridad iniciales del *cluster* en *MongoDB Atlas*

MongoDB Atlas redirigirá a la página principal y comenzará a desplegar el *cluster*, este proceso puede tardar un poco. Una vez llegados a este punto, será necesario descargar *MongoDB Compass*, la interfaz gráfica que ofrece *MongoDB* para conectarse e interactuar con las bases de datos. Esto se hará desde su página de descargas, donde se indicará la última versión estable para el sistema operativo correspondiente (Figura A.35).

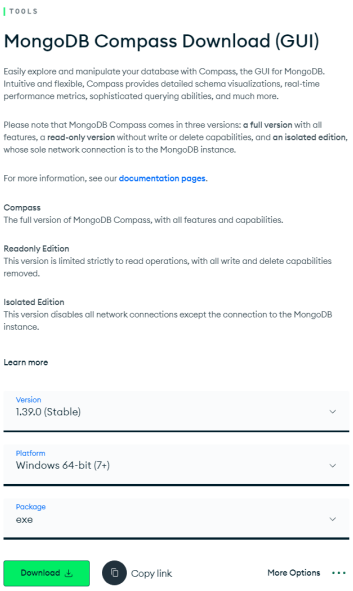

Figura A.35: Página de descargas de *MongoDB Compass*

Una vez iniciado *MongoDB Compass*, se requerirá una cadena de conexión para la conexión

al *cluster*, esta la se obtendrá en el navegador, partiendo de la última página de *MongoDB Atlas* antes de descargar *MongoDB Atlas*. En esta, al lado del nombre del *cluster*, se hará *click* en *Connect*. En el modal que aparecerá, se seleccionará *Compass* y, tras esto, se indicará *I have MongoDB Compass installed* y *1.12 or later* en el desplegable. De esta forma, se obtendrá una plantilla para la cadena de conexión que será posible introducir en *MongoDB Compass*, en la cual será necesario introducir las credenciales de cualquier usuario que se haya configurado anteriormente (Figuras A.36, A.37 y A.38).

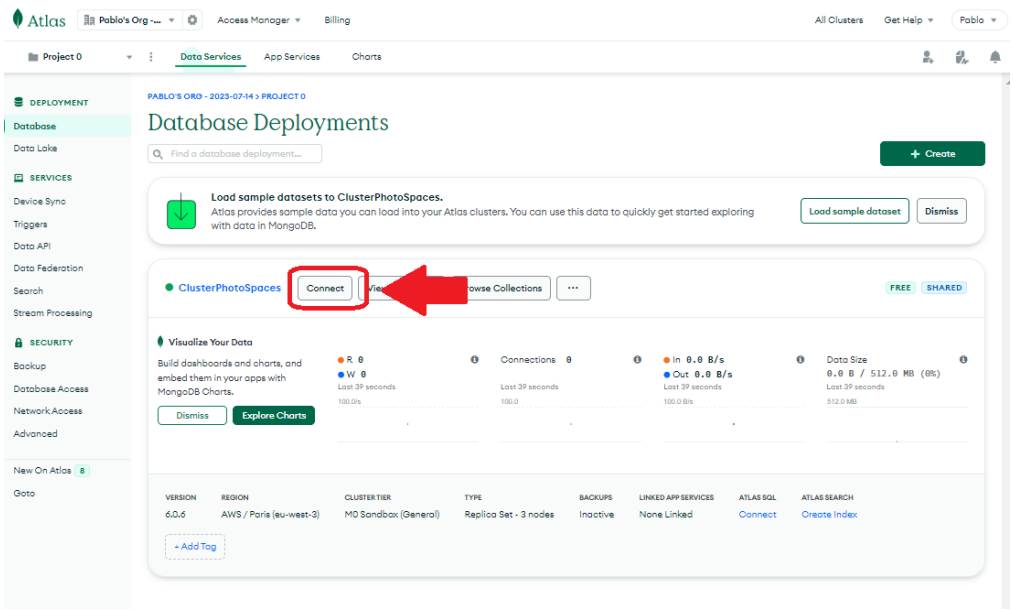

Figura A.36: Acceder al modal con las opciones de conexión en *MongoDB Atlas*

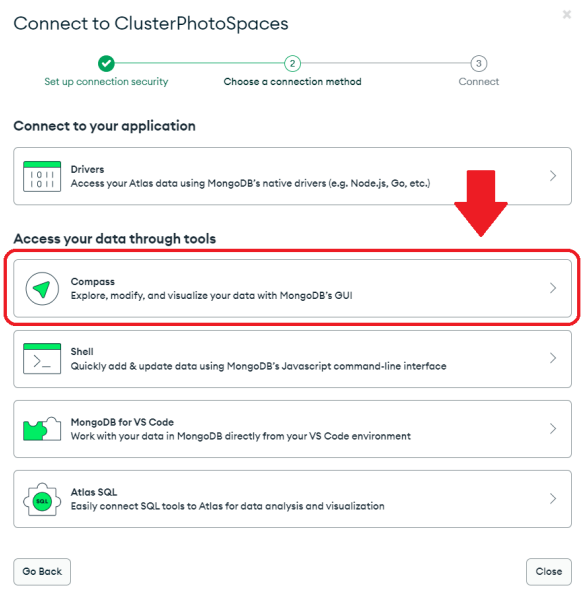

Figura A.37: Seleccionar *Compass* en las opciones de conexión en *MongoDB Atlas*

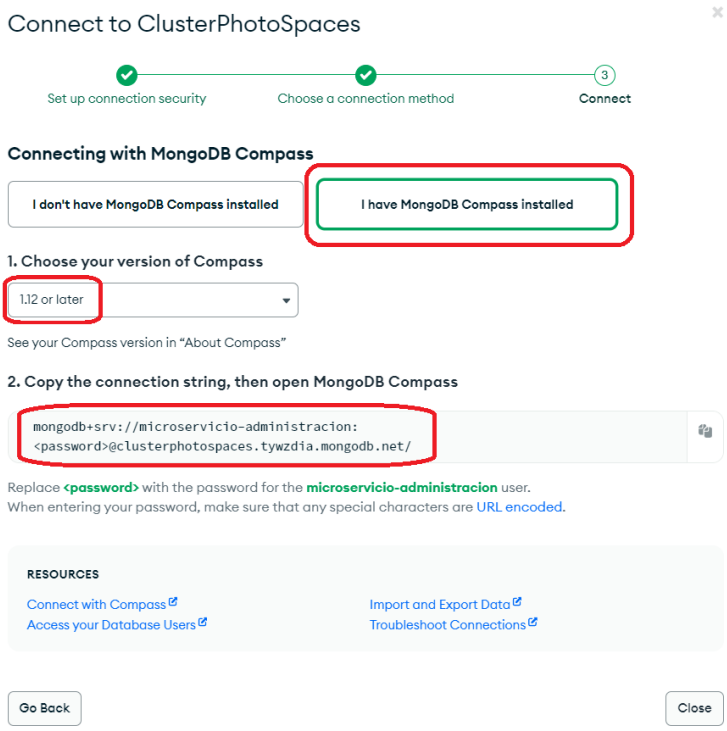

Figura A.38: Obtener plantilla para la cadena de conexión al *cluster* que se introducirá en *MongoDB Compass*

En *MongoDB Compass*, se hará *click* en *New Connection*, se introducirá la cadena modificada y se pulsará en *Connect*. Tras esto será posible interactuar con el *cluster* (Figuras A.39 y A.40).

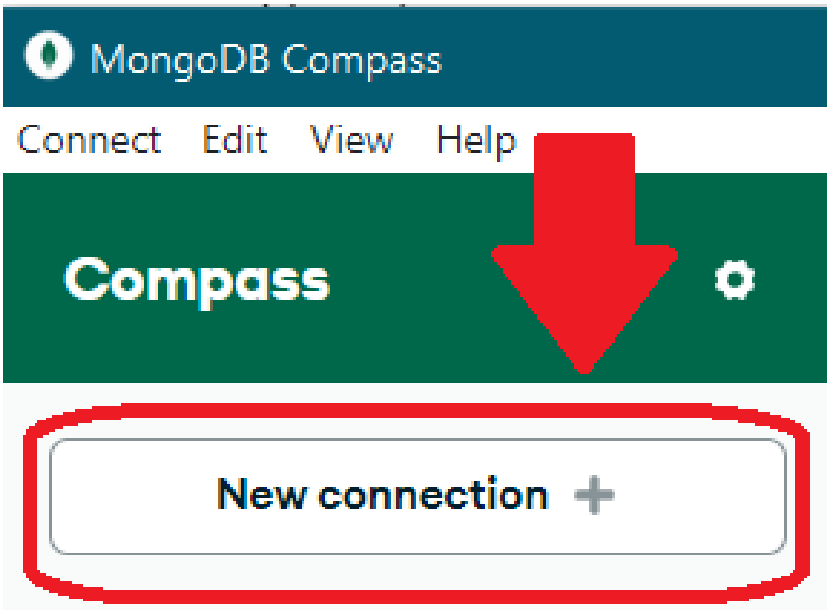

Figura A.39: Indicar que se desea añadir una nueva conexión en*MongoDB Compass*

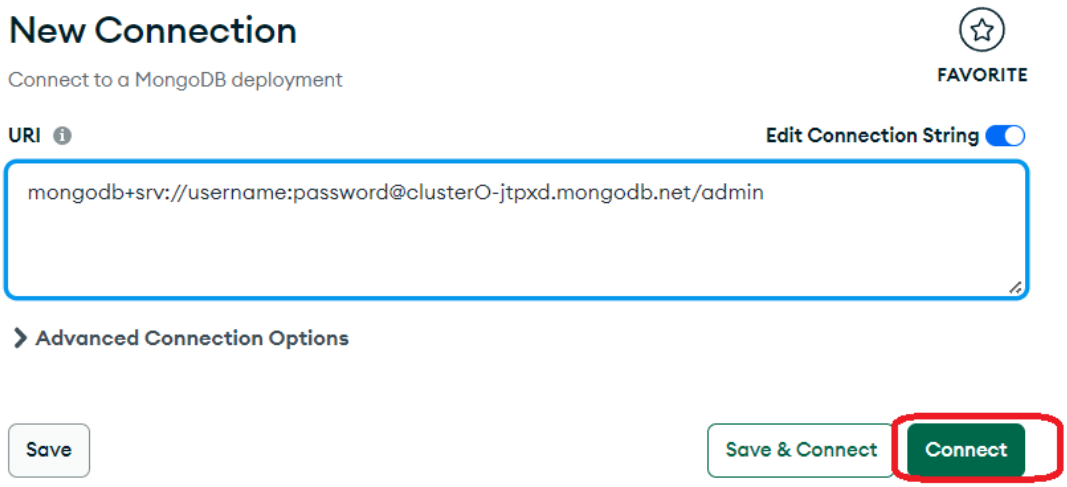

Figura A.40: Introducir cadena de conexión y conectar con el *cluster* desde *MongoDB Compass*

Una vez obtenido el acceso, solo será necesario crear una base de datos haciendo *click* en el símbolo + a la derecha del apartado *Databases* del *cluster*, a la que se dará el nombre *"photoSpacesDB"*. Las colecciones se generarán automáticamente, ya que desde el código, el módulo *mongoose* se encargará de todo. Como obliga también a especificar una colección, se indicará por ejemplo una con el nombre *"servers"*, correspondiente a una de las que se van a ser generadas (Figura A.41).

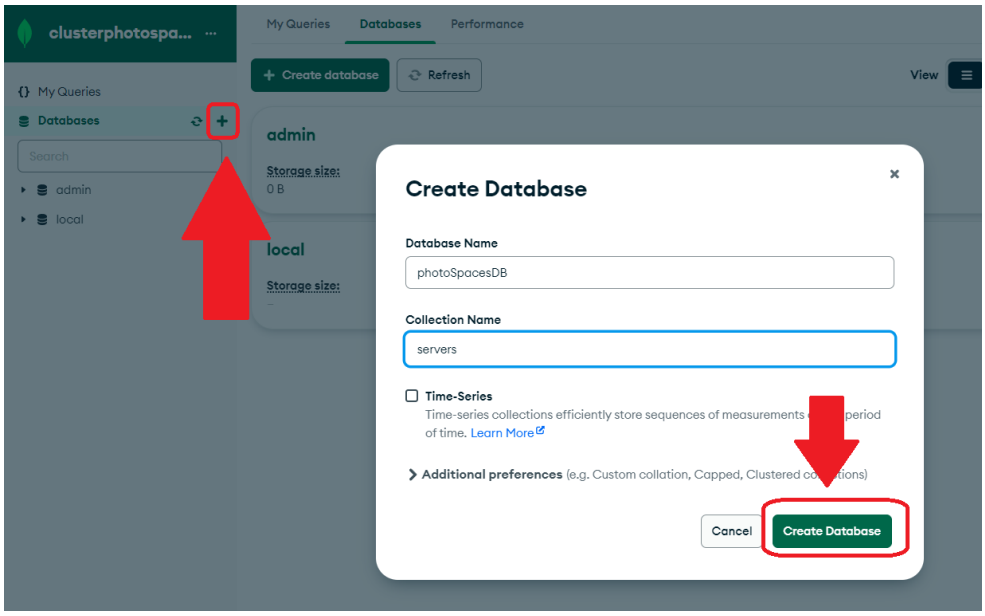

Figura A.41: Crear base de datos desde *MongoDB Compass*

Tras estos pasos, ya estaría preparada la base de datos, solo faltaría indicar las cadenas de conexión a los microservicios para que puedan conectarse a esta, lo cual se explica a partir de la sección A.9.

A.5.2. Despliegue local en *Windows*

En primer lugar, se descargará *MongoDB Community Server* accediendo a este enlace, donde se hará *click* en *Select Package* y en *Download* sin necesidad de modificar los tres parámetros configurables (Figura A.42).

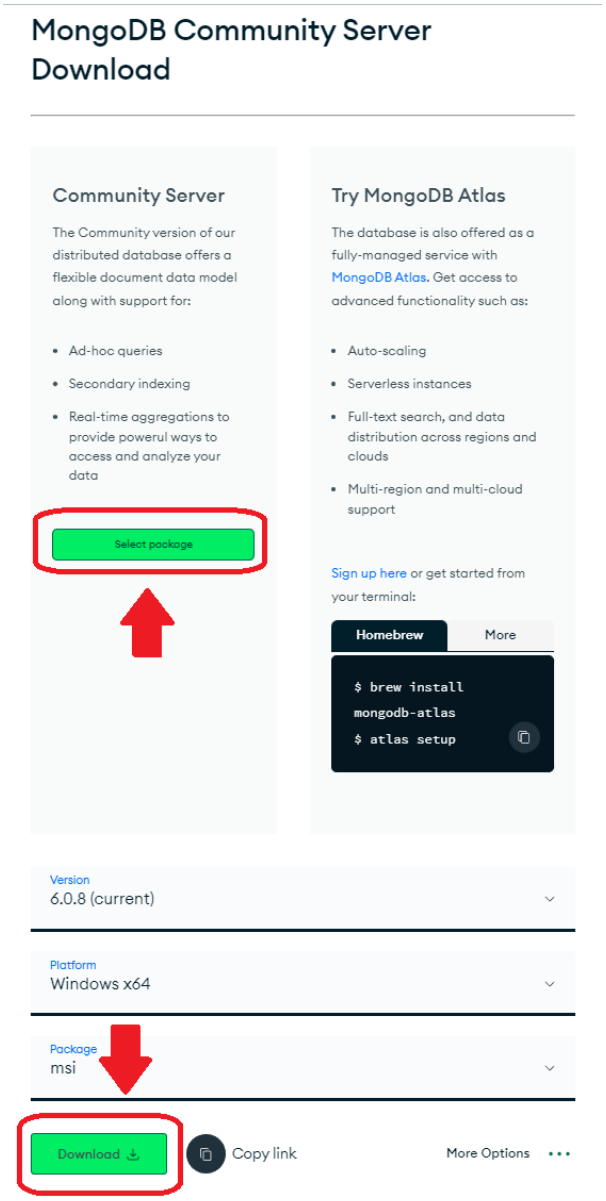

Figura A.42: Descarga de *MongoDB Community Server*

Una vez descargado el instalador, será necesario ejecutarlo. En la primera ventana se hará *click* en *Next* (Figura A.43). A continuación, se aceptará la licencia marcando la casilla correspondiente (Figura A.44).

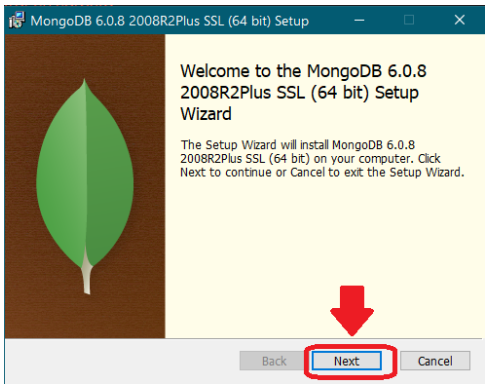

Figura A.43: Ventana inicial del instalador

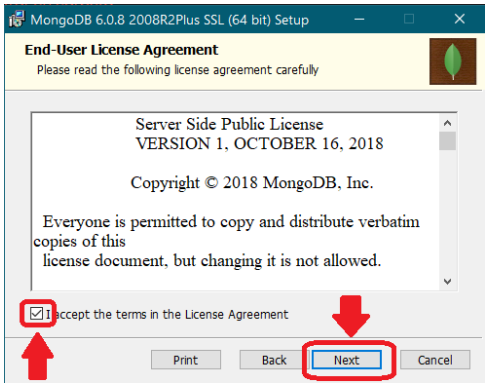

Figura A.44: Aceptar licencia

Tras esto, será necesario optar entre llevar a cabo una instalación completa o una a medida. Se seleccionará la primera opción haciendo *click* en el botón *Complete* (Figura A.45).

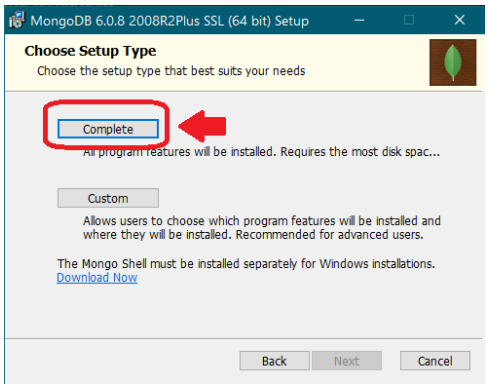

Figura A.45: Seleccionar instalación completa

Ahora se ofrecerá la opción de instalar *MongoDB* como un servicio. En este caso no se marcará la casilla, ya que simplificará el proceso, aunque a cambio será necesario poner en

marcha el servidor cada vez que se vaya a utilizar (Figura A.46).

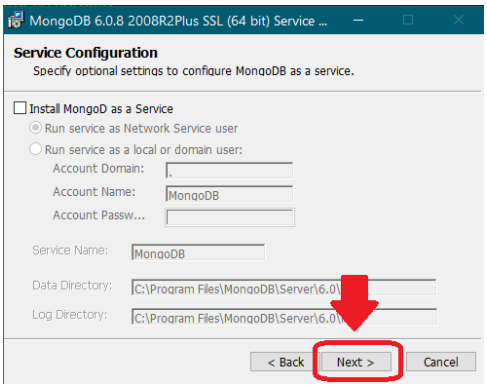

Figura A.46: No seleccionar instalación com servicio

A continuación, se ofrecerá la opción de instalar *MongoDB Compass* también, el cual permitirá interactuar con las bases de datos desde la interfaz gráfica que ofrece. Si no se dispone de una instalación de este de antes, se recomienda marcar la casilla (Figura A.47), ya que la alternativa sería usar *Mongo Shell* mediante una interfaz de línea de comandos, el cual no viene incluido en la instalación de *MongoDB Community Server* en *Windows*, por lo que sería necesario descargarlo e instalarlo de forma independiente. Además, el propio *MongoDB Compass* permite también la interacción mediante una interfaz de línea de comandos.

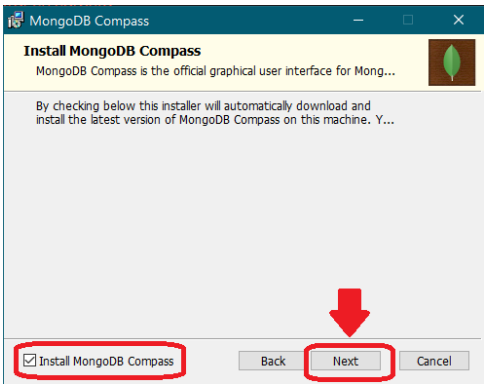

Figura A.47: Indicar instalación de adicional de *MongoDB Compass*

Finalmente, en la siguiente ventana, se hará *click* en el botón *Install* (Figura A.48) y se proporcionarán permisos de administrador si fuera necesario. Tras esto, comenzará el proceso de instalación (Figura A.49). Una vez finalice, se hará *click* en *Finish* (Figura A.50). Ya se dispondría del servidor local con *MongoDB* instalado en el sistema.

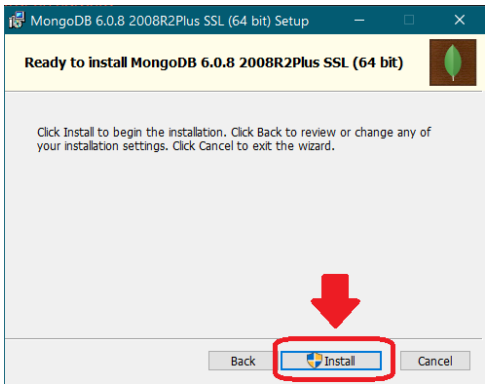

Figura A.48: Comenzar proceso de instalación

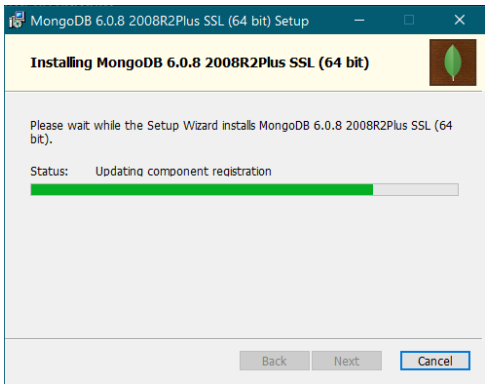

Figura A.49: Proceso de instalación en curso

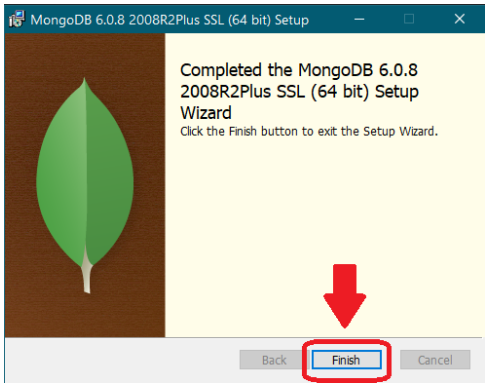

Figura A.50: Proceso de instalación finalizado

En su versión para *Windows*, *MongoDB* almacena la información en el directorio `C:\data\db`, el cual no es creado automáticamente, por lo que será necesario generarlo manualmente.

Para evitar tener que navegar hasta el directorio de instalación cada vez que se desee arrancar el servidor, es posible añadir a la variable *Path* del sistema la ruta del ejecutable, en este

caso `C:\Program Files\MongoDB\Server\6.0\bin`. Este proceso se detalla en el apartado A.4.

Para desplegar el servidor, se ejecutará el comando `mongod` (Figura A.51). Si la ejecución se para, probablemente se deba a que no se ha creado correctamente el directorio mencionado anteriormente. Si siguiera adelante sin detenerse, ya se dispondría del servidor en marcha, al cual será posible conectarse localmente utilizando la cadena de conexión *"mongodb://127.0.0.1:27017/photoSpacesDB"*.

Aunque se especifique el nombre de la base de datos en el contenido de la cadena, no será necesario crearla manualmente, ya que se generará automáticamente al establecerse la conexión desde el código con el módulo *mongoose* por primera vez.

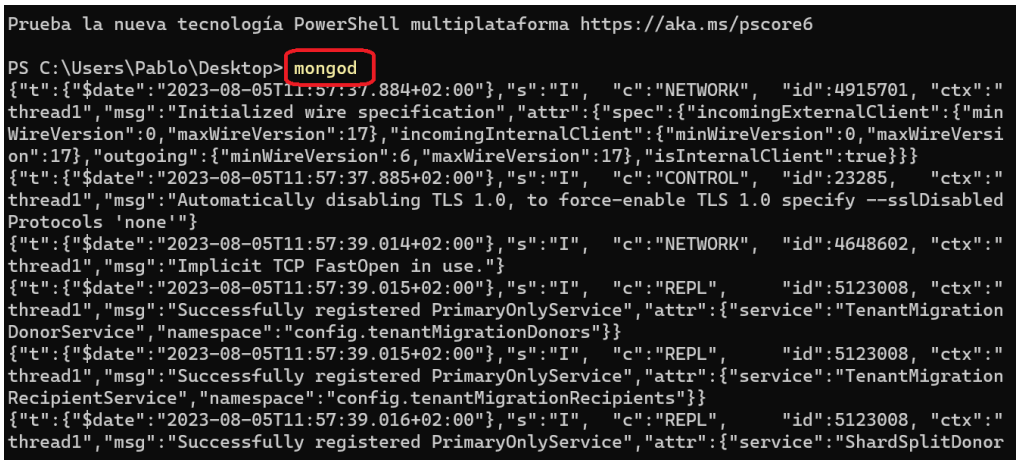

Figura A.51: Desplegar servidor *MongoDB*

A.5.3. Despliegue local en *Linux*

Para montar un servidor *MongoDB* local en una máquina, se seguirán los pasos detallados a continuación.

En primer lugar, será necesario instalar *gnupg* y *curl* si no se encuentran en el sistema utilizando el comando `sudo apt-get install gnupg curl`.

A continuación, se importará la clave pública *GPG* para el repositorio de *MongoDB* con el comando `curl -fsSL https://pgp.mongodb.com/server-6.0.asc | sudo gpg -o /usr/share/keyrings/mongodb-server-6.0.gpg --dearmor`.

Tras esto, se creará un archivo de lista para el repositorio de *MongoDB* con el comando `echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-6.0.gpg ] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list`.

Será necesario recargar la base de datos de paquetes, para ello se utilizará el comando `sudo apt-get update`.

Tras estas configuraciones, será posible descargar e instalar *MongoDB*. Se introducirá el comando sudo `apt-get install -y mongodb-org`.

Una vez instalado, para arrancar el servicio, como en este caso la distribución utiliza *systemd*, se utilizará el comando `sudo systemctl start mongod`. Tras esto, será posible comprobar el estado del servicio con el comando `sudo systemctl status mongod` (Figura A.52).

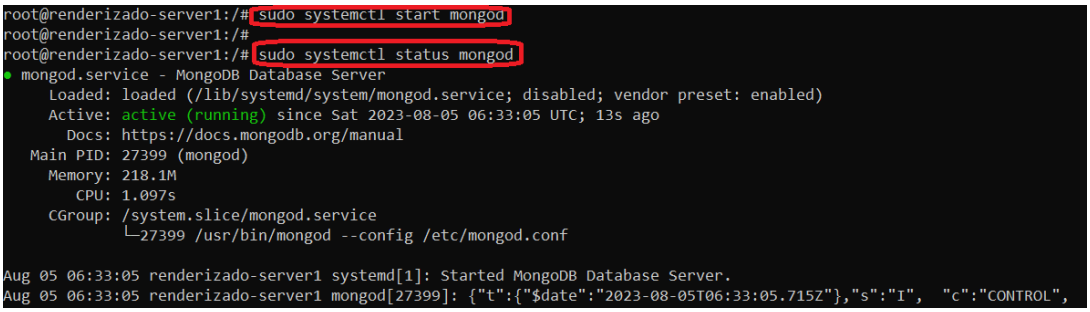

Figura A.52: Arrancar y comprobar el estado del servicio *mongod*

Una vez instalado, se tendrá acceso al comando `mongosh`, este permitirá interactuar con las bases de datos a través de comandos de *MongoDB Shell* (Figura A.53).

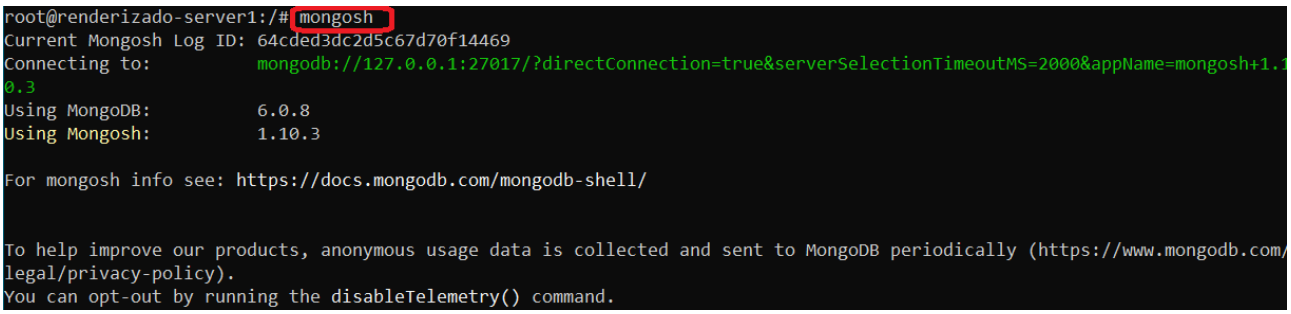

Figura A.53: Acceder a *mongosh*

Una vez desplegada la base de datos, para establecer una conexión localmente con esta desde la aplicación se utilizará la cadena de conexión *"mongodb://127.0.0.1:27017/photoSpacesDB"*.

De nuevo, al igual que en *Windows*, no será necesario crear la base de datos indicada manualmente, ya que se generará automáticamente al establecerse por primera vez la conexión desde el código con el módulo *mongoose*.

A.6. Configuración del correo electrónico

El sistema requiere una cuenta de correo electrónico que actúe como remitente cuando un usuario solicite la recepción de la imagen renderizada por este medio, por lo que será necesario configurar una.

En esta guía se explicará el proceso con una cuenta de *Gmail*, aunque es importante tener claro que hay opciones más indicadas que el servicio de *Google* para este caso de uso, ya que pueden surgir problemas de seguridad y bloqueos por detección de *spam*. Una alternativa más profesional, pero de pago, sería *Twilio SendGrid*, aunque existen bastantes más.

Desde la bandeja de entrada de la cuenta de *Gmail* que se desee utilizar, se hará *click* en la foto de perfil en la esquina superior derecha. En el modal que aparecerá, se accederá a *Gestionar tu cuenta de Google* (Figura A.54).

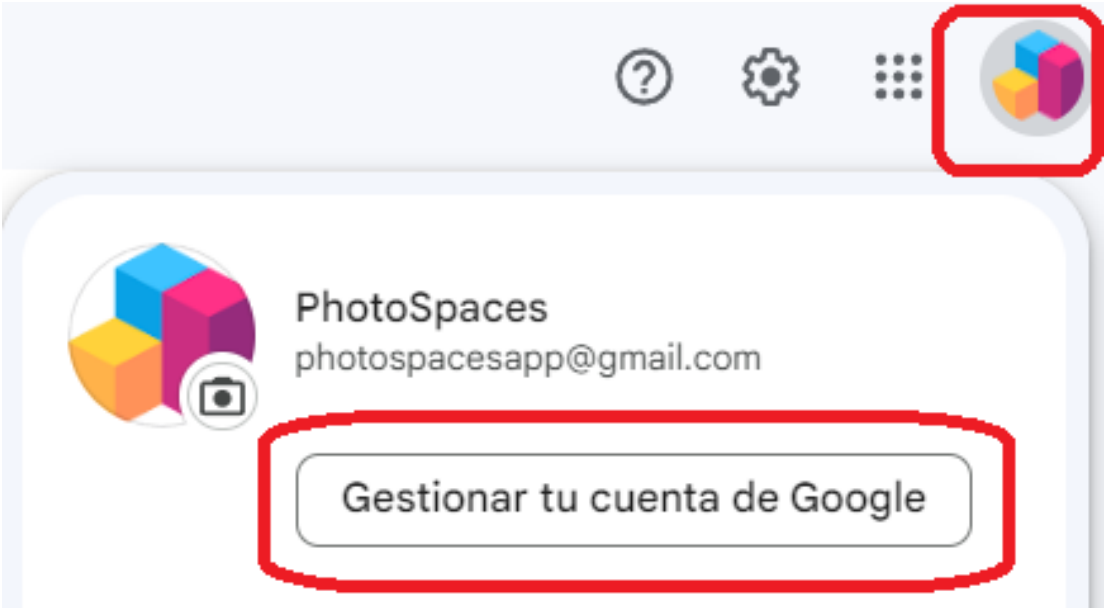

Figura A.54: Acceder a la página de gestión de la cuenta de *Google*

El primer paso consistirá en habilitar la verificación en dos pasos para la cuenta si aún no lo está. Para ello, se seleccionará *Seguridad* en el menú de la izquierda y *Verificación en dos pasos* en el nuevo menú que se mostrará (Figura A.55).

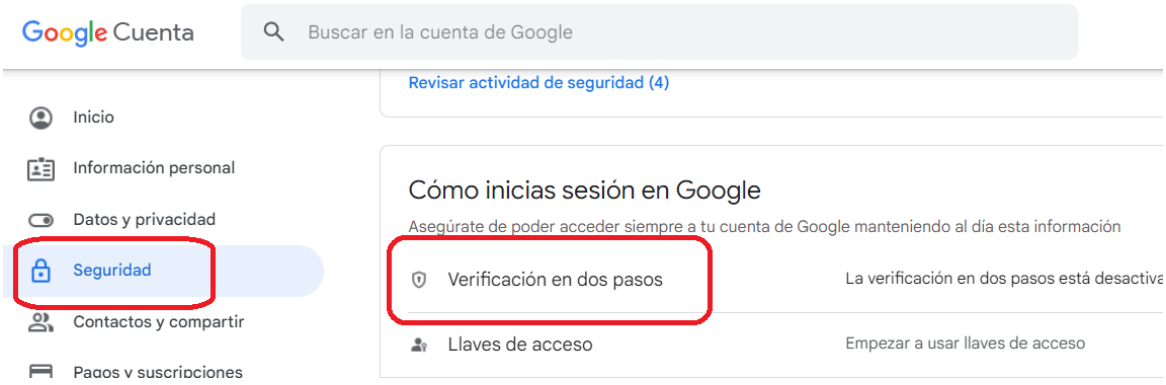

Figura A.55: Acceder a la configuración de la verificación en dos pasos

A continuación, se hará *click* en *Empezar*. Tras esto, se requerirá un número de teléfono. Se introcucirá el número en el que se desee recibir el código correspondiente y si se prefiere que este se envíe por *SMS* o mediante una llamada telefónica (Figura A.56).

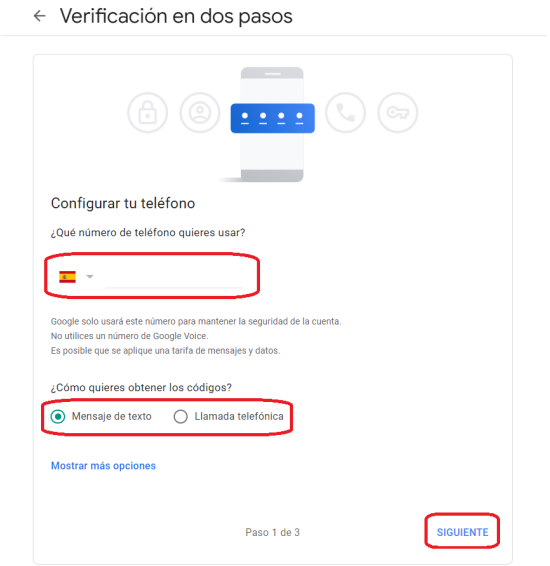

Figura A.56: Indicar número de teléfono y método de recepción del código para activar la verificación en dos pasos

Tras introducir el código recibido, será posible activar la verificación en dos pasos pulsando *Activar* en la ventana siguiente (Figura A.57).

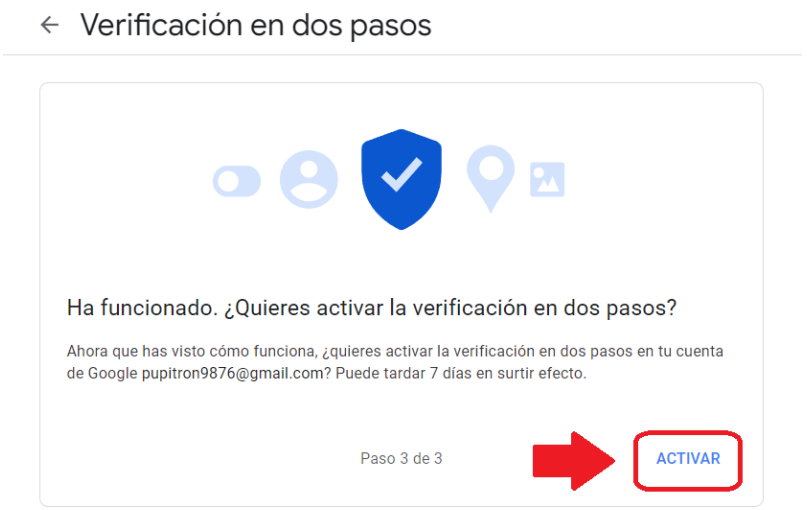

Figura A.57: Activar la verificación en dos pasos

Una vez realizada esta configuración, se desbloqueará la sección *Contraseñas de aplicaciones*, a la cual se accederá a través de la barra de búsqueda. Con introducir en esta la cadena *"aplicaciones"* será suficiente para que se muestre. Se hará *click* en esta (Figura A.58).

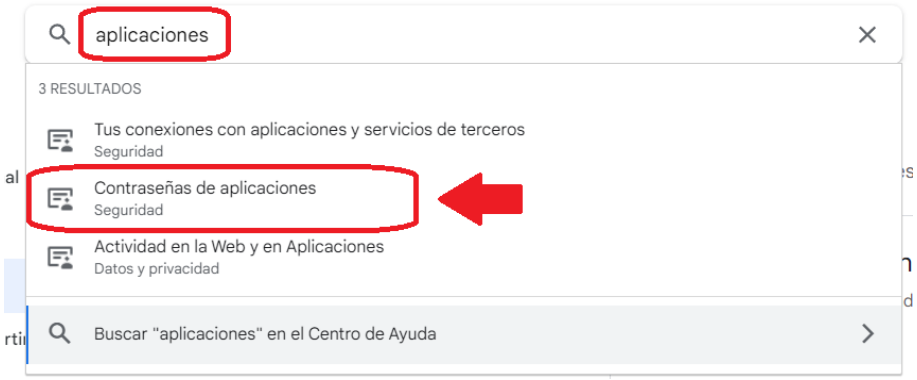

Figura A.58: Acceder a la sección *Contraseñas de aplicaciones*

Dentro de la nueva sección, se mostrarán dos desplegables. Será posible seleccionar las opciones de forma libre, ya que solo sirven para establecer el nombre de la contraseña que se va a generar. También es posible establecer un nombre personalizado. Tras esto, se hará *click* en *Generar* (Figura A.59).

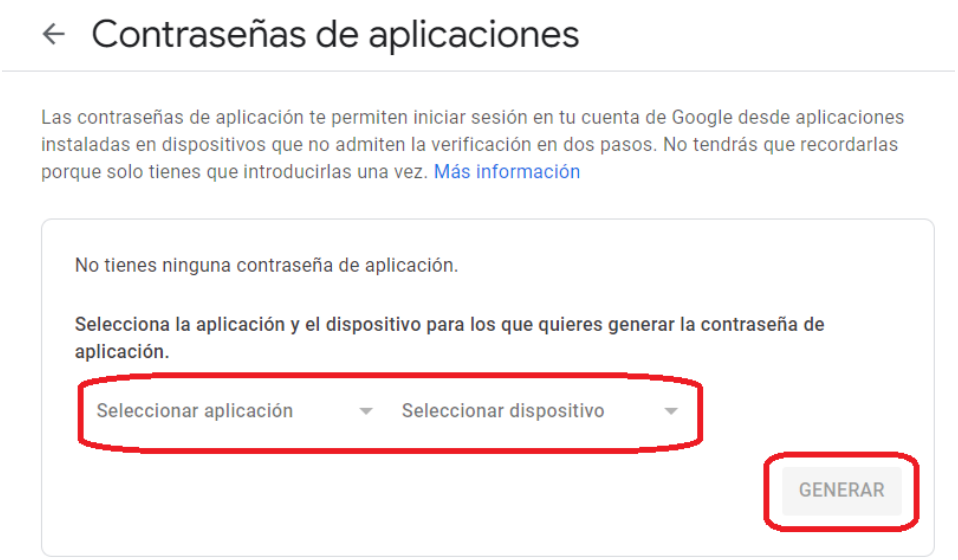

Figura A.59: Generar contraseña de aplicación

Se mostrará la contraseña generada, es muy importante apuntarla en este momento, ya que no será posible volver a consultarla. Si esta se pierde, será necesario generar otra distinta siguiendo los pasos anteriores de nuevo (Figura A.60).

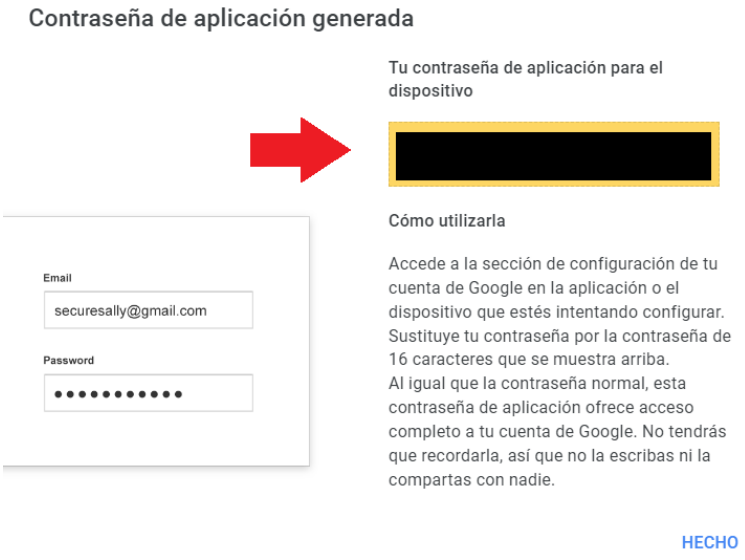

Figura A.60: Contraseña de aplicación generada

Una vez generada la contraseña, solo faltaría especificar al *microservicio de gestión de peticiones* la dirección de correo electrónico que se acaba de configurar y la contraseña generada

para que el módulo *nodemailer* pueda trabajar con estos desde el código. Este proceso de configuración se detalla más adelante en la sección A.9.

A.7. Obtención del código fuente

El código fuente de la aplicación se encuentra en este repositorio de *GitHub*.

Las dos formas más sencillas para disponer de este de forma local serían las siguientes:

- Si se dispone de la herramienta *Git*, no será necesario acceder al repositorio remoto, simplemente bastará con abrir un terminal en el directorio donde se desee clonar el proyecto e introducir el comando `git clone https://github.com/pablosanchezvecino/PhotoSpaces` (Figura A.61).

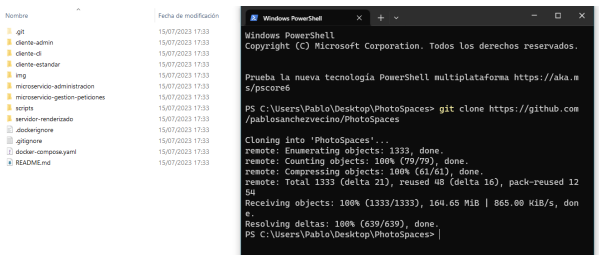

Figura A.61: Clonación del código utilizando *Git*

- Si se opta por prescindir de *Git*, también es posible descargar el código comprimido en formato *ZIP* y descomprimirlo en el directorio deseado. Para esto, será necesario hacer *click* en el botón *Code* de la página del repositorio y, tras esto, seleccionar la opción *Download ZIP* (Figura A.62).

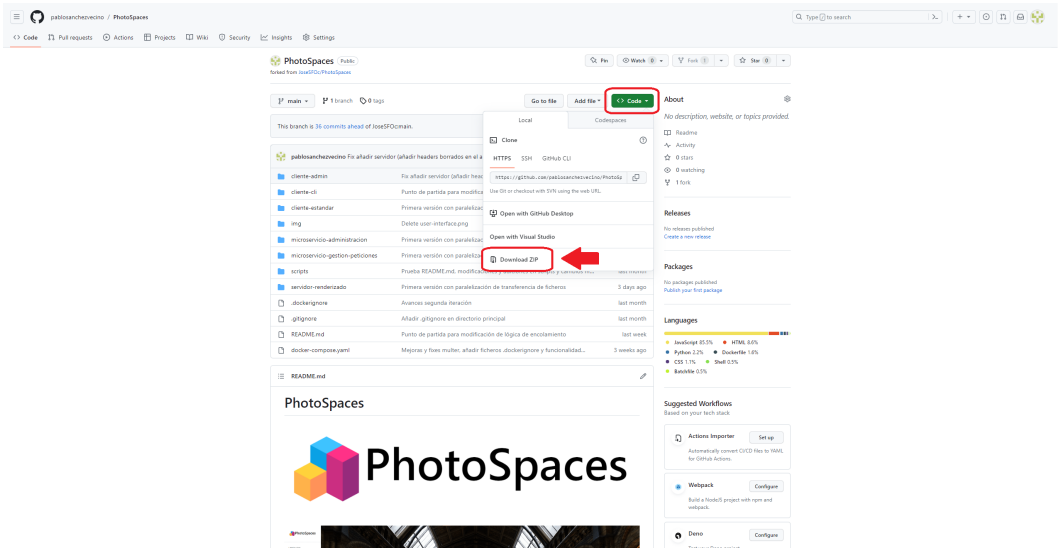

Figura A.62: Descargar código en formato *ZIP* desde la página del repositorio

Una vez descargado, será posible descomprimirlo con cualquier herramienta de descompresión compatible con ficheros *.zip*, por ejemplo *WinRar* (Figura A.63).

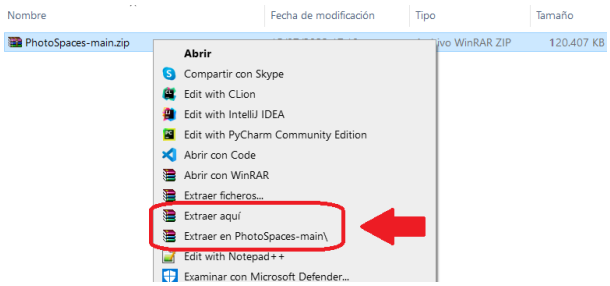

Figura A.63: Extraer archivo *ZIP*

A.8. Instalación de módulos *npm*

Una vez se disponga del código en local, será necesario instalar las dependencias de cada componente de la aplicación que se desee desplegar.

Para ello, será necesario abrir un terminal en el directorio raíz de cada componente y ejecutar el comando `npm install`, el cual instalará todas las dependencias indicadas por los ficheros *package.json* y *package-lock.json* en un nuevo directorio *node-modules* (Figura A.64).

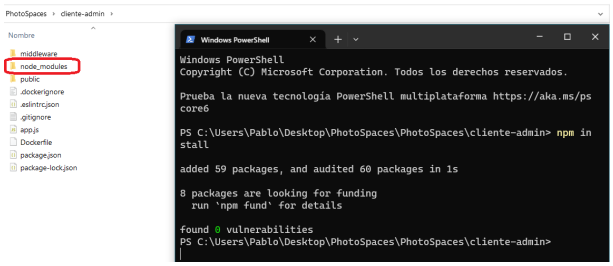

Figura A.64: Instalación de módulos *npm* en el *cliente de administración*

Al instalar las dependencias del *servidor de renderizado* se puede apreciar que *npm* indica una vulnerabilidad de seguridad, esto se debe a que por defecto se instala una versión desactualizada de *systeminformation*, el módulo que se utiliza para obtener las especificaciones del servidor, dado que la última versión da problemas en algunos equipos.

Es posible probar a actualizarla para solucionar el *warning* introduciendo los comandos `npm uninstall systeminformation` y `npm install systeminformation` (Figura A.65) y comprobar si funciona correctamente.

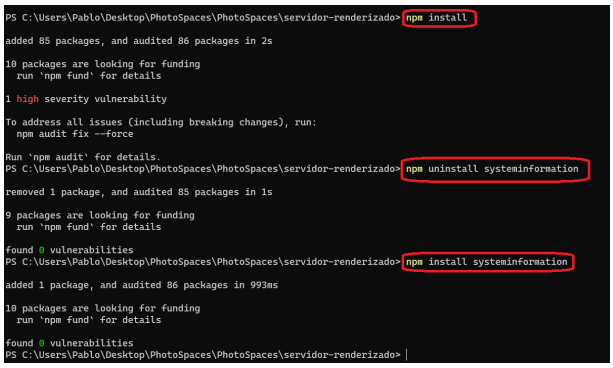

Figura A.65: Actualización de *systeminformation* a su última versión

Si no funcionara con la última versión, se podrá volver a la versión desactualizada introduciendo los comandos `npm uninstall systeminformation` y `npm install systeminformation@4` (Figura A.66).

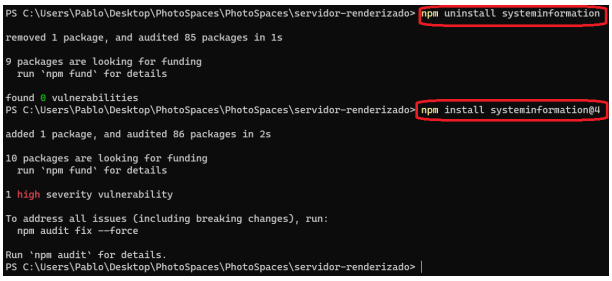

Figura A.66: Reversión de la actualización de *systeminformation*

A.9. Configuración con variables de entorno

Antes de realizar su despliegue, proceso que se detalla a partir de la sección A.12, los distintos componentes del sistema requieren una configuración previa, esta se realiza a través de una serie de valores que los distintos componentes obtienen a través de variables de entorno.

Estas variables varían de un componente a otro, por lo que a continuación se incluyen plantillas para los seis componentes. En estas, por cada parámetro, se indica el nombre, un valor de ejemplo, qué efectos tiene y qué valor toma por defecto si este no es declarado. También es posible consultarlas en los ficheros *env.js* del directorio raíz de cada componente.

La forma de especificar estas variables variará según el despliegue que se vaya a realizar:

- En el caso de que se vaya a realizar un despliegue directamente sobre el *host*, tan solo será necesario crear en el directorio raíz del componente un nuevo fichero con el nombre *.env*, en el cual se pegará el contenido de la plantilla correspondiente y se modificará con los valores deseados.
- Si se opta por un despliegue con *Docker* utilizando directamente el fichero *Dockerfile* para generar la imagen del contenedor, estas se especificarán en el contenido de este.
- Si se utilizan contenedores a través de *Docker Compose*, estas se indicarán en el archivo *docker-compose.yaml* situado en el directorio raíz del repositorio.

Estos pasos se recuerdan más adelante con más detalle en el apartado correspondiente a cada opción de despliegue.

A.9.1. Plantilla para el *cliente estándar*

```
# Puerto de escucha del propio servidor
# Valor por defecto: 8081
```

```
PORT=8081

# URL del microservicio de gestión de peticiones
# Valor por defecto: "http://localhost:9001"
REQUEST_HANDLING_MICROSERVICE_URL="http://localhost:9001"
```

A.9.2. Plantilla para el *cliente de administración*

```
# Puerto de escucha del propio servidor
# Valor por defecto: 8080
PORT=8080

# URL del microservicio de administración
# Valor por defecto: "http://localhost:9000"
ADMINISTRATION_MICROSERVICE_URL="http://localhost:9000"

# Periodo (en ms) de refresco del panel de administración
# Valor por defecto: 1000 (1s)
REFRESH_PERIOD_MS=1000

# Cantidad máxima de tarjetas que se mostrará por
# cada contenedor del panel de administración
# Valor por defecto: 0 (sin límite)
MAX_CARDS_PER_CONTAINER=2000

# Direcciones IP a los que se servirá el contenido
# (en principio deberían ser las de los administradores del sistema)
# Valor por defecto: "0.0.0.0" (se aceptan todas)
ALLOWED_IPS="0.0.0.0"
```

A.9.3. Plantilla para el *microservicio de administración*

```
# URL para la conexión con la base de datos
# Valor por defecto: "mongodb://127.0.0.1:27017/photoSpacesDB"
MONGODB_CONNECTION_STRING="mongodb://127.0.0.1:27017/photoSpacesDB"

# Puerto de escucha del propio microservicio
```

```
# Valor por defecto: 9000
PORT=9000

# Puerto de escucha de los servidores de renderizado
# Valor por defecto: 3000
RENDER_SERVER_PORT=3000

# URL del microservicio de gestión de peticiones
# Valor por defecto: "http://localhost:9001"
REQUEST_HANDLING_MICROSERVICE_URL="http://localhost:9001"

# Orígenes permitidos para la configuración de CORS.
# Idealmente, únicamente se permitirán las URLs de los servidores
# que proporcionen el contenido del cliente de administración que sean
# directamente accesibles desde el navegador.
# Valor por defecto: "*" (se permite cualquier origen)
CORS_ALLOWED_ORIGINS="http://localhost:8080"

# Direcciones IP permitidas
# (en principio deberían ser las de los administradores del sistema)
# Valor por defecto: "0.0.0.0" (se aceptan todas)
ALLOWED_IPS="::1,127.0.0.1,192.168.1.35,172.17.0.1,192.168.1.33"
```

A.9.4. Plantilla para el *microservicio de gestión de peticiones*

```
# URL para la conexión con la base de datos
# Valor por defecto: "mongodb://127.0.0.1:27017/photoSpacesDB"
MONGODB_CONNECTION_STRING="mongodb://127.0.0.1:27017/photoSpacesDB"

# Puerto de escucha del propio microservicio
# Valor por defecto: 9001
PORT=9001

# Puerto de escucha de los servidores de renderizado
# Valor por defecto: 3000
RENDER_SERVER_PORT=3000
```

```
# URL del microservicio de administración
# Valor por defecto: "http://localhost:9000"
ADMINISTRATION_MICROSERVICE_URL="http://localhost:9000"

# Orígenes permitidos para la configuración de CORS.
# Idealmente, únicamente se permitirán las URLs de los servidores
# que proporcionen el contenido del cliente estándar que sean
# directamente accesibles desde el navegador.
# Valor por defecto: "*" (se permite cualquier origen)
CORS_ALLOWED_ORIGINS="http://localhost:8081"

# Dirección de la cuenta Gmail que envía los correos automatizados
# Valor por defecto: ""
EMAIL_USER="example@example.com"

# Contraseña de aplicación de la cuenta de Gmail que envía los correos automatizados
# Valor por defecto: ""
EMAIL_PASSWORD="abcd abcd abcd abcd"

# Periodo (en ms) de la comprobación de la cola y los servidores disponibles
# Valor por defecto: 30000 (30s)
DB_CHECK_PERIOD_MS=30000

# Periodo (en ms) de limpieza de archivos temporales que se hayan podido
# quedar escritos y que ya no son necesarios
# Valor por defecto: 86400000 (24h)
CLEANUP_INTERVAL_MS=100000

# Tiempo de espera (en ms) entre las finalización de una ronda
# de reintento de los envíos de todos los emails pendientes y la siguiente
# Valor por defecto: 3600000 (1h)
EMAIL_SENDING_BACKUP_INTERVAL_MS=25000

# Tiempo de espera (en ms) entre la recepción de la respuesta a una consulta
# del tiempo restante estimado al servidor de renderizado y la siguiente
# Valor por defecto: 1000 (1s)
RENDERING_SERVER_POLLING_INTERVAL_MS=1000
```

```
# Tamaño máximo de archivo en bytes que aceptará el servidor
# Valor por defecto: Infinity (no se establece límite)
MAX_FILE_SIZE_BYTES=1073741824
```

A.9.5. Plantilla para el *servidor de renderizado*

```
# Puerto de escucha del propio servidor de renderizado
# Valor por defecto: 3000
PORT=3000

# Direcciones IP permitidas
# (en principio deberían ser las de los microservicios)
# Valor por defecto: "0.0.0.0" (se aceptan todas)
ALLOWED_IPS="::1,127.0.0.1"

# Mostrar salida del script Python
# Valor por defecto: true
SHOW_PYTHON_LOGS=true
```

A.9.6. Plantilla para el *cliente CLI*

```
# URL del microservicio de gestión de peticiones
# Valor por defecto: "http://localhost:9001"
REQUEST_HANDLING_MICROSERVICE_URL="http://localhost:9001"

# Tiempo de espera (en ms) entre la recepción de la respuesta a una consulta
# del estado de la petición enviada y la siguiente
# Valor por defecto: 1000 (1s)
POLLING_INTERVAL_MS=2000
```

A.10. Guía configuración de red

El propósito de esta sección es aclarar cualquier incertidumbre que pueda surgir en torno a la configuración de red del despliegue mediante el uso de las variables de entorno. Este proceso puede resultar complejo si no se tienen lo suficientemente claras las comunicaciones entre los distintos componentes.

En primer lugar, es importante abordar un aspecto que podría generar confusión. Cuando se hace referencia a los componentes *cliente estándar* y *cliente de administración*, resulta esencial tener clara la distinción entre los componentes que están desplegados y los clientes servidos. Los primeros consisten en servidores *Express* que se mantienen en espera en un puerto específico, listos para suministrar el contenido que hace posible la ejecución del cliente correspondiente cuando un navegador lo solicita. Por otro lado, los clientes servidos se refieren a la ejecución en el navegador web de la máquina local del usuario posibilitada por los contenidos proporcionados por el servidor, desde donde se generan las distintas peticiones a los microservicios.

A continuación, por cada componente del sistema, se indicarán los valores que habrá que asignarle a las variables de entorno relacionadas con la configuración de red

- ***Cliente estándar***:
  - ***PORT:*** El puerto de escucha del servidor *Express* que se encargará de proporcionar el contenido del cliente cuando se contacte con este desde el navegador, en el cual será necesario introducir la dirección *IP* del *host* en el que se esté ejecutando el servidor y el puerto especificado.
  - ***REQUEST_HANDLING MICROSERVICE_URL:*** Las peticiones de este cliente son atendidas por el *microservicio de gestión de peticiones*. Para que sea posible contactar con este, será necesario especificar el protocolo, dirección *IP* y puerto de escucha del despliegue correspondiente a dicho componente (ej.: *"http://127.0.0.1:8081"*). Si se hubiera obtenido un nombre de dominio, se podría incluir este en lugar de la combinación de la dirección *IP* y el puerto (ej.: *"https://microservicio-gestion-peticiones.com"*). Si se hubiera configurado un balanceador de carga entre los dos componentes, se haría de la misma forma pero con el *host* sobre el que se ejecuta este.
- ***Cliente de administración***
  - ***PORT:*** De la misma forma que en la variable *PORT* del *cliente estándar*, se especificará el puerto que se desee introducir en el navegador para que el servidor sirva los contenidos del *cliente de administración*.
  - ***ADMINISTRATION_MICROSERVICE_URL:*** El *microservicio de administración* es el encargado de satisfacer las peticiones de este cliente. Para contactar con este, de nuevo, será preciso indicar el protocolo, la dirección *IP* y el puerto del *host* donde se está ejecutando la instancia de este microservicio (ej.: *"http://127.0.0.1:8081"*), o

el nombre de dominio en caso de haber obtenido uno (ej.: *"https://microservicio-administracion.com"*).

- ***ALLOWED_IPS:*** Para evitar problemas de seguridad, es posible especificar las direcciones *IP* cuyas peticiones serán atendidas, ignorando cualquier otra. Lo ideal sería incorporar una dirección por cada dispositivo que vaya a desempeñar el papel de cliente para este microservicio, es decir, por cada dispositivo que vaya a utilizar un navegador para acceder al panel de administración del sistema. A modo de ejemplo, si se pretende acceder al panel de administración desde dos máquinas diferentes con las direcciones *IP 192.168.1.30* y *192.168.1.31* respectivamente, sería necesario especificar la cadena *"192.168.1.30,192.168.1.31"*.

- ***Microservicio de administración***

  - ***PORT:*** Puerto de escucha del servidor que implementa este componente, encargado de atender las peticiones de administración. Este debe coincidir con el puerto especificado en la variable *ADMINISTRATION_MICROSERVICE_URL* del *cliente de administración*.
  - ***RENDER_SERVER_PORT:*** Puerto que indicará el microservicio cuando se intente comunicar con los *servidores de renderizado*. Su valor debe coincidir con las variables *PORT* de estos y *RENDER_SERVER_PORT* del *microservicio de gestión de peticiones*.
  - ***REQUEST_HANDLING_MICROSERVICE_URL:*** El *microservicio de administración* notifica al *microservicio de gestión de peticiones* cuando hay un nuevo servidor disponible en el sistema que este último no es capaz de detectar por sí mismo. Para esta comunicación se utiliza la *URL* indicada en esta variable. El valor a especificar sigue las mismas reglas que la variable *ADMINISTRATION_MICROSERVICE_URL* del *cliente estándar*. Lo más normal será que coincida, aunque si ambos extremos de la comunicación se encuentran desplegados sobre el mismo *host*, también será posible especificar la dirección *IP* de *loopback* en lugar de una remota.
  - ***CORS_ALLOWED_ORIGINS:*** Esta variable se utiliza para evitar problemas de *CORS* en el navegador. Esta le indica al microservicio qué orígenes pueden tener las peticiones recibidas para que las incluya en las cabeceras de sus respuestas y que el navegador pueda confiar en este. Idealmente, se indicará una *URL* por cada instancia del componente *cliente de administración* desplegada que sea accesible directamente desde el navegador. Si no se especifica un valor para esta variable, no se restringirá ningún origen. De esta forma no se generarán problemas relacionados con *CORS*, pero la seguridad será menor.

- ***ALLOWED_IPS:*** Para evitar que se puedan realizar peticiones a este componente desde direcciones *IP* rechazadas a la hora de servir el panel de administración utilizando otros medios, estas también se le indicarán al microservicio. Estas deberían ser las mismas que las especificadas en la variable *ALLOWED_IPS* del *cliente de administración*.

- ***Microservicio de gestión de peticiones***

  - ***PORT:*** Puerto de escucha de este microservicio, que atiende las peticiones relacionadas con peticiones de renderizado del *cliente estándar* y las notificaciones antes mencionados del *microservicio de administración*. Deberá coincidir con el puerto que se especifique en las variables *REQUEST_HANDLING_MICROSERVICE_URL* de los dos componentes mencionados.
  - ***RENDER_SERVER_PORT:*** Funciona de la misma forma que la variable *RENDER_SERVER_PORT* del *microservicio de administración* y debe coincidir con el valor especificado en esta. Se trata del puerto que utilizará el microservicio para contactar con los *servidores de renderizado*.
  - ***ADMINISTRATION_MICROSERVICE_URL:*** Al igual que el *microservicio de administración* necesita contactar con el *microservicio de gestión de peticiones*, este último también necesita iniciar una comunicación con el primero cada vez que recibe una imagen de los *servidores de renderizado* cuando finaliza su procesamiento para que esta se encuentre disponible localmente en el caso de que un usuario administrador solicite descargarla. Esta comunicación se lleva a cabo utilizando la *URL* especificada en esta variable, la cual funciona de la misma forma que la variable *REQUEST_HANDLING_MICROSERVICE_URL* del *cliente estándar* y en la mayoría de casos coincidirá con esta, existiendo de nuevo la posibilidad de especificar una dirección *IP* de *loopback* si los dos componentes se ejecutan sobre el mismo *host*.
  - ***CORS_ALLOWED_ORIGINS:*** Esta se configura de la misma forma que la variable *CORS_ALLOWED_ORIGINS* del *microservicio de administración*, solo que idealmente se especificará una URL por cada instancia del componente *cliente estándar* desplegada que sea accesible directamente desde el navegador.

- ***Servidor de renderizado***

  - ***PORT:*** El puerto de escucha mediante el cual el servidor recibe las peticiones del *microservicio de gestión de peticiones* y del *microservicio de administración*. Su valor deberá coincidir con el de las variables *RENDER_SERVER_PORT* de estos dos componentes y deberá especificarse el mismo valor para todas las instancias que se vayan a desplegar.

- ***ALLOWED_IPS:*** De nuevo, con el objetivo de contribuir a la seguridad del sistema, es posible restringir las direcciones *IP* de las que puedan provenir las peticiones recibidas por los *servidores de renderizado*. Idealmente, estas se corresponderán con las direcciones de los *hosts* donde se hayan despejado instancias del *microservicio de administración* y del *microservicio de gestión de peticiones*.

- ***Cliente CLI***

  - ***REQUEST_HANDLING_MICROSERVICE_URL:*** Esta variable funciona exactamente igual que la variable *REQUEST_HANDLING_MICROSERVICE_URL* del *cliente estándar*, ya que se utilizan para lo mismo.

A.11. Posible configuración de despliegue

El sistema cuenta con un número de componentes considerable, lo que puede resultar en que la elección de una configuración de despliegue resulte complicada. Existen diversas formas de desplegar el sistema según el *host* en el que se desee ubicar cada componente y si se opta por la utilización de contenedores o no.

Es esencial tener en cuenta que, para acceder a todas las funcionalidades del sistema, será necesario desplegar todos los componentes, excepto el *cliente CLI*. Este último no aporta ninguna funcionalidad adicional a la que ya ofrece el *cliente estándar* y, por lo tanto, su uso es opcional. Esto no es de extrañar, ya que fue desarrollado con el propósito de facilitar la realización de pruebas en lugar de formar parte integral del sistema.

A continuación se presenta una posible configuración de despliegue del sistema completo para facilitar este aspecto. Esta debe considerarse como una guía, ya que la elección dependerá de las necesidades que se tengan y de los recursos disponibles.

En este caso, solo se requieren dos *hosts*:

- El primer *host* se utilizará exclusivamente para desplegar el componente *servidor de renderizado*. Esto permitirá maximizar los recursos del equipo de los que dispondrá *Blender* al reducir la cantidad de procesos con los que tiene que compartirlos durante el proceso de renderizado.

- El segundo se usará para desplegar el resto de componentes, excepto el *cliente CLI*. Dado que este último es una aplicación de consola que no se proporciona desde un servidor, debe instalarse directamente en la máquina del usuario. Además, como se ha mencionado anteriormente, no es necesario para el funcionamiento del sistema al completo.

Al contar con ambos microservicios (*microservicio de administración* y *microservicio de gestión de peticiones*) en el mismo *host*, las comunicaciones y transferencias de archivos entre ellos se facilitarán. Sin embargo, esta configuración implica que compartirán recursos, lo que podría ser un inconveniente a considerar.

Con respecto a los dos clientes, su ubicación de despliegue no tiene mucha importancia, ya que las comunicaciones se realizarán con la máquina del usuario después de haberle servido el contenido estos.

A.12. Despliegue directamente sobre el *host*

En primer lugar, para especificar las variables de entorno, bastará con copiar los contenidos de las plantillas en un archivo *.env* que se añadirá al directorio raíz de cada componente y modificarlas con los valores deseados (Figura A.67).

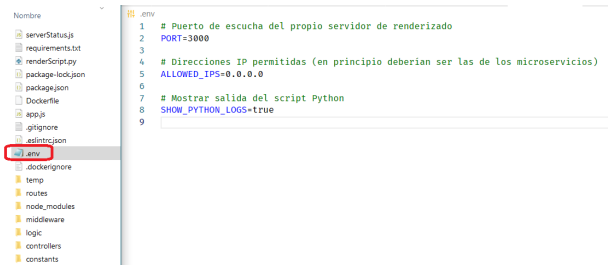

Figura A.67: Fichero *.env* para el *servidor de renderizado*

Una vez realizadas todas las instalaciones y configuraciones anteriores, será posible desplegar los distintos componentes directamente sobre el *host* sin problemas. Este proceso funcionará de la misma forma para todos, ofreciéndose dos opciones:

- La primera consiste en abrir una consola en el directorio raíz del componente que se desee desplegar e introducir el comando `npm run start`. Tras esto, se podrá observar la salida del programa a través del terminal, la cual informará de posibles problemas (Figura A.68).

Figura A.68: Ejemplo de despliegue de una instancia del componente *microservicio de administración* mediante la introducción del comando `npm run start`

- También es posible realizar esta misma secuencia de pasos, pero de forma automatizada, ejecutando el *script* correspondiente al despliegue del componente. Para acceder a estos, partiendo del directorio raíz del repositorio, se navegará a la ruta `/scripts/<sistema operativo>/host` (Figura A.69).

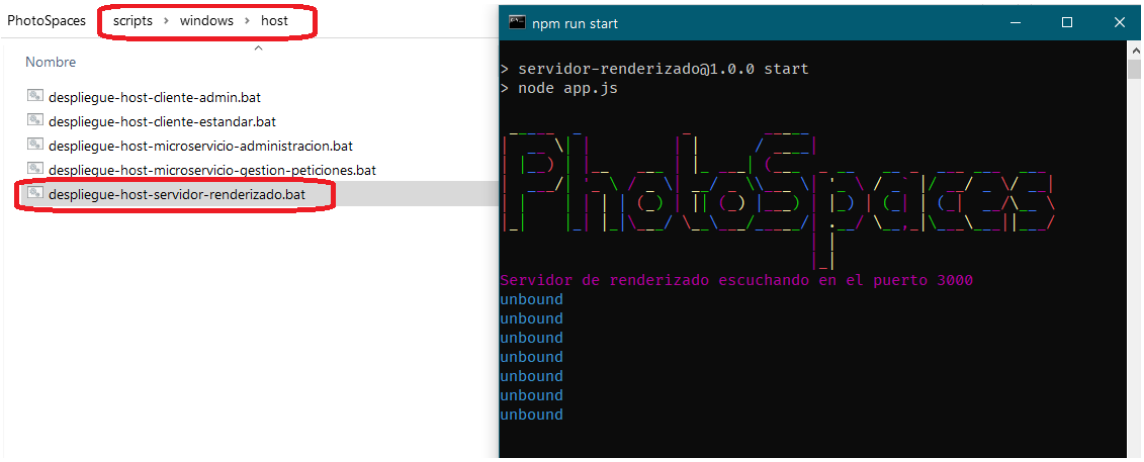

Figura A.69: Ejemplo de despliegue de una instancia del componente *servidor de renderizado* utilizando su *script* correspondiente

A.13. Despliegue con *Docker*

Si se opta por realizar el despliegue en contenedores utilizando *Docker*, se ofrecen dos opciones principales: desplegar cada componente por separado, o utilizar *Docker Compose* para una puesta en marcha de forma conjunta.

Una ventaja de estas es que no requieren de la instalación manual ni de *Node.js* ni de *Blender*, ya que estos se instalan de forma automática al generar las imágenes de los contenedores cuando son necesarios. Sin embargo, seguirá siendo necesario configurar el correo y la base de datos.

Conviene tener en cuenta que también es posible combinar componentes desplegados directamente sobre *hosts* con componentes ejecutándose sobre contenedores.

A.13.1. Despliegue de los contenedores *Docker* de forma independiente

Se recurrirá a esta opción cuando requiramos de una configuración de despliegue más flexible, ya que permitirá desplegar solo un componente donde se desee.

En primer lugar será necesario configurar las variables de entorno. En este caso, estas se especifican en el archivo *Dockerfile* situado en el directorio raíz del componente (en las líneas que comienzan por la palabra reservada *ENV*) (Figura A.70).

```
3   ARG MONGODB_CONNECTION_STRING
4   ARG EMAIL_USER
5   ARG EMAIL_PASSWORD
6
7   ENV MONGODB_CONNECTION_STRING=${MONGODB_CONNECTION_STRING}
8   ENV PORT=9001
9   ENV RENDER_SERVER_PORT=3000
10  ENV EMAIL_USER=${EMAIL_USER}
11  ENV EMAIL_PASSWORD=${EMAIL_PASSWORD}
12  ENV DB_CHECK_PERIOD_MS=60000
13  ENV CLEANUP_INTERVAL_MS=36000000
14  ENV EMAIL_SENDING_BACKUP_INTERVAL_MS=60000
15  ENV RENDERING_SERVER_POLLING_INTERVAL_MS=1000
```

Figura A.70: Variables de entorno en el *Dockerfile* correspondiente al *microservicio de gestión de peticiones*

Hay que tener en cuenta que algunas variables de entorno son delicadas (credenciales del correo y cadenas de conexión a la base de datos) y es mejor tratar de evitar escribirlas directamente en el *Dockerfile*, por lo que no se modificará el contenido de estas y se pasarán los valores a través de argumentos, a continuación se verá que esto se logra utilizando el argumento `--build-arg` en el comando encargado de la generación de la imagen.

Se podrá llevar a cabo este despliegue de dos formas distintas, iniciando en primer lugar *Docker* para que funcionen los comandos correspondientes:

- La primera consiste en navegar al directorio raíz de cada componente y generar la imagen que utilizará el contenedor con el comando `docker build -t <nombre para la imagen>:latest .` para los contenedores que no requieran secretos para su configuración. En el caso del *microservicio de administración*, se utilizará el comando `docker build --build-arg MONGODB_CONNECTION_STRING=<cadena de conexión> -t <nombre para la imagen>:latest .` y, en el caso del *microservicio de gestión de peticiones*, el comando `docker build --build-arg MONGODB_CONNECTION_STRING=<cadena de conexión> --build-arg EMAIL_USER=<dirección del correo electrónico> --build-arg EMAIL_PASSWORD=<contraseña de aplicación del correo electrónico> -t <nombre para la imagen>:latest ..`

  Tras esto, bastaría con arrancar el contenedor a partir de la imagen generada con el comando `docker run -t --name <nombre del contenedor> --rm <nombre de la imagen generada>:latest`.

- La segunda se basa en realizar exactamente los mismos pasos, pero a través de los *scripts* definidos en el repositorio. Los que se encargan de la generación de la imagen se encuentran en el directorio `/scripts/<sistema operativo>/docker/generacion-imagen`. De nuevo, en el caso del *microservicio de administración* y del *microservicio de gestión de peticiones*, será necesario editarlos para incluir los secretos en los comandos (Figura A.71).

  Los que se encargan de poner en marcha el contenedor se encuentran en la ruta `/scripts/<sistema operativo>/docker/despliegue-contenedor`.

```
1  @echo off
2  cd ../../../../microservicio-gestion-peticiones
3
4  docker build --build-arg MONGODB_CONNECTION_STRING=<nuestra cadena de conexión> ^
5  --build-arg EMAIL_USER=<dirección del correo electrónico> ^
6  --build-arg EMAIL_PASSWORD=<contraseña de aplicación del correo electrónico> ^
7  -t microservicio-gestion-peticiones:latest .
```

Figura A.71: Ejemplo del *script* que genera la imagen para el *microservicio de gestión de peticiones* en *Windows*

A.13.2. Despliegue con *Docker Compose*

Esta opción permite desplegar todos los componentes sobre contenedores en un mismo *host* con un solo comando.

Primero será necesario configurar las variables de entorno, en este caso se encuentran todas en el archivo *docker-compose.yaml* del directorio raíz del repositorio (Figura A.72). Para evitar conflictos, se recomienda comentar las variables de entorno de los ficheros *Dockerfile*.

```
37  microservicio-administracion:
38    tty: true
39    build:
40      context: ./microservicio-administracion
41    container_name: microservicio-administracion
42    ports:
43      - 9000:9000
44  environment:
45      - DOCKER_CONTAINER_EXECUTION=true
46      - MONGODB_CONNECTION_STRING=${ADMINISTRATION_MICROSERVICE_MONGODB_CONNECTION_STRING}
47      - PORT=9000
48      - RENDER_SERVER_PORT=3000
49      - REQUEST_HANDLING_MICROSERVICE_HOST=microservicio-gestion-peticiones
50      - REQUEST_HANDLING_MICROSERVICE_PORT=9001
51      - ALLOWED_IPS=0.0.0.0,::1
```

Figura A.72: Variables de entorno del *microservicio de administración* en el fichero *docker-compose.yaml*

Para los secretos, a diferencia de los archivos *Dockerfile*, *Docker Compose* soporta la declaración de variables de entorno (no las que se están definiendo para configurar la aplicación, sino otras que se puedan usar de forma dinámica en el fichero declarativo), por lo que se aprovechará esta funcionalidad para evitar declarar estos valores sensibles directamente. De esta forma, se incluirá un fichero *.env* en el directorio raíz del repositorio que los especifique. *Docker Compose* se encargará de cargarlos a partir de este en el archivo *docker-compose.yaml* (Figura A.73).

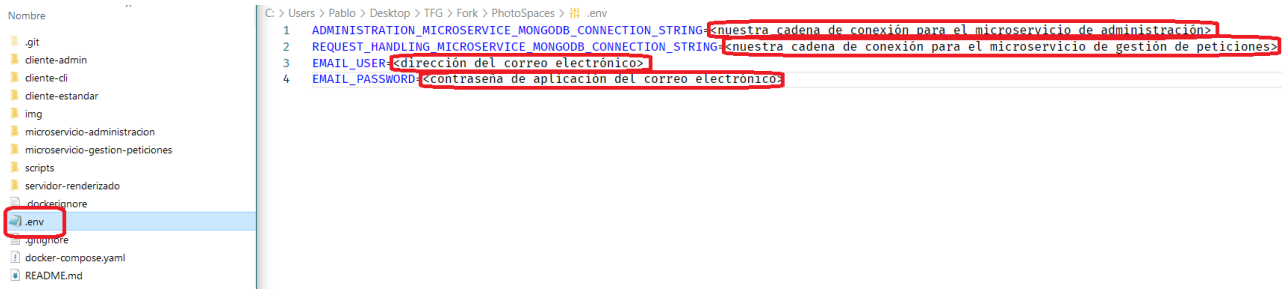

Figura A.73: Fichero *.env* del que obtendrá los secretos el fichero *docker-compose.yaml*

Una vez configuradas las variables de entorno, de nuevo se presentan dos opciones partiendo de que *Docker* ya haya sido inicializado:

- Introducir en el directorio raíz del repositorio el comando `docker-compose up`
- Utilizar el *script* que se encuentra en el directorio `/scripts/<sistema operativo>/docker/docker-compose`.

Independientemente del método que se seleccione, si todo va bien, se desplegarán todos los componentes y se mostrarán en el terminal las salidas que vayan imprimiendo (Figura A.74).

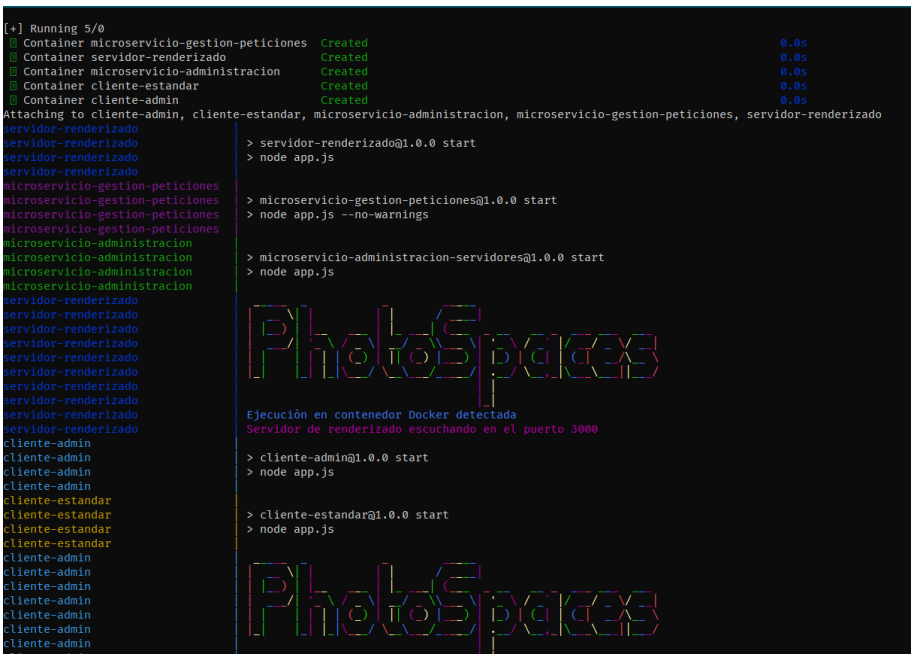

Figura A.74: Salida de *Docker Compose* tras desplegar todos los componentes

A.14. Comprobación del correcto despliegue del sistema

Por último, una vez arrancado el sistema, es posible realizar algunas pruebas simples para verificar que el despliegue se ha configurado correctamente. Estas pruebas tienen como objetivo ser sencillas y rápidas para probar la configuración de despliegue, sin abarcar todas las funcionalidades en detalle, para lo cual se requeriría un enfoque más exhaustivo.

Las pruebas consistirán en seguir los siguientes pasos, que se explicarán de manera general. Para obtener instrucciones detalladas sobre cómo realizar cada acción, es posible consultar consultar el manual de usuario (B).

Para probar las funcionalidades de administración, se podrán realizar las siguientes acciones:

1. Acceder al panel de administración utilizando el nombre de dominio o la combinación de dirección *IP* y puerto configurados para el *cliente de administración*. Verificar que el panel esté vacío y asegurarse de que no se generen alertas en la página que indiquen errores.

2. Añadir un servidor de renderizado mediante el formulario ubicado en la esquina inferior izquierda, donde se ingresará la dirección *IP* del servidor configurado y desplegado.

3. Verificar que el servidor de renderizado se haya vinculado correctamente. Esto asegurará que el *cliente de administración*, el *microservicio de administración* y el *servidor de renderizado* han sido desplegados correctamente. Además, se pueden realizar pruebas adicionales deshabilitando y rehabilitando el servidor añadido.

Para comprobar el correcto funcionamiento de las funcionalidades orientadas a los usuarios finales de la aplicación, se pueden seguir los siguientes pasos:

1. Acceder a la interfaz web estándar de *PhotoSpaces* mediante el nombre de dominio o la combinación de dirección *IP* y puerto establecidos para el *cliente estándar*.

2. Cargar una escena, configurar los parámetros de la barra lateral y hacer *click* en el botón *Renderizar* para generar la petición de renderizado.

En el caso de no disponer de ficheros válidos para cargarlos en el visor, es posible descargar escenas de ejemplo a través de esta carpeta compartida.

3. Verificar que la imagen renderizada es recibida correctamente. Esto asegurará que el *cliente estándar*, el *microservicio de gestión de peticiones* y el *servidor de renderizado* han sido desplegados satisfactoriamente. Se recomienda probar ambos métodos de obtención de la imagen (síncrono mediante descarga en el navegador y asíncrono mediante el envío de un correo electrónico).

Tras seguir estas instrucciones, será posible tener una primera evaluación del correcto funcionamiento del sistema después del despliegue.

Apéndice B
Manual de Usuario

En este anexo se incluyen los detalles y explicaciones necesarias para facilitar el uso de la aplicación *PhotoSpaces* de manera satisfactoria. Se tratan tanto las funcionalidades originales que ofrece el cliente estándar a los usuarios, como las que son accesibles desde el nuevo cliente basado en una interfaz de línea de comandos y desde el nuevo panel de administración.

B.1. Cliente estándar

El acceso a esta interfaz web consiste en la introducción de la *URL* correspondiente al despliegue realizado del componente *cliente estándar* en un navegador. Al recibir la petición, el servidor devolverá los recursos necesarios para contar con una instancia del cliente funcionando de manera local.

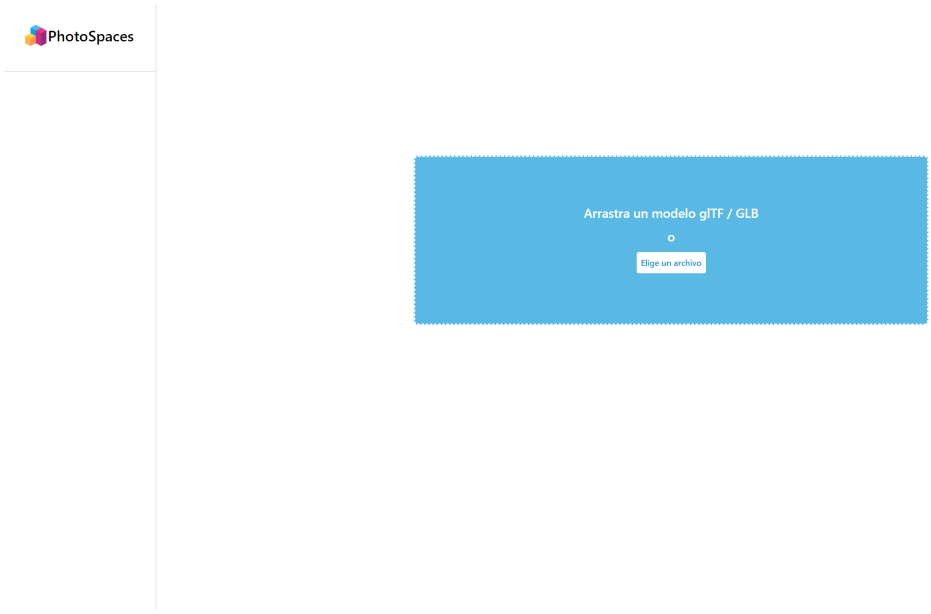

Figura B.1: Selección del fichero a cargar en el visor

Al acceder a la aplicación, lo primero que se puede apreciar es un selector de fichero que solicitará un archivo con extensión *.gltf* o *.glb* para cargar una escena tridimensional y poder

utilizar el visor 3D sobre esta (Figura B.1). Se puede optar por hacer *click* en *Elige un archivo* y seleccionar uno desde el explorador de archivos, que solo mostrará ficheros válidos, o arrastrar directamente uno, comprobando que tiene una de las extensiones solicitadas, ya que si no, el sistema notificará del error y no permitirá continuar.

Existe la posibilidad de descargar escenas de ejemplo utilizando esta carpeta compartida en el caso de que no se contara con ningún fichero de este tipo en el sistema.

Una vez seleccionado el fichero, la aplicación carga la escena correspondiente en el visor. Dependiendo de su complejidad y de las especificaciones del equipo utilizado, este proceso puede tardar más o menos. Mientras, la aplicación indicará que está cargando la escena. Una vez cargada, esta aparecerá en el visor de la aplicación. Se mostrará una interfaz formada por el visor y una barra lateral a la izquierda (Figura B.2). A continuación, se explican todas las funcionalidades que esta ofrece.

Figura B.2: Interfaz de *PhotoSpaces* una vez se ha cargado una escena

B.1.1. Acciones sobre la escena

En primer lugar, se tratan las principales acciones que es posible realizar sobre la escena.

- **Desplazamiento:** Si se hace *click* izquierdo sobre el visor, se accederá a los controles del visor, los cuales permiten navegar por la escena 3D. Los controles son los siguientes, aunque se pueden consultar en la propia interfaz (Figura B.3):

Figura B.3: Controles del visor

- **Desplazar cursor** para establecer hacia dónde mira la cámara.
- **Tecla W** para desplazar la cámara hacia delante.
- **Tecla S** para desplazar la cámara hacia atrás.
- **Tecla A** para desplazar la cámara hacia la izquierda.
- **Tecla D** para desplazar la cámara hacia la derecha.
- **Barra espaciadora** para para desplazar la cámara hacia arriba.
- **Shift izquierdo** para desplazar la cámara hacia abajo.
- **Desplazamiento de la rueda del ratón** para modificar la velocidad de desplazamiento (hacia arriba para aumentarla y hacia abajo para disminuirla)

Para salir de este modo, bastará con volver a hacer *click izquierdo* sobre el visor.

- **Menú de objetos:** Si se hace *click* derecho sobre el visor, aparecerá un pequeño menú que permitirá añadir fuentes de luz a la escena. Será posible seleccionar entre luces de punto y luces direccionales. Una vez añadidas, en la esquina superior derecha del visor aparecerá un menú por cada una desde el que podrán ser controladas modificando sus parámetros. También se añadirán a una lista en la parte más baja de la barra de opciones, desde la cual podrán ser eliminarlas haciendo *click* en las papeleras (Figura B.4).

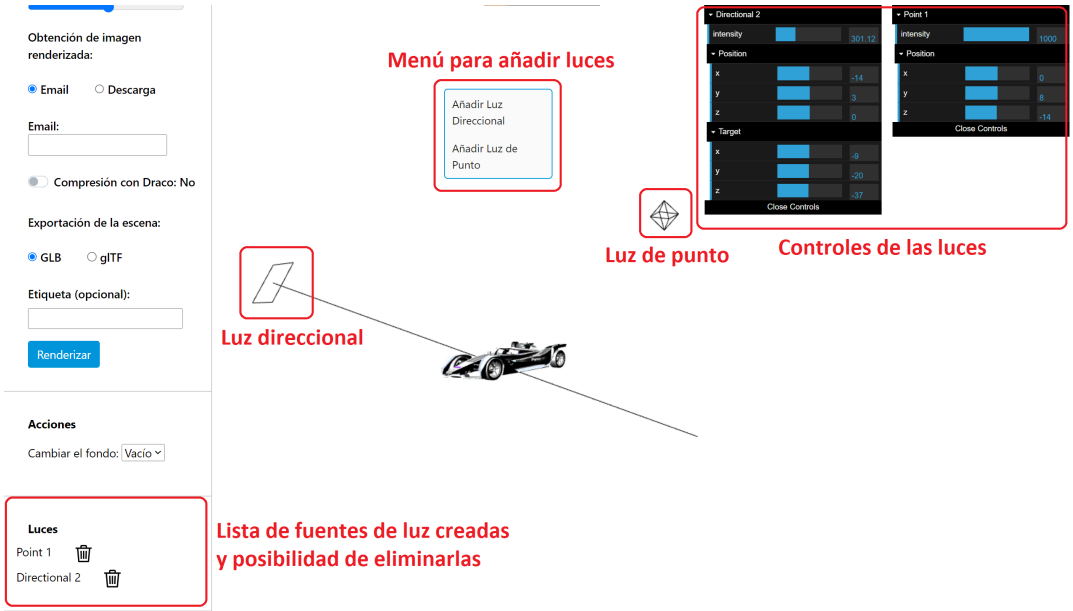

Figura B.4: Trabajar con luces en el visor

- **Limpiar visor:** Una vez se haya terminado de trabajar con una escena y se desee cargar otra, se podrá pulsar el botón *Limpiar* para volver a la pantalla inicial de la aplicación, la cual solicitará de nuevo un fichero para cargar otra escena (Figura B.1).

- **Cambiar fondo:** Desde el apartado *Acciones* de la barra lateral, es posible seleccionar una imagen de las predeterminadas que incluye la aplicación para que se muestre como fondo de la escena. Esto solo afectará al visor, si se solicita un renderizado, la imagen resultante no lo incluirá.

Figura B.5: Añadir un fondo a la escena

B.1.2. Generación de peticiones de renderizado

La principal funcionalidad ofrecida consiste en la obtención de una imagen en formato *PNG* con la escena renderizada. Para esto, será preciso generar primero la petición de renderizado correspondiente.

En primer lugar, se situará la cámara de forma que el visor muestre todo lo que se desee que aparezca en la imagen y se indicarán una serie de parámetros en la barra lateral. Estos serían los siguientes:

- **Resolución:** Las dimensiones (en píxeles) que se desee que tenga la imagen renderizada. Se ofrecen las principales resoluciones con formato 16:9. A mayor resolución, mayor será el tiempo que tardará el proceso de renderizado (Figura B.6).

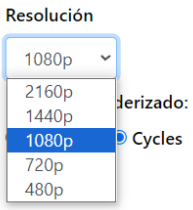

Figura B.6: Selección de la resolución en la barra lateral

- **Motor de renderizado:** Permite seleccionar cuál de los dos motores de *Blender* se utilizará, *Cycles* o *Eevee* (Figura B.8).

Eevee utiliza rasterización para ofrecer menores tiempos de espera, pero también una fidelidad menor con respecto a la realidad. En cambio, *Cycles* utiliza trazado de rayos y ofrece resultados de mayor calidad y realismo, pero el proceso de renderizado requiere más tiempo.

Figura B.7: Selección del campo de visión de la cámara en la barra lateral

Si se selecciona el motor *Eevee*, se podrán especificar parámetros de renderizado adicionales, estos son:

- **Oclusión ambiental:** Aplicar oclusión ambiental, una técnica que permite simular cómo la luz se acumula o es bloqueada en las áreas más oscuras y cerradas de la escena, contribuyendo al realismo.
- **Resplandor:** Aplicar resplandor o *bloom*, un efecto que añade un resplandor suave y brillante alrededor de las fuentes de luz intensas, como luces, reflejos o destellos, lo que crea una sensación de brillo y realismo en la imagen resultante.
- ***SSGI*:** Aplicar *SSGI* (*Super Sampling Global Illumination*), una técnica que permite una aproximación de la iluminación global en tiempo real, mejorando la calidad de la imagen prescindiendo del rendimiento computacional que implicaría un enfoque de iluminación global más preciso y detallado.

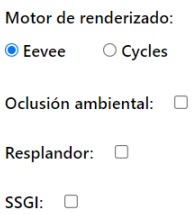

Figura B.8: Selección del motor de renderizado en la barra lateral

- **FOV:** Corresponde con el campo de visión de la cámara, es decir, el área total que puede observar la cámara en una sola toma (Figura B.7).

- **Método de obtención de la imagen renderizada:** Cómo se desea obtener la imagen una vez haya sido procesada la petición generada. Se podrá elegir entre recibirla de forma síncrona y descargarla en el navegador, o recibirla por correo electrónico de forma asíncrona. En caso de que se elija el envío por correo, será necesario especificar la dirección donde se quiera recibir la imagen renderizada (Figura B.9).

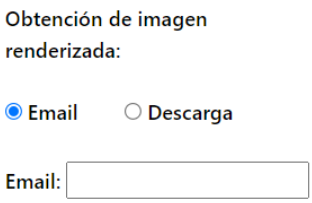

Figura B.9: Selección del método de obtención de la imagen renderizada en la barra lateral

- **Compresión *Draco*.** Opcionalmente, se podrá solicitar que se le aplique compresión *Draco* a la escena. Esta solo se recomienda si el tamaño de la escena es excepcionalmente grande, ya que ayuda a reducirlo a cambio de perder un poco de calidad. En caso de que se indique, será necesario establecer qué nivel de compresión se desea aplicar, teniendo en cuenta que el nivel 0 ya aplica compresión, y que, a mayor nivel, mayor reducción de tamaño y mayor pérdida de calidad (Figura B.10).

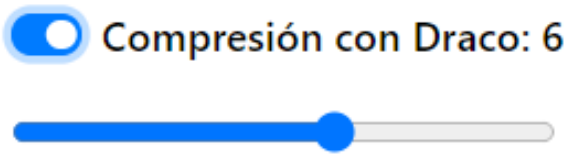

Figura B.10: Opciones de compresión *Draco* en la barra lateral

- **Exportación de la escena:** Es posible indicar qué tipo de fichero (*glTF* o *GLB*) se generará a partir de la escena del visor, que será el que maneje el sistema. Aquí se recomienda especificar *GLB*, ya que supone un tamaño de archivo ligeramente menor. Esta opción solo se ofrece por consistencia con el resto de métodos para generar una petición, los cuales permiten los dos tipos de archivo (Figura B.11).

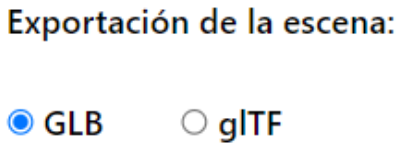

Figura B.11: Opciones de exportación de la escena 3D en la barra lateral

- **Etiqueta opcional para la petición:** Opcionalmente, será posible especificar una cadena de texto que se asociará a la petición de renderizado generada, cuyo único fin es facilitar su identificación en el sistema (Figura B.12).

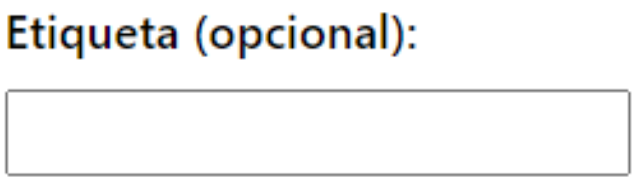

Figura B.12: Posibilidad de introducir una etiqueta para la petición en la barra lateral

Una vez configurados todos estos parámetros, para generar la petición de renderizado se hará *click* en el botón *Renderizar* en la parte baja de la barra lateral.

Si se ha indicado que se desea recibir la imagen renderizada por correo electrónico, la aplicación avisará de que la petición ha sido recibida y que se enviará a la dirección de correo especificada cuando esté lista (Figura B.13). Tras esto, no se requerirá ninguna acción adicional, por lo que sería posible cerrar la aplicación o empezar a trabajar con otra escena.

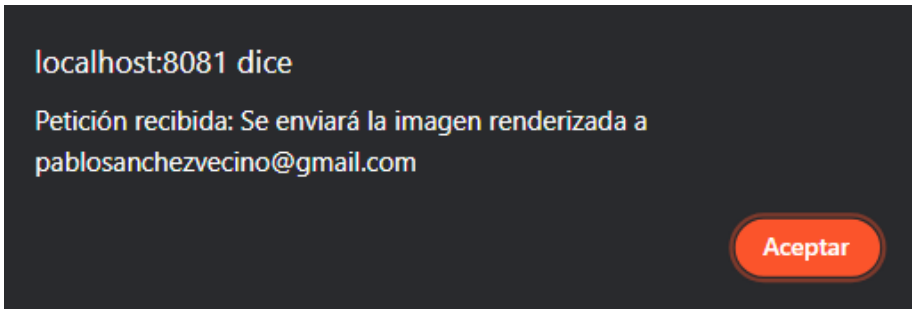

Figura B.13: Respuesta del sistema tras recibir petición con envío por correo electrónico

En cambio, si se ha especificado una recepción síncrona de la imagen en el navegador, aparecerá un modal de carga que mostrará la siguiente información (Figura B.14):

- **Estado de la petición:** Si la petición se encuentra encolada o ya ha sido enviada a un servidor de renderizado. En caso de que se encuentre encolada, se mostrará su posición en la cola.

- **Tiempo restante estimado:** Si la petición ya ha sido enviada a un servidor, el tiempo restante estimado del proceso de renderizado. Es necesario tener en cuenta que este no estará disponible si el renderizado se está realizando con el motor *Eevee*. Si la petición se encuentra encolada, se mostrará el tiempo restante estimado para su avance en la cola en el caso de que sea posible obtenerlo.

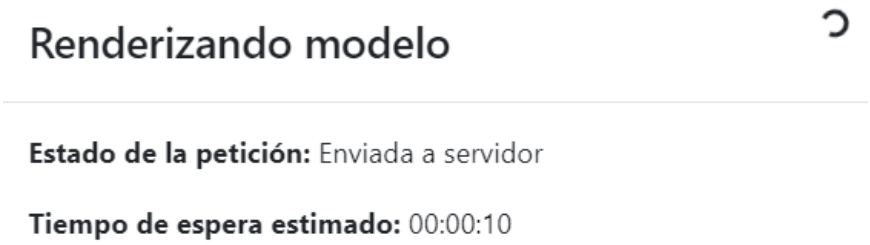

Figura B.14: Modal con información sobre el estado de la petición que se muestra tras enviar la petición de renderizado

Una vez esté lista la imagen, en la barra lateral aparecerá un botón *Descargar imagen* (Figura B.15). Si se hace *click* en este, se descargará esta en el navegador (Figura B.16).

Figura B.15: Modal con información sobre el estado de la petición que se muestra tras enviar la petición de renderizado

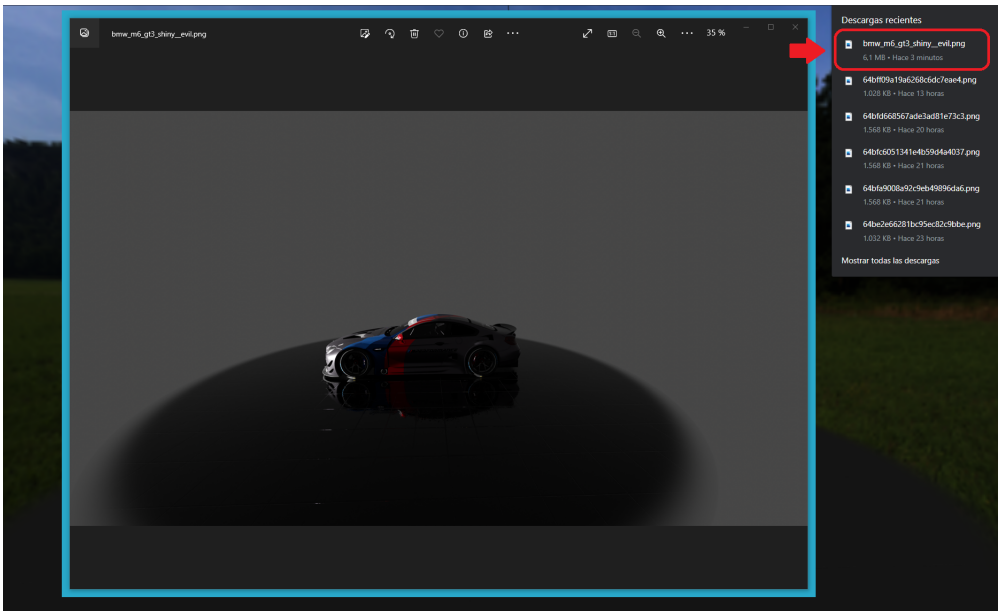

Figura B.16: Imagen renderizada descargada

B.2. Cliente CLI

El cliente basado en una interfaz de línea de comandos permite generar peticiones de renderizado. La diferencia con el *cliente estándar* reside en que no se tendrá acceso al visor ni a la barra lateral para configurar la escena a enviar, por lo que los distintos parámetros serán especificados en forma de entradas introducidas por consola.

Para poder acceder a este, el equipo utilizado debe cumplir los siguientes requisitos:

- Contar con una instalación de *Node.js* (detallado en la sección A.2).
- Disponer de una copia local del código fuente del componente (detallado en la sección A.7).
- Haber realizado en este la configuración con variables de entorno de forma correcta (detallado en la sección A.9).
- Haber iniciado una ejecución del programa (detallado en la sección A.12).

Tras iniciar el programa, este solicitará una cadena de texto con los parámetros de renderizado en formato *JSON*. Es posible utilizar la siguiente como plantilla. Es necesario cuidar los saltos de línea a la hora de introducirla en la consola:

```
{
```

```
    "resolution":"1080p",
    "lens":12.326339984789739,
    "clip_start":0.1,
    "clip_end":10000,
    "location":{"x":-0.0010785156548955774,"y
  ":-5.279649897972276,"z":0},
    "qua":{"_x":0.6941158209388606,"_y":-0.03473474149027019,"
  _z":-0.0359362643382868,"_w":0.7181262491800342},
    "engine":"CYCLES",
    "gtao":false,
    "bloom":false,
    "ssr":false
}
```

Una vez introducidos, el programa se encargará de validarlos y notificará si se detectara algún valor incorrecto. En el caso de que todo vaya bien, mostrará los valores que ha leído (Figura B.17).

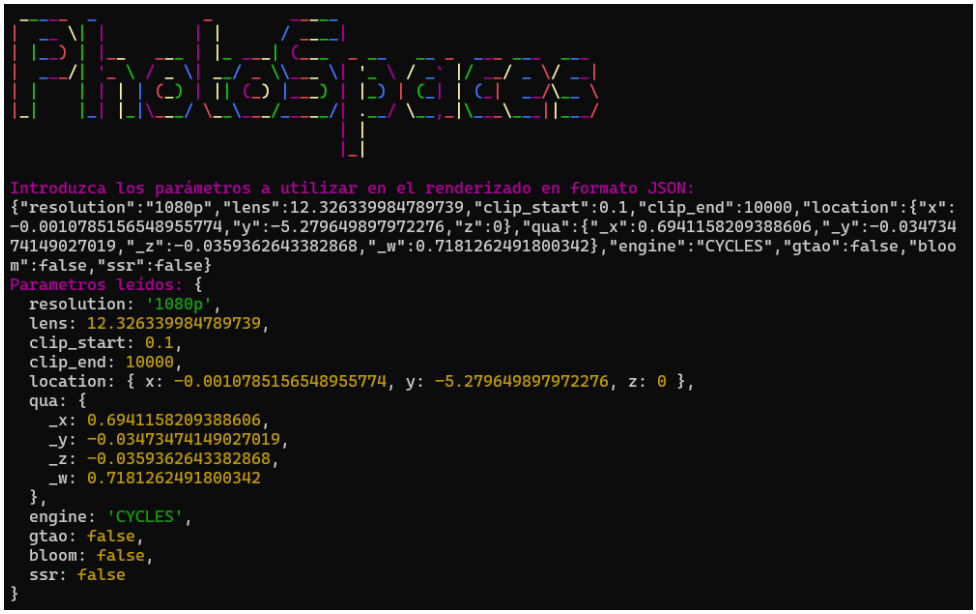

Figura B.17: Especificación textual de los parámetros de renderizado

A continuación, solicitará la ruta donde se encuentra el fichero con la escena 3D que se enviará junto a a la petición. Al igual que con el *cliente estándar*, en esta carpeta compartida se incluyen algunas escenas de ejemplo en el caso de que resultaran necesarias porque no se disponga de ninguna en la máquina.

Será necesario especificarle la ruta correspondiente al fichero *.gltf* o *.glb* que se vaya a utilizar. Si el programa no logra localizarlo, solicitará la ruta de nuevo. Si lo encuentra, lo indicará y avanzará al siguiente paso (Figura B.18).

También es posible indicar la ruta a un fichero *.txt* si se desea probar la optimización basada en este tipo de ficheros parcialmente implementada, teniendo en cuenta que esta no constituye una funcionalidad adicional del sistema, ya que cuenta con numerosas limitaciones en su estado actual.

```
Introduzca ruta del fichero con la escena 3D:
C:\Users\Pablo\Desktop\models\bmw_m6_gt3_shiny__evil.glb
Fichero localizado
```

Figura B.18: Especificación textual del fichero con la escena 3D

Tras esto, se pasará a la especificación de la compresión con *Draco*. Si se desea aplicar, se indicará el nivel correspondiente (del 0 al 10). Si se opta por prescindir de esta, se pulsará *Enter* para continuar. Si se introduce un valor no válido, el programa avisará y volverá a solicitar una entrada, en caso contrario, seguirá adelante (Figura B.19).

```
Si desea utilizar la compresión con Draco, introduzca el nivel de compresión
 a aplicar (0-10).
Si no desea aplicar compresión, no introduzca nada y pulse Enter.

No se aplicará compresión con Draco
```

Figura B.19: Especificación textual de la compresión con *Draco*

Por último, será necesario especificar el método de obtención de la imagen. Para recibirla de forma asíncrona por correo electrónico se indicará la dirección correspondiente (Figura B.20). Si se opta por recibir la imagen de manera síncrona en el directorio *out*, simplemente se pulsará *Enter* .

```
Indique la dirección de correo electrónico al que enviar la imagen renderizada.
Para recepción síncrona en la carpeta out, no escriba nada y presione la tecla E
nter
pablosanchezvecino@gmail.com
La imagen renderizada se enviará a pablosanchezvecino@gmail.com
```

Figura B.20: Especificación textual de recepción asíncrona por correo electrónico

Si se solicita envío por correo, el programa avisará cuando reciba la petición, indicando que se enviará la imagen a la dirección especificada (Figura B.21).

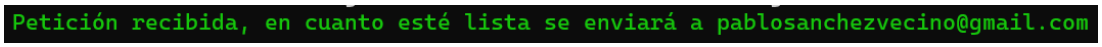

Figura B.21: Notificación de recepción de petición con recepción asíncrona

Si se indica una recepción síncrona, el programa comenzará a monitorizar el estado de la petición, mostrándolo por consola hasta que se reciba la imagen renderizada (Figuras B.22 y B.23).

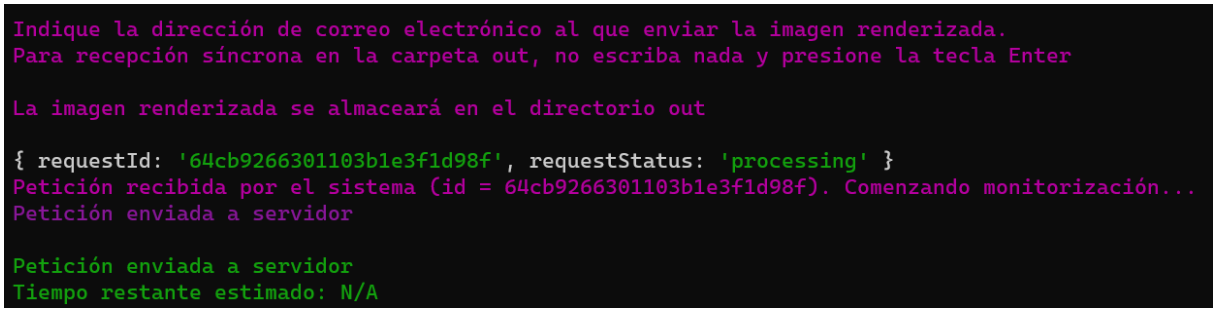

Figura B.22: Comienzo de la monitorización

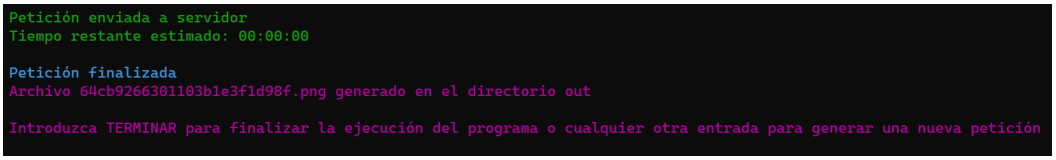

Figura B.23: Recepción de la imagen síncrona

Cuando se reciba la petición (en el caso de recepción asíncrona) o cuando se genere la imagen en el directorio *out* (en el caso de recepción síncrona), será posible introducir la cadena *"TERMINAR"* para finalizar la ejecución, o cualquier otra entrada para comenzar a generar una nueva petición (Figura B.24).

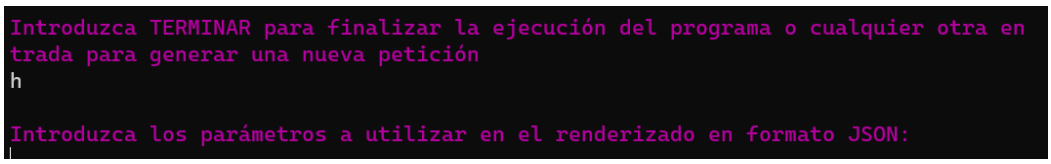

Figura B.24: Finalización de programa o generación de una nueva petición

B.3. Panel de administración

Para acceder al panel de administración se indicará en un navegador la *URL* correspondiente al despliegue configurado para el *cliente de administración*.

Una vez se tenga acceso a este, se mostrará una interfaz de usuario basada en tarjetas dividida en cuatro secciones. Las dos de arriba permiten la monitorización y la interacción con los servidores de renderizado (izquierda) y las peticiones de renderizado (derecha). Cada una está formada por tres contenedores encargados de mostrar las tarjetas correspondientes a los servidores y peticiones registrados en el sistema organizados por su estado.

Para los servidores, estos estados serían:

- **Disponible:** El servidor no se encuentra procesando ninguna petición y será elegible para el reenvío de peticiones si no tiene ninguna otra encolada.
- **Ocupado:** El servidor se encuentra renderizando una petición que le ha sido asignada.
- **Deshabilitado:** El servidor ha sido deshabilitado por un administrador y no será tenido en cuenta por el sistema a la hora de asignar las peticiones hasta que vuelva a ser habilitado.

En el caso de las peticiones, se hace una distinción entre los tres estados siguientes:

- **En proceso:** La petición ha sido asignada a un servidor que se encuentra procesándola en ese momento.
- **Encolada:** La petición no ha podido ser reenviada directamente a ningún servidor de renderizado, por lo que ha tenido que ser encolada a alguno de estos. También se puede dar el caso de que no se pueda encolar a ningún servidor porque no hay ninguno disponible ni ocupado en el momento de su llegada.
- **Finalizada:** La petición ya ha sido procesada y ya se ha obtenido su imagen renderizada correspondiente.

Las dos secciones de abajo se corresponden con el formulario que permite añadir un nuevo servidor al sistema (esquina inferior izquierda) y con las estadísticas generales del sistema (esquina inferior derecha).

En la Figura B.25 es posible observar el estado inicial del panel de administración. En este caso no muestra tarjetas aún y no resulta muy complejo distinguir las cuatro secciones y los seis contenedores de tarjetas.

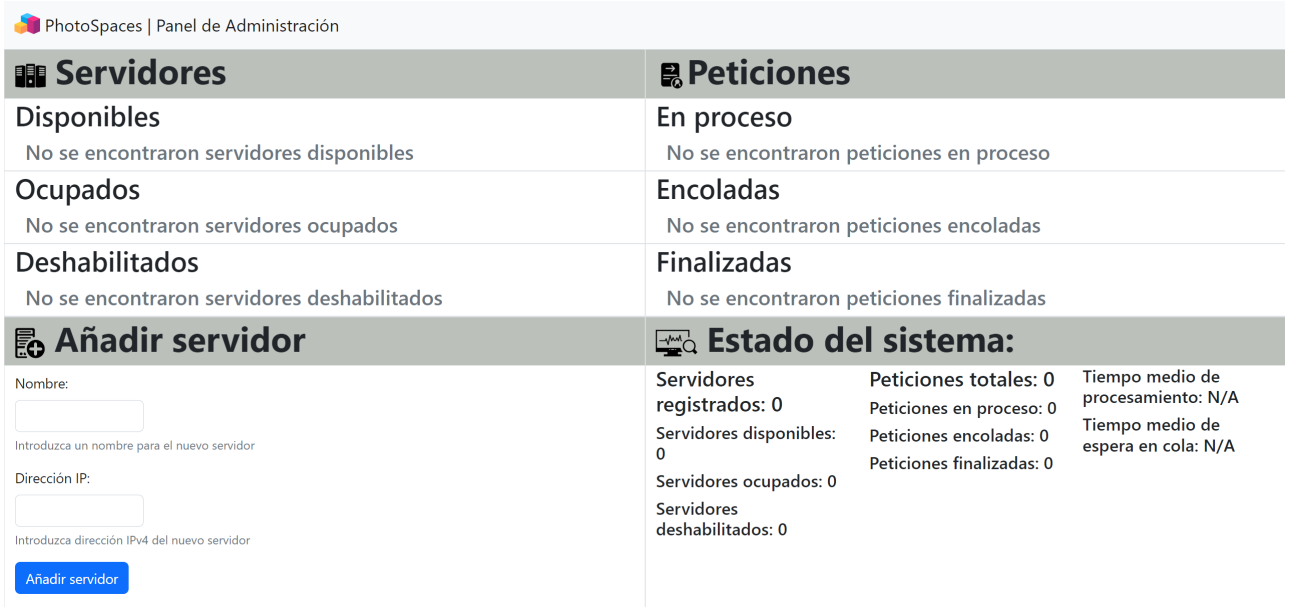

Figura B.25: Panel de administración en su estado inicial

B.3.1. Consulta de información de los servidores de renderizado

Cada tarjeta mostrada en los contenedores situados a la izquierda se corresponde con un servidor de renderizado registrado en el sistema. Estas incluyen bastante información, motivo por el cual, para evitar que ocuparan demasiado espacio, no muestran directamente a qué se corresponde la información de cada línea, sino un icono representativo. Es posible consultar una descripción de la información manteniendo el cursor sobre dicho icono. Independientemente del estado del servidor, la información mostrada será la misma.

A continuación, se detalla a qué corresponde cada atributo. Para facilitar la identificación, en la Figura B.26 se ha numerado cada uno de estos:

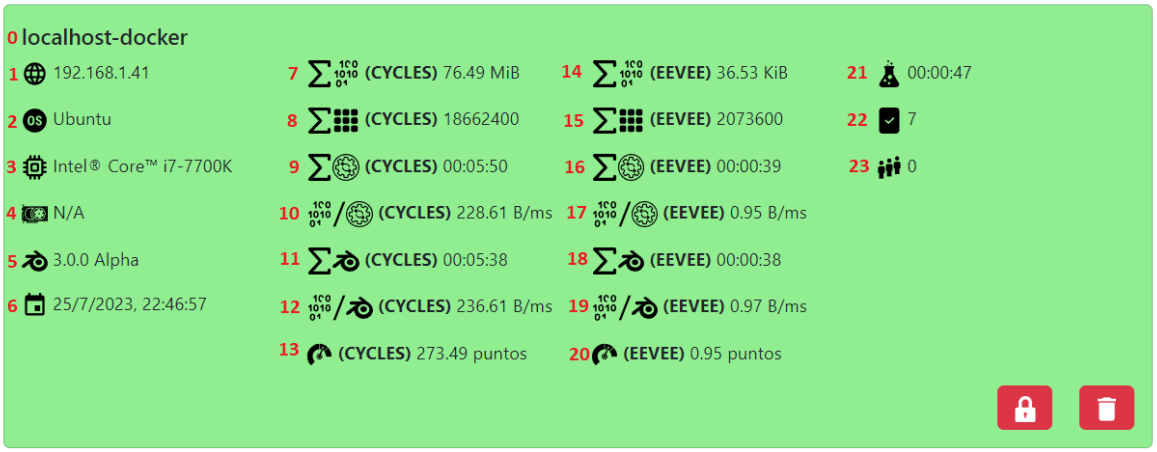

Figura B.26: Tarjeta correspondiente a un servidor de renderizado

- **0.** Nombre que se le asignó al servidor al registrarlo en el sistema.
- **1.** Dirección *IP* del servidor.
- **2.** Sistema operativo del servidor.
- **3.** Procesador del servidor.
- **4.** Tarjeta gráfica del servidor.
- **5.** Versión de *Blender* que utiliza el servidor para renderizar.
- **6.** Fecha y hora en la que se registró el servidor en el sistema.
- **7 y 14.** Total de información procesada por el servidor, es decir, la suma de los tamaños de los ficheros asociados a las peticiones que se han reenviado a este. Se diferencia entre las peticiones que solicitan el motor *Cycles* (7) y las que solicitan *Eevee* (14).
- **8 y 15.** Total de píxeles procesados por el servidor. Se diferencia entre las peticiones que solicitan el motor *Cycles* (8) y las que solicitan *Eevee* (15).
- **9 y 16.** Total de tiempo que ha necesitado el sistema en su conjunto para satisfacer todas las peticiones asignadas al servidor, teniendo en cuenta tanto los tiempos de transferencias como los de renderizado. Se diferencia entre las peticiones que solicitan el motor *Cycles* (9) y las que solicitan *Eevee* (16).
- **10 y 17.** Cantidad media de información procesada por el servidor por milisegundo de tiempo que ha requerido el sistema, es decir, el cociente entre los valores 7 y 9 y el cociente entre los valores 14 y 16. Se diferencia entre las peticiones que solicitan el motor *Cycles* (10) y las que solicitan *Eevee* (17).

- **11 y 18.** Total de tiempo que ha necesitado la instancia de *Blender* del servidor para satisfacer todas las peticiones asignadas. Se diferencia entre las peticiones que solicitan el motor *Cycles* (11) y las que solicitan *Eevee* (18).

- **12 y 19.** Cantidad media de información procesada por el servidor por milisegundo de tiempo que ha requerido su instancia de *Blender*, es decir, el cociente entre los valores 7 y 11 y el cociente entre los valores 14 y 18. Se diferencia entre las peticiones que solicitan el motor *Cycles* (12) y las que solicitan *Eevee* (19).

- **13 y 20.** Puntuación que se utiliza para dar prioridad a un servidor o a otro a la hora de asignarles una petición. Cuanto mayor sea, mayor prioridad tendrá. Este valor tiene en cuenta la cantidad de información procesada, los tiempos requeridos por el sistema y las resoluciones solicitadas en cada petición. Se diferencia entre las peticiones que solicitan el motor *Cycles* (13) y las que solicitan *Eevee* (20).

- **21.** Tiempo empleado por el servidor para completar el renderizado de prueba que tiene que realizar a modo de desafío y de *benchmark* para poder ser añadido al sistema.

- **22.** Número total de peticiones satisfechas por el servidor.

- **23.** Número de peticiones encoladas al servidor.

B.3.2. Consulta de información de las peticiones de renderizado

En la parte derecha del panel de administración se encuentran las tarjetas correspondientes a cada petición de renderizado registrada en el sistema. Al igual que con las tarjetas referentes a los servidores, estas incluyen iconos representativos y una descripción de a qué se corresponde cada línea si se mantiene el cursor sobre este.

De nuevo, se utilizan Figuras con los distintos valores numerados para facilitar la identificación. Sin embargo, en este caso, se hace uso de una Figura por cada estado, ya que, a diferencia de las tarjetas de los servidores, la información varía un poco de un estado a otro. Véase por ello las Figuras B.27 correspondiente a una petición finalizada, la Figura B.28 correspondiente a una petición que está siendo procesada y la Figura B.29 correspondiente a una petición que se encuentra encolada. A continuación, se detalla a qué corresponde cada valor:

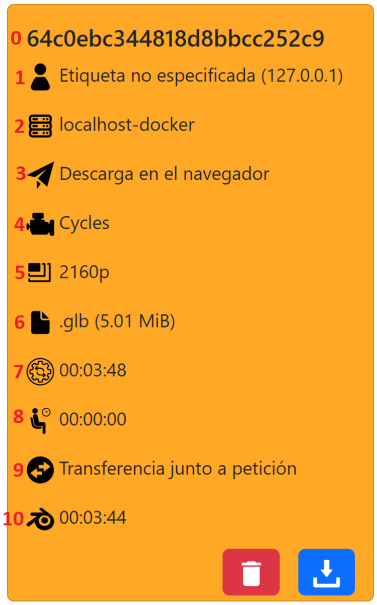

Figura B.27: Tarjeta correspondiente a una petición de renderizado que ya ha sido satisfecha

Figura B.28: Tarjeta correspondiente a una petición de renderizado que está siendo procesada

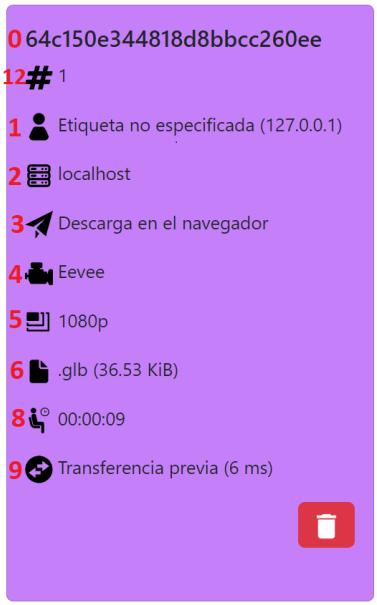

Figura B.29: Tarjeta correspondiente a una petición de renderizado que se encuentra encolada

- **0.** Id de la petición.
- **1.** Información del usuario que ha generado la petición. Incluye la etiqueta opcional que puede haber especificado este y la dirección *IP* desde la que se ha recibido la petición.
- **2.** Nombre del servidor al que ha sido asignada la petición. Si en el momento de su recepción no ha sido posible asignarla a ningún servidor, se indicará aquí.
- **3.** Método de recepción de la imagen que ha indicado el usuario. Si ha seleccionado envío por correo electrónico, se mostrará la dirección que ha indicado.
- **4.** Motor de renderizado que ha indicado el usuario.
- **5.** Resolución que ha solicitado el usuario.
- **6.** Extensión del archivo asociado a la petición que maneja el sistema (*.gltf*, *.glb* o *.drc*) junto al tamaño de este.
- **7.** Si ha finalizado, tiempo que le ha llevado al sistema procesarla teniendo en cuenta tiempos de transferencia y tiempos de renderizado. Si se está procesando, tiempo que ha transcurrido desde que ha sido enviada al servidor.
- **8.** Tiempo que ha pasado la petición encolada. Si aún se encuentra encolada, tiempo que lleva esperando en la cola.

- **9.** Estado de la transferencia del fichero asociado a la petición al servidor asignado. Se podrán dar cuatro casos:
  - La petición se envía junto al fichero de forma conjunta porque ha podido ser reenviada directamente al ser recibida.
  - El fichero se ha enviado previamente al servidor porque su petición correspondiente ha tenido que ser encolada y ya se ha completado la transferencia. Se mostrará el tiempo que ha requerido la transferencia.
  - El fichero se ha enviado previamente al servidor porque su petición correspondiente ha tenido que ser encolada pero aún no se ha completado la transferencia.
  - La petición no ha podido ser asignada a ningún servidor, por lo que no se ha podido ni enviar directamente la petición con el fichero ni iniciar la transferencia previa de este último.
- **10.** Tiempo de renderizado requerido por *Blender* al procesar la petición.
- **11.** Solo en el caso de que se esté procesando, el tiempo de espera estimado para que finalice el renderizado si este está disponible (Figura B.28).
- **12.** Solo en el caso de que se encuentre encolada, la posición que ocuparía si se ordenaran todas las peticiones encoladas independientemente del servidor asignado por la cantidad de tiempo que llevan esperando.

B.3.3. Consulta de estadísticas generales del sistema

Las estadísticas generales del sistema se pueden consultar en la esquina inferior derecha del panel de administración. Esta sección incluye la siguiente información:

- **Servidores registrados:** Número total de servidores de renderizado registrados en el sistema.
- **Servidores disponibles:** Del total de servidores de renderizado registrados en el sistema, cuántos de estos se encuentran disponibles.
- **Servidores ocupados:** Del total de servidores de renderizado registrados en el sistema, cuántos de estos se encuentran procesando una petición.
- **Servidores deshabilitados:** Del total de servidores de renderizado registrados en el sistema, cuántos de estos se encuentran deshabilitados.

- **Peticiones totales:** Número total de peticiones de renderizado registradas en el sistema.
- **Peticiones en proceso:** Del total de peticiones de renderizado registradas en el sistema, cuántas de estas se encuentran siendo procesadas en un servidor de renderizado.
- **Peticiones encoladas:** Del total de peticiones de renderizado registradas en el sistema, cuántas de estas se encuentran encoladas.
- **Peticiones finalizadas:** Del total de peticiones de renderizado registradas en el sistema, cuántas de estas ya han sido satisfechas.
- **Tiempo medio de procesamiento:** Valor medio de tiempo requerido por el sistema para satisfacer todas las peticiones finalizadas, teniendo en cuenta tanto los tiempos requeridos para la transferencia de ficheros como los que precisan los procesos de renderizado.
- **Tiempo medio de espera en cola:** Valor medio de tiempo que han tenido que esperar las peticiones finalizadas que han pasado por alguna cola. Solo se tienen en cuenta las peticiones finalizadas, las encoladas se tendrán en cuenta una vez sean satisfechas.

Para estas dos últimas medidas, resulta necesario tener en cuenta que las peticiones utilizadas para realizar el cálculo son las mostradas en el panel, es decir, tanto una petición que se desee eliminar o bien otra que no se muestre porque se ha limitado el número máximo de tarjetas a mostrar en el panel de administración, no se tendrán en cuenta para calcular el valor medio.

B.3.4. Añadir servidor de renderizado

Para registrar un nuevo servidor de renderizado y que el sistema lo tenga en cuenta para el envío de peticiones, bastará con rellenar el formulario situado en la esquina inferior izquierda del panel de administración .

En este, será necesario rellenar el primer campo con un nombre para el servidor y el segundo con la dirección *IP* de este, teniendo en cuenta las siguientes restricciones (si alguna no se cumpliera, el sistema lo notificará):

- El nombre tendrá a lo sumo 20 caracteres.
- El nombre debe ser único, es decir, no puede existir un servidor registrado en el sistema con el mismo nombre.

- La dirección *IP* seguirá el formato *IPv4*.
- La dirección *IP* debe ser única, es decir, no puede existir un servidor registrado en el sistema con la misma dirección *IP*.

Una vez rellenados los campos, se hará *click* en el botón *Añadir servidor* (Figura B.30).

Figura B.30: Formulario para registrar un nuevo servidor de renderizado en el sistema

El sistema realizará las comprobaciones necesarias e intentará comunicarse con el servidor de renderizado. Este proceso puede tardar un tiempo significativo, ya que debe esperar a que el servidor demuestre que es capaz de renderizar con un renderizado de prueba. Mientras se realiza todo esto, se mostrará un modal de carga (Figura B.31).

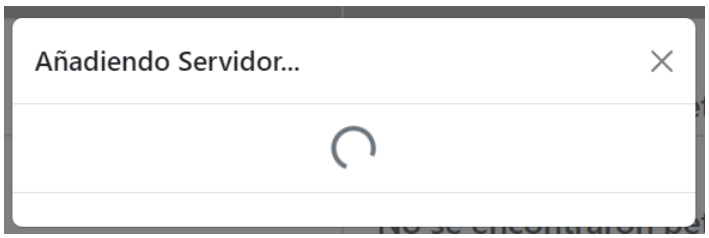

Figura B.31: Modal de carga mostrado mientras se intenta añadir el servidor

Una vez el sistema haya logrado registrar el servidor, este notificará del éxito en la operación (Figura B.32). Si se produjera algún error, informará de la misma forma.

Se podrá apreciar en el panel que se ha generado una nueva tarjeta con la información del nuevo servidor y que este se encuentra disponible inicialmente (Figura B.33).

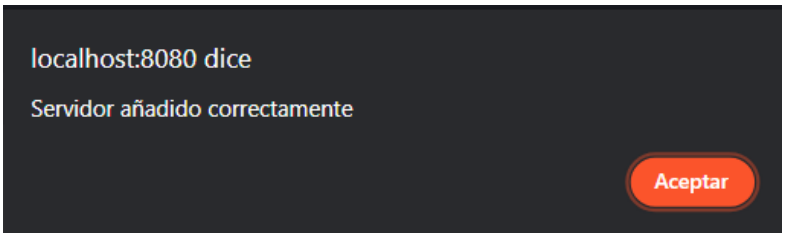

Figura B.32: Notificación del sistema tras llevarse a cabo el registro del servidor exitosamente

Figura B.33: Tarjeta correspondiente al nuevo servidor en el panel de administración

B.3.5. Eliminar servidor de renderizado

Es posible eliminar del sistema cualquier servidor del sistema que haya sido registrado previamente.

En primer lugar, se hará *click* en el botón con el icono de la papelera situado en la esquina inferior derecha de la tarjeta correspondiente al servidor que se desee eliminar (Figura B.34).

Figura B.34: Botón para eliminar el servidor de renderizado del sistema

Tras esto, el sistema mostrará un modal para que se confirme la intención de eliminar el servidor seleccionado. En este, se pulsará el botón *Eliminar* (Figura B.35).

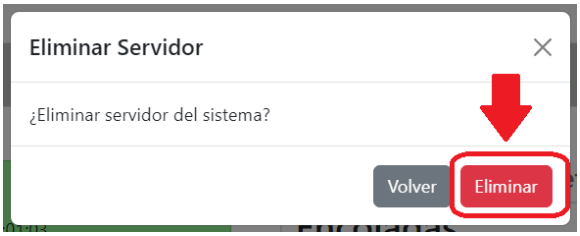

Figura B.35: Modal de confirmación mostrado tras solicitar la eliminación de un servidor

El sistema comenzará a realizar las comprobaciones y acciones necesarias para eliminar el servidor. Mientras tanto, el modal indicará que se está llevando a cabo la operación (Figura B.36).

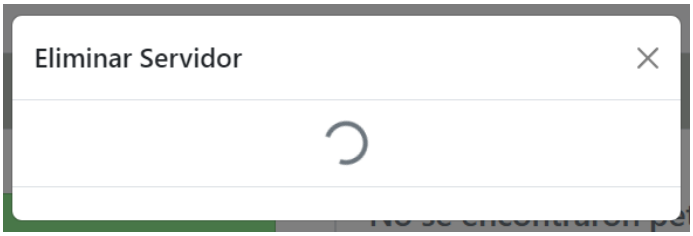

Figura B.36: Modal de carga mostrado mientras se intenta eliminar el servidor

Una vez el servidor haya sido eliminado, el sistema notificará del éxito de la operación (Figura B.37) y, se podrá apreciar en las tarjetas del panel de administración que ya no se encuentra la correspondiente al servidor eliminado.

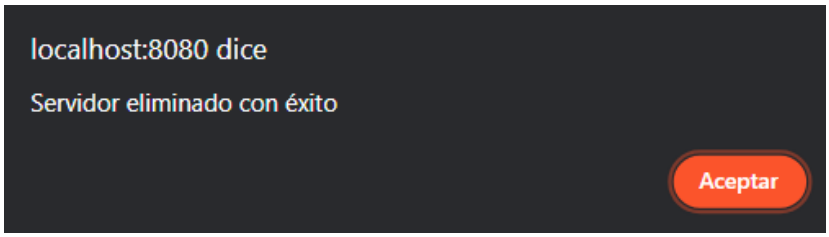

Figura B.37: Notificación del sistema tras llevarse a cabo la eliminación del servidor exitosamente

B.3.6. Deshabilitar servidor de renderizado

Es posible deshabilitar un servidor de renderizado para evitar que el sistema lo tenga en cuenta a la hora de asignar peticiones de forma temporal sin que sea necesario eliminar su

información.

Para ello, se hará *click* en el botón con el icono del candado cerrado situado en la esquina inferior derecha de la tarjeta correspondiente al servidor que se desee deshabilitar. Este botón será accesible solo si la tarjeta corresponde a un servidor disponible (Figura B.38).

Figura B.38: Botón para deshabilitar el servidor de renderizado

Tras esto, el sistema mostrará un modal para confirmar la intención de deshabilitar el servidor seleccionado. En este, se pulsará el botón *Deshabilitar* (Figura B.39)

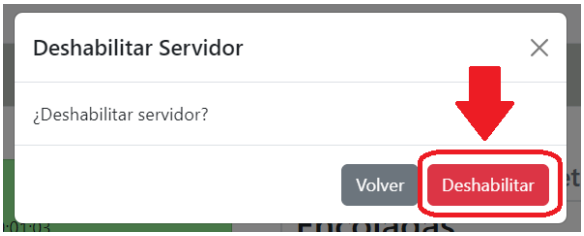

Figura B.39: Modal de confirmación mostrado tras deshabilitar un servidor

El sistema comenzará a realizar las comprobaciones y acciones necesarias para deshabilitar el servidor. Mientras tanto, el modal indicará que se está llevando a cabo la operación (Figura B.40).

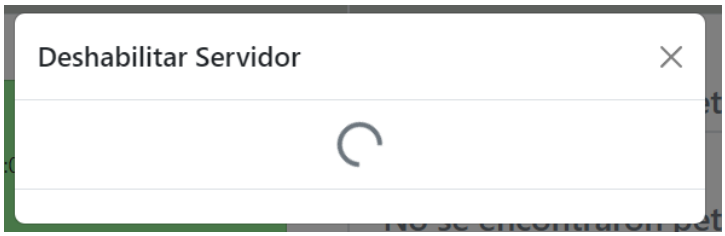

Figura B.40: Modal de carga mostrado mientras se intenta deshabilitar el servidor

Una vez el servidor haya sido deshabilitado, el sistema notificará del éxito de la operación (Figura B.41) y se podrá apreciar en las tarjetas del panel de administración que la correspondiente al servidor deshabilitado ha pasado del contenedor de servidores disponibles al de servidores deshabilitados (Figura B.42).

A partir de este momento, ya no se enviarán ni se encolarán peticiones a este servidor hasta que se vuelva a habilitar.

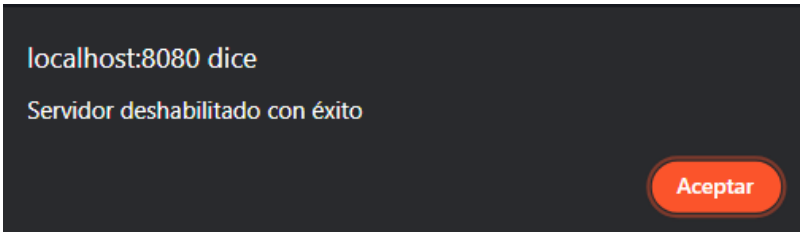

Figura B.41: Notificación del sistema tras lograr deshabilitar el servidor exitosamente

Figura B.42: Tarjeta actualizada tras deshabilitar con éxito el servidor

B.3.7. Habilitar servidor de renderizado

Cuando se haya deshabilitado un servidor, este podrá ser habilitado de nuevo para que vuelva a ser tenido en cuenta para la asignación de peticiones.

Para lograrlo, se hará *click* en el botón con el icono del candado abierto situado en la esquina inferior derecha de la tarjeta correspondiente al servidor que se desee habilitar. Este botón será accesible solo si la tarjeta corresponde a un servidor deshabilitado (Figura B.43).

Figura B.43: Botón para habilitar el servidor de renderizado

Tras esto, el sistema mostrará un modal para confirmar la intención de habilitar el servidor seleccionado. En este, se pulsará el botón *Habilitar* (Figura B.44).

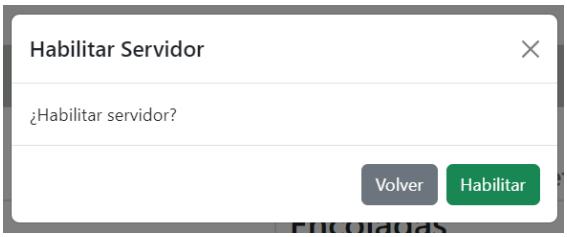

Figura B.44: Modal de confirmación mostrado tras solicitar habilitar un servidor

El sistema comenzará a realizar las comprobaciones y acciones necesarias para habilitar el servidor. Mientras tanto, el modal indicará que se está llevando a cabo la operación (Figura B.45).

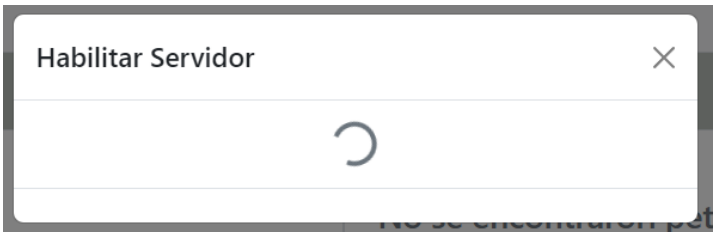

Figura B.45: Modal de carga mostrado mientras se intenta habilitar el servidor

Una vez habilitado de nuevo el servidor, el sistema notificará del éxito de la operación (Figura B.46). Por su parte, el panel de administración mostrará de nuevo la tarjeta correspondiente al servidor habilitado en el contenedor de servidores disponibles (Figura B.47).

A partir de este momento, el servidor volverá a ser tenido en cuenta por el sistema a la hora de asignar peticiones.

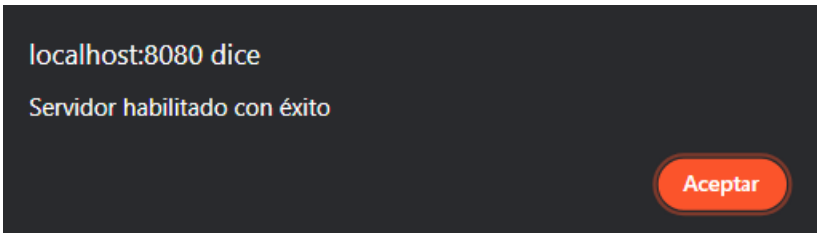

Figura B.46: Notificación del sistema tras lograr habilitar el servidor exitosamente

Figura B.47: Tarjeta actualizada tras habilitar con éxito el servidor

B.3.8. Abortar procesamiento en servidor de renderizado

Cuando un servidor se encuentre renderizando una petición, será posible abortar este proceso desde el panel de administración. Es importante tener en cuenta que la petición que estaba siendo procesada no se eliminará, sino que será enviada al final de la cola. Esta funcionalidad puede resultar útil cuando se perciba que algo no va bien en el procesamiento y se quiera extraer la petición del servidor sin eliminarla, o cuando se desee que entre la siguiente petición encolada del servidor sin tener que esperar a que finalice la actual.

El primer paso será hacer *click* en el botón con el icono con un círculo que contiene una *X* situado en la esquina inferior derecha de la tarjeta correspondiente al servidor en el que se quiera abortar el procesamiento. Este botón será accesible solo si la tarjeta corresponde a un servidor que se encuentra procesando una petición (Figura B.48).

Figura B.48: Botón para abortar el procesamiento en el servidor de renderizado

El sistema mostrará un modal para confirmar la intención de abortar el procesamiento en el servidor seleccionado. En este, se pulsará el botón *Abortar* (Figura B.49).

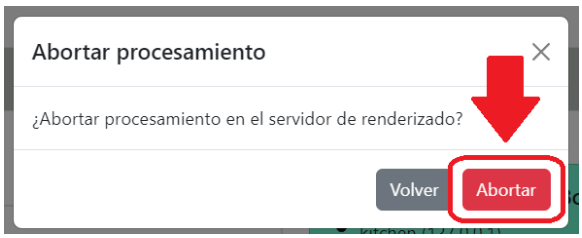

Figura B.49: Modal de confirmación mostrado tras solicitar abortar el procesamiento en un servidor

El sistema comenzará a realizar las comprobaciones y acciones necesarias para abortar el procesamiento en el servidor. Mientras tanto, el modal indicará que se está llevando a cabo la operación (Figura B.50).

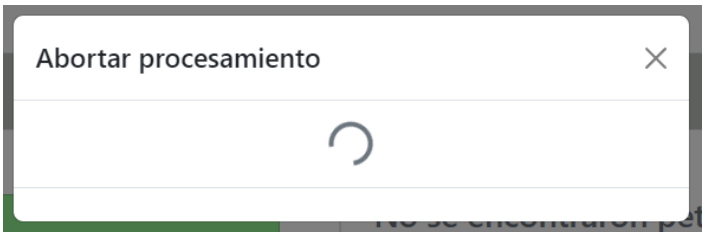

Figura B.50: Modal de carga mostrado mientras se intenta abortar el procesamiento en el servidor

Una vez abortado el procesamiento en el servidor, el sistema notificará del éxito de la operación (Figura B.51).

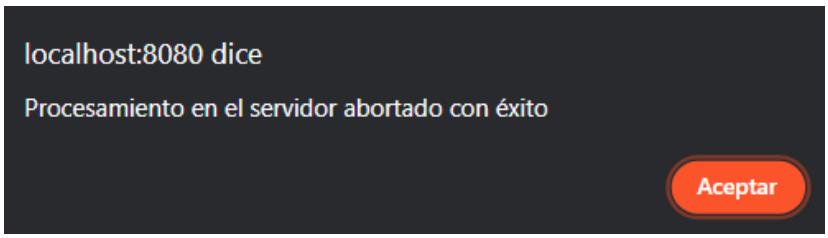

Figura B.51: Notificación del sistema tras lograr abortar el procesamiento en el servidor exitosamente

Los cambios reflejados en el panel de administración dependerán de la situación en la que se encontrara el sistema. Por ejemplo, si la petición abortada era la única en el sistema, volverá a ser reenviada a un servidor inmediatamente.

La Figura B.52 muestra una situación en la que hay un solo servidor y dos peticiones. Una está siendo procesada (*kitchen*) y otra se encuentra encolada (*ferrari*). Al abortar el procesamiento en el servidor, es posible observar en la Figura B.53 como *ferrari* sale de la cola y es enviada al servidor. *Kitchen* pasa de estar siendo procesada a la cola. Si hubiera más peticiones encoladas al mismo servidor, estarían por delante.

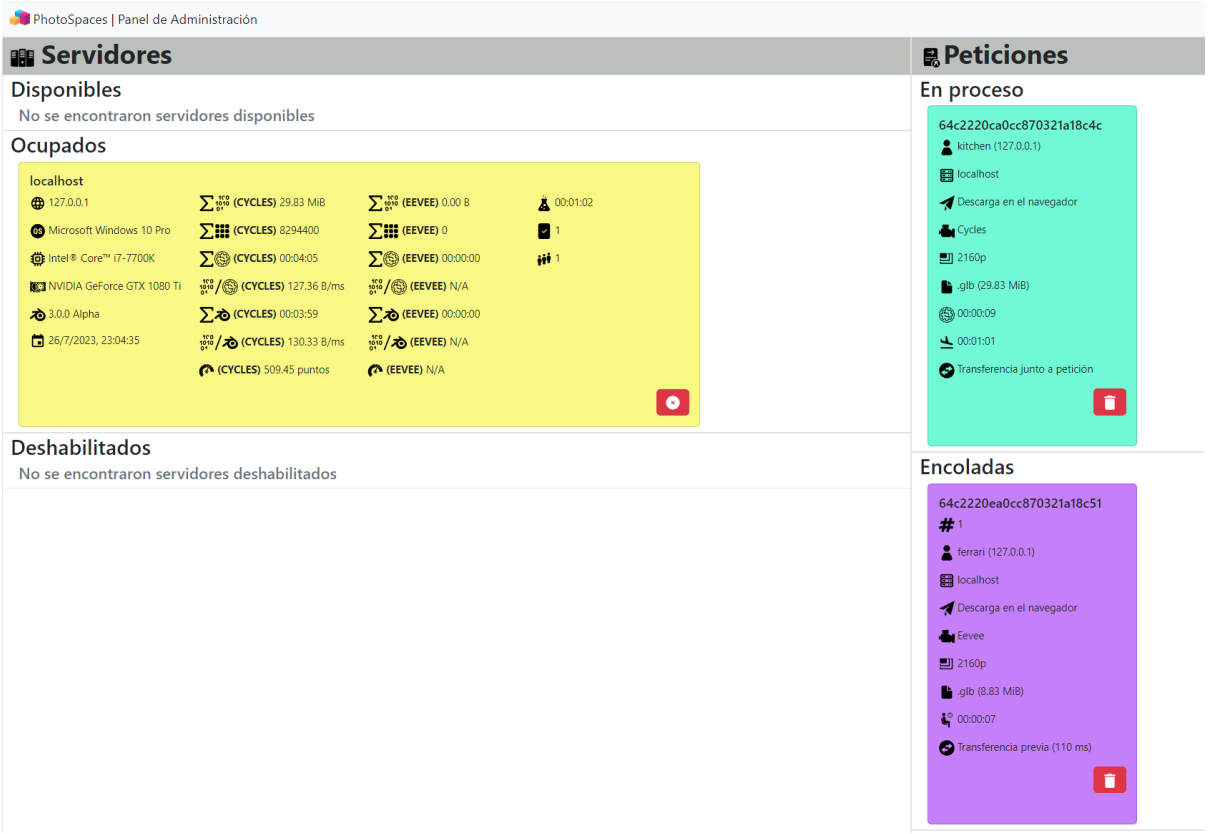

Figura B.52: Panel de administración antes de abortar el procesamiento en el único servidor

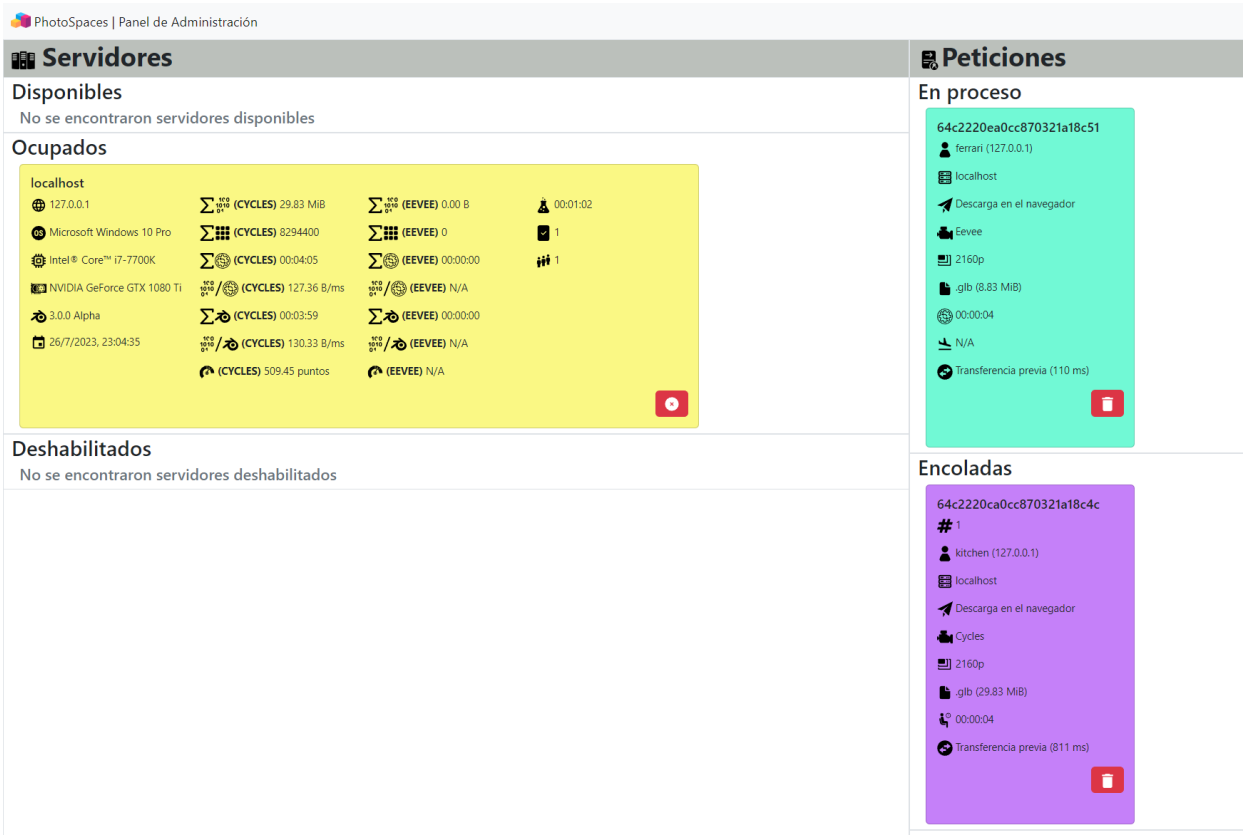

Figura B.53: Panel de administración tras abortar el procesamiento en el único servidor

B.3.9. Eliminar petición de renderizado

Es posible eliminar una petición de renderizado del sistema independientemente de su estado, teniendo en cuenta que, si está siendo procesada, se abortará el procesamiento en el servidor que se le había asignado.

En primer lugar, será necesario hacer *click* en el botón con el icono de papelera situado en la esquina inferior derecha de la tarjeta correspondiente a la petición que se desee eliminar. En este caso, este botón se encontrará accesible en cualquier tarjeta de petición (Figura B.54).

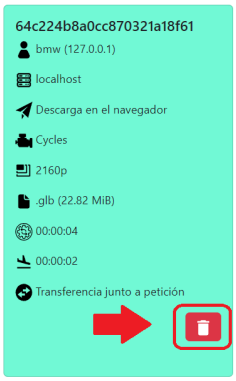

Figura B.54: Botón para eliminar la petición de renderizado del sistema

A continuación, el sistema mostrará un modal para confirmar la intención de eliminar la petición seleccionada. En este, se pulsará el botón *Eliminar* (Figura B.55)

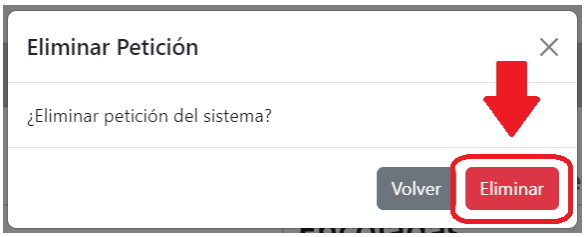

Figura B.55: Modal de confirmación mostrado tras solicitar la eliminación de una petición

Una vez pulsado el botón, el sistema comenzará a realizar las comprobaciones y acciones necesarias para eliminar la petición. Mientras tanto, el modal indicará que se está llevando a cabo la operación (Figura B.56).

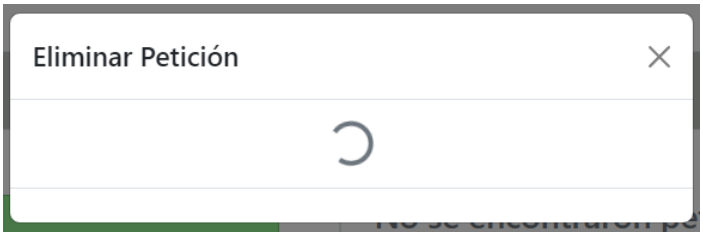

Figura B.56: Modal de carga mostrado mientras se intenta eliminar la petición

Una vez la petición haya sido eliminada, el sistema notificará del éxito de la operación (Figura B.57) y se podrá apreciar en el panel que ya no se muestra la tarjeta correspondiente a la petición eliminada.

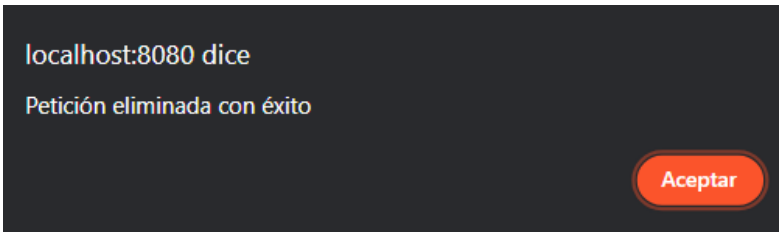

Figura B.57: Notificación del sistema tras llevarse a cabo la eliminación de la petición exitosamente

Además, si la petición eliminada se encontraba siendo procesada, el servidor que la estaba procesando habrá pasado a procesar la siguiente petición encolada en caso de existir esta, o a encontrarse disponible en caso de que no.

B.3.10. Descargar imagen renderizada asociada a petición finalizada

También es posible descargar las imágenes renderizadas que se han generado para los usuarios desde el panel de administración.

Para obtener estos ficheros, bastará con hacer *click* en el botón con el icono de descarga situado en la esquina inferior derecha de la tarjeta correspondiente a la petición cuya imagen asociada se desee descargar. Este botón solo será accesible si la tarjeta corresponde a una petición finalizada (Figura B.58).

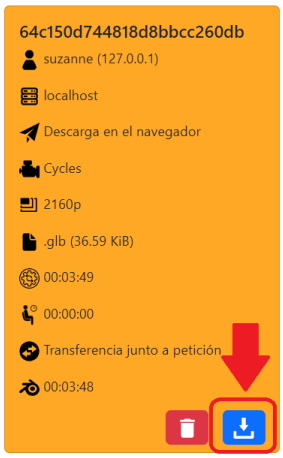

Figura B.58: Botón de descarga de la imagen renderizada

Una vez pulsado el botón, comenzará la descarga de esta. Cuando finalice la transferencia, el navegador solicitará una ubicación para almacenar la imagen descargada y un nombre para el fichero. Tras indicarlos, se dispondrá de la imagen de forma local en el equipo utilizado

(Figura B.59).

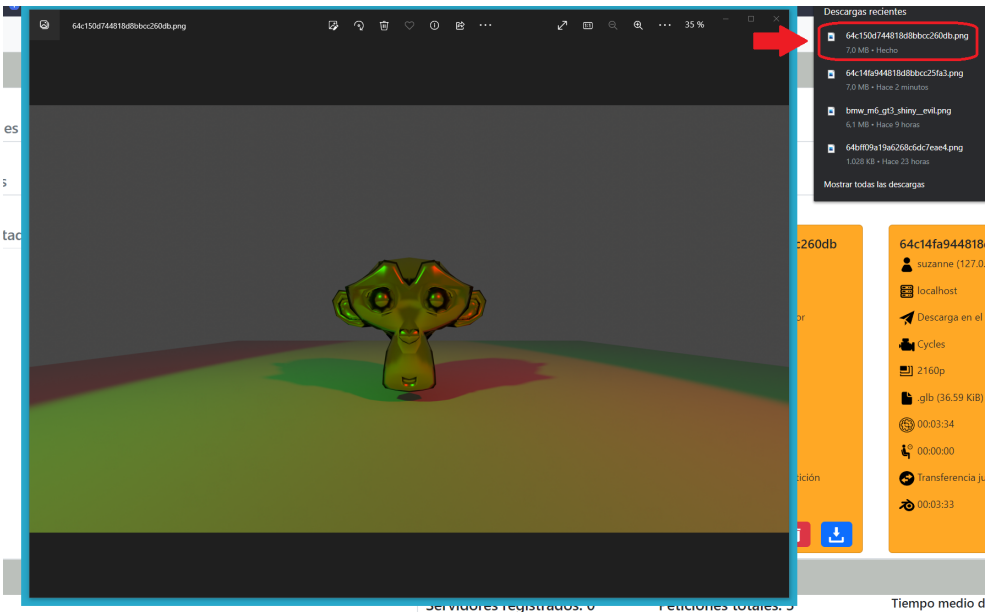

Figura B.59: Imagen renderizada descargada en el navegador a través del panel de administración

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA