Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación web para la visualización de objetos 3D fragmentados en piezas

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Carlos Durán Roca

Tutor: Eduardo Vendrell Vidal

Junio 2018

Resumen

Proceso de implementación y diseño de una aplicación web cuyas funcionalidades principales son mostrar objetos 3D y realizar acciones de visualización y tratamiento sobre ellos. Un ejemplo podría ser obtener medidas directamente sobre la superficie del objeto.

Las figuras mostradas serán obras restauradas o en proceso de restauración, por lo que cada modelo estará formado por distintas piezas. Estas se mostrarán en la posición pertinente en el visor. Estarán alojadas en un almacén de datos y habrá un sistema que facilitará la gestión de las mismas.

Palabras clave: *Ruby on Rails*, 3D, aplicación web, obras de arte.

Abstract

Implementation and design process of a web application which its main functionalities are showing 3D objects and perform visualization and treatment actions on them. An example about that is getting measures directly from the object's surface.

The shown objects will be restored artworks or in restoration process objects. This is why each model will be composed by several pieces. These will be shown in its pertinent position. They will be located in a data warehouse and there will be a system which make easy the gestion of them.

Keywords: Ruby on Rails, 3D, web application, artwork.

Resum

Procés d'implementació i disseny d'una aplicació web que té com a funcionalitats principals mostrar objectes 3D i realitzar accions de visualització i tractament. Un exemple podria ser obtenir mesures directament sobre la superfície de l'objecte.

Les figures mostrades seran obres d'art restaurades o en procés de restauració, per la qual cosa cada model estarà format per diverses peces. Aquestes es mostraran en la posició pertinent en el visor. Estaran allotjades en un magatzem de dades i haurà un sistema que facilitarà la gestió d'aquestes.

Paraules clau: *Ruby on Rails*, 3D, aplicació web, obras d'art.

Tabla de contenidos

1 Introducción 8
1.1 Estructuración del trabajo 9
1.2 Motivación 10
1.3 Requisitos 11
1.4 Objetivos 13
1.5 Metodología y estructura 13
1.6 Convenciones 14
1.7 Impacto 16
2 Estado del arte 16
2.1 Crítica 19
2.2 Solución y propuesta de mejora 20
3 Planificación 21
3.1 Análisis de problemas 21
3.1.1 Seguridad 21
3.1.2 Eficiencia 22
3.1.3 Legal y ético 23
3.2 Planificación temporal 23
3.3 Planificación económica y de recursos 25
4 Desarrollo del proyecto 26
4.1 Herramientas utilizadas 26
4.2 Arquitectura del sistema 27
4.3 Diseño detallado 29
4.4 Creación de base de datos: modelos y servidor 30
4.5 Implementación del visor con *Javascript* y *ThreeJS* 33
4.5.1 Carga de piezas 33
4.5.2 Descarga asíncrona con *web-workers* 33
4.5.3 Herramientas de visualización 34
4.5.4 Herramientas interactivas 37
4.6 Aplicación *Rails* 41
4.6.1 Despliegue 41
4.6.2 Vistas y controladores 42
5 Producto final 43

5.1 Interfaz de usuario 43
5.1.1 Interfaz 44
5.1.2 Interfaz del visor 49
5.2 Carga automática de piezas 51
5.3 Objetivos y requisitos cumplidos 51
5.4 Pruebas 53
6 Conclusiones 55
6.1 Análisis del producto 56
6.2 Satisfacción con el resultado 57
6.3 Conocimientos obtenidos durante el desarrollo 57
7 Trabajos futuros 58
8 Bibliografía 59
9 Anexo: Manual de usuario de la aplicación 61
9.1 Descripción general de la aplicación 61
9.2 Creación de cuenta e inicio de sesión 61
9.3 Carga de piezas 62
9.4 Descripción general del visor 64
9.5 Herramientas del visor 65
9.5.1 Modo de visibilidad 65
10 Anexo: tablas con información sobre la base de datos 70
11 Glosario 71

Tabla de Figuras

Figura 1. Representación del porcentaje de las obras no expuestas. Artículo de Quarz, desarrollado por Christopher Groskopf 9
Figura 2. Juego Interland, desarrollado por Google haciendo uso de ThreeJS 11
Figura 3. Crecimiento de la inversión en tecnologías de Gráficos por Computador, Artículo de investigación en EETimes, escrito por Jon Peddie 17
Figura 4. Imagen hiperrealista realizada con una GPU NVIDIA 18
Figura 5. Representación 3D de un objeto y su textura en el visor 19
Figura 6. Parte delantera del cuadro 20
Figura 7. Parte trasera del cuadro 20
Figura 8. Diagrama de Gantt con la planificación del proyecto 24
Figura 9. Presupuesto del proyecto 25
Figura 10. Relación uno a muchos 28
Figura 11. Relación muchos a muchos con tabla intermedia 28
Figura 12. Esquema principal arquitectura de la aplicación 30
Figura 13. Esquema principal de la base de datos 32
Figura 14. Tabla comparativa de tiempos de descarga 34
Figura 15. Gráfica de descarga en Google Chrome 34
Figura 16. Representación temporal de descarga en Google Chrome 34
Figura 17. Modelo vista con pieza seleccionada 36
Figura 18. Modelo donde se muestran las piezas que bordean a la seleccionada 36
Figura 19. Vista plano XZ 37
Figura 20. Vista plano XY 37
Figura 21. Vista plano ZY 37
Figura 22. Vista superior segundo corte 38
Figura 23. Vista superior primer corte 38
Figura 24. Vista interior segundo corte 39
Figura 25. Vista interior primer corte 39
Figura 26. Prueba de medición con cero puntos 40
Figura 27. Prueba medición de perímetro 40
Figura 28. Punto de luz superior en modelo 41
Figura 29. Punto de luz paralelo al modelo 41
Figura 30. Vista principal en formato móvil 44
Figura 31. Vista principal en tableta 44
Figura 32. Cabecera principal 45
Figura 33. Vista para registrarse en la web 45
Figura 34. Vista para acceder a la web 45
Figura 35. Vista para la recuperación de contraseña 45
Figura 36. Página principal enriquecida con una serie de modelos libres 3D 46
Figura 37. Vista del panel de figuras destacadas, mostrando un modelo libre 3D 46
Figura 38. Vista de una de las categorías, mostrando una tarjeta ejemplo 47
Figura 39. Vista del índice de objetos restaurados 47
Figura 40. Formulario de información de una pieza 48
Figura 41. Interfaz del visor con figura de prueba “cake” 49

Figura 42. Vista del modo visibilidad 49
Figura 43. Vista del botón “Play” 50
Figura 44. Vista del botón “Pausa” 50
Figura 45. Vista del menú de selección de perspectiva 50
Figura 46. Vista del menú de corte 50
Figura 47. Vista del menú del panel de herramientas 51
Figura 48. Resultado promedio de las pruebas realizadas 53
Figura 49. Tiempos promedio de los test realizados 54
Figure 50. Gráfica con los tiempos de la figura 49 54
Figura 51. Vista de la interfaz en un iPad air 2 55

1 Introducción

La representación de figuras y objetos en un ordenador ha sido un reto que ha abordado la ingeniería desde la aparición de los primeros entornos virtuales, y que a día de hoy sigue siendo un gran desafío al que enfrentarse.

Una primera aproximación fue la visualización de los objetos en un espacio bidimensional. Se realizaban representaciones de figuras utilizando las vistas ortogonales del alzado, planta y perfil. Esto permitió dar grandes pasos en el campo del diseño y fabricación industrial, asentando las bases para realizar el primer programa de edición y visualización en 1963, llamado *Sketchpad*.

En el modelo mencionado se podían mostrar objetos tridimensionales en un espacio bidimensional, teniendo una vista parcial del mismo. El uso de esta técnica suponía una pérdida de información y una abstracción compleja del objeto que se quería mostrar.[1] Para realizar vistas más significativas y con perspectivas diferentes, se recurrió a la representación llamada 2.5D que emulaba las vistas en 3D haciendo uso de representaciones bidimensionales. Sin embargo, no contenía información de las tres dimensiones completa, solo una aproximación.

Las carencias que presentaba el modelo anterior fueron suplidas con el modelo tridimensional, que fue introducido a principio de los años 70 en los gráficos por computador. La información que aporta este modelo es más completa y sencilla de interpretar. No hace falta una visualización simultánea de varias vistas para tener una concepción geométrica de la figura.

El ser humano es capaz de entender con mayor facilidad la representación tridimensional de objetos y figuras en espacios virtuales. Esto se debe a que el cerebro no tiene que adaptarse a un nuevo marco dimensional, donde muchas veces la información es parcial, como en los modelos de dos dimensiones.

Es por esta causa que las nuevas tecnologías emergentes están haciendo uso del 3D. Encontramos ejemplos en la realidad aumentada o la realidad mixta que están teniendo un éxito creciente. El pasado año 2017, se invirtió un capital total de 13,7 millones de dólares y se prevé que esta cifra supere los 80 millones en 2018[2], demostrando el prometedor futuro que tendrán.

Con la representación 3D, el usuario comprende de una forma muy sencilla el espacio donde se encuentra y la mecánica que tiene, ya que la relaciona directamente con la realidad donde vive. Su uso se extiende desde la formación mediante simulaciones de entornos reales hasta el aprendizaje como usuarios pasivos. Tecnologías como la realidad aumentada se han utilizado en diversos campos como son la industria pesada, la medicina, la arquitectura, los videojuegos o incluso para las fuerzas armadas, siempre de una forma discreta y experimental, sin embargo, su crecimiento está asegurado.

La representación 3D nos permite acercar y estudiar piezas, modelos y fragmentos que no están a nuestro alcance. Un ejemplo lo podemos encontrar en las obras de arte. Estas se encuentran

[1] Ivan Edward Sutherland, “Sketchpad: A man-machine graphical communication system” ,2003, https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-574.pdf
[2] Paul Lamking, “VR And AR Headsets to Hit 80 Million By 2021”, 2017, Forbes: https://www.forbes.com/sites/paullamkin/2017/09/29/vr-and-ar-headsets-to-hit-80-million-by-2021

distribuidas por todo el mundo. Muchas de las piezas están en sótanos de museos a los que el público general no puede acceder, debido a restricciones de conservación. En la *National Gallery*, en el MET de Nueva York, en el MOMA, muestran menos de un 50% de todas las obras albergadas.[3]

En la figura 1 se puede observar un diagrama de barras que muestra el porcentaje de obras que se muestran al público en algunos de los museos más notorios del mundo.

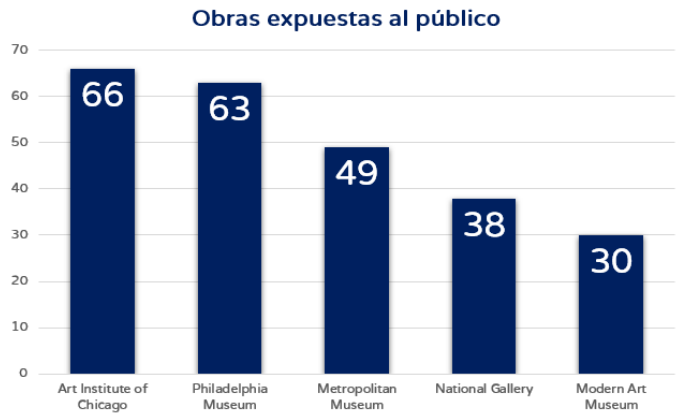


Figura 1. Representación del porcentaje de las obras no expuestas. Artículo de Quarz, desarrollado por Christopher Groskopf

Frecuentemente encontramos piezas que están fragmentadas, es decir, debido al paso del tiempo se han seccionado. Esto no permite mostrar la pieza como originalmente se creó y su reconstrucción podría implicar un deterioro mayor.

El proyecto que se va a realizar aborda este triple problema de visualización, accesibilidad y reconstrucción, proporcionando una aplicación que permita mostrar de manera sencilla obras de arte en proceso de restauración. La representación en 3D ayudará al usuario a tener una concepción concreta de cómo eran originalmente estas figuras. En adición a esto con una serie de herramientas podrá manipular los objetos para tener una experiencia más interactiva.

1.1 Estructuración del trabajo

A continuación, realizaremos un repaso punto por punto de todo lo que se va a mencionar en este trabajo, haciendo una breve explicación de los temas tratados en cada sección. Dichos apartados se pueden consultar en el índice en las primeras páginas del trabajo.

En la primera sección hablaremos del proyecto en términos generales, cómo se ha planificado, qué objetivos tiene que cumplir y han motivado a hacerlo. Además, se hará un análisis en profundidad de los requisitos que se quieran cumplir y metodología a seguir y estilo que se empleará para llevar a cabo el proyecto.

[3] Christopher Groskopf," Museums are keeping a ton of the world's most famous art locked away in storage", Quartz: https://qz.com/583354/why-is-so-much-of-the-worlds-great-art-in-storage/

En el apartado número dos realizaremos un breve comentario sobre el estado del arte, mencionando algunos de los ejemplos encontrados en internet y trabajos de años posteriores realizados en la escuela de informática.

En el tercer punto se desarrollará la planificación que se va a seguir en el proyecto. Se comentarán los posibles riesgos que pueden surgir durante el desarrollo, la planificación del trabajo y el presupuesto final del mismo.

En la cuarta sección se hablará del desarrollo, este punto se dividirá en cuatro secciones. Diseño de la aplicación y elección de su arquitectura y tecnologías. Diseño de los modelos y creación de la base de datos. Implementación del proyecto en *Ruby on Rails* y del visor de piezas 3D. Y la integración de todo el conjunto.

En la quinta sesión explicará el resultado final logrado tras el desarrollo, haciendo hincapié en las interfaces que el usuario final usará en la aplicación. Se harán breves valoraciones sobre ellas y se comentará su diseño.

En este apartado también se han descrito una serie de pruebas que se realizaron a un conjunto de usuarios para validar la aplicación.

Las conclusiones se realizarán en el apartado sexto, se hablará sobre el desarrollo y el conjunto de tareas que se realizaron. Se valorarán los objetivos completados y se explicará qué aspectos han tenido repercusión en el resultado final.

Tras explicar la implementación y realizar las conclusiones pertinentes, en el séptimo apartado se hablará sobre el futuro de ella y las posibilidades que tiene de cara al mercado, valorando la viabilidad del producto y su posible ampliación.

En los apartados ocho, nueve y diez, encontramos la bibliografía, un anexo adicional donde explica cómo utilizar la aplicación de forma correcta y un glosario donde encontraremos una lista de términos específicos para el lenguaje en el ámbito técnico informático.

1.2 Motivación

Como hemos comentado en los últimos párrafos introductorios, es importante e imprescindible poder acercar y dar a conocer la cultura y el arte que existe en nuestro planeta a toda la sociedad, independientemente de las circunstancias que rodeen a este conjunto. Internet y en concreto la web, tal y como los conocemos, ofrecen herramientas que facilitan hacer accesible la divulgación de exposiciones, esculturas y colecciones, entre otras muchas representaciones artísticas al gran público. Únicamente necesitaremos un dispositivo con un navegador moderno y una conexión a internet.

Para llegar a tener un público más amplio, no solo se podrá utilizar la plataforma con fines meramente lúdicos. Se quiere tener un lugar donde los historiadores y científicos, puedan guardar, visualizar, compartir y estudiar las figuras que estén en sus centros de investigación o

trabajos. De este modo, tendrán acceso a un conjunto de piezas inmenso desde cualquier lugar y contarán con una serie de herramientas que permitirán analizarlas de forma no invasiva.

Con este proyecto, se quiere intentar emular un escenario real de trabajo, con fechas de entrega, requisitos específicos e imprevistos que se tienen que resolver de forma eficaz. Se deben de aplicar las competencias, habilidades y conocimientos adquiridos a lo largo del grado, haciendo uso de una tecnología novedosa y cambiante.

ThreeJS, ha permitido desarrollar proyectos web 3D que han destacado por su eficiencia y su complejidad técnica. Muchos de ellos los podemos encontrar en la web oficial de la librería: https://threejs.org/ . Se ha querido destacar "*Interland*" desarrollado por Google (figura 2), un juego web implementado con esta tecnología con unos gráficos sublimes y un diseño magnífico.[4]

Figura 2. Juego Interland, desarrollado por Google haciendo uso de ThreeJS

Se quiere conseguir un resultado que intente llegar a igualar la calidad y complejidad de los ejemplos mencionados. Cabe destacar que ya se había hecho uso de esta librería, sin embargo, los proyectos donde se ha utilizado no han sido de la importancia de este, por lo que se tendrán que adquirir más conocimientos a medida que el proyecto avance.

1.3 Requisitos

Este proyecto tiene como objetivo desarrollar una aplicación web que tenga un sistema de gestión de usuarios que permita realizar las operaciones básicas de registro y autentificación. Estos podrán subir sus modelos fragmentados. También tiene que contener un visor 3D integrado en el cual se podrán visualizar las piezas que hayan subido y utilizar una serie de herramientas.

El usuario podrá crear una cuenta con una contraseña y nombre, a la cual estará asociado un correo electrónico. Cuando se haya creado la cuenta, se podrán subir obras de arte formadas por

[4] Google, Interland, 2017, https://beinternetawesome.withgoogle.com/en/interland/

distintas piezas, cada una corresponde a un archivo 3D (stl). Para ser subidos al almacén de datos, los archivos podrán estar incluidos en un fichero comprimido, junto con la información de posición. El sistema se encargará de leerlo, procesarlo y cargarlo en el almacén de forma automática. También existirá la posibilidad de subir los fragmentos de forma individual, sin necesidad de hacer uso del archivo comprimido, por lo tanto, la información adicional se tendrá que añadir rellenando campos de texto en la interfaz.

La información que acompaña a los fragmentos en el fichero comprimido son sus matrices de traslación, matrices de rango cuatro que indican la posición en la que se encuentra esa pieza en el espacio. Estas son suministradas por el usuario. Las matrices se podrán conseguir haciendo uso de un algoritmo de recomposición de piezas.

Para la representación de la figura el visor accederá a un almacén de datos para obtener las piezas, y a la base de datos para conocer la información de posición y clasificación. Será capaz de situar las piezas en el espacio tridimensional del visor, usando los datos suministrados, y mostrarlas correctamente al usuario. El tiempo para realizar esta acción debe de ser mínimo y el sistema de descarga tiene que ser transparente al usuario.

Se ha planificado crear un conjunto de herramientas para la visualización y tratamiento de figuras que permiten explorarlas más a fondo. La aplicación tiene que ser capaz de desempeñar todas las funciones que se enumerarán a continuación y que son requisitos básicos:

- Mostrar el conjunto de piezas faltantes.
- Mostrar el conjunto de piezas no faltantes.
- Tener una visión en alámbrico del modelo.
- Modificar la iluminación que afecta a las piezas.
- Situar la cámara en cualquier punto de la escena de forma sencilla.
- Situar la cámara de forma precisa en los planos de corte XY, YZ y XZ.
- Cortar la pieza en los planos X, Y y Z.
- Rotar de forma automática la pieza en modo presentación.
- Medir una serie de puntos sobre la superficie de la pieza.
- Seleccionar una pieza determinada y obtener información concreta de ella.
- Obtener la información del conjunto de las piezas.
- Tiempo de iniciación de herramientas menor a un segundo.

Todas estas funcionalidades tienen que ejecutarse en tiempo real, sin cortes, esperas o deceleraciones gráficas. Para las medidas mencionadas, *a priori*, es difícil establecer los requisitos mínimos que tiene que tener un ordenador para ejecutar la aplicación. Para marcar un límite en el hardware, se han listado los componentes mínimos que debe de tener el ordenador:

- Procesador Intel Core i5-6400u 2.5GHz o equivalente.
- Tarjeta gráfica NVIDIA GeForce 920MX con 2 GB de RAM dedicada o equivalente.
- 4 GB DDR3 de memoria RAM.
- Disco duro magnético de 128 GB.
- Pantalla con resolución FullHD 1080.

En el resto de las funcionalidades que no forman parte de las herramientas para el visor, pero que también forman parte de los requisitos esenciales para un correcto y completo funcionamiento de la aplicación. Son los siguientes:

- Gestión de usuarios de forma simple y eficaz (explicado en el primer párrafo).
- Carga de piezas intuitiva y transparente.
- Carga de las piezas en el visor con un tiempo inferior a 60 segundos.

La implementación de las tareas es esencial para poder considerar el proyecto como terminado y exitoso, sin embargo, durante su proceso se podrán añadir más requisitos que se comentarán en el apartado de análisis y conclusiones 6.

1.4 Objetivos

El propósito principal es crear una aplicación web capaz de cubrir todos los requisitos expuestos en el apartado 1.3, entre los que nos gustaría destacar: visor de piezas 3D fragmentadas accesible y una plataforma de gestión de los recursos que abastecerán a dicho visor.

Esta aplicación tiene que ser sencilla de desplegar y de instalar, garantizando en todo momento la seguridad de sus usuarios. Estos podrán acceder desde cualquier navegador moderno y que pueda ejecutar *JavaScript* sin sufrir ningún tipo de problema asociado al rendimiento como se ha comentado en el apartado 1.3, donde también se especifican los requisitos mínimos donde se garantizarán las opciones de rendimiento óptimo.

Se realizará una implementación eficiente y compacta, para que su ejecución sea rápida y pueda ser visualizado en la mayor cantidad de dispositivos posible. Se usarán tecnologías estándar tanto en el visor como en la aplicación *Rails*, tales como *JavaScript*, CSS y HTML. El objetivo de utilizar tecnologías estándar es el mismo que de programar de una forma eficiente: llegar al máximo número de usuarios.

Existe la posibilidad en todo momento de que los requisitos cambien ligeramente, esto no debe suponer un riesgo para los plazos de entregas críticos. Se adoptarán las medidas necesarias para que este hecho no repercuta negativamente en el trabajo.

La fecha límite para finalizar la implementación terminada es el día 5 de mayo de 2018, durante el desarrollo y tras la fecha indicada se redactará la memoria sobre este proyecto, la cual se expondrá el día 13 de Julio.

1.5 Metodología y estructura

Se considera que es de gran importancia hacer una descripción detallada del método de trabajo que se va a realizar, esto nos permitirá estudiar posteriormente los aspectos negativos y positivos en los que este ha tenido impacto durante la realización del proyecto.

Se ha adaptado la metodología ágil o *agile* a este proyecto, enfocándonos en la interacción con el usuario final. Esta forma de trabajo se centra en la realización y producción del software. Se pretende ir realizando entregas continuas de software de forma periódica para que se pueda ver el avance del proyecto y los resultados de la implementación de forma gradual.

Se permite realizar entregas de forma flexible, siempre que se respete el principio de que las entregas sean continuas. Esto permite al desarrollador trabajar de forma cómoda y sin una presión constante. Este trabajo tendrá revisiones periódicas en las que se evaluará el diseño y el software desarrollado cada vez que se finalice un *Sprint*.[5]

Un *Sprint* es el periodo de tiempo en el que se realizarán tareas concretas de implementación y diseño, como el desarrollo de los modos de visualización. Su duración puede variar dependiendo del tiempo que se estima que dura la tarea a realizar, en este proyecto se calcula que se realizarán varios *Sprints* de una semana y uno final de dos semanas. Al final de cada uno de ellos se realizará una valoración del trabajo y se publicará para que se pueda ver y comentar.[6]

Agile permite realizar cambios en los requerimientos aun empezado el proyecto, esta flexibilidad agrada al cliente y usuario final y los integra en el proceso de desarrollo y test. Si el entorno de trabajo es agradable, repercutirá positivamente sobre clientes, trabajadores y *software*.

Para el reparto de tareas se utilizará una pizarra *Kanban*, en ella se puede controlar el flujo de trabajo y qué integrantes del equipo están realizando las distintas tareas del proyecto. En este trabajo en concreto todas las tareas serán realizadas por el mismo desarrollador. En la pizarra existen tres columnas (pueden ampliarse) que indican el estado de las tareas: por hacer, haciéndose y realizadas.

GitHub tiene una herramienta que permite realizar esta actividad y enlazarla con los requisitos anotados en la plataforma, por lo que es cómodo trabajar con ella. Los diálogos se pueden replicar las veces que se quiera, para organizar mejor los distintos subproyectos dentro del proyecto.

1.6 Convenciones

Para que el proyecto pueda tener una continuidad en el tiempo, sea escalable y fácil de retomar por la comunidad o incluso por un equipo nuevo de desarrollo para mantenerlo o ampliarlo, se han establecido una serie de normas a seguir para la escritura e implementación del código. Muchas de estas normas están inspiradas en los estilos de programación de las comunidades de *JavaScript y Ruby on Rails*.

Las convenciones tienen que seguirse durante todo el desarrollo de la aplicación y sus posteriores modificaciones. Se deberá justificar su no utilización en algún caso particular, y en caso de realizar cambios en alguna de ellas se deberá especificar y añadir información sobre el cambio en un anexo nuevo de convenciones.

[5] Emerson Taymor, AGILE HANDBOOK ,2018, http://agilehandbook.com/agile-handbook.pdf
[6] Scrum.org, What is a Sprint in Scrum? 2018, https://www.scrum.org/resources/what-is-a-sprint-in-scrum

Desarrollo del código en *JavaScript* y *Ruby*:

- Utilizar *indentación* a dos espacios.
- Programación desacoplada, realizando un método por cada función.
- Las funciones no deben superar las 25 líneas de longitud.
- Las líneas no pueden exceder los 80 caracteres de longitud, idealmente se tienen que ajustar a 60 caracteres.
- Las variables globales son declaradas en el encabezado del código, en el ámbito en el que se encuentren.
- Las variables locales se deben declarar próximas a su inicialización y declaración.
- Los nombres de las variables tienen que ser descriptivos y se utilizará *camelCase* para su codificación en *JavaScript* y *snake_case* en *Ruby.*
- Los nombres de las funciones tienen que ser descriptivas y se utilizará *camelCase* para su codificación en *JavaScript* y en *snake_case* en *Ruby*. A ser posible utilizando como nombre la acción que van a realizar.
- Seguir el principio DRY (*don't repeat yourself*), no repetir el código[7].
- Seguir el principio KIS (*keep it simple*), realizar las tareas de la forma más sencilla posible.

Diseño en CSS y HTML:

- Por lo general se utilizarán las clases predefinidas de *Bootstrap* para realizar los diseños de la interfaz.
- Minimizar el código *Ruby* embebido en las vistas HTML.
- Las clases e identificadores específicos tienen que realizarse de manera concreta y escueta, de forma modular y reutilizable.
- Las clases e identificadores se ordenarán de forma alfabética, independientemente de su jerarquía (clase o identificador).
- Las clases e identificadores de CSS tienen como nombre el estilo que representan, si este está compuesto por varias palabras, irán separadas por guión.
- Utilizar el CSS y HTML estándar y nunca ninguna función propia de algún navegador, así se garantizará que todos los componentes funcionen en todos los navegadores.
- El estilo debe estar definido en los archivos que *Rails* indica para desempeñar esta tarea y nunca en cualquier otra localización.
- La indentación de los archivos HTML y CSS se debe de realizar a dos espacios.

Se utilizará *Git* para la tarea de control de versiones, actualmente se está utilizando la plataforma *GitHub*, pero si el proyecto lo requiere no habrá problema en migrarlo a otro lugar. Las recomendaciones para utilizar el control de versiones son las siguientes:

- Realizar *commit* de cada función implementada y subirla al repositorio.
- Los comentarios de los *commits* tendrán que ser breves y descriptivos con la estructura: verbo + función implementada + aclaración (opcional).
- En el caso de que el *commit* cierre una *issue* determinada indicar al final del comentario cual ha sido cerrada.

---

7 Mush Tahir Aziz Rahman, RUBY ON RAILS DEVELOPMENT PRINCIPLES, EXPLAINED, 2014 http://www.nascenia.com/ruby-on-rails-development-principles

- Se trabajará haciendo el uso de ramas, por cada funcionalidad nueva que se quiere implementar se creará una rama. La programación sobre la rama máster se reserva a pequeños cambios o urgentes.

1.7 Impacto

Esperamos que esta aplicación sea utilizada por museos, galerías o cualquier tipo de organización que pretenda acercar sus figuras representadas en modelos 3D a comunidades y colectivos que no tienen el acceso a ellas. Con esta tarea se quiere fomentar la difusión cultural en la sociedad.

Esta plataforma pretende superar en utilidades y eficiencia a las ya existentes, esto puede fomentar la creación de una comunidad abierta y libre para compartir las piezas. Una parte de los usuarios no serán meros espectadores como antes se ha comentado, serán especialistas en sus distintos campos que utilice la plataforma para estudiar y realizar investigaciones.

Con ella asentaremos las bases para el diseño de futuras aplicaciones que tomarán como referente la sencillez y la utilidad de esta. Se piensa que el futuro de las aplicaciones está en realizar diseños muy minimalistas que permitan realizar tareas complejas sin esfuerzo. No se debería estar especializado para utilizar una aplicación.

Se contribuirá en el desarrollo de software libre, ya que esta aplicación se liberará al público general para que pueda descargarla, comentarla, participar en ella o incluso realizar sus propias versiones de la misma.

Por supuesto esta página web simplemente es un inicio de todo un proyecto que pretende crecer con el paso del tiempo, añadiendo más funcionalidades que se puedan especificar o incluso, que los usuarios reporten como necesarias.

2 Estado del arte

Los gráficos por computador han llegado a su época más álgida en esta última década, en gran medida se debe por la incursión de los videojuegos en el ámbito doméstico, la utilización masiva de software de edición 3D y el interés por esta tecnología en sectores en auge como la simulación, que harán uso de ella. Compañías privadas como *Ubsioft* o *Unity* se han dedicado a realizar avances en este ámbito de forma privada, esto ha llevado a una mejora técnica en los gráficos por computador.

El renderizado fotorrealista a día de hoy, es uno de los métodos más complejos y avanzados para la síntesis y creación de gráficos. Tiene como objetivo realizar composiciones de escenas estáticas y cinemáticas afines a la realidad que intentan representar. Para conseguir este resultado se

utilizan algoritmos que hacen uso de sistemas basados en la física tales como la iluminación o la física de partículas.[8]

La implementación del fotorrealismo en animaciones y cortos hasta ahora era un proceso lento y complejo, que precisa de un hardware potente para poder realizarse. Sin embargo, actualmente se han desarrollado técnicas que parten del trazado de rayos a tiempo real.

El trazado de rayos es una técnica innovadora que consiste en generar un rayo por cada fuente de luz que se encuentra en la escena. Este interactuará con el entorno, creando reflejos y brillos en las superficies. Para realizar este proceso de una forma computable, se explota el principio de reciprocidad de los rayos, trazando únicamente las trayectorias que repercutan en la escena final. El flujo de rayos se puede controlar por software, dependiendo de la calidad final que se quiera obtener.

En el ámbito comercial se ha popularizado el software especializado para el tratamiento de los gráficos por computador. Dentro de este nicho, podemos hablar desde editores 3D, como *Blender* o *3DStudioMax*, hasta software de diseño industrial como el AutoCAD.

Un ejemplo de este serían los motores gráficos, que proporcionan al usuario un entorno de programación para desarrollar, en este caso Videojuegos, cómodo y fácil de utilizar. Implementa los últimos avances en tecnología, tales como motores de físicas y de renderizado, incluso implementan sistemas de IA (inteligencia artificial). Facilitan su uso a los programadores.[9]

La implementación de estos sistemas se debe en gran parte a que existen APIs (*aplication programmig interface)* que han facilitado su creación. Algunas de ellas tuvieron su aparición a principios de los años 90, cuando los computadores se introdujeron en la industria del cine.

Una gran parte del aumento de la potencia, eficiencia y del rendimiento en los gráficos por computador se lo debemos a las tarjetas gráficas, que implementan una gran variedad de algoritmos, de forma que ejecutarlos es muy eficiente.[10]

En la gráfica siguiente podemos ver el aumento de la cantidad de dinero (en billones de dólares) invertida en los diferentes sectores hasta el año 2014.

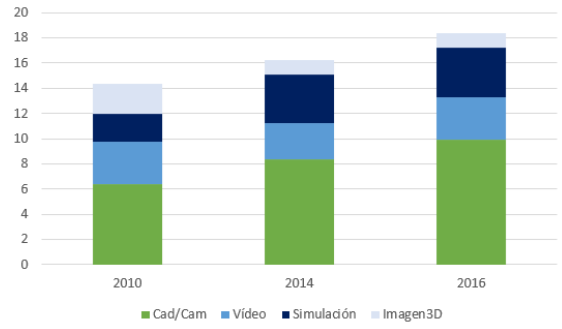


Figura 3. Crecimiento de la inversión en tecnologías de Gráficos por Computador, Artículo de investigación en EETimes, escrito por Jon Peddie

[8] David Cardinal, How Nvidia's RTX Real-Time Ray Tracing Works, 2018, Extremetech https://www.extremetech.com/extreme/266600-nvidias-rtx-promises-real-time-ray-tracing
[9] The Unreal dev Team, Unreal Engine Features, 2018, https://www.unrealengine.com/en-US/features
[10] Jon Peddie, The state of the art of graphics world, 2018, https://www.eetimes.com/author.asp?section_id=36&doc_id=1266025

Poco a poco el hardware se ha ido diversificando y se han creado dispositivos específicos para tareas particulares. De esta forma se optimiza la potencia y se focaliza para un solo cometido. Un ejemplo claro es cómo NVIDIA ha implementado la tecnología de trazado de rayos, que simula la luz natural en entornos virtuales en sus nuevos prototipos de tarjetas gráficas. [11]

En la figura 4 se puede observar un ejemplo de una imagen obtenida haciendo uso de la tecnología de trazado de rayos, obtenida con el soporte de NVIDIA® OptiX™ *Ray Tracing Engine*.[12]

Figura 4. Imagen hiperrealista realizada con una GPU NVIDIA

El futuro del hardware está directamente relacionado con la eficiencia de los algoritmos que se utilicen. El hardware especializado, es un fuerte apoyo al software y un gran paso para los gráficos por computador. La evolución del software y el hardware implicará la irrupción masiva de su uso en campos como el cine (en el que ya se ha utilizado), el arte o los videojuegos. Se introducirán e implementarán conceptos como el hiperrealismo.

Haciendo una referencia más concreta a nuestro trabajo, existe una página web la cual ha implementado la carga de objetos 3D desde una base de datos al igual que nosotros queremos hacer, sin embargo, queremos añadir funcionalidades que esta web no tiene, además de hacerla más eficaz y escalable y que sea genérica para poder ser desplegada en cualquier lugar.

El visor del que hablamos es el realizado por la institución *Smithsonian*, concretamente en el departamento de digitalización: "*Smithsonian DIGITIZATION*". A pesar de lo comentado antes hay que destacar que actualmente es uno de los visores referencia y que funciona de manera rápida e intuitiva.

Este visor fue creado por un equipo de cinco personas especializadas en el 3D. La finalidad y la filosofía de este proyecto es muy similar a la del nuestro: poder dar a conocer las obras no expuestas de forma permanente.

Este visor se puede encontrar en la siguiente dirección web: https://3d.si.edu/[13] , en ella se pueden encontrar diferentes figuras para explorar y un blog para participar en la plataforma. Los modelos que se encuentran en esta plataforma han sido digitalizados por la propia institución y

[11] David Cardinal, How Nvidia's RTX Real-Time Ray Tracing Works, 2018, Extremetech https://www.extremetech.com
[12] NVIDIA® OptiX™ Ray Tracing Engine, 2018, https://developer.nvidia.com/optix/index.html
[13] Smithsonian 3D team, About Smithsonian X 3D, 2013, https://3d.si.edu/about

son de una calidad excelente. Podemos encontrar desde obras de arte antiguas hasta artefactos modernos como son cápsulas de aterrizaje del *Apollo*.

En la figura 5 podemos observar la representación de una figura 3D con sus texturas originales, este visor servirá de referencia para realizar el nuestro ya que ambos comparten muchas características y funcionalidades.

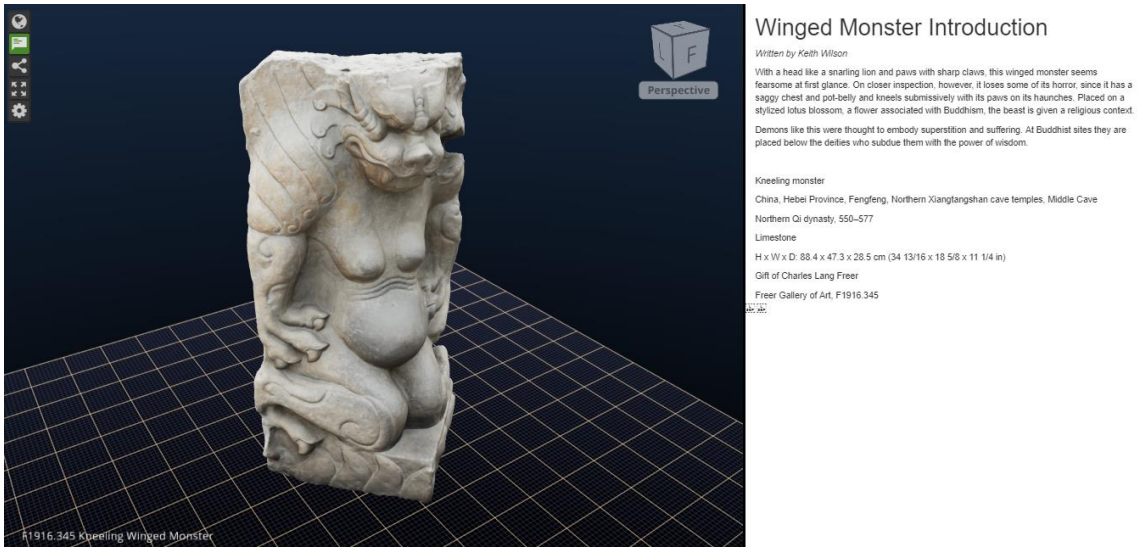


Figura 5. Representación 3D de un objeto y su textura en el visor

Se estudió un trabajo anterior de la misma temática realizado por un compañero en el año 2017. En este se pretendía hacer un visor 3D utilizando *Unity*, que posteriormente se convertiría en una plataforma web para la visualización de objetos tridimensionales.

Utiliza *SQLite* para almacenar las tablas de información de la piezas u objetos y un almacén local para la figura. Para levantar el servicio utiliza *XAMPP* (*Apache*, *MariadB*, *MySQL*, *PHP* y *PERL*), un paquete para la gestión de las bases de datos. Al lanzar el servicio ofrece la posibilidad de cargar figuras enterizas y visualizarlas en el navegador.

En este trabajo se implementó una galería en la que se podían ver los objetos y se podía interactuar con ellos con los controles básicos de *Unity*; escalado, rotación y translación. Además, introduce funciones más complejas como la de medición en caras planas con dos puntos de referencia y visualizar la información asociada al objeto.

2.1 Crítica

Valorando mucho el esfuerzo de los equipos en la realización de estas dos aplicaciones, se han encontrado ciertos puntos donde es posible que se puedan introducir mejoras. Nuestro proyecto pretende aprender de las carencias y aplicar mejoras en la solución que proponemos.

La aplicación realizada por el equipo de digitalización de la institución *Smithsonian*, a pesar de estar bien realizada y tener un rendimiento excelente, tiene ciertos *bugs* a la hora de representar imágenes, tal como el que se muestra a continuación, en las figuras seis y siete.

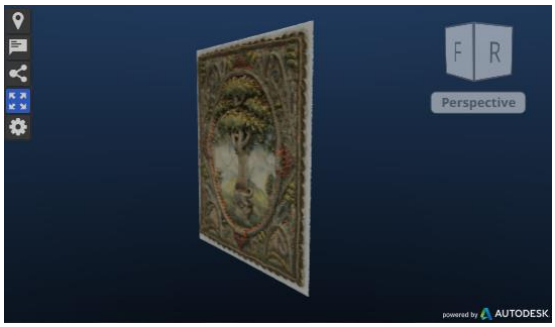


Figura 6. Parte delantera del cuadro

Figura 7. Parte trasera del cuadro

No se llega a mostrar el interior de las figuras, debido a que han escaneado solo la parte frontal, dejando un hueco transparente en el reverso de la pieza que puede generar confusión al usuario. Este puede pensar que la aplicación está mal o que realmente existe un hueco en esa posición.

La información que suministra la página web es muy extensa y está servida de una forma poco atractiva para el usuario, esto hace que pierda interés en la lectura. A pesar de tener un menú muy completo su uso es confuso ya que no se sabe exactamente qué representa cada imagen en las herramientas y los ajustes.

Utilizan software de *Autodesk* para realizar su visor, cuando existen alternativas más económicas con las que se puede contribuir a la comunidad del conjunto de desarrolladores de software libre. De esta forma el proyecto puede ser aplicable a más instituciones y no solo a las que pueden permitirse pagar por este servicio.

En cuanto al software desarrollado por nuestro compañero, no se sabe exactamente si el visor creado en *Unity* forma parte de la aplicación web o simplemente es un modelo de referencia. En todo caso, se critica que se utilice un motor de videojuegos para realizar un simple visualizador de elementos 3D. El hecho de utilizar *Unity* acota las posibilidades de la aplicación ya que la reduce a ser distribuida mediante un instalador.

El método de despliegue empleado es XAMPP, este es conocido por ser difícil de manejar ya que es necesario tener conocimientos específicos sobre él para poder realizar una configuración concreta para cada dispositivo. Pensamos que puede haber alternativas más fáciles de usar y más escalables. Con XAMPP se realizaron las pruebas preliminares de nuestro proyecto.

2.2 Solución y propuesta de mejora

Para atajar los problemas que sugeridos al visor de la institución del *Smithsonian,* se propone realizar una aplicación únicamente empleando tecnologías estándar y *Open Source* como *ThreeJS*, librería que explicaremos en el apartado 4.1. Con esta además se puede solucionar fácilmente el

problema de las caras invisibles, ya que tiene una directiva para hacer visibles ambas caras de cada polígono utilizando el "sistema de normales con dirección opuesta".

En cuanto a la usabilidad de la interfaz, pensamos que se puede mejorar incorporando etiquetas de texto que se muestren junto a las fotos. Además, las imágenes también representarán visualmente la acción que se va a realizar.

Para mejorar el software producido por el compañero cambiaríamos el concepto de la aplicación de forma drástica, abandonando *Unity* para el desarrollo del este visor, para realizar una aplicación web que lo incluya. Con esto aclararíamos el concepto del proyecto, dejando claro que su finalidad es realizar un visor web y aumentaríamos la accesibilidad del mismo, evitando que los usuarios tengan instalados paquetes especiales es sus máquinas.

3 Planificación

3.1 Análisis de problemas

Realizar un proyecto de estas características correctamente implica que se debe realizar un estudio metódico de los problemas que podrían surgir durante el desarrollo. Gracias a él se podrán prevenir o afrontar sin que implique una pérdida en el rendimiento o el incumplimiento de los requisitos.

En los siguientes puntos se comentarán los problemas más críticos y usuales que se pueden encontrar en una aplicación web, divididos en las categorías de seguridad, eficiencia y legalidad.

3.1.1 Seguridad

Los *frameworks* web, como *Rails*, implementan una serie de directivas y protocolos de seguridad, que si son respetados por el programador que lo utilizará se creará una aplicación segura y estable.

RoR, nativamente impide, gracias a *Active Record,* la inyección SQL. Esta práctica consiste en realizar consultas y acciones no deseadas a la base de datos que soporta nuestra aplicación. La finalidad de este ataque puede ser el ocasionar el borrado integral de la base de datos, arruinando por completo el sistema. También se pueden averiguar datos comprometidos, como contraseñas y correos electrónicos. Es importante seguir el patrón MVC para evitar este tipo de percances.

El ORM (*mapeo objeto relacional*) de *Rails, Active Record*, implementa consultas a la base de datos mediante métodos nativos. No hay que realizar las consultas SQL directamente sobre la base de datos implementadas por los programadores. Confiar en un *framework* estable y que ha sido puesto a prueba en diversas ocasiones es más seguro que implementar las propias consultas.

La seguridad en las cuentas de usuario es primordial para que no haya fugas de datos, se puedan suplantar identidades, o tener accesos indeseados. Para esto también se ha recurrido a una implementación de terceros, utilizando una gema llamada *Devise*. La causa para no realizar esta implementación de forma propia es la misma expuesta en el párrafo anterior; la librería ya ha sido probada en diversas aplicaciones.

Devise es una librería que se encarga de la gestión de usuarios, garantizando su seguridad ya que posee un sistema de autenticación y cifrado para las contraseñas de cada usuario. Además, permite las opciones de registro mediante confirmación, recuperación de contraseña y bloqueo por exceso de fallos al registro, entre otras muchas funcionalidades.[14]

Existen problemas en la seguridad en las páginas web debido a que se suele utilizar *flags* escondidos en el código HTML, imperceptibles a simple vista, pero fáciles de encontrar por atacantes que tengan conocimientos básicos de programación.

Se ha pensado que para evitar este tipo de contratiempos era necesario apartar toda la lógica de la vista y servir un HTML limpio. Esta tarea con *Rails* es muy sencilla ya que nos fuerza a albergar la lógica en el controlador separándola de la vista. En esta únicamente se implementará cómo aparecerá la información.

3.1.2 Eficiencia

Para trabajar en la web, hace falta optimizar los recursos que se tienen y hay que tener en cuenta diversos tipos de problemas, tales como: la baja latencia en la red, topologías que no ofrecen una conexión rápida o segura, equipos que no tienen suficiente potencia para ejecutar el *backend* del servidor, entre otros muchos. Para suplir dichos problemas hemos diseñado un sistema híbrido que mezcla las tecnologías de *JavaScript y Ruby on Rails*.

En *Rails* utilizamos una gema llamada *Turbolinks* para acelerar la carga de vistas. Su función principal es mantener invariantes los fragmentos en la página web que permanecen inmutables y cargar los nuevos, así se reduce significativamente el tiempo de espera entre las vistas. Esto lo consigue guardando fragmentos específicos en caché del HTML de nuestra web. *Turbolinks* se tiene que desactivar al iniciar el visor ya que esta gema no permite que se cargue de una forma correcta.

Para mantener la eficiencia en *Rails,* solo se ha embebido código *Ruby* en los archivos HTML únicamente cuando ha sido imprescindible y se necesitaba mostrar por pantalla información procedente de la base de datos. La inserción de *Ruby* en las vistas genera retrasos a la hora de servirlas. Esto está explicado con más detalle en el apartado 4.6.2.

En *JavaScript* se ha intentado explotar al máximo su asincronía, además, la comunicación con la base de datos para descargar los modelos de las piezas se ha realizado de forma concurrente, minimizando el tiempo de espera en la descarga.

[14] Devise team, Devise gem README & DOC,2009-2018 https://github.com/plataformatec/devise

3.1.3 Legal y ético

Todo el código empleado de terceros, sistema de control de versiones, y materiales de apoyo software utilizados para desarrollar esta aplicación se ha buscado que fuera *Open Source*, no solo para no tener problemas con temas referentes a la propiedad intelectual, sino por la filosofía del proyecto. Su idea de acercar la cultura a la gente y la forma de llevarla a cabo comparte muchos de los valores que se dan en el *Open Source*, como el de generar y compartir código.

La aplicación desarrollada será de código abierto y será publicada en plataformas para compartir código como *GitHub* el día que se finalice completamente su desarrollo. Tendrá una licencia *Creative Comons,* de reconocimiento y no comercial. Esto implica que cualquier obra derivada, copia o reproducción, tiene que atribuir la autoría a los creadores originales y que este software no se podrá utilizar con fines comerciales. [15]

La plataforma albergará las figuras y objetos subidos por expertos y profesionales en el campo de la historia y la restauración. Por lo general las obras de arte subidas a esta plataforma no están ligadas a ningún tipo de licencia o restricción debido al carácter que tienen. En ningún momento se ofrecerá la posibilidad de descargar la pieza que se está mostrando, simplemente la plataforma permite su visualización.

3.2 Planificación temporal

Este proyecto ha tenido una duración de 45 días aproximadamente desde su inicio hasta la versión final, sobre la cual se ha realizado este trabajo. Se ha esquematizado un diagrama de Gantt con la planificación temporal que se va a seguir.

Durante las tres primeras semanas, se realizará un proceso de estudio y diseño de la aplicación, analizando sus puntos fuertes, los imprevistos y problemas que puedan darse en las distintas fases.

La parte de implementación tendrá una duración de cuatro semanas, tres para la implementación y una para la integración de las distintas partes del sistema. Durante este periodo de tiempo se realizarán las labores de desarrollo. La definición basada en el diseño realizado de la base de datos y la creación de modelos se realizará durante la primera semana y posteriormente en paralelo se implementará el visor y la aplicación en *RoR*.

Por último, se realizarán las labores de redacción del proyecto, documentación y revisión. No se descarta realizar alguna implementación puntual ya que durante esta fase se prevé que se pueda modificar el código para mejorar su compresión sin modificar en ningún momento su lógica.

[15] Universidad Politécnica de Cartagena,"Las licencias creative comons",2016, Univerdad Politéncia de Cartagena, http://www.bib.upct.es/las-licencias-creative-commons

En este diagrama de Gantt que encontramos en la figura 8, se pueden visualizar mejor las tareas que se van a realizar en el tiempo que dure el proyecto.

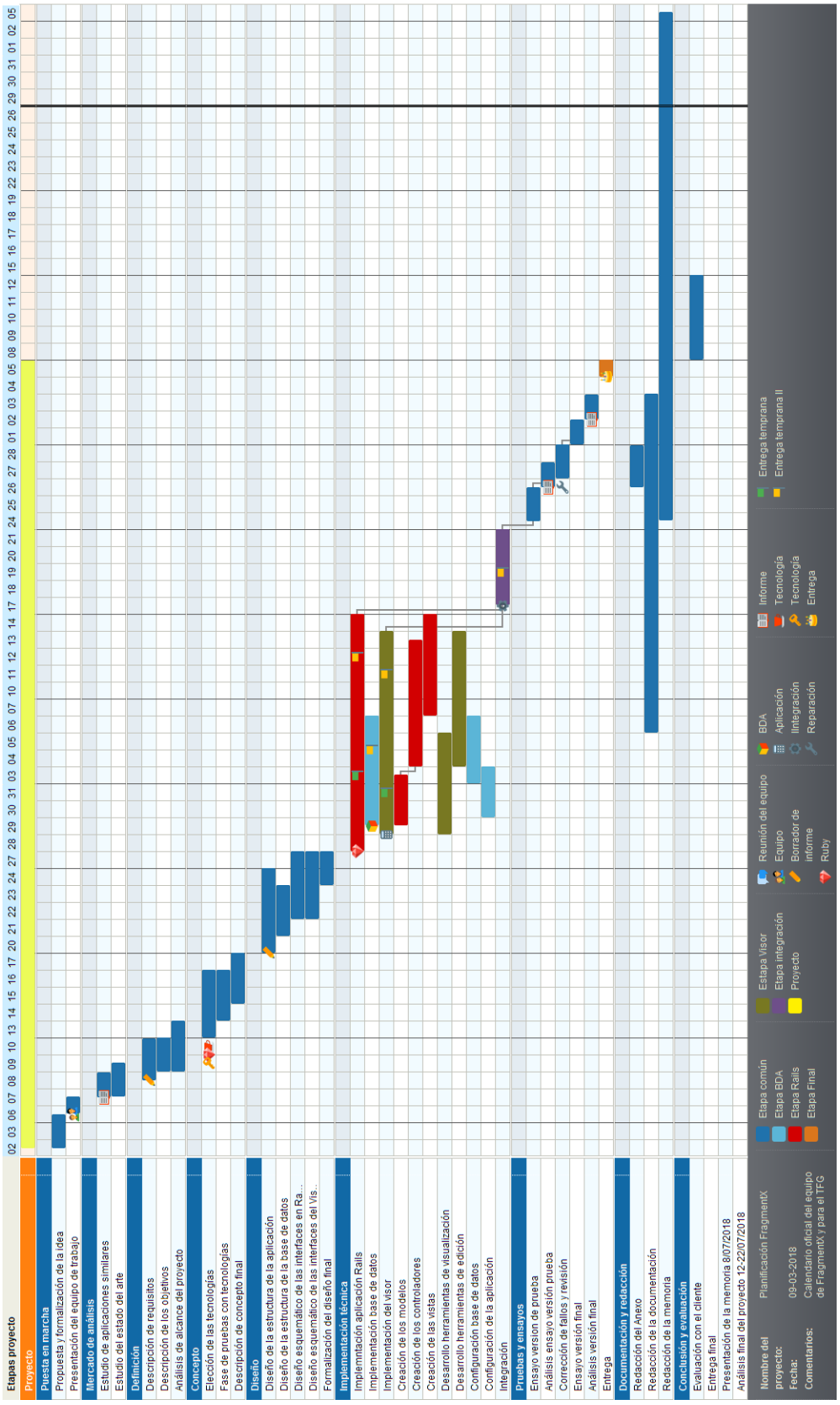


Figura 8. Diagrama de Gantt con la planificación del proyecto

3.3 Planificación económica y de recursos

Todo proyecto conlleva una serie de gastos los cuales implican un desembolso inicial por parte del equipo de desarrollo. Es importante ser conscientes del coste económico de los materiales y recursos utilizados, para poder formalizar un presupuesto que agrade para ambas partes.

Se tendrán en cuenta factores reales como, el coste promedio de desarrolladores en *Rails*, precio de sistemas de almacenamiento y servidores, recursos de coste fijo y variable entre otros factores.

El puesto a desempeñar necesita de un desarrollador *Full-Stack* que tenga conocimientos en *Rails* y con experiencia en proyectos previos. El salario medio de este tipo de trabajadores se encuentra en España aproximadamente desde los 14.000 hasta los 27.000 con experiencia de 1 a 4 años a tiempo completo. Esta estimación se ha realizado en base a distintas ofertas de trabajo vistas para proyectos similares combinado con los datos extraídos de una plataforma web que suministra datos de esta índole. Este trabajador realizará una jornada parcial.[16]

Se estima que el coste por hora dedicada por el trabajador será de 13,2 euros resultando un total a lo largo de los 45 días de 2376 euros. Este necesitará un entorno de desarrollo para realizar las pruebas y la implementación. En este caso el desarrollador posee su propia infraestructura y espacio de trabajo para realizar las tareas. Por lo tanto, se establecerá un precio de alquiler por los bienes empleados.

Los bienes serán un portátil de desarrollo para la implementación, un servidor en la nube para realizar las pruebas de despliegue, un almacén de datos ya que es necesario para cumplimentar los requisitos especificados y un despacho habilitado con conexión a internet y una estación de trabajo.

Durante el periodo de tiempo que se trabaje en el proyecto, el uso de las infraestructuras no queda exclusivamente restringido al mismo proyecto, por lo que se aplicará una parte proporcional al coste y no la parte integral. El coste total de todo el material mencionado con los gastos no fijos que conlleve se recoge en la siguiente tabla en la figura 9. Al final de esta se encuentra el presupuesto total del proyecto.

Presupesto final de la aplicación					
Personal	precio/hora	horas/dia	dias		
	13,2	4	45	TOTAL:	2376
Material	PC desarrollo	Servidor de desarrollo	Material oficina		
	120	30	20	TOTAL:	170
Espacio de trabajo	Infraestructura básica	Aproximación de gastos variables (luz/conexion)			
	200	20		TOTAL:	220
Total					
				TOTAL:	2766

Figura 9. Presupuesto del proyecto

Los presupuestos de mantenimiento posteriores se pactarán una vez se realice la entrega de la implementación final del software y esta se despliegue en el sistema de alojamiento proporcionado, si finalmente se requiere.

[16] Glassdor, Auto generated Rails FullStack dev salary, 2018, https://www.glassdoor.com/Salaries/ruby-on-rails-full-stack-developer-salary

4 Desarrollo del proyecto

4.1 Herramientas utilizadas

Hemos utilizado *Ruby on Rails (RoR)* como estructura base del sistema desarrollado. Se ha elegido esta opción frente a otras como *Django*, *Hibernate* o *Node* debido a que ya se tenía experiencia con el *framework* y se habían realizado proyectos similares con él. Además, *Rails* es seguro y eficiente y ofrece una serie de comodidades al programador que le hacen centrarse más en la aplicación y menos en tareas triviales como es la configuración.

RoR es un *framework Open Source* escrito en *Ruby,* enfocado a la productividad y comodidad del desarrollador, está basado en lenguajes como *Perl*, *Python o Eiffel*. Este leguaje es tipado dinámicamente y con él se abordan diferentes paradigmas (Orientado a objetos, Scripting etc.). Su sintaxis está diseñada para que sea entendible por cualquier persona. [17]

Entre las utilidades que ofrece, se destaca que *Rails* posee una serie de comandos para la consola con los que permite crear los modelos, controladores y *tests* entre otras cosas. Desde ella también se pueden consultar los datos almacenados en la BDA y modificarlos a tiempo real.

Ruby on Rails automatiza las tareas de creación y gestión de la base de datos, configuración y comunicación con servicios de terceros (como el servidor de Amazon S3), desarrollo de las interfaces y paso de datos al visor realizado en *JavaScript*. Ofrece un sistema muy intuitivo de despliegue y puesta en producción.

La intención de realizar esta aplicación multiplataforma y web condicionó la realización del visor utilizando *JavaScript*, ya que la mayoría de los navegadores lo soportan. Este lenguaje de programación era conocido y se había utilizado en trabajos anteriores. Estas dos razones fueron determinantes para la elección de no solo el lenguaje, sino la arquitectura entera del proyecto, ya que la librería *ThreeJs* funciona sobre este.

La versatilidad y agilidad de *JavaScript* ha permitido un desarrollo eficiente del visor, en un tiempo limitado y generando unos resultados excepcionales. Al ser un estándar universal y tener tantos *frameworks desktop* compatibles como *Electron*[18], no se necesitaría reescribir el código en otro lenguaje para realizar la exportación a una aplicación de escritorio. Simplemente realizando una pequeña adaptación del *script* sería suficiente.

Como alternativa a *JavaScript* encontramos *WebAssembly*[19], este lenguaje de programación está basado en un sistema de instrucciones binarias para una máquina virtual basada en pila. Obtiene unos resultados muy buenos en cuanto eficiencia y optimización de tiempos de carga. Sin embargo, se descartó su uso por estar en una fase temprana de su desarrollo y necesitábamos una aplicación estable.

[17] Adrian Mejia, *RoR* Architectural Design, 2011, https://adrianmejia.com/blog/2011/08/11/ruby-on-rails-architectural-design
[18] Riot Williams, Bringing Your Web Application to Desktop Life with Electron, 2016, http://rion.io/2016/12/02/bringing-your-web-application-to-desktop-life-with-electron
[19] Daniel Simmons, "Get started with WebAssembly", 2018, https://medium.freecodecamp.org/get-started-with-webassembly

ThreeJs es una librería escrita en *JavaScript* que permite la visualización, manejo y operaciones sobre objetos 3D. Proporciona una interfaz para la utilización de sus métodos. *ThreeJs* funciona sobre *webGL*, esta es una interfaz de aplicación que, mediante una serie de instrucciones puede ejecutar las directivas gráficas en la GPU de nuestro ordenador, explotando recursos hardware que aumentan enormemente la eficiencia. Así pues, podemos combinar la representación de nuestros objetos tridimensionales con un diseño elaborado el lenguaje de marcas HTML.

HTML y CSS, imprescindibles para realiza el diseño de la interfaz, son estándares web utilizados en todas las aplicaciones que encontramos en la red. Ofrecen las herramientas con las que construiremos nuestra interfaz. Utilizaremos como apoyo una librería llamada *Bootstrap* que tiene implementados diversos diseños adaptativos para páginas web. Proporcionará una mejor adaptabilidad al diseño ya que está creada específicamente para ello.

Como candidato alternativo a CSS se tuvo en cuenta a SASS, su sintaxis era más cómoda de utilizar y los archivos que potencialmente se podrían realizar serían de menor tamaño. Esta opción se descartó porque se quería optimizar al máximo la eficiencia del servidor y convertir los archivos de SASS a CSS era una tarea que podíamos ahorrar al servidor. Por esta misma razón no se utilizó *CoffeeScript* en sustitución a JS, en última instancia *Cofee* se transpilaría a *JavaScript* cargando con una tarea extra al servidor.

Utilizaremos una serie de gemas para realizar distintas funcionalidades en nuestro sistema. Las más destacadas son:

Devise nos permite gestionar de forma segura a los usuarios y proporciona interfaz muy cómoda para realizar las acciones de registro, acceso y recuperación.

Aws-sdk, para poder comunicarnos con el almacén de archivos donde se encuentran las piezas e imágenes asociadas a las figuras.

G*eocoder*, un *API* que usa los mapas de Google para mostrar la ubicación de una coordenada determinada.

Rubyzip, para poder descomprimir archivos zip en el *backend* de nuestra aplicación y poder gestionar el contenido de esta acción.

Gon, que nos permite comunicar los controladores de *Rails* programados en *Ruby* con el script del visor, programado en *JavaScript*.

4.2 Arquitectura del sistema

Rails propicia la utilización de la estructura MVC (modelo vista controlador). Este patrón de diseño separa la lógica (operaciones, filtrados, etc.), su interfaz o vistas y las plantillas con las que definimos los objetos. MVC es muy utilizado para programación web ya que ordena el código de una forma lógica y coherente para que se puedan introducir cambios de forma cómoda y rápida.

Los modelos son las clases que se definirán en la base de datos. Cada uno de los modelos contiene las relaciones que establece cada clase con las demás clases del sistema. Existen tres tipos básicos de relaciones.

La relación uno a uno, es decir, dos objetos se relacionan entre ellos. En *RoR* para declarar este tipo de relación utilizaremos las sentencias: "*has_one*" y "*belongs_to*" dentro de los modelos correspondientes.

La relación uno a muchos, un modelo se relaciona con diversos modelos de una clase concreta. En nuestro sistema podemos encontrar varios ejemplos de este tipo de relación, uno de ellos es la relación entre un usuario y un objeto restaurado. Un usuario puede tener varios objetos restaurados, sin embargo, un objeto restaurado solo pertenece a un usuario.

Figura 10. Relación uno a muchos

Esta relación en *rails* se definiría, añadiendo la directiva "*has_many: restored_objects"* en el modelo de usuario y añadiendo *"belongs_to user"* en el modelo de objeto restaurado.

La relación muchos a muchos, diversos objetos de un mismo tipo están relacionados con diversos objetos de otra clase. Esta relación necesita de una tabla intermedia para poder realizarse. En nuestra aplicación también tenemos ejemplos para este caso. Un material puede pertenecer a muchas piezas, pero una pieza puede estar compuesta de distintos materiales.

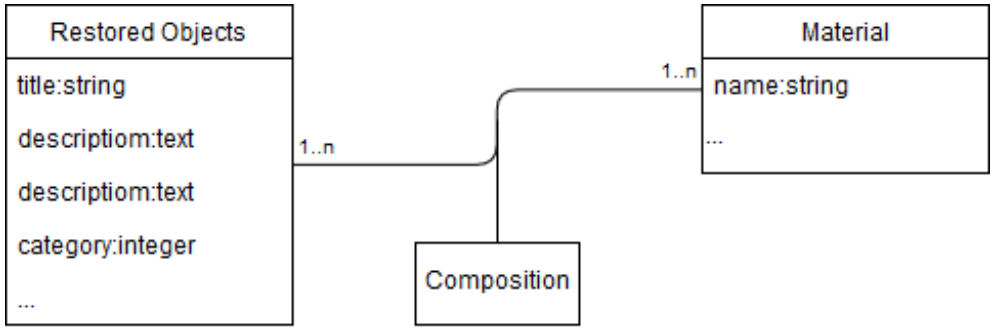


Figura 11. Relación muchos a muchos con tabla intermedia

En este caso como podemos ver tenemos las dos tablas principales unidas mediante una tabla intermedia. En *Rails* la relación se definiría del siguiente modo: desde Objetos restaurados y Material apuntaríamos hacia la clase Composición haciendo uso de la directiva "*has_many: Composition*" y desde Composición apuntaríamos a las dos anteriores haciendo uso de "*belongs_to: Restored_objects*" y "*belongs_to: Materials*". Así que para definir que un objeto

tiene muchos materiales tendremos que crear necesariamente la clase Composición intermedia, esta hará de identificador inequívoco para esta relación.[20]

En *Ruby on Rails*, los modelos también establecen las validaciones de las entradas a la base de datos. Estas sirven para comprobar que los campos cumplen unos requisitos determinados. Un ejemplo sería controlar la presencia del campo "nombre" en la tabla de objetos y que este no supere los 512 caracteres.

El controlador sirve para definir los métodos CRUD, es decir, la creación, destrucción, actualización y lectura de cada objeto que participe en el sistema. En cada uno de ellos se definirán un conjunto de reglas y acciones que se ejecutarán en cada llamada a los métodos.

En él también se debe implementar la lógica que va a ser utilizada en la vista, como la inicialización de variables que contengan los modelos de la base de datos, operaciones de búsqueda etc.

En nuestra aplicación el intercambio de datos entre el visor y el *framework* se efectúa utilizando una gema llamada *gon* que se inicializa en el controlador de lectura de la clase "Objeto restaurado".

Es en el controlador de creación de Objetos restaurados donde se realizan también las acciones de descompresión del fichero con las piezas de la figura, su procesamiento y almacenado en el almacén de datos.

Ambas operaciones mencionadas anteriormente se explicarán en el apartado 4.6.2, donde se ampliará la información sobre los controladores y las vistas. Los modelos se explicarán en el apartado 4.4.

La vista es el espacio de representación donde se va a mostrar la información de los modelos y donde el usuario podrá interactuar con ellos. En nuestro trabajo, está formada por ficheros propios de *RoR*.

Estos archivos permiten que se empotre código *Ruby* en el *HTML* de la vista, pudiendo así realizar webs dinámicas y complejas. Este código se ejecuta antes de servir los datos generado previamente un *HTML* puro que es el fichero que realmente recibe el navegador.

4.3 Diseño detallado

La aplicación se divide en tres grandes componentes que se comunican entre sí y comparten toda la información del sistema, estos elementos son: la base y el almacén de datos, el visor 3D y la aplicación *Rails*.

Aunque estas tres partes estén muy diferenciadas la integración entre ellas es total ya que las tres son imprescindibles para ofrecer el servicio. La aplicación *Rails* se encuentra en la base del

[20] Rails Guides Community, Active Record Associations,2018, http://guides.rubyonrails.org/association_basics.html

producto, se encargará de gestionar la base de datos y ofrecer las vistas, en una de ellas, concretamente en la de objetos restaurados se encontrará el visor.

En este diagrama mostrado en la figura 12 se puede ver de una forma más representativa cómo están conectados el conjunto de herramientas, sistemas y aplicaciones de las que se hace uso para desarrollar el producto final. En la parte inferior izquierda podemos ver la tecnología que está haciendo uso y las flechas señalan cómo se realiza la comunicación entre los diferentes nichos de trabajo.

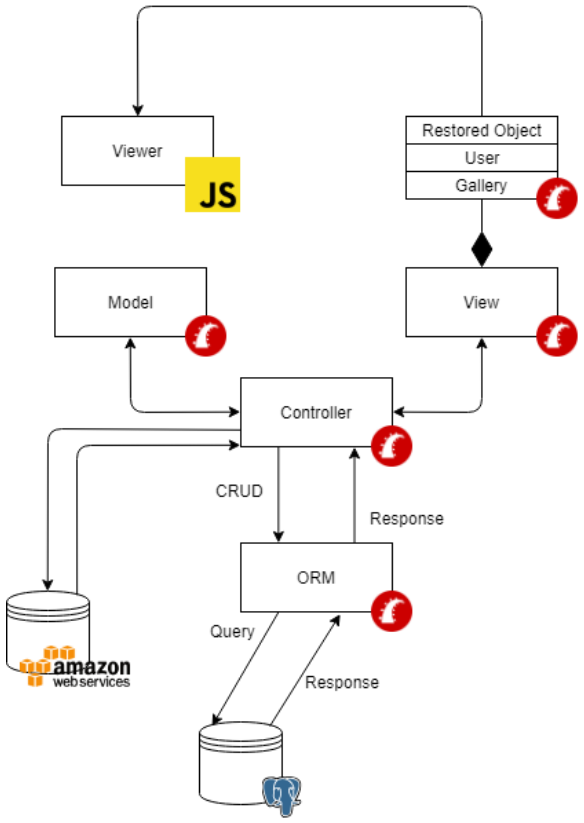


Figura 12. Esquema principal arquitectura de la aplicación

En los siguientes apartados se realizará una descripción más detallada sobre los aspectos más específicos *PostgreSQL*, la aplicación *Rails* y visor.

4.4 Creación de base de datos: modelos y servidor

Antes de comenzar las descripciones más detalladas del sistema, cómo funcionan y qué tareas realizan se quiere remarcar que en las secciones (4.4, 4.5 y 4.6) no se encontrarán figuras con implementaciones realizadas en código. Se ha pensado que es más interesante realizar una descripción detallada de la funcionalidad, comentando la tecnología usada y algunos aspectos técnicos.

Todo el código de la aplicación se puede ver, descargar y compartir sin ninguna restricción en el repositorio del proyecto, que es el siguiente(https://github.com/Duxy1996/eduproject).

Se debe aclarar que hay que realizar una diferenciación entre la base de datos y el almacén de datos. El primer elemento alberga el conjunto de datos simples que el usuario proporciona a la aplicación, normalmente suelen estar guardados en tablas y se accede a ellos a través de un ORM, en el caso de *Ruby on Rails*, es *Active Redcord*. En nuestra aplicación estos datos serán el conjunto de campos que contienen la información de cada objeto. El almacén de datos, guarda elementos más complejos y pesados que no son fáciles de serializar, como las piezas en formato ply de nuestro proyecto. La gestión para el almacenamiento de estos datos y su posterior recuperación deberá de ser realizada por el desarrollador.

Todos los modelos creados en *Rails* están relacionados con una clase que está almacenada en la base de datos. En los modelos se definen las relaciones entre las clases como se ha explicado en el apartado 4.2. En esta aplicación los modelos creados han sido los siguientes:

- Objeto Restaurado
- Categoría
- Colección
- Deterioros
- Materiales
- Piezas
- Prioridades
- Estados
- Usuarios

Para las clases deterioros, colecciones y categorías se han establecido tablas intermedias para poder establecer las relaciones muchos a muchos.

La base de datos utiliza *PostgreSQL*. Se eligió esta alternativa frente a otras debido a que su desarrollo y su implementación es libre, cumple con el principio ACID (atomicidad, consistencia, aislamiento, durabilidad), cumple el estándar SQL en gran parte, tiene un rendimiento excelente y se integra fácilmente con *Rails*. Otros candidatos fueron MySQL, que se descartó por pertenecer a Oracle y no respetar el principio ACID en algunos casos de uso y SQLite, que se descartó por su sencillez y poca escalabilidad.[21]

Rails abstrae las peticiones a la base de datos mediante *Active Record*, el cual implementa métodos para los accesos. Ofrece todos los métodos para realizar las operaciones CRUD, además de operaciones de búsqueda, filtrado y comparación.

El almacén de datos está alojado en Amazon S3, fue elegido porque es económico y rápido, pero el factor decisivo fue que ya se tenían proyectos albergados en ese espacio y se conocía la gema escrita en *Rails* AWS, que permite su comunicación. Los servicios de Google de almacenamiento son muy eficaces, pero un poco más costosos económicamente.

El *framework* que estamos utilizando tiene una serie de comandos por consola que ayuda al programador a crear los modelos para la base de datos. Se puede realizar de forma progresiva, sin tener que generar toda la base de datos en el mismo momento, incluso se pueden añadir campos en una clase ya creada.

Es en los modelos donde se establecen las relaciones como se ha explicado en el apartado 4.2 “Arquitectura del sistema”. Es necesario el uso de un esquema para describir el resultado final de

[21] 2ndQuadrant, PostgreSQL vs MySQL, 2015, https://www.2ndquadrant.com/es/postgresql/postgresql-vs-mysql

la base de datos, en este diagrama UML[22] se puede ver toda la estructura, se han reducido los campos en las clases, éstos se pueden ver de forma completa en el anexo de especificación de la base de datos 10.

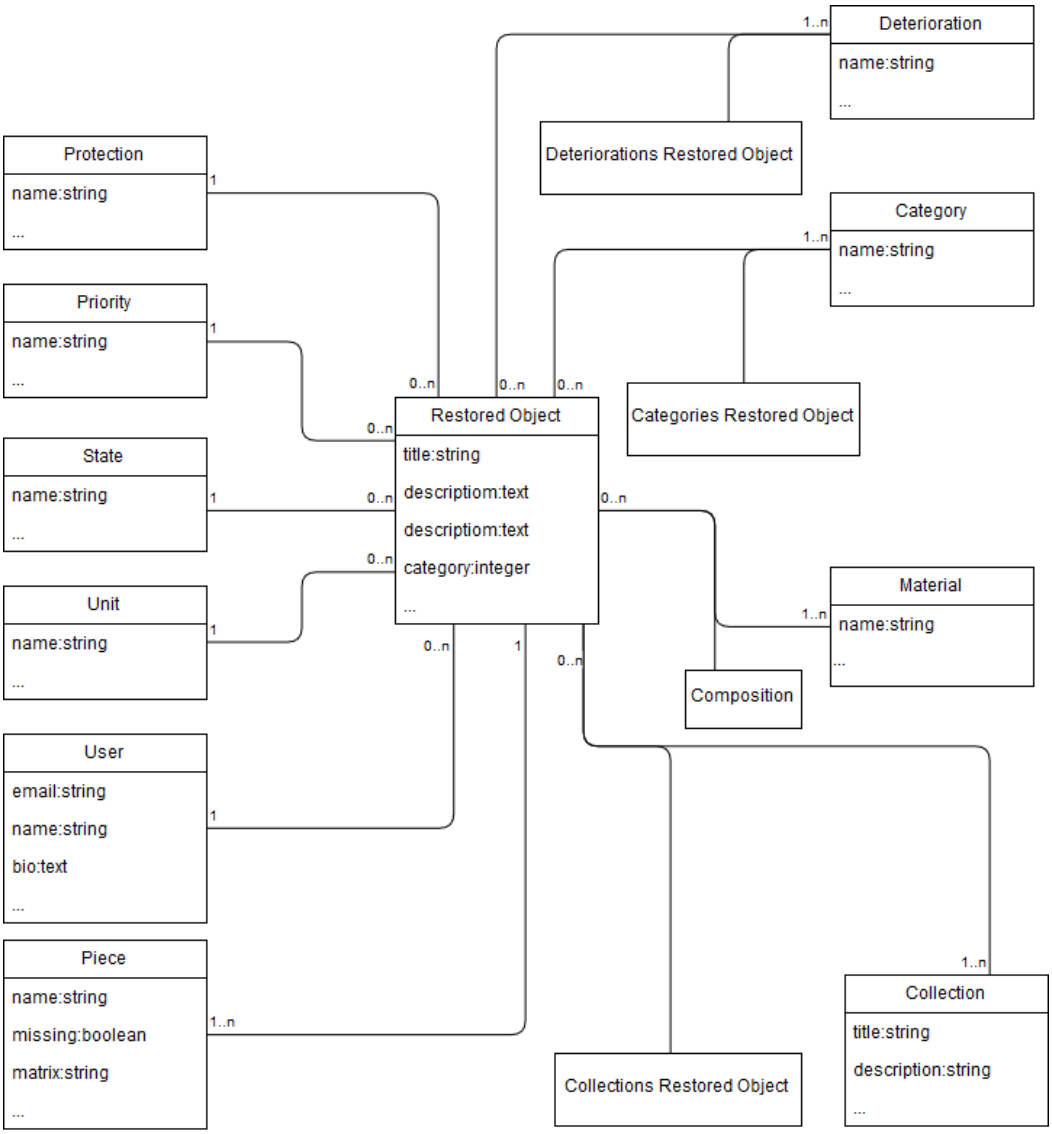


Figura 13. Esquema principal de la base de datos

Como podemos percibir, la base de datos tiene como principal actor la clase *Objeto Restaurado,* todas las demás clases tienen alguna relación con ella. Sin embargo, su no existencia no condiciona a que no puedan almacenarse otros datos como colecciones, materiales o usuarios. Para que existan piezas sí debe de existir obligatoriamente una instancia de objeto restaurado.

Las relaciones que tienen como identificador final 1 implican obligatoriedad en el modelo, por ejemplo, una pieza tiene que tener un usuario. De lo contrario habrá un 0. El sufijo (..n) indica que puede contener o pertenecer a contener varias instancias de una clase.

[22] Donald Bell, An introduction to the UML, 2003, https://www.ibm.com/developerworks/rational/library/769.html

4.5 Implementación del visor con *JavaScript* y *ThreeJS*

4.5.1 Carga de piezas

ThreeJs ofrece una serie de métodos llamados *loaders* [23](cargadores), que permiten de una forma muy simple cargar en el *canvas* los archivos que contienen los objetos 3D. Los métodos *loaders* no se encuentran en la librería nativa de *ThreeJs*, pero se pueden obtener del repositorio oficial y añadir igual que el archivo principal de *ThreeJs*.

Para cargar estos ficheros con las figuras 3D lo único que necesitamos es saber dónde se encuentran. Esta información se añade como atributo al método *load* y este se ejecuta. Una vez cargado el modelo se inicializa su función *callback* donde realizaremos las operaciones oportunas para añadirlo a la escena y operar con él.

Esta función se encuentra encapsulada en un archivo independiente y que tenemos que importar a nuestra aplicación, junto *ThreeJs*. Hacemos uso de este *loader* en la implementación del *worker,* encargado de cargar las piezas. El uso de los *workers* se explicará en la sección siguiente 4.6.2.

4.5.2 Descarga asíncrona con *web-workers*

Cada una de las piezas que conforma una figura tiene un tamaño medio de 25 Megabytes, y cada una de estas puede llegar a tener 12 piezas o incluso más. Esto se traduce en una duración prolongada de la carga en la web, debido a que el tiempo medio de descarga de un archivo de 25 Mb con un ancho de banda de 10 *Mbits*/s es de 8 segundos.

Para agilizar el proceso de carga del visor, se decidió utilizar un sistema de descarga asíncrona de las piezas, realizado en *JavaScript* y que ejecutará el cliente. Este sistema es conocido como *web-workers* [24] y funcionan como hilos independientes de peticiones, que descargan de forma independiente las figuras. La definición de nuestros *workers* se encuentra en la carpeta *js*, dentro de *public*. Tiene la simple tarea de recibir la URI (identificador uniforme de recursos) donde se encuentra la pieza que queremos visualizar, descargarla y servirla al hilo de ejecución principal. Este aplicará las acciones pertinentes para poder visualizarla. Nuestro sistema lanza un *worker* por cada pieza que necesita descargar, teóricamente con esta arquitectura todas las piezas se descargan a la vez, teniendo un tiempo de carga máximo equivalente al tiempo de carga de la pieza más grande. Hemos podido comprobar que esto no es así y que depende de factores muy diversos, tales como la topología de la red, versión de navegador o la máquina en la que se ejecuta.

Realizando una pequeña prueba experimental se hace latente que la mejora de esta forma de carga se hace presente cuando la figura tiene varias piezas que cargar. Adjuntamos a continuación

[23] ThreeJs Community, Loading 3D models, 2018, https://threejs.org/docs/#manual/introduction/Loading-3D-models
[24] Matilde Rocha Aguilera, Introducción a los Web Workers, 2017, https://medium.com/techwomenc/introducci%C3%B3n-a-los-web-workers-f8d44b7ad6e

los resultados obtenidos con las pruebas realizadas antes y después de la implementación de los *web-workers* en la figura 14.

				Piezas	1	2	5	11
Tipo	Pin	Descarga		Medida				
Asíncrono	14	8Mb/s		Segundos	3,06	5,54	12,02	27,35
Síncrono	26	8Mb/s		Segundos	2,47	4,48	16,37	48,12

Figura 14. Tabla comparativa de tiempos de descarga

El navegador *Google Chrome* nos ha permitido realizar un análisis de la descarga asíncrona de las piezas. En la imagen se puede ver una lista con las comunicaciones que ha realizado nuestra aplicación. Cada llamada a los *workers* se corresponde con la aparición del nombre *PLYLoader.js*, ya que este es el nombre del archivo donde fueron implementados.

Se puede visualizar el tiempo de descarga de las piezas y su orden de llegada si nos fijamos en los archivos que comienzan por *cake_part.* El tiempo se encuentra en la columna *Time*, y también está representado en color azul en la última columna. Este tiempo simplemente es el de descarga, el intervalo total para que se pueda visualizar, es la suma de esta más la duración del procesamiento de la pieza en el *canvas*.

Al realizarse todas las peticiones a la vez, los archivos llegan desordenados, independientemente de su nombre o posición. El sistema es capaz de gestionar esta situación ya que cada *worker* tiene preasignada la pieza que tiene que procesar.

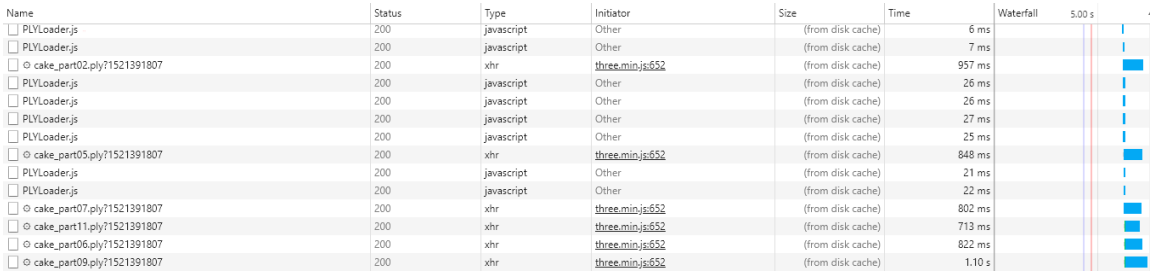


Figura 15. Gráfica de descarga en Google Chrome

También podemos generar un diagrama temporal, en el que se muestra exclusivamente el tiempo que han tardado las peticiones en realizarse y las piezas en descargarse.

El tiempo de las peticiones se muestra con los colores verde y gris. El color azul es para representar la duración de la descarga de cada pieza. En esta gráfica se puede ver con claridad como hasta 6 piezas se han descargado a la vez.

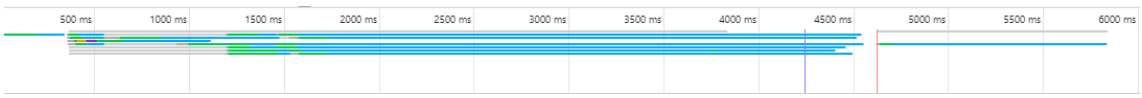


Figura 16. Representación temporal de descarga en Google Chrome

4.5.3 Herramientas de visualización

Se han considerado opciones de visualización todas aquellas que en su uso no implique el cambio en la estructura del objeto 3D. Con ellas podemos realizar acciones sobre la cámara, posición de la pieza, o presencia de la misma.

La implementación de todas las herramientas se ha realizado en *JavaScript* y se encuentra en la vista del modelo "*Restored Objects*". A continuación, realizaremos una breve descripción de cada una de ellas y explicaremos cómo se ha desarrollado de una manera clara y en alto nivel.

Mostrar todas las piezas del objeto. Esta función permite visualizar todas las piezas que componen la figura subida por el usuario, tanto las que son faltantes como las que no. Una pieza faltante es aquella que se ha reconstruido digitalmente, una no faltante es la que ha sido escaneada. Existe un campo en inicializado *PostgreSQL* que permite diferenciarlas.

Para implementar esta funcionalidad, se decidió crear dos listas qué almacenaban las piezas faltantes y no faltantes. Las cuales están replicadas, una como objeto *ThreeJS* y otra como un array normal de *JavaScript*. Esta estructura nos permitirá agilizar procesos futuros.

Un objeto *ThreeJS* puede funcionar como una lista, ya que se le pueden añadir otros objetos *ThreeJS* como: figuras, piezas y texturas. Todas las propiedades que se apliquen al objeto padre serán heredadas por los objetos hijos. Cuando nos refiramos a lista-objeto durante este texto, nos referiremos a la propiedad de acumular otras entidades que tienen los objetos.

Después de inicializar las listas anteriormente mencionadas, añadimos las listas-objeto a la escena creada. Esta escena es el lugar de representación donde posteriormente se mostrarán los objetos 3D.

Tras la carga asíncrona realizada con los *web-workers* se aplican las funciones de trasformación pertinentes para la ubicación de la pieza en el lugar correspondiente a la matriz de transformación. Después de esto se consulta la base de datos mediante una gema llamada *gon*[25], que nos proporciona la información necesaria para establecer si una pieza es faltante o no faltante. Con esta información podemos decidir en qué array meter la pieza descargada.

El simple hecho de añadir las piezas a las listas-objeto, que a su vez se han añadido a la escena, da como resultado que las piezas se muestren en el navegador donde se está ejecutando la aplicación.

Para hacer desaparecer las piezas simplemente eliminamos las listas-objeto que contienen las piezas (tanto las faltantes como las no faltantes) de la escena y todos los objetos se eliminarán de la interfaz.

Mostrar las piezas escaneadas o reconstruidas, con esta funcionalidad, el usuario puede seleccionar si desea visualizar las piezas faltantes o no faltantes individualmente. Si se quisieran volver a visualizar todas en conjunto otra vez, utilizaríamos la función anterior.

Para que esta función sea desarrollada correctamente es necesario un campo en la base de datos que proporcione la información pertinente de la pieza como se ha explicado en el apartado anterior. *Rails* se comunicará con el visor mediante la gema *gon* que proporciona el dato de clasificación (faltante, no faltante). Esta función de comunicación está explicada en el apartado 4.6.2 en la sección de controladores.

[25] Alexey Gaziev , Gon gem — get your Rails variables in your Js ,2018, https://github.com/gazay/gon

Esta implementación es una ampliación de la funcionalidad "mostrar todas las piezas de un objeto". Comparten el código de inicialización. Cuando se seleccione una de las opciones para mostrar el conjunto de piezas deseado, se añade la lista-objeto seleccionada a la escena y se elimina la complementaria. Esto permite mostrar solo los objetos de la lista seleccionada. Esta acción no implica un borrado, es una desindexación, siendo eficiente el intercambio entre vistas.

Visualizar las piezas que rodean a una seleccionada, con esta acción permitimos al usuario seleccionar una pieza determinada y que se muestren las piezas de su alrededor. Esta función es útil para realizar una exploración parcial del objeto.

Al seleccionar una pieza, se mostrará de un color diferente al resto. Cuando realicemos la acción de "Mostrar piezas cercanas", las piezas que no estén alrededor de la seleccionada se verán con una textura que tendrá la opacidad reducida. Podemos ver ese efecto en las figuras 17 y 18.

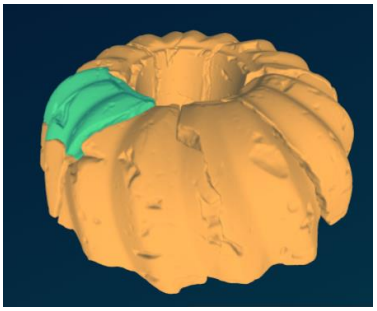


Figura 17. Modelo vista con pieza seleccionada

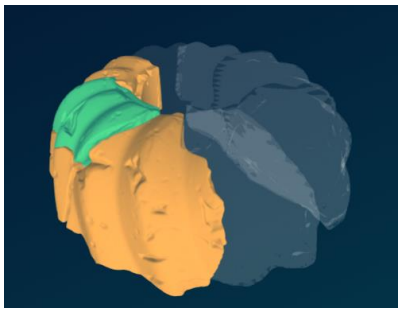


Figura 18. Modelo donde se muestran las piezas que bordean a la seleccionada

Podemos diferenciar dos acciones en esta funcionalidad: la acción de selección y la visualización de las piezas cercanas. La primera de estas se utilizará en más métodos a lo largo de la aplicación, pero es en este primer apartado donde realizaremos su explicación.

Cuando se realiza una selección en una zona cualquiera del *canvas* (parcela de representación de los objetos en el navegador), se ejecuta un método *listener* que realiza una serie de comprobaciones mediante código para seleccionar los objetos pertinentes dependiendo del modo en el que se encuentra el visor y el punto escogido.

En este método se comprueba que la selección no haya durado más de 300 milisegundos, ya que esto implicaría que se está realizando una acción de translación de la cámara y realmente no se desea seleccionar el objeto. Posteriormente y si la duración es menor de 300 milisegundos se crea un objeto rayo, que va desde la cámara hasta la posición en el infinito marcada con el ratón. Si este rayo intersecta con algún objeto lo seleccionaremos, para realizar esta operación se tiene una variable global que almacena la dirección del objeto al que se está haciendo referencia. Si no se ha seleccionado nada, el objeto se deselecciona de forma automática, borrando la indexación del objeto y consecuentemente siendo deseleccionado. Solo se puede seleccionar un objeto.

Una vez se tenga seleccionado el objeto, se puede ejecutar la función de "mostrar las piezas cercanas" ya que, si no hay nada seleccionado, esta acción no tendrá ningún efecto.

Cuando se presione el botón para mostrar las piezas colindantes, se comprobará pieza por pieza cual es su posición respecto a la seleccionada utilizando sus cajas de contención. Si estas no intersectan, es decir, no son colindantes, se les aplicará la textura con opacidad reducida.

Rotar el objeto. Al aplicar esta funcionalidad, el objeto en la pantalla rotará sobre su propio eje de forma uniforme y continua. Este modo permite al usuario ver toda la figura sin necesidad de mover la cámara, es un modo de presentación. Al inicio de la ejecución, este comenzará en reposo.

ThreeJS permite de forma sencilla aplicar trasformaciones sobre cualquier objeto. Los objetos hijo que este contenga heredarán este cambio, por lo cual, al aplicar una rotación a las listas-objeto que contienen nuestras piezas todas ellas rotarán.

Durante la acción de renderizado, que permite el refresco de la pantalla, se evalúa una condición que valida si nos encontramos en el modo de rotación. Si es así, las listas-objeto rotarán haciendo que las piezas giren alrededor del eje Y, hasta que se desactive esta función.

Situar la cámara en el eje XZ, XY y ZY permite al usuario situar su punto de vista de forma precisa en los tres ejes, pudiendo visualizar de forma sencilla el alzado, la planta y el perfil. Además, sirve para llevar al punto de origen la cámara si el usuario ha perdido la vista del objeto durante el uso del visor.

Esto es posible gracias a que en *ThreeJS* existe un objeto cámara que se declara al inicio del *script* y que podemos manejar como deseemos. Entre otras opciones, podemos comunicarle en qué posición queremos que se encuentre y hacia qué punto en el espacio debe de mirar, construyendo así el vector conocido como *lookAt*.

Para situar la vista en el plano XY, el punto X de la cámara tiene que estar en 30, el Y en 0 y el Z en 0. Posteriormente aplicaremos el punto hacia donde queremos que enfoque la cámara, X = 0, Y = 0 y Z = 0. Para las demás vistas se realiza el mismo procedimiento cambiando el punto el que se sitúa la cámara, estas vistas se mostraran como se pueden ver en las figuras 18,20 y 21.

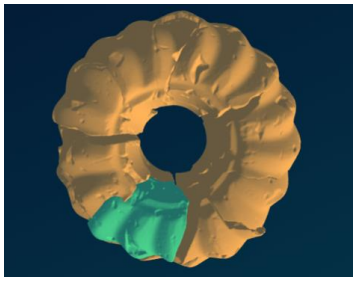


Figura 19. Vista plano XZ

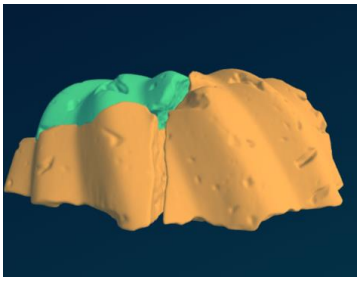


Figura 20. Vista plano XY

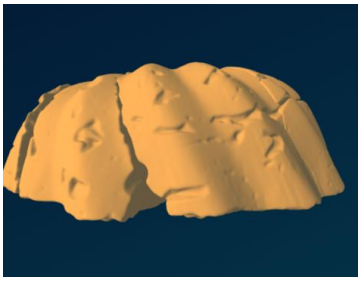


Figura 21. Vista plano ZY

4.5.4 Herramientas interactivas

Las tres vistas anteriores se relacionan directamente con los planos de corte que explicaremos en este apartado.

Esta implementación está situada en el mismo lugar y sigue las mismas características que hemos comentado en el apartado 4.5.3. Ambas partes fueron programadas durante el mismo periodo de tiempo. Explicaremos estas herramientas como en el apartado que ya se ha mencionado.

Cortar el objeto con un plano nos permite seccionar la figura a lo largo del eje seleccionado mostrando su parte interior. El control de la posición del plano de corte se realiza utilizando una barra de desplazamiento lateral. Para cada uno de los platos existe un *slider*.

Un *slider* es una barra que contiene un punto de referencia que se puede mover a lo largo de su longitud aportando un valor comprendido entre un rango.

Para implementar esta funcionalidad al inicio del script se inicializan los tres planos (XZ, XY y ZY) de corte en un vector y otro array con valores booleanos para saber cuál de ellos ha sido activado. Estos planos de corte se añaden al material que forma parte de las piezas en su creación. Así es como *ThreeJS* permite que luego se realice el corte, ya que asocia el plano a la pieza.

Al hacer *click* en los botones de la interfaz que se encuentran al lado de los *sliders*, se ejecuta el método que actualiza el estado de activación de los planos según el botón que se haya seleccionado, actualizando los valores del vector de booleanos.

Este método conoce exactamente qué botón ha sido pulsado ya que recibe un parámetro con un número entero que hace referencia al botón pulsado (X = 1, Y = 2, Z = 3). A continuación, tras la actualización comprueba si el plano seleccionado tiene que ser cortado, en caso negativo, el corte se redefine en un lugar no visible y la tarea termina. De lo contrario, si el booleano es cierto (*true*), adquiere el valor de su deslizador correspondiente y posiciona el plano en el punto adquirido.

Cada vez que el deslizador se mueve, se realiza una llamada que actualiza la posición del plano, pudiendo elegir así qué parte de la pieza será seccionada.

En estas imágenes podemos observar desde una vista isométrica y desde el plano XZ, cómo se ha realizado un corte en el eje X sobre la pieza. Podemos llegar a ver el relieve en el interior de la figura, como ose muestra en las imágenes.

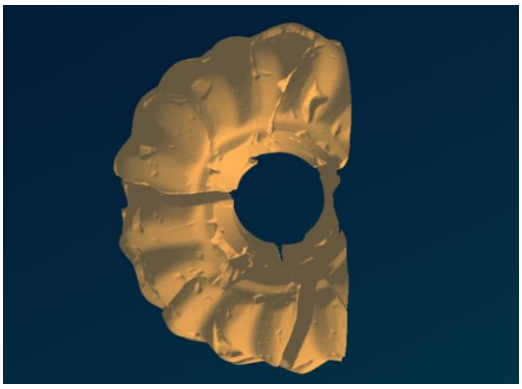


Figura 23. Vista superior primer corte

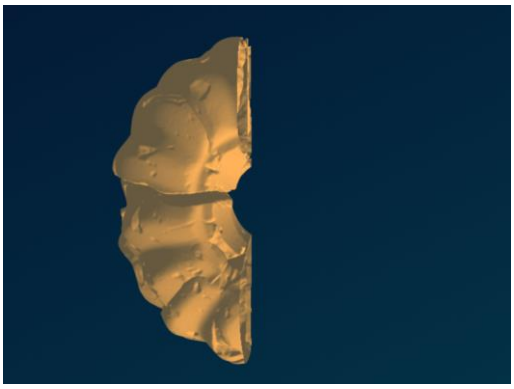


Figura 22. Vista superior segundo corte

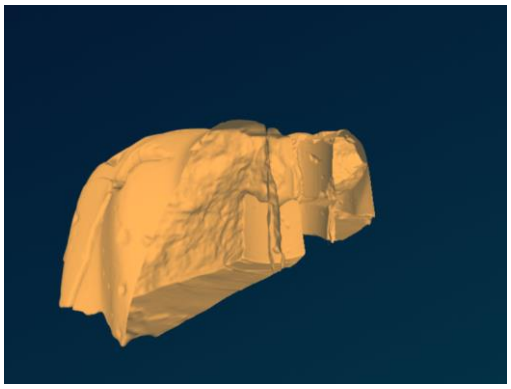


Figura 25. Vista interior primer corte

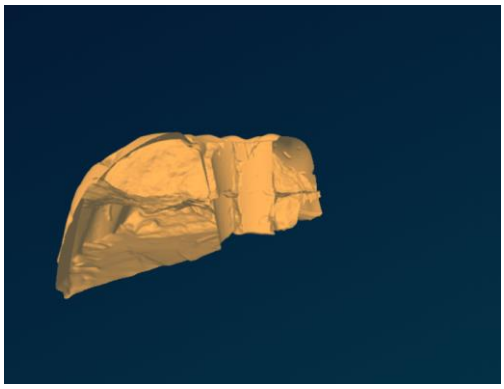


Figura 24. Vista interior segundo corte

Medir la distancia entre puntos del objeto es útil para conocer la longitud exacta entre dos o más puntos de la figura y calcular perímetros. Con esta herramienta seremos capaces de seleccionar una serie de puntos para calcular la distancia, en línea recta, entre los adyacentes. La distancia medida, es acumulada, por lo que podemos adaptarnos al contorno de la figura.

La precisión de la herramienta depende del usuario y de la cantidad de puntos que quiera añadir. Si los puntos están juntos se adaptarán mejor a la forma, obteniendo un resultado más preciso, sin embargo, si están separados las medidas serán más bastas.

En los primeros instantes de la ejecución del visor se crea una lista-objeto donde almacenaremos las esferas que representan los puntos que el usuario ha seleccionado y una lista donde se almacenan los puntos propiamente dichos. Cuando activemos la función, reescribiremos en una variable global, también declarada al inicio del script, el valor contrario en el que se encuentre en dicho momento. Si al finalizar esta acción esta variable contiene el valor "verdadero" desplegaremos un menú secundario utilizando una animación que simula un deslizamiento hacia la parte superior.

Durante este modo cada vez que seleccionemos algo en el *canvas* del visor se generará un rayo que se dirigirá hasta este punto que partirá desde la posición de nuestra cámara (nuestro punto de vista). Si este rayo interseca con alguna pieza de la figura, se calculará su punto de corte, y este será el punto que el usuario ha seleccionado, se añadirá a la lista de puntos y se colocará una esfera verde en el lugar seleccionado. Esta esfera verde se añade simplemente creando un objeto esfera con unas características predefinidas y añadiéndolo a la lista objeto contenida en el entorno, siendo su punto central el punto de corte que hemos obtenido anteriormente.

Cada vez que el usuario añada un punto se calculará la distancia entre los elementos de la lista utilizando la fórmula de distancias vectoriales. Se ha decidido realizar de esta manera y no de forma incremental para poder eliminar puntos haciendo *click* con el botón derecho sobre ellos, independientemente de su posición.

También podemos eliminar el último punto añadido con el botón que se sitúa en la parte inferior de la interfaz a la derecha dentro del menú desplegable o eliminarlos todos con el botón que se encuentra a la parte izquierda de este. El primer método realiza desencola los objetos de las listas y el siguiente las reinicializa, dejándolas vacías.

Por último se quiere mencionar que durante la fase de selección y una vez calculada la distancia, esta y el número de puntos se muestran en la interfaz accediendo al DOM (*Document Object Model*) mediante la instrucción: `document.getElementById("idDelObjeto").innerHTML.`

En las imágenes 26 y 27, podemos ver la interfaz cómo él muestra antes y después de haber seleccionado un área con siete puntos sobre la figura. En este caso el cálculo es de 21.3 unidades de medida, estas se habrán seleccionado en el momento de subida de la pieza y se ha supuesto un conocimiento previo de la escala antes de su carga a la plataforma.

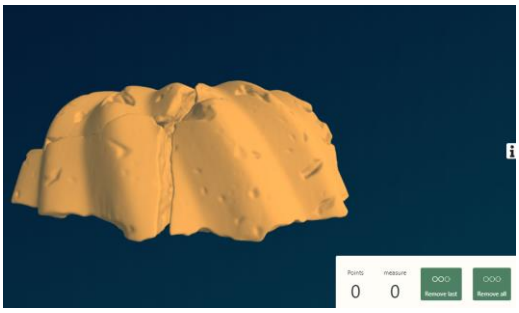


Figura 26. Prueba de medición con cero puntos

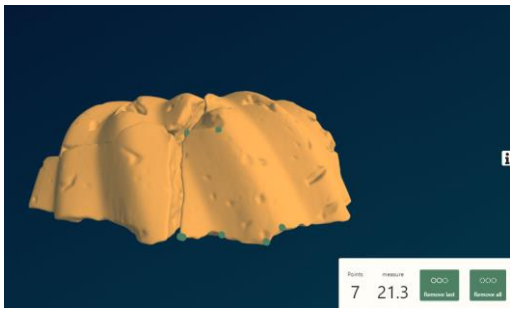


Figura 27. Prueba medición de perímetro

Mostrar el objeto en alámbrico es una forma de conocer detalles estructurales de la figura sin tener en cuenta la textura de la misma. Cuando activamos este modo, se puede visualizar la malla de la pieza, es decir, el conjunto de vértices que forman la figura. Éstos están unidos por aristas formando triángulos. Cambiaremos la textura del objeto por defecto por una que tenga la malla.

Situar un punto de luz en el espacio nos permite añadir una luz focal en diferentes puntos del *canvas*, para poder iluminar la pieza, ver las sombras que proyecta, mejorar la visibilidad de cierta zona, etc.

El foco se inserta simplemente pulsando la zona donde se quiere que se sitúe. Mediante una técnica de posicionamiento, se coloca en el punto correspondiente. Es necesario este control para garantizar que el foco quedará próximo de la figura.

Para realizar este control de posición, se implementaron una serie de tareas que el script ejecuta cada vez que se pulsa para colocar el punto de luz. Al inicio de la ejecución se introduce una esfera con una textura translúcida que envuelve la figura y que nos permite situar el foco. También se activa la proyección de sombras y se habilita la iluminación focal.

Cuando activamos este modo y seleccionamos un punto cualquiera del *canvas,* emitimos un rayo desde la posición de la cámara al punto seleccionado. Esta posición estará usualmente dentro de la esfera anteriormente mencionada. En el punto que el rayo intersecte con la esfera situaremos el punto de luz. Este siempre mirará hacia el centro del *canvas*, es decir, tendrá orientado el sector *lookat* hacia el punto (*X=0, Y=0, Z=0).*

En el momento que se pulse en otro lugar, el foco anterior se eliminará de su posición y se situará en la nueva seleccionada. Si se desactiva el modo de iluminación se eliminará del *canvas* el punto de luz que hayamos puesto y las opciones de iluminación volverán a ser las que se muestran cuando iniciamos por primera vez la aplicación.

A continuación, en las figuras 28 y 29, tenemos el resultado visual tras la implementación y ejecución del método para añadir puntos de luz.

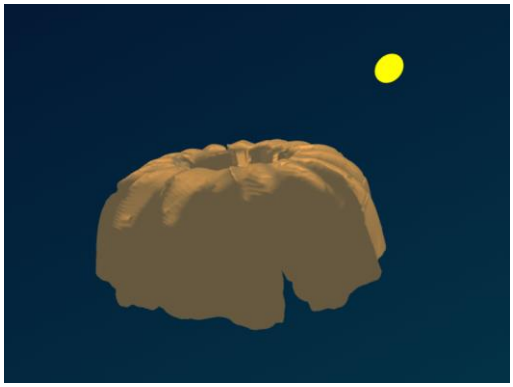


Figura 28. Punto de luz superior en modelo

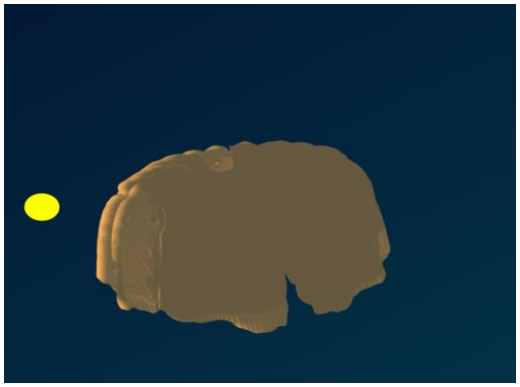


Figura 29. Punto de luz paralelo al modelo

Podemos ver como la sombra se proyecta a lo largo de la superficie de la figura, dependiendo del punto de luz donde se encuentre. Ambas imágenes han sido obtenidas con la cámara en la misma posición.

Realizar una captura de pantalla del *canvas.* Puede ser interesante realizar una captura de la visión que tenemos en el navegador. Para no utilizar una herramienta externa que haga que la imagen obtenida pierda calidad, se ha implementado una herramienta que descarga directamente la imagen que hay en el *ráster* del *canvas.*

Esta acción esta implementada con dos métodos, el primero accede al *canvas* de *ThreeJS* y extrae su contenido en 2D, como si fuera una fotografía con el formato deseado. Una vez tiene esta información la entrega al siguiente método como argumento. Este se encarga de realizar la acción de descarga en el navegador. Crea un nuevo elemento descargable en el que está contenida la imagen. Todos estos procesos son transparentes para el usuario.

4.6 Aplicación *Rails*

4.6.1 Despliegue

Uno de los objetivos que se ha requerido es que el despliegue del producto se realice de la manera más sencilla posible. Se ha considerado oportuno comentar el proceso de instalación del *framework* y la posterior puesta en producción de la aplicación. Se quiere demostrar que este proceso es muy rápido y fácil y cualquier administrador de sistemas lo podría realizar sin problemas sin tener conocimientos específicos de *Rails*.

La mayoría de las plataformas de servicio como *Heroku* o *Hostinger* tienen *Rails* preinstalado por lo que no es necesario realizar todo un proceso de instalación para su puesta en producción si hacemos uso de los servicios mencionados. Sin embargo, para su desarrollo es imprescindible tenerlo instalado en la máquina en la que se está trabajando.

Este proceso tanto en Windows, Linux y MacOs es muy sencillo. En Windows se tiene que hacer uso de los instaladores proporcionados por *railsinstaller.org*. Una vez se tenga instalado esto y la base de datos requerida, en nuestro caso *Postgre,* se podrá realizar el uso del *framework*. Linux y MacOs, obtienen todas las dependencias accediendo a los repositorios oficiales con los comandos *curl* y *apt-get* respectivamente.

Para el despliegue de cualquier aplicación *Rails* se requiere tener el conjunto de archivos que forman parte de la aplicación. Estos pueden ser descargados de los repositorios de código que existen como *GitHub*, donde está alojada la aplicación desarrollada. Existen dos posibilidades para realizar la descarga de esta plataforma: haciendo uso de la interfaz web de *GitHub*, pinchando en el botón de descargar en la vista de la aplicación o mediante línea de comandos ejecutando el comando: *git clone,* atacando al repositorio *https://github.com/Duxy1996/eduproject.git*

Una vez tengamos el repositorio clonado, configuraremos la base de datos que se quiera utilizar (en caso de ser *Postgre*, no hace falta realizar nada) y añadiremos las variables de entorno con las claves para utilizar los distintos servicios de la aplicación como el almacén de piezas (Amazon).

Cuando terminemos la configuración, podremos desplegar la aplicación haciendo uso del comando: *rails start,* que se ejecutará en una consola abierta en el directorio donde tengamos contenida la aplicación. En esta consola se mostrarán todas las peticiones que se hagan al servidor resaltadas en colores según el tipo, esto es muy útil para detectar errores o un funcionamiento anómalo en la aplicación. También se mostrarán registros, accesos a la base de datos y comunicaciones internas de *Rails*.

4.6.2 Vistas y controladores

Los controladores en *RoR* implementan las operaciones CRUD con las que posteriormente el usuario interactuará de forma transparente. Estas estarán relacionadas con una serie de métodos que implementa el controlador y que usualmente se corresponden a vistas determinadas. Estos métodos son los siguientes:

- *Index*, se relaciona con la lectura de todos los objetos referentes a una clase determinada.
- *Show,* se corresponde con la acción de lectura de un objeto en concreto. En este lugar se suele inicializar una variable que apunta al objeto actual.
- *Edit and Update,* en ellos se implementan las operaciones para añadir o actualizar nuevos objetos de la base de datos. Normalmente comparten características o incluso llegan a ser idénticos.
- *Delete,* en él se establecen las acciones pertinentes que hay que realizar para borrar el elemento que se haya seleccionado.

Para que se muestren las vistas en la aplicación hay que añadir sus rutas al archivo *routes* en la configuración. Esto modifica la configuración inicial del servicio para añadir las URL que queremos que se muestren. Todas las rutas se pueden consultar introduciendo el comando *rails routes* en la consola, es muy útil para implementar los links de acceso a otras vistas.

Entre los controladores que tiene la aplicación hay que destacar el de la clase "Objetos Restaurados". En este se implementan todas las acciones para servir la información al visor y para cargar las piezas al almacén de Amazon. En los dos párrafos siguientes explicaremos con más detalles estas acciones.

Cuando el usuario accede a una figura para hacer uso del visor, se realizan una serie de operaciones previas en el método *show* del controlador de "Objetos Restaurados". Se añade al objeto "*gon*" la información de las piezas (url, matriz y valor booleano). Este objeto es el encargado de la transmisión de esta información al visor, que está escrito en *JavaScript* y que nativamente no se puede comunicar con *RoR*. Para acceder al objeto "*gon*" utilizaremos la sintaxis de una estructura en C accediendo a sus campos con un punto después del nombre de la variable.

Al cargar una figura en la plataforma, el método *create* del controlador de objetos restaurados se encarga de descomprimir el fichero que alberga las matrices y las piezas de la figura. Empleando este método no se pueden clasificar. Para relacionar cada matriz con su pieza el fichero comprimido tiene que tener una estructura determinada. Las matrices tienen que tener el mismo nombre de fichero que los archivos que contienen las piezas cambiando la extensión por *".txt"*.

Una vez tenemos los archivos de piezas y de matrices separados, almacenamos de forma rutinaria las matrices en la base de datos y nos comunicamos con Amazon S3 para realizar la subida de archivos con la ayuda de su gema, llamada AWS-sdk.

La aplicación en *Rails* implementada en este proyecto tiene un número de vistas reducido, en comparación a otras aplicaciones realizadas. Esto se debe a que las vistas son realmente un complemento del visor, que hace tener al usuario una experiencia más completa. Estas son el índice principal, galerías por clasificación, página de registro de una figura y la vista de cada figura. La última vista menciona es la que contiene el visor. Implementa lógica en *Ruby* y en *JS*

Las vistas de *RoR* tienen la particularidad de que pueden albergar código escrito en *Ruby* en los documentos HTML para enriquecer su visualización al usuario. De esta manera podemos implementar una lógica mínima en la vista para que la visualización sea más completa y detallada. El diseño creado utilizando ambas tecnologías permite mostrar la información de los modelos en estructuras complejas, como listas y tablas, haciendo uso de la información que estaba almacenada previamente en la base de datos.

Sin embargo, esta utilidad descrita puede llevar a sobre poblar de lógica el visor, cosa que violaría los principios del patrón VMC. Se ha de tener en cuenta que el lugar de la lógica es el controlador y en la vista solo tienen que estar las sentencias indispensables.

5 Producto final

5.1 Interfaz de usuario

La sencillez y la claridad son las características principales de la interfaz diseñada. Esto se debe a que el usuario final en muchas de las ocasiones no será un especialista en el tratamiento de estas

figuras. Sin embargo, el hecho de que sea sencilla no implica una pérdida de potencia en las acciones a realizar.

La integración de *Rails* en el proyecto ha dado la posibilidad de implementar vistas dinámicas que cambien y se adapten conforme a los datos que se tengan almacenados. Este dinamismo es bueno para el usuario ya que implica que tiene un *feedback* inmediato de los cambios que está realizando.

Nos centraremos en la interfaz del visor, ya que es aquí donde ha recaído la mayor parte del trabajo de diseño. Como ya hemos comentado en el apartado 4.1, se han utilizado las tecnologías HTML y CSS para la estructuración y estilo de la página, y *JavaScript* para realizar algunas acciones dinámicas sobre esta.

Los iconos utilizados para obtener una interfaz más intuitiva se encuentran en formato *svg* o *png*. Ambos tipos de archivo permiten la manipulación del canal *Alpha*. Este nos permite cambiar la opacidad de ciertas partes de la imagen. Algunos de los botones han sido realizados por nosotros, pero la mayoría de ellos han sido obtenidos de la web *The noun Project.* Al final del trabajo mencionaremos a los artistas que los han creado.

5.1.1 Interfaz

La aplicación está formada por una serie de vistas con distintas funcionalidades. Realizaremos una descripción de cada una de ellas, y de cómo ha sido diseñada, comentando la serie de elementos que influyen en ellas.

Las interfaces han sido diseñadas haciendo uso de *Bootstrap*, se adaptan a cualquier tamaño de pantalla y navegador, haciendo accesible la aplicación desde cualquier dispositivo.

Podemos ver las vistas de la página principal en un dispositivo móvil con resolución 1920x1080 en la figura 30 y en un Tablet con resolución de 2048x1536, figura 31.

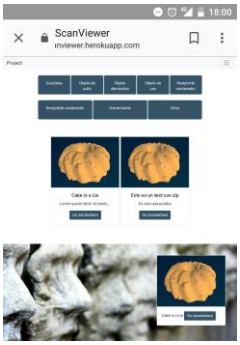


Figura 31. Vista principal en formato móvil

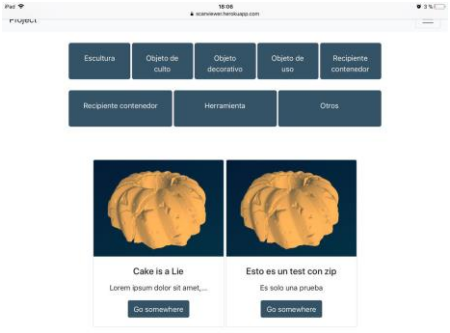


Figura 30. Vista principal en tableta

En todas las vistas nos encontraremos en el encabezado de la página una barra que contiene un menú el cual nos permite desplazarnos a las vistas más importantes de la aplicación simplemente seleccionando sus enlaces.

De izquierda a derecha nos encontramos con las palabras "*Project*", esta posteriormente será sustituida con el nombre o logo que se le quiera dar a la página web y sea suministrado. "*Home*", nos llevará a la página de inicio, esta es la que podemos observar en la figura 38. "Objetos Restaurados" nos llevará a la lista donde se mostrarán todos los objetos guardados en la base de datos, figura 36. "*Sign in*" y "*Log in*" serán para registrarse en la página web o simplemente acceder a la plataforma con el usuario y la contraseña.

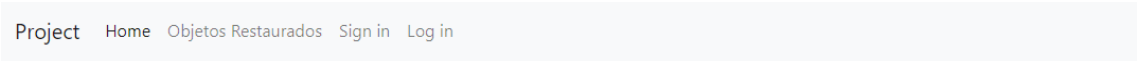


Figura 32. Cabecera principal

Para poder iniciar sesión, *Devise* proporciona una interfaz minimalista que ofrece las utilidades básicas para la gestión de usuarios. Las tres vistas que se muestran a continuación son las que permiten iniciar sesión, crear una cuenta y recuperar la contraseña.

Para crear una nueva cuenta haremos uso de la vista mostrada en la figura 34, en esta tendremos los campos para introducir nuestro correo y la verificación de la contraseña. La figura 33, muestra la vista para iniciar sesión una vez que tengamos nuestra cuenta creada. Al pulsar en el botón de acceder "*Sign up*" o "*Log in*" y si todos los datos han sido correctamente rellenados nos dirigirá a la vista principal.

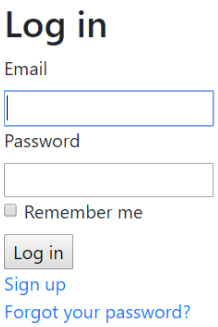


Figura 33. Vista para registrarse en la web

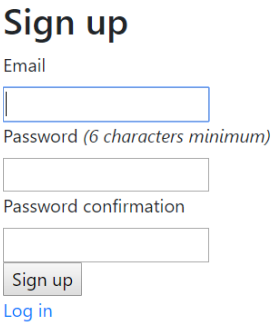


Figura 34. Vista para acceder a la web

Si en algún momento el usuario ha olvidado la contraseña puede recurrir a un formulario de recuperación, en este únicamente tendremos que introducir el correo electrónico con el que se inició sesión por primera vez, como se muestra en la figura 35.

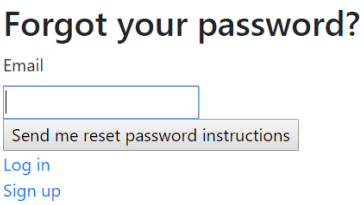


Figura 35. Vista para la recuperación de contraseña

En la página principal, representada en la figura número 36, se muestran las categorías en las que se pueden clasificar los objetos añadidos a la base de datos. Algunos de los objetos se mostrarán en la vista ordenados cronológicamente según su fecha de adición.

Además, se podrá ver un objeto destacado aleatorio en una franja destacada. Debajo de esta sección aparecerían otra fila de tarjetas como las que se muestran encima de la franja resaltada, si en la base de datos existen más de 6 figuras. Para ver todas las figuras se tiene que acceder al índice de objetos restaurados.

Las tarjetas que contienen los modelos tienen su avatar, el título de la obra y su descripción limitada a 30 caracteres. Se añade un botón inferior que redirige a la vista donde se encuentra el visor. Se ha decidido mostrar las categorías en dos filas para no recargar tanto la aplicación y que los textos de dentro de las etiquetas se puedan leer sin problemas.

Tanto la figura 36, como la 37 están integradas en la misma vista, la imagen de fondo usada para la sección destacada es libre de derechos de autor.

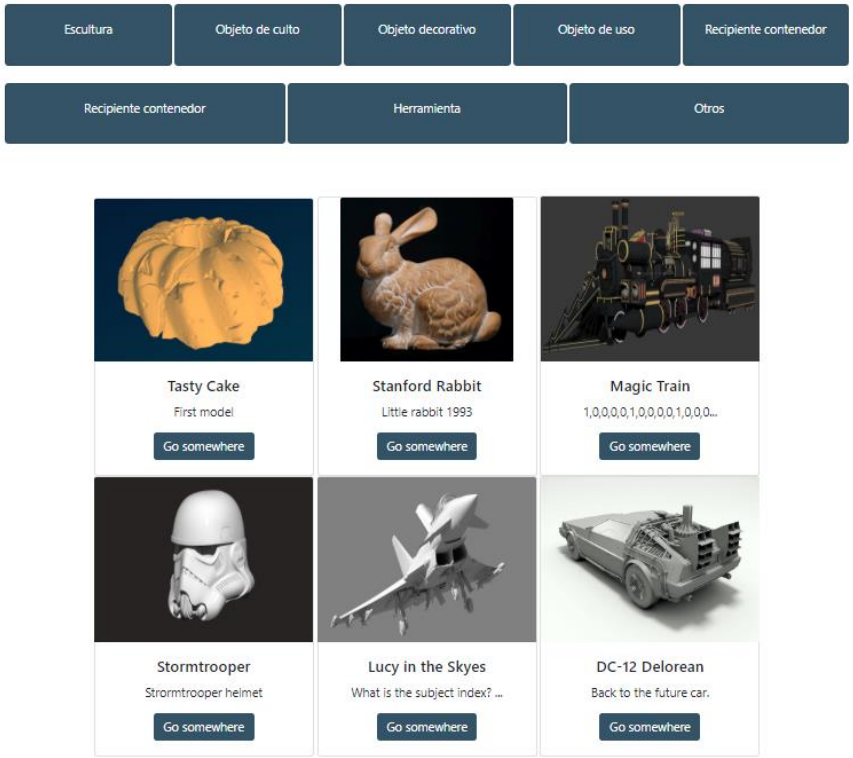


Figura 36. Página principal enriquecida con una serie de modelos libres 3D

En la siguiente figura podemos observarla la sección destacada. En ella se muestra un objeto aleatorio destacado en una tarjeta sobre un fondo estático que al hacer *scroll* queda inmóvil. La información que se muestra del objeto en la tarjeta es reducida, aparece solo: la foto y el título además de un botón que redirige al visor.

Figura 37. Vista del panel de figuras destacadas, mostrando un modelo libre 3D

Al seleccionar sobre una categoría nos lleva al índice, en este se mostrará el título de la categoría y los objetos que pertenecen. Cada objeto mostrará su título, las categorías a las que pertenece, y su descripción, limitada a 30 caracteres.

Las tarjetas se han diseñado manteniendo un estilo similar para que el usuario pueda identificar fácilmente las funcionalidades implementadas en ellas.

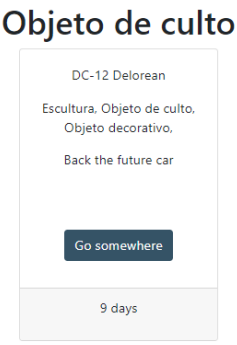


Figura 38. Vista de una de las categorías, mostrando una tarjeta ejemplo

En el índice de los objetos restaurados se podrá encontrar una lista de tarjetas con los objetos subidos a la plataforma ordenados por fecha de creación. En cada tarjeta podremos encontrar información sobre la figura, como su nombre, una breve descripción limitada a 300 caracteres, el número de piezas y en el *footer*, el tiempo transcurrido desde su subida a la plataforma.

Al pulsar en el nombre de la pieza nos redirigirá hacia el visor, donde podremos interactuar con ella y tendremos la interfaz que veremos en el apartado 5.1.2. El botón situado arriba en la parte izquierda de la figura 39, que actualmente es de color azul nos permitirá ir a la interfaz para cumplimentar el formulario para subir nuevas piezas.

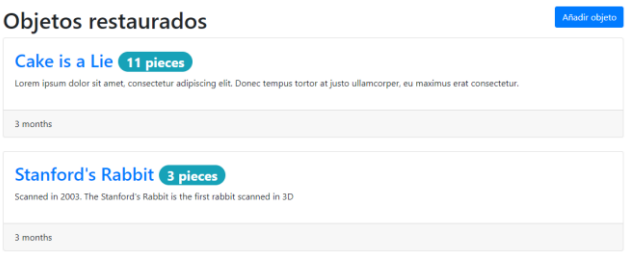


Figura 39. Vista del índice de objetos restaurados

La vista empleada para añadir piezas a la aplicación ofrece un formulario básico para completar los campos que posteriormente serán almacenados en la base de datos *PostgreSQL*. También se usará para establecer las relaciones con otras clases también almacenadas en la base de datos.

En la vista encontraremos cinco tipos de entrada de datos, campos de texto normal, en ellos se puede insertar texto de forma libre. *Checkboxes*, para realizar elecciones múltiples, son muy útiles para establecer las relaciones muchos a muchos en la BDA. *ComboBoxes*, para seleccionar una opción entre un conjunto. Esto permite establecer relaciones unitarias entre objetos. Para añadir la URL donde se encuentra la imagen que será el avatar de la pieza mostramos un botón, al hacer *click* sobre él se abrirá el diálogo del explorador de Windows. Y para concluir campos numéricos,

que permiten solo la entrada de números, garantizan el formato para ese campo en la base de datos.

En la parte inferior nos encontramos con la sección para añadir figuras. Nos permite realizar la carga de dos maneras, las cuales estarán explicadas en el apartado 5.2.

Finalmente, con el botón azul, mostrado en la parte baja de la figura 40, podremos guardar todos los datos introducidos anteriormente, durante su almacenamiento en la BDA aparecerá una pantalla de carga. Cuando los datos estén guardados nos redireccionará al visor.

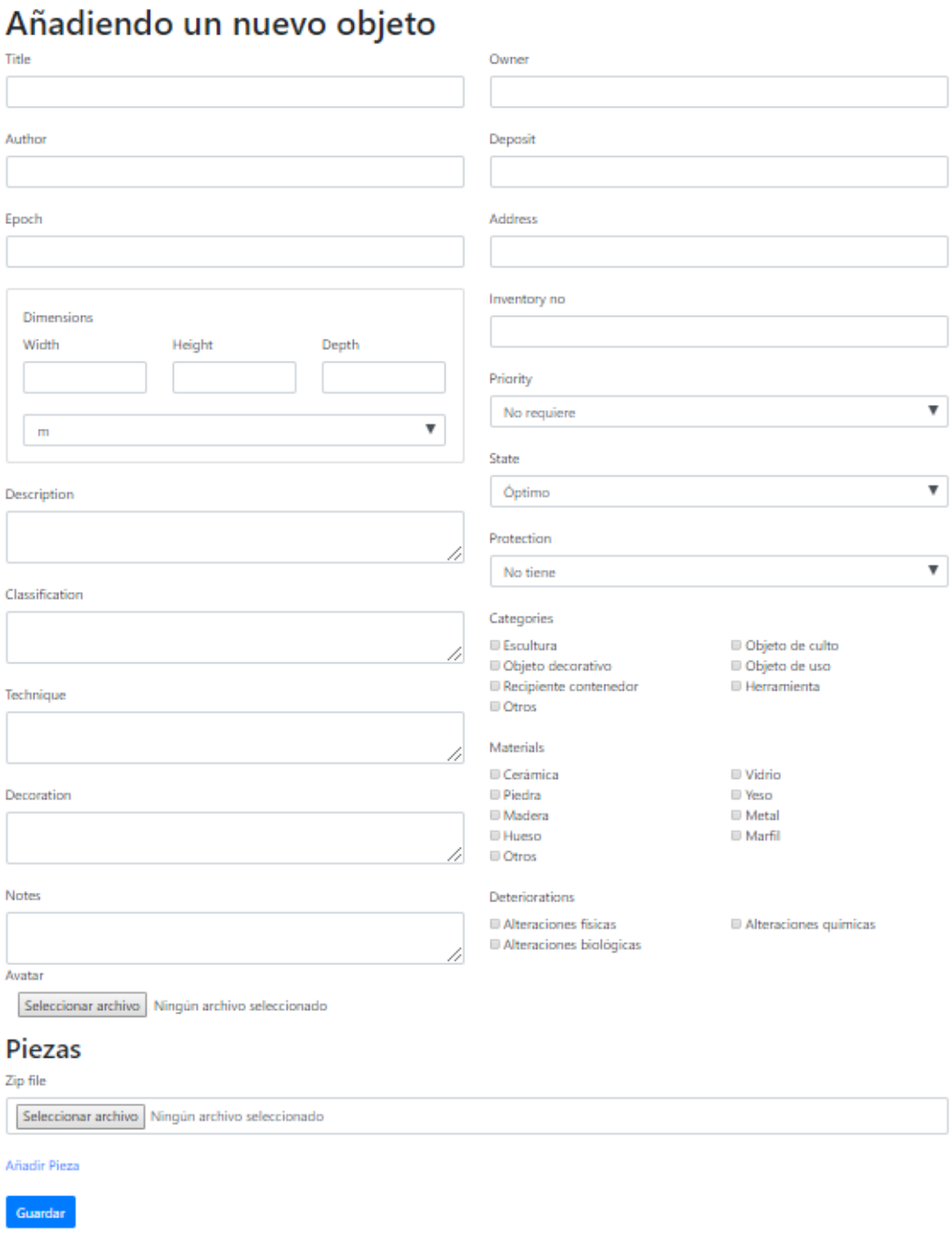


Figura 40. Formulario de información de una pieza

5.1.2 Interfaz del visor

Este es el diseño final del visor en la aplicación, haremos una breve explicación de cada uno de los objetos que contiene la interfaz. La figura que se encuentra inmediatamente a continuación muestra el resultado final de la interfaz del visor.

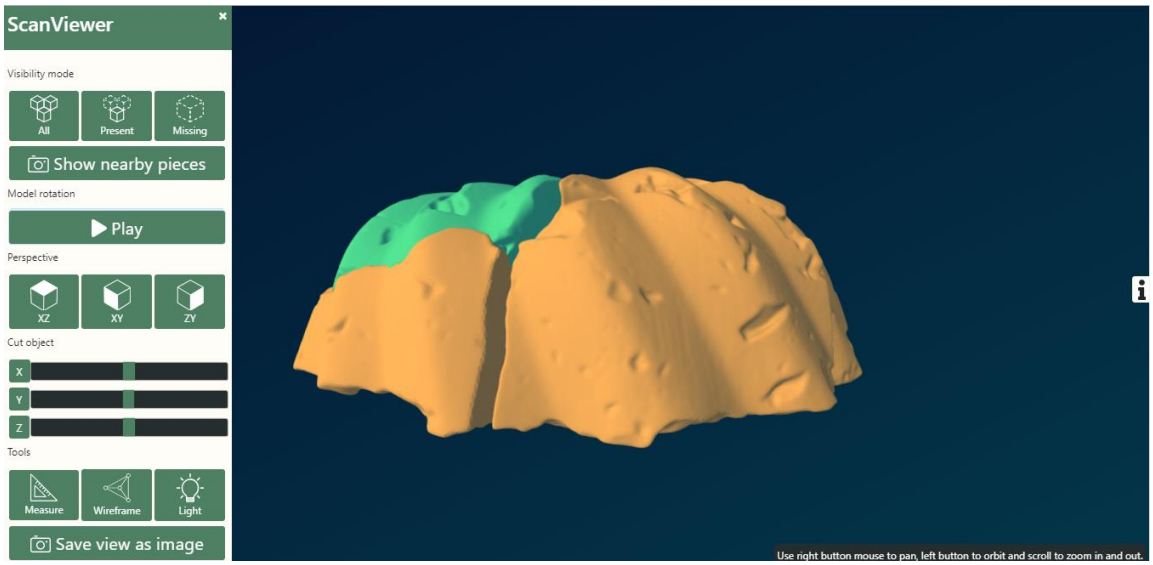


Figura 41. Interfaz del visor con figura de prueba "cake"

En la parte izquierda nos encontramos con una pestaña, la cual podemos replegar o desplegar según nos convenga. En ella se encuentran las acciones básicas de visualización y las herramientas interactivas. Tiene una animación de despliegue que hace más intuitivo su uso, es rápida para no entorpecer el uso del sistema.

Desde la parte superior hasta la inferior en esta pestaña podemos encontrar las siguientes divisiones con sus respectivas acciones a realizar.

Modo de visibilidad (*Visibillity mode*), en esta sección se muestra una serie, los cuales realizan acciones sobre la visibilidad del conjunto de piezas. Esto quiere decir que nos permiten mostrar o esconder las piezas de nuestro interés.

En el apartado 9 de este trabajo encontraremos una explicación más detallada de las funcionalidades de los botones. Esta aclaración se aplica a todas las secciones de este apartado.

Los botones están posicionados en una matriz (figura 42), cada uno de los situados en la parte superior ocupa un 30% de la anchura de la pestaña. El inferior ocupa toda la parte inferior de esa sección, con esto evitamos que los 4 se sitúen en la misma línea y quede una interfaz recargada.

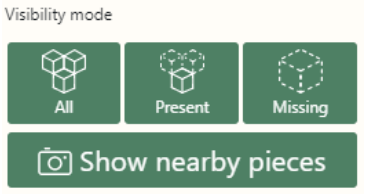


Figura 42. Vista del modo visibilidad

Modo de rotación (*rotation mode*), contiene un solo botón que ocupa el total de la anchura de la pestaña. Este botón al pulsarlo contiene una pequeña animación la cual cambiará el botón de *play* por el de pausa.

Las animaciones de este botón están realizadas haciendo uso de *JavaScript*, cada vez que detecta el cambio, se realiza un cambio de imagen accediendo al DOM de HTML. El botón de "*play*" se mostrará cuando el sistema no esté en rotación, este es como el de la figura 43.

Figura 43. Vista del botón "Play"

El botón de "*pause*" se mostrará en el caso contrario, cuando las piezas estén rotando. Figura 44.

Figura 44. Vista del botón "Pausa"

Perspectiva (*perspective*), en esta sección podemos modificar la posición de la cámara haciendo uso de estos tres botones, posicionados de una forma idéntica a los 3 botones del modo de visibilidad.

Las imágenes representan un cubo mostrando el alzado la planta y el perfil, con los correspondientes ejes en la que se situará la cámara como se muestra en la figura 45.

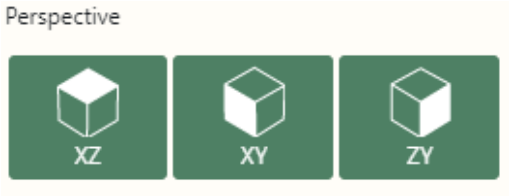


Figura 45. Vista del menú de selección de perspectiva

Cortar objetos (*cut objects*), esta vista se compone de tres barras deslizantes y tres botones, todo diseño se ha realizado con *HTMLy CSS*, sin embargo, para obtener los datos que otorga este selector se ha utilizado *JavaScript*.

Los botones de selección habilitan el uso de la herramienta, sin embargo, las barras deslizantes siempre estarán habilitadas. Podemos encontrar estos actuadores en la figura 46.

Figura 46. Vista del menú de corte

Panel de herramientas (*tools*), tiene exactamente el mismo diseño que se ha aplicado en el marco de visibilidad. Los iconos, representan la herramienta que se va a utilizar de una forma visual e intuitiva. El menú que se muestra en pantalla en el visor es el que podemos ver en la figura 47.

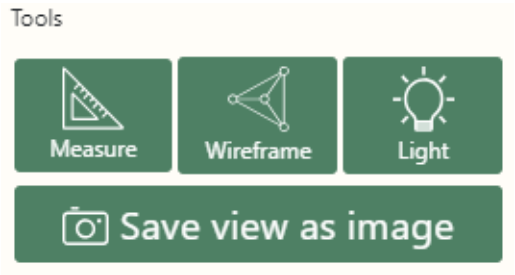


Figura 47. Vista del menú del panel de herramientas

En el botón horizontal se decidió poner el texto "*Save view as image*" ya que poner únicamente un icono de una cámara no era representativo y mejoraba la estética general de la interfaz.

5.2 Carga automática de piezas

En el formulario para subir las piezas, se permite cargar un zip que contendrá las piezas y las matrices de translación con la estructura que se ha explicado en el apartado 4.6.2. De esta manera facilita la carga de las piezas a los usuarios.

En la carga automática de piezas se contemplan ataques producidos en los ficheros que están contenidos en el archivo comprimido. Por lo tanto, con Ruby se comprueba que estos archivos sean ficheros de texto plano compatibles.

En la interfaz mostrada en el apartado 5.1.1 se muestran el conjunto de campos que deben ser rellenados para cumplimentar el formulario de la pieza. Una vez se complete se puede proceder a realizar la carga de las piezas utilizando los dos métodos implementados.

La forma manual, en la que los usuarios tienen que añadir las piezas una a una, simplemente tienen que seleccionar el botón añadir pieza mostrado en la figura 40 (parte inferior). Por el contrario, si ya se tiene el fichero con extensión .zip con el conjunto de piezas listo para subir, se seleccionará directamente la opción de carga en la sección *Zip File.*

5.3 Objetivos y requisitos cumplidos

Se han cumplido todos los requisitos expuestos y desglosados en el apartado 1.3 del trabajo. No ha habido retrasos ni demoras en las fechas de entrega, pudiendo cumplir con las expectativas esperadas y sin perder tiempo ante los contratiempos que nos han afectado.

A continuación, enumeraremos las soluciones propuestas e implementadas que resuelven los problemas que se han planteado en los requisitos, realizando un breve comentario sobre cada una de ellas.

- La función de representación de piezas, y sus modos de uso de faltantes y no faltantes explicado en el apartado 4.5.3 resuelve los requisitos de: mostrar el conjunto de piezas faltantes y no faltantes.
- El modo explicado en el punto 4.5.3 que realiza un cambio de texturas para obtener el diseño alámbrico resuelve el requisito de: tener una visión en alámbrico del modelo.
- El método para añadir focos y puntos de luz sobre una esfera resuelve el requisito: modificar la iluminación de la escena.
- La implementación de controles orbitales en la escena y redimensionamiento de la misma hace que cumplamos el requisito de situar la cámara en cualquier punto de la escena de forma sencilla.
- Con la implementación para posicionar la cámara en cualquier punto del plano de forma sencilla, en concreto con los atajos para los planos de corte resolvemos el requisito de situar la cámara de forma precisa en los planos de corte XY, YZ y XZ.
- Añadiendo los tres métodos de corte para cada plano resolvemos el requisito de cortar la pieza en los planos X, Y y Z.
- El modo rotado que se introduce para hacer girar el modelo 3D introducido suple el requisito de la creación de un modo presentación.
- La herramienta de medición implementada suple los requisitos de poder medir dos puntos sobre la superficie de una prueba y la de medir una colección de puntos para calcular perímetros.
- La selección básica de piezas programada permite resolver el requisito de selección de piezas mediante el ratón
- El acceso a la base de datos bien estructurada mediante *Active Record* y su visualización en las vistas permite resolver el requisito de obtener la información del conjunto de las piezas.

En cuanto al rendimiento general de la aplicación, podemos decir que es óptimo ya que ha sido capaz de ejecutarse de forma fluida en el hardware con requisitos mínimos incluso en dispositivos con recursos menores como en un iPad Air 2. Los tiempos de carga de las piezas podemos encontrarlos comentados en el apartado 4.5.2 junto con los comentarios de la implementación del sistema de descarga.

La gestión de usuarios está implementada en *Rails* gracias a la gema *Devise* y tiene las funciones básicas requeridas como son: realizar un registro en la aplicación, ingresar una contraseña y gestionar las piezas desde la cuenta de usuarios.

Utilizar la programación asíncrona para la carga de piezas y los *plugins* aceleradores de *Rails* nos ha proporcionado que se tenga una carga de las vistas rápidas reduciendo el tiempo del requisito inicial que era de un minuto hasta los 37 segundos de media por pieza.

Los aspectos más subjetivos especificados en los requisitos se han intentado medir haciendo una serie de ejercicios y pruebas a posibles usuarios finales o pruebas de campo. En general éstos

arrojaron muy buenos resultados y críticas, ayudaron a mejorar el diseño de la interfaz durante los últimos rediseños. Todos estos aspectos se explican en el siguiente apartado 5.4 con más detalle. En cuanto a que la plataforma sea utilizable en la mayor cantidad de dispositivos posibles hemos realizado pruebas en distintos dispositivos con diferentes navegadores, en muchos de ellos la aplicación funcionaba perfectamente, entre los más inusuales encontramos: IPad Air 2, celulares con resolución 1080 y monitores grandes 4K, los cuales requieren de un hardware un poco más potente o Smart TVs.

5.4 Pruebas

Durante la última fase del proyecto, cuando ya se tenía una interfaz completamente funcional y sencilla de utilizar se realizó un test a potenciales usuarios para ver qué aspectos se podían mejorar en posteriores versiones.

El test se realizó a 10 sujetos de prueba, todos ellos voluntarios. Entre sus características destacamos que eran mujeres y varones, con edades comprendidas entre 12 hasta 55 años. Ninguno de ellos había visto antes la aplicación y sus conocimientos en informática eran de nivel de usuario. Para todas las tareas se dio un tiempo máximo de 60 segundos, exceptuando la que tiene por nombre "Tiempo muestra información" que se dio 30 segundos extra.

A continuación, se presenta la tabla con los resultados de los tiempos medios de todos los participantes en las distintas tareas que se pidieron realizar en la figura 48. La unidad de medida del tiempo es el segundo.

Test sin conocimiento		Test con conomimiento	
Tiempo acceso a la pieza	3,36	Tiempo acceso a la pieza	1,79
Tiempo medir pieza	22,65	Tiempo medir pieza	2,28
Tiempo cortar pieza	33,40	Tiempo cortar pieza	14,08
Tiempo captura pieza	11,90	Tiempo captura pieza	6,46
Tiempo solo faltantes	2,70	Tiempo solo faltantes	1,94
Tiempo recomponerlas	22,40	Tiempo recomponerlas	2,02
Tiempo añadir luz	60,00	Tiempo añadir luz	18,07
Tiempo medir 3 puntos	7,09	Tiempo medir 3 puntos	7,52
Tiempo mostrar información	82,56	Tiempo mostrar información	3,12
Desplazarse lejos	2,20	Desplazarse lejos	1,68
Seleccionar pieza	1,61	Seleccionar pieza	1,99
Vista plano XY	16,34	Vista plano XY	1,86
TOTAL	266,21	TOTAL	62,81667

Figura 48. Resultado promedio de las pruebas realizadas

Cuando se analizaron los resultados del test se recomendaron las siguientes modificaciones, todas ellas están fundamentadas en los datos recabados y en las conversaciones con los voluntarios.

La tarea de añadir un punto de luz en la escena requiere de una explicación previa, ya que hay que realizar varios pasos en la interfaz usando un botón poco usual del ratón. Ninguno de los usuarios consiguió realizarla en la primera fase. Para solucionar este problema se ha propuesto la redacción de un manual de usuario donde se indicarán todos los pasos a seguir.

Varios participantes no pudieron completar la tarea de acceder a la información de la pieza. Muchos de ellos comentaron que el icono era muy pequeño y no era fácil de identificar, además no estaba posicionado con el conjunto de herramientas. Se propuso el aumento del tamaño del icono y añadir un apartado en el anexo donde se explican todas las funcionalidades de los botones.

Los participantes realizaron con rapidez el conjunto de tareas restantes, nunca superando los 40 segundos por ejercicio. Esto nos indica que la interfaz es fácil de utilizar e intuitiva para un primer usuario. Sin embargo, hay que matizar algunos conceptos si se le quiere dar un uso más avanzado.

Alguna curiosidad que encontramos durante este proceso de evaluación fueron que los usuarios más jóvenes en los primeros ejercicios tienden a realizar un barrido de la aplicación explorando todas las funcionalidades sin leer las etiquetas que hemos utilizado, se guían más por los dibujos de los botones. Este tipo de participantes no se daba por vencido fácilmente e intentaba por todos los medios realizar la tarea. El resto de los usuarios dedicaban más tiempo a leer las etiquetas cometiendo menos errores hasta encontrar la solución final.

Al finalizar el proyecto se realizó un test a otros 10 sujetos, cinco de los cuales no habían visto aún la aplicación en el que se facilitaba el Anexo que adjuntamos en el apartado nueve con las explicaciones para utilizar correctamente la aplicación. Antes de realizar las tareas se dejó cinco minutos para que el usuario se familiarizara con el documento. Los tiempos medios de las baterías de ejercicios realizadas por los usuarios se muestran en la tabla.

		Iteración con manual	Iteración sin conocimiento	Iteración con conocimiento
1	Tiempo acceso a la pieza	1,69	3,36	1,79
2	Tiempo medir pieza	6,7	22,65	2,28
3	Tiempo cortar pieza	26,88	33,4	14,08
4	Tiempo captura pieza	4,26	11,9	6,46
5	Tiempo solo faltantes	3,25	2,7	1,94
6	Tiempo recomponerlas	0,62	22,4	2,02
7	Tiempo añadir luz	34,69	60	18,07
8	Tiempo medir 3 puntos	7,412	7,09	7,52
9	Tiempo mostrar información	2,14	82,56	3,12
10	Desplazarse lejos	0,63	2,2	1,68
11	Seleccionar pieza	0,98	1,61	1,99
12	Vista plano XY	6,62	16,34	1,86

Figura 49. Tiempos promedio de los test realizados

La gráfica muestra en naranja el tiempo medio del test a primera vista sin manual; azul, segunda iteración y gris se corresponde al test con el manual. Se recomendaron mejoras en el manual como aumentar el tamaño de las ilustraciones. Estos cambios se realizaron par la versión final del manual.

Figure 50. Gráfica con los tiempos de la figura 49

En este test ya se habían realizado las modificaciones de diseño de interfaz que se habían solicitado o que habían sido propuestas durante la anterior prueba. Tras comparar todos los

resultados podemos concluir que todos los usuarios han podido realizar todas las tareas con el manual, sin necesidad de que un agente externo les dijera como proceder. Se puede ver también que, aunque el manual sea de ayuda sigue habiendo tareas costosas.

A parte de los test y pruebas realizados con los usuarios voluntarios para modificar y validar la interfaz, se hicieron pruebas de ejecución en dispositivos variados que podían abrir páginas web, para comprobar la flexibilidad del software desarrollado. En el apartado de interfaz 5.1.1, las figuras 30 y 31 muestran cómo en distintos dispositivos con resoluciones variadas la interfaz diseñada se adapta sin ningún problema.

En cuanto al visor se realizaron pruebas en distintos dispositivos. El resultado fue satisfactorio, ya que teniendo dispositivos con unas especificaciones menores a las requeridas el sistema funcionaba sin problemas. En la siguiente figura, la número 51 se muestra un ejemplo de estas pruebas, el editor siendo ejecutado en un iPad Air 2.

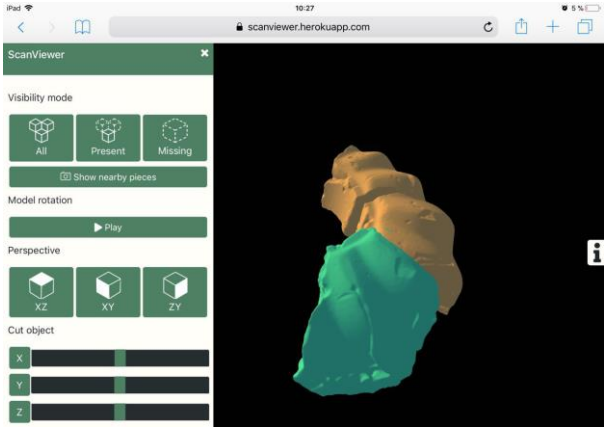


Figura 51. Vista de la interfaz en un iPad air 2

6 Conclusiones

Se ha realizado una aplicación web cumpliendo todos los requisitos acordados en el primer momento, y los que fueron añadidos posteriormente. El desarrollo de la misma no ha tenido ningún percance ni retraso, se han cumplido los plazos de entrega y la relación entre usuario final y desarrollador ha sido cordial y efectiva. Debido a estas razones se piensa que el proyecto ha sido un éxito.

La integración de la metodología ágil durante el desarrollo del producto se piensa que fue una buena elección ya que usuario final ha visto de forma periódica el trabajo que se realizaba y se ha involucrado en las fases de desarrollo y diseño.

Los plazos y fechas de hitos y entregas se han respetado en base a lo acordado, haciendo latente que se ha tenido una buena planificación para la ejecución de este proyecto. La metodología anteriormente comentada pensamos que ha tenido una repercusión positiva y se tendrá en cuenta su uso para otros proyectos futuros.

Las impresiones finales han sido positivas, dando lugar a una segunda fase del proyecto que integrará más funcionalidades y donde se intentará mejorar la eficiencia del visor, así como sus tiempos de carga. Se pondrá en contacto con posibles usuarios finales que quieran integrar esta aplicación en su infraestructura.

A nivel profesional se ha realizado una buena tarea y ha ampliado conocimientos sobre los campos desarrollados, implementación y formas de trabajo nuevas y el ambiente que ofrece un escenario real laboral.

Las tecnologías empleadas han sido correctamente elegidas, *ThreeJS* ha ofrecido un conjunto de herramientas y métodos básicos imprescindibles para realizar el visor. Además, el motor de esta librería es muy eficiente, dando como resultado una fluidez idónea en el sistema entero. *JavaScript* ha sido un lenguaje versátil y potente, que ha permitido realizar las descargas de las piezas en tiempos muy reducidos gracias a la implementación de los *web-workers*. Finalmente, *RoR* ha estructurado la aplicación, ha realizado la composición de todos los sistemas y ha gestionado la base de datos de una forma simple.

Para concluir hay que destacar que en este proyecto se han puesto en práctica las técnicas de aprendizaje adquiridas durante nuestro proceso de formación en la Universidad Politécnica de Valencia.

Específicamente se han utilizado conocimientos impartidos en las asignaturas de SGI, Sistemas gráficos e interactivos donde se aprendió *OpenGL,* algoritmos y scripts para la manipulación de objetos 3D y cálculo de fórmulas y aproximaciones para gráficos 3D. EDA, estructuras de datos y algoritmos, ya que es imprescindible tener una buena organización de los datos y como acceder a ellos de una forma rápida y eficiente. BDA, bases de datos, para tener un conocimiento estricto de la gestión de la base de datos *Postgre* que hemos utilizado. A pesar de que se utiliza un ORM que facilita enormemente la tarea, es imprescindible entender su funcionamiento interno. Finalmente, las asignaturas de ISW, ingeniería del software y GPR, gestión de proyectos, estas inspiraron a buscar nuevas metodologías y formas de trabajo que hicieran la producción de software más eficiente.

Estudiar el grado ha sido de gran ayuda para la realización de este trabajo de una forma profesional.

6.1 Análisis del producto

Gracias a las pruebas realizadas en el apartado 5.4 se pudieron corregir algunos fallos de diseño que hicieron que la interfaz fuera más intuitiva. En estas, a pesar de que reflejan algunas carencias en la aplicación, que ya han sido corregidas, también muestran como el usuario en muchos otros aspectos se siente cómodo al usar la aplicación. No se reportaron errores en el funcionamiento de la aplicación durante las pruebas.

Tras las entrevistas posteriores a los participantes, todos ellos coincidían en que la interfaz funcionaba con fluidez, era sencilla y estéticamente era atractiva. Algunos de ellos comentaron

que sería de gran ayuda el manual que ya se estaba redactando y que sus tiempos de respuesta hubieran sido menores.

Los tiempos de carga entre las distintas vistas son adecuados conforme a los requisitos, y los usuarios que realizaron los test, generalmente sintieron que la duración de los tiempos fue corta. Sin embargo, sí hubo dos sujetos que dijeron que la duración podía ser excesiva si el uso de la aplicación era muy frecuente.

6.2 Satisfacción con el resultado

Las encuestas realizadas a los voluntarios demostraron que la interfaz era sencilla de utilizar, intuitiva y contaba con un buen diseño estético. El tiempo de adaptación a la interfaz fue rápido y todos los participantes pudieron realizar todas las tareas tras una exposición mínima de corta duración.

En una segunda encuesta realizada se comprobó que todos los participantes con el manual del usuario que podemos encontrar en el Anexo, punto nueve de este trabajo, pudieron realizar todas las actividades que se propusieron. Estas pruebas comentadas pueden verse explicadas en más profundidad en el apartado 5.4.

El usuario final se vio involucrado durante el proceso de diseño y desarrollo, los cuales podemos ver en la planificación en el apartado 3. Se dieron una serie de indicaciones claras para el diseño y construcción de la interfaz que se cumplieron estrictamente. Esta involucración supuso una serie de mejoras en el producto final, ya que se subsanaron los errores cometidos y se mejoraron las funcionalidades.

En la fase final de este proyecto, se pudo demostrar que se habían cumplido los requisitos y objetivos impuestos durante la fase de constitución del proyecto. Podemos decir que tanto los usuarios que probaron las aplicaciones en la fase de integración y desarrollo como los que probaron la versión final quedaron satisfechos con el resultado del producto.

El proceso de desarrollo ha sido un éxito en sí mismo ya que se ha cumplido la planificación que se propuso y se ha trabajado de forma cómoda. Los conocimientos adquiridos son muy amplios y serán útiles para futuros proyectos.

6.3 Conocimientos obtenidos durante el desarrollo

A pesar de que previamente a este trabajo se conocían y se habían utilizado las tecnologías que se iban a emplear en este proyecto se tuvieron que adquirir nuevos conocimientos para solucionar problemas e introducir mejoras. El aprendizaje ha sido gradual durante todo el desarrollo. Enumeraremos ahora específicamente qué recursos se tuvieron que aprender.

Para realizar la carga asíncrona de las piezas en el visor se necesitaron varios días de estudio del uso y funcionamiento de los *web-workers* integrados con *ThreeJS*. Se consultó tanto en las guías oficiales de esta librería como en foros y repositorios de código. Se considera que la implementación de los *workers* ha sido la más complicada de toda la aplicación.

Los *web-workes* han demostrado ser una tecnología que agiliza mucho los procesos de carga y que no se tenía consciencia de que existía algo así para *JavaScript* y *ThreeJS*. El hecho de realizar esta implementación concreta ayuda a que futuros programadores no tengan que volver a estudiar y a reimplementar este sistema de carga ya que puedan copiarlo de nuestra aplicación. Un proyecto futuro derivado de este hecho será realizar una API para automatizar este proceso.

Se adquirieron conocimientos más avanzados en el uso de la librería *ThreeJS*, entre ellos podemos destacar el uso de matrices de transformación para el posicionamiento de objetos, utilizar objetos auxiliares como planos, rayos, la cámara para realizar tareas complejas y como realizar cálculos matemáticos de forma nativa para realizar operaciones en el espacio tridimensional.

Se ha explorado más a fondo el *framework Rails* y las gemas que este usa para realizar acciones específicas. En concreto destacamos que se debió de adquirir conocimientos sobre las gemas gon, railszip y AWS-sdk. Su utilización no fue complicada y encontramos una documentación excelente en los repositorios oficiales de las mismas.

Se aprendió a gestionar un almacén de datos como es Amazon S3, realizar todas las configuraciones pertinentes para que este pueda conectarse con la aplicación en *Ruby on Rails*.

HTML y CSS por mucho que se utilicen siempre es recomendable seguir realizando trabajos con ellos ya que los problemas que surgen en cada ocasión son muy diferentes y podemos aprender mucho de las oportunidades, sobre todo con CSS.

7 Trabajos futuros

Aun estando muy satisfechos con el resultado obtenido y habiendo cumplido todos los requerimientos y objetivos, se han pensado y propuesto un conjunto de mejoras que podrían realizarse en el futuro.

Estas nuevas funcionalidades ofrecerían al usuario una experiencia más completa, ya que se integraría el algoritmo de reconstrucción de las piezas en la propia aplicación. De esta forma el usuario únicamente tendría que preocuparse de conseguir las piezas y subirlas a la web. El sistema se encargaría de procesarlas, generar las matrices y almacenarlas en la base de datos.

Al estar accesible para todo el público en un repositorio de software libre, el proyecto podrá ser clonado y replicado por cualquier programador, haciendo distintas versiones del proyecto que posteriormente se pueden agregar al original como mejoras.

También se quiere traducir la página a múltiples idiomas, como el inglés, el italiano o el francés haciendo uso de las facilidades que ofrece el *framework RoR* para que el alcance de la aplicación sea más amplio.

La web a medio y largo plazo necesitará el mantenimiento propio de una aplicación web implementada en *Rails*, así como la migración al entorno donde se quiera lanzar esta aplicación, ya que actualmente está ejecutándose en un servicio gratuito de *hosting.*

Para dar por finalizado el proyecto completamente se pretende escribir un artículo con información sobre el proyecto, explicando cómo se ha desarrollado, qué pretende conseguir y finalmente, cómo podría implementarse en un mercado.

8 Bibliografía

[1] Ivan Edward Sutherland, "Sketchpad: A man-machine graphical communication system" ,2003, https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-574.pdf

[2] Paul Lamking, "VR And AR Headsets to Hit 80 Million By 2021", 2017, Forbes: https://www.forbes.com/sites/paullamkin/2017/09/29/vr-and-ar-headsets-to-hit-80-million-by-2021/

[3] Christopher Groskopf," Museums are keeping a ton of the world's most famous art locked away in storage", Quartz: https://qz.com/583354/why-is-so-much-of-the-worlds-great-art-in-storage/

[4] Google, Interland, 2017, https://beinternetawesome.withgoogle.com/en/interland/

[5] Emerson Taymor, AGILE HANDBOOK ,2018 http://agilehandbook.com/agile-handbook.pdf

[6] Scrum.org, What is a Sprint in Scrum? 2018, https://www.scrum.org/resources/what-is-a-sprint-in-scrum

[7] Mush Tahir Aziz Rahman, RUBY ON RAILS DEVELOPMENT PRINCIPLES, EXPLAINED, 2014 http://www.nascenia.com/ruby-on-rails-development-principles

[8] David Cardinal, How Nvidia's RTX Real-Time Ray Tracing Works, 2018, Extremetech https://www.extremetech.com/extreme/266600-nvidias-rtx-promises-real-time-ray-tracing

[9] The Unreal dev Team, Unreal Engine Features, 2018, https://www.unrealengine.com/en-US/features

[10] Jon Peddie, The state of the art of graphics world, 2018, https://www.eetimes.com/author.asp?section_id=36&doc_id=1266025

[11] David Cardinal, How Nvidia's RTX Real-Time Ray Tracing Works, 2018, Extremetech https://www.extremetech.com

[12] NVIDIA® OptiX™ Ray Tracing Engine, 2018, https://developer.nvidia.com/optix/index.html

[13] Smithsonian 3D team, About Smithsonian X 3D, 2013, https://3d.si.edu/about

[14] Devise team, Devise gem README & DOC,2009-2018 https://github.com/plataformatec/devise

[15] Universidad Politécnica de Cartagena,"Las licencias creative comons",2016, Univerdad Politéncia de Cartagena, http://www.bib.upct.es/las-licencias-creative-commons

[16] Glassdor, Auto generated Rails FullStack dev salary, 2018, https://www.glassdoor.com/Salaries/ruby-on-rails-full-stack-developer-salary

[17] Adrian Mejia, Ruby on Rails Architectural Design, 2011, https://adrianmejia.com/blog/2011/08/11/ruby-on-rails-architectural-design/

[18] Rion Williams, Bringing Your Web Application to Desktop Life with Electron, 2016, https://electronjs.org/ , http://rion.io/2016/12/02/bringing-your-web-application-to-desktop-life-with-electron

[19] Webassembly Team, Webassembly about, 2017, https://webassembly.org

[20] Rails Guides Community, Active Record Associations,2018, http://guides.rubyonrails.org/association_basics.html

[21] 2ndQuadrant, PostgreSQL vs MySQL, 2015, https://www.2ndquadrant.com/es/postgresql/postgresql-vs-mysql

[22] Donald Bell, An introduction to the UML, 2003, https://www.ibm.com/developerworks/rational/library/769.html

[23] ThreeJs Community, Loading 3D models, 2018, https://threejs.org/docs/#manual/introduction/Loading-3D-models

[24] Matilde Rocha Aguilera, Introducción a los Web Workers, 2017, https://medium.com/techwomenc/introducci%C3%B3n-a-los-web-workers-f8d44b7ad6e

[25] Daniel Simmons, Get started with WebAssembly — using only 14 lines of JavaScript, 2018, https://medium.freecodecamp.org/get-started-with-webassembly-using-only-14-lines-of-javascript-b37b6aaca1e4

9 Anexo: Manual de usuario de la aplicación

Este apéndice es una guía para para el usuario de la aplicación FragmentX, donde podremos encontrar todas las explicaciones para el uso de la aplicación. Para encontrar una descripción más detallada de la implementación y funcionamiento se recomienda mirar la memoria realizada del proyecto.

9.1 Descripción general de la aplicación

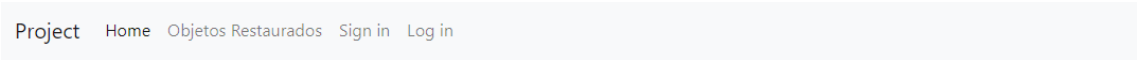
La aplicación FragmentX, es un visor 3D acoplado en una plataforma web para la gestión de obras de arte. En ella puedes crear una cuenta, regístrate, subir tus modelos y obras de arte fragmentados y posteriormente visualizarlos.

Cuenta con una serie de herramientas de edición y visualización para poder explorar mejor el modelo. Su utilización es muy sencilla, pero en esta guía se encontrará la explicación detallada para realizar un uso adecuado de ellas.

9.2 Creación de cuenta e inicio de sesión

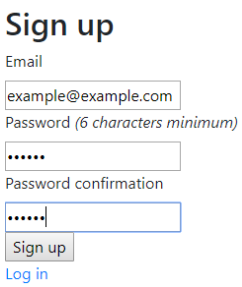
El primer paso para realizar un uso de todas las posibilidades y herramientas que ofrece la aplicación es la creación de una cuenta de usuario en la plataforma. Una vez la tengamos podremos realizar la carga de objetos propios a la plataforma. Para crear nuestra cuenta simplemente tendremos que realizar estos pasos:

Acceder a la página de "*Sign Up*" accesible desde la barra de menú.



Anexo figura 1. Menu principal superior

Hay que rellenar los datos como aparece en la siguiente imagen, teniendo en cuenta que se tiene que introducir un e-mail válido y una contraseña con al menos 6 caracteres, una vez realizada esta tarea pulsaremos el botón "Log in" que nos redirigirá a la pantalla de inicio.

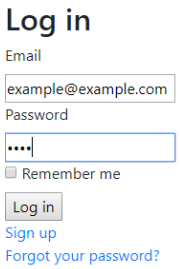


Anexo figura 2. Vista de registro

Si se quiere volver a entrar una vez se ha creado la cuenta desde otro dispositivo en otro momento, seguiremos unos pasos muy similares a los anteriores:

Acceder a la página de "Log in" accesible desde la barra de menú que se muestra en la figura 1 del anexo.

Rellenar los datos como aparece en la siguiente imagen, con los datos que se pusieron la primera vez que se accedió a la web. Pulsar en el botón "Log in". Una vez pulsado te redirigirá a la pantalla de inicio, tus credenciales ya estarán guardadas.



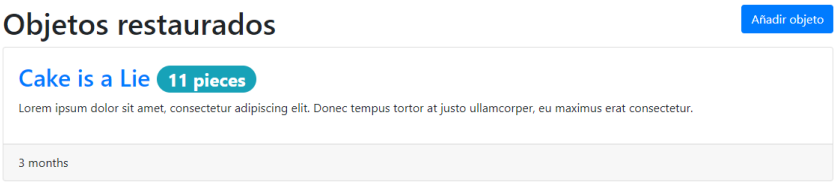
Anexo figura 3. Vista de acceso a la plataforma

9.3 Carga de piezas

Una vez estemos registrados en la plataforma y con la sesión iniciada en el navegador, podremos añadir nuestros objetos y figuras a la web. Para realizar esta acción seguiremos los pasos que podemos ver a continuación.

Hay que dirigirse a la pestaña de objetos restaurados haciendo uso de la barra de menú de la web, localizada en la parte superior de la misma, ver figura 2.

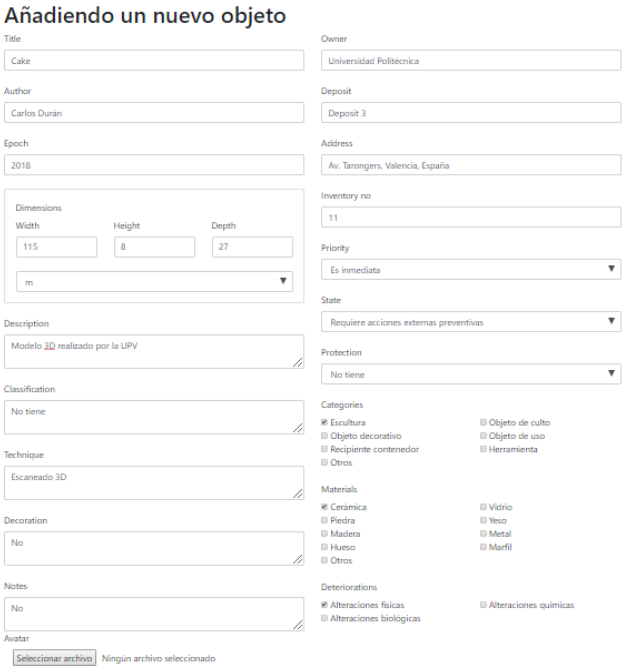
Una vez estemos en la vista de objetos restaurados, seleccionaremos el botón azul situado en la parte superior derecha de la interfaz mostrada en la figura 4. Una vez seleccionado, nos redirigirá a la vista para introducir los datos de la figura.



Anexo figura 4. Lista de objetos restaurados

En el formulario tendremos que rellenar los campos que se muestran a continuación (figura 5) con la información correspondiente. Algunos de los huecos son obligatorios de rellenar, estos son: el título y la descripción. También es necesario marcar una clase en los *checkboxes* que se encuentran en la parte inferior derecha (categorías, materiales y deterioros). Se puede añadir

también una imagen que representará al objeto y se mostrará en las tarjetas de la página principal.



Anexo figura 5. Formulario de una figura

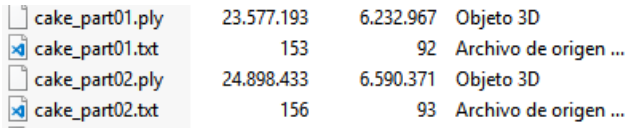
Podemos añadir figuras a la plataforma de dos maneras posibles; pieza a pieza o utilizando el método del archivo comprimido, es decir todas a la vez.

Anexo figura 6. Formulario acceso a una pieza

Para subir las piezas por separado tendremos que seleccionar el texto azul que pone “Añadir pieza”, que se muestra en la figura 6. Una vez realizada esa acción se desplegará un cuadro donde podremos rellenar la información correspondiente a cada pieza. Esta se corresponde a su clasificación, su matriz de traslación, su título, una breve descripción y el objeto 3D. Por cada pieza que se quiera añadir repetiremos este proceso. Si se desea eliminar una pieza simplemente se tiene que seleccionar el botón rojo de la figura siete.

Anexo figura 7. Formulario información de una pieza

Para añadir todas las piezas a la vez, deberemos generar un archivo comprimido en nuestro ordenador que contenga todas las figuras que queramos subir junto con sus matrices de traslación. Simplemente nombraremos los pares de ficheros matriz-pieza con el mismo nombre. En la siguiente imagen (figura 8) podemos ver un ejemplo de cómo se estructura.



Anexo figura 8. Archivos dentro de un fichero comprimido

Utilizando este método no podremos añadir una descripción a las piezas ni su clasificación en faltantes y no faltantes.

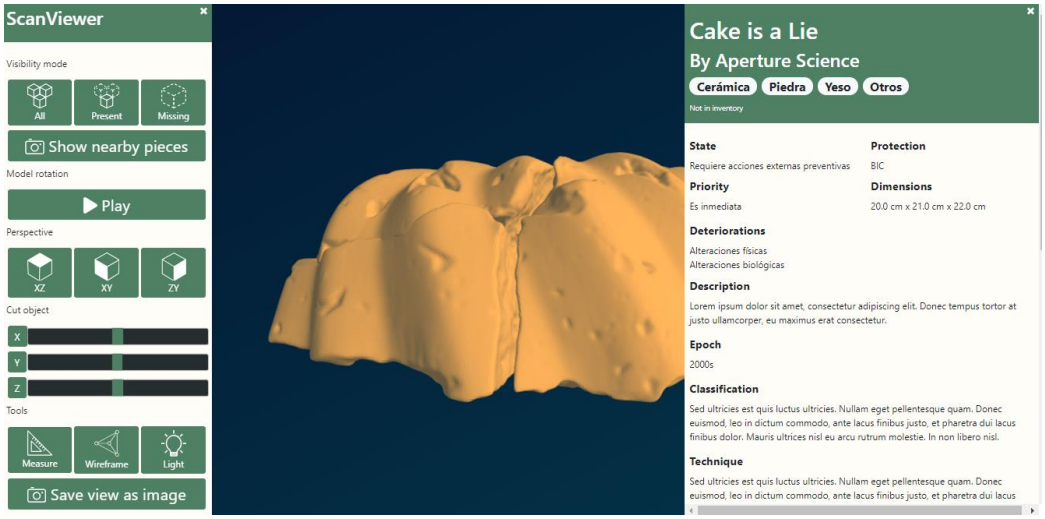
Una vez tengamos el archivo comprimido lo seleccionaremos haciendo uso del botón que se encuentra debajo del texto “Zip file”. Una vez se haya seleccionado no se tendrá que realizar ninguna acción más.

Para finalizar presionaremos el botón azul guardar que nos redirigirá a la página web donde se encuentra el visor con la visualización de nuestra pieza.

9.4 Descripción general del visor

Al acceder al visor podemos diferenciar tres partes dentro de la interfaz, el panel de herramientas y acciones, desde donde se activan todos los modos de representación, el visor donde veremos la figura representada y finalmente el panel de información que por defecto permanecerá replegado.

El panel de herramientas se encuentra en la parte izquierda de la pantalla y puede esconderse dejando todo el espacio para el visor. La representación ocupa la parte central de la pantalla y la información la parte izquierda como podemos ver en esta imagen de la figura 8.



Anexo figura 8. Interfaz del visor con información desplegada

9.5 Herramientas del visor

El visor tiene un conjunto de herramientas que permiten realizar tareas como medir, añadir puntos de luz en la imagen, cortar o guardar capturas de pantalla. A continuación, realizaremos un repaso de todas las funcionalidades y cómo utilizarlas.

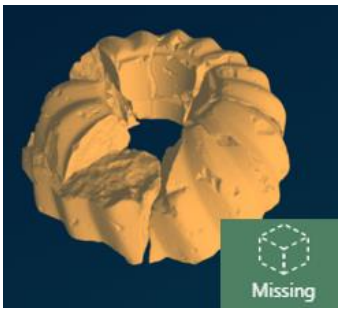
9.5.1 Modo de visibilidad

El modo visibilidad permite realizar una serie de acciones sobre los fragmentos que se están mostrando en el visor. Partiendo del objeto inicial podremos realizar tres acciones básicas, de izquierda a derecha nos encontramos con mostrar figuras presentes, mostrar figuras faltantes y mostrar la pieza entera. Figura 10.

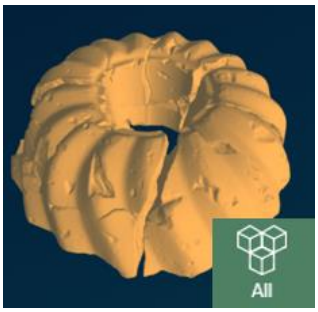
Anexo figura 10. Menú de modo visibilidad

Al hacer *click* sobre estos botones se realizan las acciones sobre la pieza tal y como se muestra en las figuras siguientes, 11, 12 y 13. Estas figuras tienen en la esquina derecha el botón pulsado para obtener el resultado.

Anexo figura 11. Piezas presentes

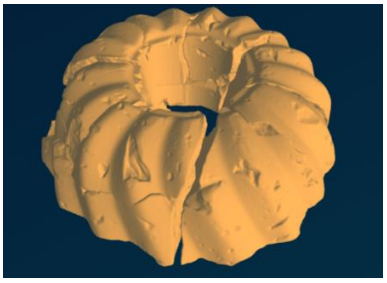


Anexo figura 12. Piezas faltantes

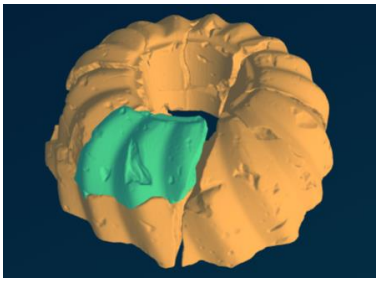


Anexo figura 13. Todas las piezas

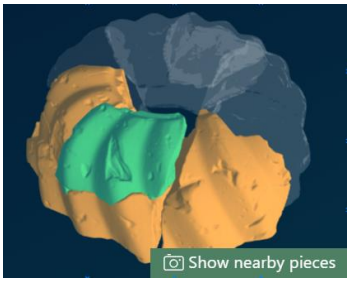
Para mostrar las piezas colindantes con la seleccionada hay que realizar una serie de pasos: en primer lugar, seleccionar la pieza deseada, y una vez aparezca la selección presionar el botón "*Show nearby pieces*". Los pasos se pueden ver gráficamente en las figuras: 14, 15 y 16.



Anexo figura 14. Todas las piezas



Anexo figura 15. Pieza seleccioanada



Anexo figura 16. Piezas colindantes

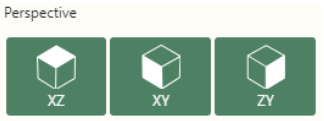
Para regresar a la vista normal, simplemente pulsaremos el botón "*All*" que se muestra en la primera figura de la colección previa a esta.

Para hacer rotar la pieza, tendremos que darle al botón "*Play*" (figura 17) que se encuentra debajo del texto "*Model rotation*". Al realizar esta acción el objeto empezará a rotar y el botón cambiará a "*Pause*" (figura 18). Si se pincha sobre este, el objeto volverá a quedarse estático.

Anexo figura 17. Botón de "Play"

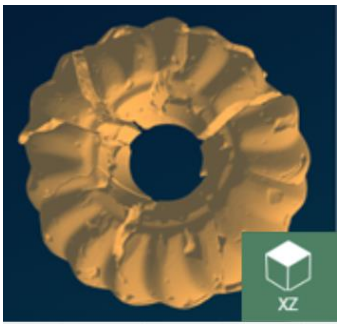
Anexo figura 18. Botón de "Pause"

Para poder seleccionar de forma rápida la perspectiva deseada tenemos una serie de botones que hacen referencia a los ejes donde nos podemos posicionar. Estos son los mostrados en la figura 19.

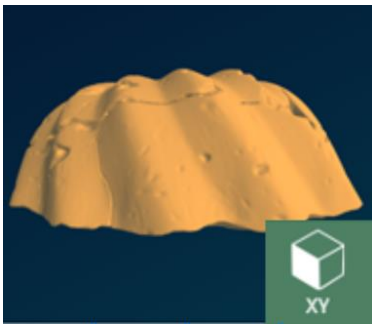


Anexo figura 19. Botones selección vista

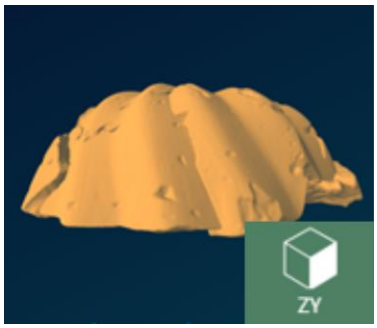
Al hacer *click* sobre estos botones se realizan las acciones sobre la pieza tal y como se muestra en las figuras siguientes. Las figuras 21, 22 y 23 tienen en la esquina derecha el botón pulsado para obtener el resultado.



Anexo figura 21. Vista plano XZ



Anexo figura 22. Vista plano XY



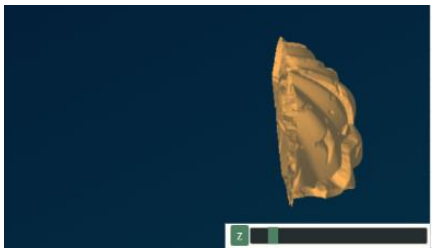
Anexo figura 23. Vista plano ZY

Al seleccionar el botón con la etiqueta "XZ", obtendremos una vista desde arriba, hacia el origen de coordenadas. Si pulsamos "XY" o "ZY" nos mostrará la vista sobre el plano seleccionado.

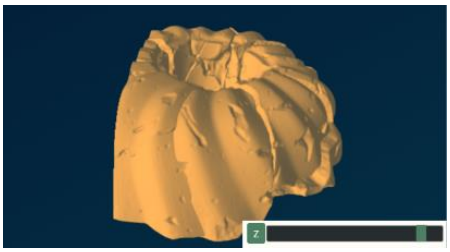
Si queremos realizar un corte en alguno de los planos nos dirigiremos al menú de corte. En él encontraremos tres barras deslizantes que utilizaremos para seleccionar la posición del corte. Los cortes se activarán pulsando el botón asociado a la barra en la parte derecha.

Anexo figura 24. Sliders de corte

Al pulsar el botón activaremos el modo y con la barra seleccionaremos el punto de corte, tal y como se puede ver a continuación en las figuras 25 y 26, en el que la pieza esta seccionada en distintos puntos.



Anexo figura 25. Figura cortada al 15%



Anexo figura 26. Figura cortada al 90%

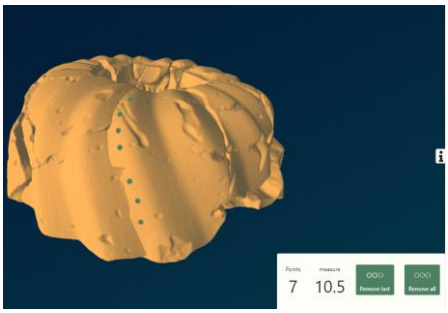
Para medir la distancia entre una serie de puntos en la superficie de la imagen hay que realizar una serie de sencillos pasos que explicaremos. En primer lugar, seleccionaremos el modo de medida en la sección de herramientas en la interfaz, el cual se corresponde al cartabón con el texto "*Measure*", en la figura 27.

Anexo figura 27. Menú de herramientas

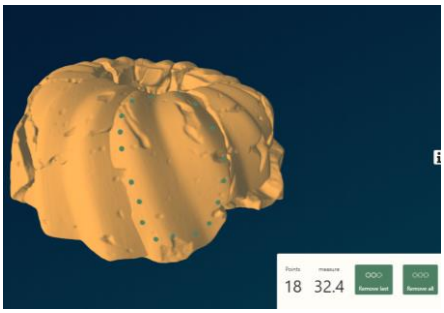
Una vez seleccionado entraremos y se mostrará un desplegable (figura 28) en la parte inferior izquierda que contendrá la información referente a las acciones de medida que se van a realizar. Nos mostrará la cantidad de puntos que hemos insertado y la distancia total entre ellos. Tendremos además dos acciones adicionales, una para eliminar el último punto de la lista y otra para eliminarlos todos. Los puntos también se pueden eliminar seleccionándolos con el botón derecho del ratón.

Anexo figura 28. Menú de medición

Este diálogo irá actualizando los valores de distancia y cantidad de puntos conforme añadamos o quitemos puntos sobre la superficie del objeto. Podemos ver la actualización de este en las figuras 29 y 30.



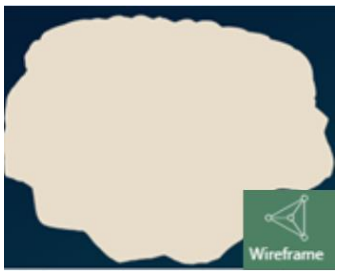
Anexo figura 29. Ejemplo con 7 puntos



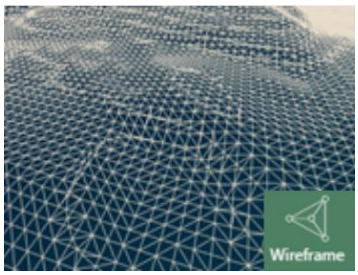
Anexo figura 30. Ejemplo con 12 puntos

Para poder ver la maya que compone nuestra figura 3D, se tiene acceso a un modo de trabajo que proporciona una visualización del objeto con esta textura. Para activarlo, tendremos que hacer *click* en el botón con la etiqueta "*Wireframe*" (figura 327), que se encuentra a la derecha del botón utilizado para activar el sistema de medición.

Si el objeto tiene una buena resolución *a priori* parece que simplemente se le haya aplicado una textura mate, sin embargo, si nos acercamos a este podremos ver los polígonos que forman su superficie. Esto se puede observar en las figuras 31 y 32, esta última es la aplicación de la primera.



Anexo figura 31. Modelo en wireframe

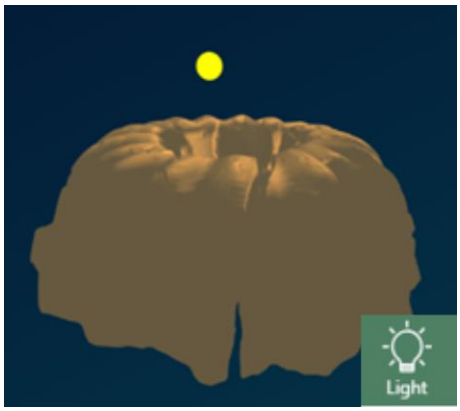


Anexo figura 32. Modelo zoom

El modo de iluminación manual estará disponible al pulsar el último botón de la columna de herramientas, el cual tiene la etiqueta de "*light*". Tras esta selección y para colocar el punto de luz donde se crea pertinente, se tendrá que hacer un *click* con el botón central del ratón en cualquier punto del visor.

Con esta acción añadiremos una luz direccional en el escenario que enfocará hacia el centro de la figura. Si se quiere cambiar el punto de luz, simplemente tendremos que realizar otra selección en el punto deseado, el anterior punto se eliminará. Se muestra el resultado final en la figura 33, en la parte inferior izquierda de esta podemos ver el icono que tenemos que seleccionar para activar este modo.

Para desactivar el punto de luz, tendremos que pulsar el botón "light" de nuevo.

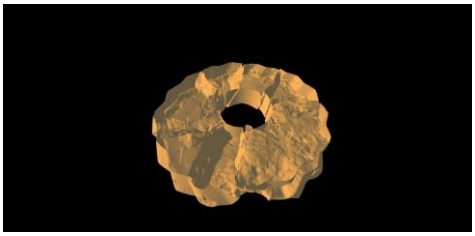


Anexo figura 33. Punto de luz en la figura

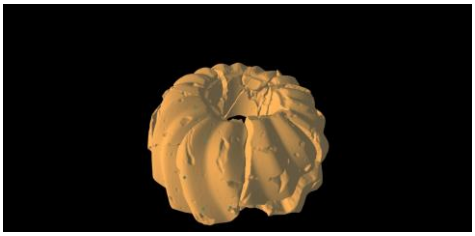
Por último, podremos realizar capturas de pantalla de nuestra aplicación de forma nativa sin utilizar aplicaciones de terceros. Para ello nos dirigiremos al último botón de la columna de herramientas (en la figura 43 aparece el icono) y simplemente lo seleccionaremos.

Anexo figura 34. Botón para realizar capturas de pantalla

Unos ejemplos de capturas realizadas utilizando la herramienta son los que se muestran a continuación. En la figura 35 se hace una captura del corte y en la 36 de los puntos de medida.



Anexo figura 35. Corte



Anexo figura 36. Puntos de medida

10 Anexo: tablas con información sobre la base de datos

En este anexo se adjuntan las tablas con todos los campos que tiene la base de datos:

Categories contiene:

- String: name

Collections contiene:

- String: title
- Text: description

Deteriorations contiene:

- String: name

Materials contiene:

- String: name

Pieces contiene:

- String: name
- Text: description
- Bool: missing
- String: matrix
- Integer: restored_object_id
- String: model_file_name
- String: model_content_type
- Integer: model_file_size
- Date: model_updated_at
- float: width
- float: height
- float: depth

Priorities contiene:

- String: name

Protections contiene:

- String: name

Restored_objects contiene:

- String: title
- Text: description
- Integer: category_id
- Integer: user_id
- String: author
- Text: classification
- Integer: material_id
- String: technique
- String: decoration
- Integer: units_id
- float: width
- float: height
- float: depth
- String: inventory_no
- Bool: in_inventory
- String: owner
- String: deposit
- String: address
- String: location
- t.float latitude
- t.float longitude
- String: epoch
- Integer: state_id
- Integer: deterioration_id
- Integer: priority_id
- Integer: protection_id
- Text: notes
- String: avatar_file_name
- String: avatar_content_type
- Integer: avatar_file_size
- Date: avatar_updated_at

States contiene:

- String: name

Units contiene:

- String: name

Users contiene:

- String: email, default: null: false
- String: encrypted_password, default: null: false
- String: reset_password_token
- Date: reset_password_sent_at
- Date: remember_created_at
- Integer: sign_in_count, default: 0, null: false
- Date: current_sign_in_at
- Date: last_sign_in_at

11 Glosario

- 3D: geométricamente es la representación de un objeto en un espacio tridimensional, es decir que tiene altura, anchura y profundidad.
- APIs: *Application programming interface* en inglés e interfaz de programación de aplicaciones, en castellano, proporciona una serie de funciones y métodos los cuales nos proporcionan información o datos de una aplicación, un sistema operativo o cualquier otro servicio software
- *Active Record*: es la capa que se encarga de la gestión de los modelos y la información actuando como un ORM.
- *Apache server*: servidor web *Open Source* que está construido sobre una arquitectura Unix, generalmente.
- *Back-end*: parte del servidor de un servicio o plataforma web, es transparente al usuario.
- *Canvas*: superficie de representación de las vistas tridimensionales en el visor web .
- *CoffeeScrip*: lenguaje de programación con una sintaxis cómoda y legible, cómodo para el programador que posteriormente se transforma a *JavaScript.*
- *Creative Comons*: organización norteamericana sin ánimo lucrativo que pretende promover la divulgación cultural.
- *Devise*: gema implementada para *Ruby on Rails* que permite la gestión de usuarios de forma segura.
- *Front-end*: parte visible de un servicio o plataforma web con la que el usuario interactúa.
- *Full-Stack*: programador que tiene conocimientos en *front-end* y *back-end.*
- *Heroku*: plataforma que ofrece servicios y recursos a los usuarios, soportando varios lenguajes de programación.
- *Hibernate*: es una herramienta para el ORM implementado para el lenguaje Java que facilita el acceso a los datos.
- *Listener:* método que espera la ejecución de una determinada acción para ejecutarse.
- *Loader:* método que tiene la implementación de carga de un fichero.
- *NodeJs*: es un entorno de programación que se ejecuta sobre *JavaScript*, que está ubicado en la capa del servidor de una aplicación.
- MySQL: tecnología que nos permite gestionar bases de datos de forma relacional, asíncrona y accesible por varios usuarios.
- *Open Source*: software libre y accesible por cualquiera que puede ser redistribuido y modificado.
- ORM: *Object-relational mapping* (en castellano: mapeo objeto-relacional) es sistema procedural referente a la programación que encapsula el acceso a los datos en una serie de funciones que realizan accesos a las bases de datos.
- *Postgre*: base de datos relacional, libre y orientada a objetos.
- *Rails:* abreviación de *"Ruby on Rails".*
- *Raster*: conjunto de píxeles que conforman un sistema de salida digital como una pantalla.
- *Ruby*: lenguaje de programación de tipado dinámico inspirado en *Python*, *Perl* y *Effiel*.

- ***Ruby on Rails, RoR****: framework* web escrito en *Ruby* que ofrece comodidad al programador y sigue el patrón VMC.
- ***Slider:*** botón o barra que puede ser desplazada vertical u horizontalmente dentro de un límite establecido.
- ***snake_case:*** práctica de programación que consiste en separar las palabras de un nombre de clase, variable y objeto mediante el símbolo barra baja y usando minúsculas en todas las palabras.
- ***STL:*** formato de edición 3D que conserva la información de la superficie de la figura sin textura o color.
- ***SVG:*** formato de imágenes vectoriales.
- ***Tests:*** pruebas de implementación, integración o de validación que se realizan durante el desarrollo.
- ***ThreeJS:*** librería escrita en *JavaScript* que ofrece una serie de métodos y acciones para manipular objetos tridimensionales.
- ***VMC****:* patrón de diseño, conocido como modelo, vista controlador. Intenta separar la lógica, vista y la obtención de datos en su sistema de almacenado. En inglés *model, view, controller*.
- ***WebGL:*** es una librería de gráficos web estándar implementada originalmente en C
- ***Web worker, Worker:*** fragmentos de Código que permiten ejecutar tareas en segundo plano no bloqueantes. Permiten que el navegador funcione con fluidez mientras realiza diversas acciones.