Universitat Oberta de Catalunya

enlaz.art

Memoria de Proyecto Final de Grado

Grado en Multimedia

Mención en Desarrollo de Aplicaciones Interactivas

Autor
Luis Joaquín Simón Lázaro

Consultor
Kenneth Capseta Nieto

Profesor
Carlos Casado Martínez

Junio de 2018

Créditos

El presente proyecto hace uso de las siguientes librerías, software, tipografías e imágenes, propiedad de terceros:

- Laravel (https://laravel.com). Framework open-source, bajo licencia MIT.
- ngnix (https://nginx.org). Servidor web ligero, bajo licencia BSD simplificada.
- MariaDB (https://mariadb.org). Servidor de base de datos, bajo licencia GPL versión 2.
- jQuery (https://jquery.org). Librería javascript, bajo licencia MIT.
- Bootstrap (https://getbootstrap.com). Bajo licencia MIT, copyright 2011-2018 Twitter y los autores de Bootstrap.
- Simple QrCode (https://www.simplesoftware.io). Generador php de códigos QR. Bajo licencia MIT.
- Intervention Image (http://image.intervention.io). Manipulación de imágenes. Bajo licencia MIT.
- Carbon (https://carbon.nesbot.com). Manipulación de fechas. Bajo licencia MIT.
- Axios (https://github.com/axios/axios). Librería javascript, bajo licencia MIT.
- Ckeditor (https://ckeditor.com). Librería javascript, bajo licencia MPL versión 1.1.
- Dropzone (http://www.dropzonejs.com). Librería javascript, bajo licencia MIT.
- Open Iconic (https://useiconic.com). Conjunto de iconos, bajo licencia MIT.
- SVGInjector (https://github.com/iconic/SVGInjector). Librería javascript, bajo licencia MIT.
- Europa (https://www.europatype.com). Tipografía web, bajo acuerdo de licencia Adobe Typekit.
- Freight Text (https://philsfonts.com/index.php/fonts/overview/GF060014X1). Tipografía web, bajo acuerdo de licencia Adobe Typekit.
- Open Sans (https://www.fontsquirrel.com/fonts/open-sans). Tipografía, bajo licencia Apache.
- Aquellas imágenes usadas en el sitio web no elaboradas por el autor están accesibles en https://stocksnap.io, bajo licencia Creative Commons CC0.

Dedicatoria

A Laura, que me empujó a dar el primer paso y me acompaña en todos los siguientes.

A nuestros hijos, Erik y Adriana, que tuvieron paciencia.

Abstract

En la sociedad actual proliferan las exposiciones o muestras de arte, tanto a nivel profesional, como amateur. Entre las múltiples facetas desarrolladas en la organización de estos eventos, se encuentran el etiquetado de las obras y, cada vez de forma más habitual, la página web con más información sobre la exposición y las obras. Si bien existen soluciones orientadas al sector profesional (museos, galerías de arte, artistas profesionales), se observan carencias de soluciones orientadas al sector no profesional, que a su vez es más numeroso y en el que encontramos, mayoritariamente, organizaciones con recursos muy limitados (organizaciones sin ánimo de lucro, asociaciones, colegios y centros educativos, entre otras).

enlaz.art es una aplicación web que viene a ofrecer una solución sencilla a este colectivo no profesional, dando respuesta al escenario descrito. La organización se registrará en **enlaz.art** para dar de alta las obras de su exposición, introduciendo la información necesaria para su etiquetado, así como para la página web de ampliación de la información. **enlaz.art** permitirá la generación de las etiquetas descriptivas e identificativas de las obras, que se ubicarán físicamente en la exposición y que incluirán un código QR único. Mediante la lectura de dicho código QR, el visitante accederá a una web donde se mostrará información adicional de la obra y que se generará automáticamente por la aplicación a partir de la información introducida en el proceso de alta por la organización.

El desarrollo de la plataforma se realizará a partir de tecnologías de código abierto, entre otras el *framework* MVC Laravel (versión 5.6), basado en PHP (versión 7+), el servidor de bases de datos MariaDB, el servidor web ngnix y el *framework* CSS Bootstrap (versión 4). Para la generación de las etiquetas de las obras y de los códigos QR se usarán las librerías de PHP Intervention Image (versión 2) y Simple QrCode (versión 2).

Palabras clave: aplicación interactiva, web, arte, php, laravel, QR, trabajo final de grado, tfg.

Abstract (English version)

In today's society, exhibitions or art exhibitions proliferate, both professionally and in an amateur manner. Among the many aspects developed in the organization of these events, are the labeling of the works and, more and more frequently, the website with more information about the exhibition and the works. While there are solutions aimed at the professional sector (museums, art galleries, professional artists), there are shortcomings in solutions aimed at the non-professional sector, which in turn is more numerous and in which we find, mostly, organizations with very limited resources (non-profit organizations, associations, schools and educational centers, among others).

enlaz.art is a web application that comes to offer a simple solution to this non-professional group, responding to the described scenario. The organization will be registered in **enlaz.art** to enter the works of its exhibition, introducing the necessary information for its labeling, as well as for the web page as an extension of the given information. **enlaz.art** will allow the creation of descriptive and identifying labels of the works, which will be physically located in the exhibition, and which will include a unique QR code. By reading this QR code, the visitor will access a website where additional information of the work will be automatically generated by the application, based on the information entered in the registration process, and displayed.

The development of the platform will be based on open source technologies as MVC framework Laravel (5.6 version), based on PHP (7+ version), the MariaDB database server, the ngnix web server and the framework CSS Bootstrap (4 version), among others. The PHP Intervention Image (2 version) and Simple QrCode (2 version) libraries will be used to generate the labels of the works and QR codes.

Keywords: interactive application, web, art, php, laravel, QR, end-of-degree project, tfg.

Notaciones y convenciones

A continuación, se ejemplifican los estilos y convenciones utilizados en la presente memoria.

Título de primer nivel

Título de segundo nivel

Título de tercer nivel

Texto normal y **fragmento de código fuente** en el texto normal.

Texto resaltado

Términos técnicos o palabras en otro idioma

Fragmento de código fuente

```
/* Comentarios en código fuente */
// Esto también es un comentario en el código fuente
Palabras clave en código fuente
(...) /* Indica un bloque o fragmento de código omitido*/
```

Encabezado de tabla 1	Encabezado de tabla 2
Columna 1, fila 1	Columna 2, fila 1
Columna 1, fila 2	Columna 2, fila 2
Columna 1, fila 3	Columna 2, fila3

Figura/Tabla. Descripción de figura o tabla

- Elemento de lista no ordenada
- Elemento de lista no ordenada

1. Elemento de lista numerada
2. Elemento de lista numerada

Índice

Figuras y tablas

Índice de figuras

Índice de tablas

Índice de fragmentos de código fuente

1. Introducción

1.1. Contexto y justificación

La realización de una exposición artística requiere del equipo organizador la gestión de la información sobre la colección expuesta. Esta información llega al visitante en distintas formas, desde los catálogos de la exposición, si los hay, al etiquetado de las obras. El desarrollo de cada forma comunicativa se limita por la capacidad de la organización, en recursos y especialización o experiencia, encontrando una amplia heterogeneidad en la tipología de organizadores de exposiciones.

Respecto a la tipología de organizaciones, pueden distinguirse actores profesionales (como museos, salas de arte o galerías) y actores no profesionales (por ejemplo: asociaciones, empresas u organizaciones con colecciones, artistas aficionados, colegios, entidades públicas y, en definitiva, cualquier entidad que organice algún tipo de muestra).

Mientras que el colectivo profesional dispone de soluciones profesionales verticalizadas (específicas del sector), habitualmente de pago y/o propietarias, el colectivo no profesional suele recurrir a herramientas genéricas y, por tanto, inespecíficas, cuya capacidad de respuesta al escenario descrito suele ser insatisfactoria.

enlaz.art es una aplicación web orientada al colectivo organizador no profesional, cuyo objetivo es proporcionar una herramienta sencilla para la realización del etiquetado de exposiciones artísticas, que mediante la inserción de un código QR en dichas etiquetas, permitirá al usuario acceder a la ficha técnica de cada obra en entorno web.

Figura 1. Logotipo de enlaz.art.

1.2. Objetivos

Objetivos principales

- Desarrollar una aplicación web para la generación de etiquetas identificativas de una exposición de obras y sus fichas técnicas, las cuales serán accesibles vía web mediante la lectura de un código QR en la etiqueta por parte del usuario.

- Ofrecer una primera versión de la aplicación en producción, abierta al público en general, al finalizar el TFG.
- Aprender nuevas tecnologías de desarrollo web, por ejemplo, el *framework* MVC[1] Laravel.
- Implementar ambientes de desarrollo y producción basados en las tecnologías y mejores prácticas actuales de desarrollo web, por ejemplo, entorno de producción *cloud* y publicación de la aplicación mediante *webhooks*.

Objetivos secundarios

- La aplicación deberá permitir el registro y gestión de usuarios organizadores de exposiciones.
- La aplicación generará automáticamente los códigos QR únicos para cada obra.
- El usuario organizador podrá editar la información de cada obra.
- La aplicación generará las etiquetas identificativas de cada obra individualmente o por lotes.
- El usuario visitante, mediante un software lector de códigos QR, accederá a la ficha web detallada de la obra de cuya etiqueta lee el código QR.

1.3. Enfoque y método seguido

El desarrollo del TFG es un proyecto de envergadura, a llevar a cabo en un tiempo acotado. El marco temporal nos viene fijado por el calendario de entregas de la asignatura, por lo que los requerimientos de estas han de ser tenidos en cuenta a la hora de establecer la planificación a seguir. Así, el primer paso será diseñar una planificación que nos ayude a estructurar el desarrollo del proyecto de cara a su culminación con éxito, facilitándonos la monitorización del avance de este y permitiéndonos corregir nuestras actuaciones de forma proactiva frente a posibles desviaciones. El diagrama de Gantt resultante se incorpora en el capítulo Planificación.

En líneas generales, el proyecto se estructura en dos planos. El **plano técnico**, en el que se contemplan el diseño de la arquitectura de la aplicación, su diseño e implementación, la puesta en producción, la creación, diseño y aplicación de una imagen de marca y, en resumen, todo lo relacionado con la aplicación enlaz.art. Por otro lado, el **plano académico**, relacionado con el desarrollo de un TFG, que contempla la realización de la memoria final del proyecto, las entregas parciales, la gestión de la comunicación con el consultor, la realización y envío de los entregables necesarios y todas las cuestiones requeridas para la superación de la asignatura final del Grado en Multimedia. Ambos aspectos deben ser contemplados, coordinados e integrados en la planificación.

[1] Modelo-Vista-Controlador

De este modo, el proyecto se divide en cuatro fases consecutivas, cuyo hito final corresponde en cada caso a la entrega de cada PEC. En cada entrega, además, se recibirá un *feedback* del consultor, por lo que se contempla en la planificación dedicación a la gestión e incorporación de este *feedback* al proyecto. Alineados con las fases referidas, se establecerán objetivos parciales de desarrollo técnico del proyecto, tal y como se recoge en la planificación. En consecuencia, se realizarán revisiones periódicas de seguimiento del proyecto en estos hitos principales, implantando las medidas correctivas pertinentes en caso necesario.

Respecto al equipo humano, el presente proyecto se desarrolla de manera individual —salvo por la colaboración, en el marco de la asignatura, que desarrollan consultor y equipo docente—, por lo que la metodología a seguir difiere de la que se emplearía en un entorno real. Mientras que en un entorno real y en función del equipo de trabajo, se propondría usar algún tipo de metodología ágil, como *SCRUM*, para el desarrollo del presente proyecto, en un entorno académico y en el que podemos obviar las tareas propias del desarrollo de proyectos en equipo, se usará una metodología propia, basada en *Kanban* e inspirada en la filosofía de *Desarrollo Ágil*.

Así, se considerará cada fase del proyecto de manera independiente y se planificará semanalmente el trabajo a desarrollar. Puede considerarse este ciclo semanal como un *sprint*, al estilo de *SCRUM*. Por motivos de organización personal, se realizará seguimiento parcial del estado del *sprint* los martes y revisión final del *sprint* los viernes. En la revisión final, se realizará una retrospectiva sobre el trabajo desarrollado en el *sprint*, los puntos fuertes de la semana y las áreas de mejora y el estado general del proyecto.

2. Descripción

El presente proyecto recoge integralmente el proceso de desarrollo de la plataforma enlaz.art. enlaz.art es una aplicación web que permitirá a usuarios registrados la generación de etiquetas de obras de arte, insertando en estas un código QR único mediante el cual un visitante podrá acceder a una ficha técnica de dicha obra.

Para la realización del proyecto se han seleccionado un conjunto de tecnologías web modernas y abiertas, siendo el *framework* Modelo-Vista-Controlador Laravel (versión 5.6), desarrollado en PHP, el elemento central del proyecto. La elección de Laravel frente a otros productos similares, como Symfony, Code Igniter o CakePHP, responde a varias cuestiones.

La primera, dado que uno de los objetivos del proyecto es aprender un marco de trabajo MVC basado en PHP, seleccionar un producto sencillo, pero a la vez consolidado y con recorrido a futuro. En este sentido, Laravel ofrece equilibrio entre complejidad y rendimiento, una amplia documentación de referencia y una fuerte comunidad de desarrolladores.

La segunda, como plataforma de futuro, Laravel ofrece productos relacionados tanto para proyectos de mayor envergadura -como Statamic o Spark—, como de menor tamaño (Lumen), o herramientas específicas (Horizon, para gestión de colas, Cashier, para integración de pagos y suscripciones, y Envoyer, para *deployments*, entre otras).

En tercer lugar, Laravel nos provee de ambientes de desarrollo adecuadas para equipos de diferentes tamaños. Mientras que para un equipo de tamaño medio/grande podemos desplegar Homestead[2] sobre Vagrant[3], para un equipo pequeño —o unipersonal, como es el caso— podemos usar Valet[4].

Adicionalmente, Laravel nos facilitará la libertad de elección de otros elementos en el despliegue de la plataforma. Por ejemplo, las aplicaciones de Laravel son independientes de la base de datos, del sistema de archivos o de la interfaz de notificaciones. Esto nos permite configurar diferentes ambientes de producción/desarrollo de manera sencilla y diferenciada. Podríamos emplear un servidor de base de datos MariaDB en producción y una base de datos ligera SQLite en desarrollo —sin apenas configuración—, o usar en producción espacio de almacenamiento de Amazon S3, mientras que en desarrollo usar el disco duro de nuestro ordenador.

En relación con la persistencia de datos, para el presente proyecto, se ha optado por usar un servidor de base de datos MariaDB para ambos ambientes de producción y desarrollo. MariaDB es una de las bases de datos más

[2] Homestead es un entorno de desarrollo local de Laravel, basado en una máquina virtual de Ubuntu.

[3] Vagrant es una herramienta para la creación de ambientes de desarrollo virtuales.

[4] Valet es un entorno de desarrollo minimalista de Laravel para usuarios de Mac.

populares de la actualidad, cumpliendo a la perfección los requisitos de la plataforma. Para el archivo de las imágenes de las obras, los archivos QR y las etiquetas generadas, se empleará el propio disco de almacenamiento del entorno (ordenador del autor en el ambiente de desarrollo, VPS[5] en Digital Ocean[6] en el ambiente de producción).

Para finalizar con el *backend*, el proyecto se basará en un servidor web ngnix. Si bien Laravel puede desplegarse en distintos tipos de servidores, como Apache o ngnix, se ha seleccionado ngnix por ser un servidor más liviano en su configuración.

La gestión de versiones del código fuente se realizará mediante GIT, creando un repositorio privado en Bitbucket[7]. Ello nos facilitará configurar la publicación automática del sitio en producción, mediante la configuración de *pipelines* (antes llamados *webhooks*). También, trabajar con GIT nos permite realizar pruebas en ramas diferenciadas del código, testeo, solución de errores, *upgrades*, etc., así como mantener copias de seguridad del proyecto en distintas ubicaciones.

La plataforma se alojará en un servidor privado virtual, basado en Ubuntu Linux, para lo que se creará un VPS en el proveedor Digital Ocean.

La gestión de usuarios (registro, restablecimiento de contraseñas, autenticación) se basará en la funcionalidad por defecto de Laravel. Se configurará el servicio de correo mediante Zoho Mail[8], para las notificaciones a usuarios, y se ampliará la funcionalidad para soportar la gestión de usuarios de distintos tipos: administradores de la plataforma y organizadores de muestras (con diferentes tipos de planes de usuario).

enlaz.art permitirá al usuario aportar imágenes de cada obra dada de alta. Para ello, se integrará la librería javascript Dropzone, que nos facilita una interfaz intuitiva y agradable para la gestión de subidas de archivos al servidor de la aplicación, tanto para el usuario final, como para el desarrollador. Además de las validaciones de dichos archivos en la capa de servidor, Dropzone nos ofrece la posibilidad de realizar validaciones de las subidas en cliente, con lo que reforzaremos la seguridad de la aplicación.

Para la generación de los códigos QR, se integrará la librería PHP Simple QrCode (versión 2), que nos ofrece una interfaz sencilla y configurable para la personalización de dichos códigos según las necesidades de la aplicación. La generación de las etiquetas de las obras requerirá la integración de la librería PHP Intervention Image (versión 2), que facilita la manipulación y creación expresiva de imágenes. Las etiquetas de las obras se generarán como

---

[5] VPS, Virtual Private Server.

[6] Digital Ocean, proveedor de computación en la nube. https://www.digitalocean.com

[7] Bitbucket, de Atlassian, es un servicio de alojamiento de repositorios GIT en la nube.

[8] Zoho es un proveedor de servicios virtuales para la empresa como email, aplicaciones, inventario, etc.

archivos de imagen e, inicialmente, se ofrecerá un único formato de etiqueta. La tipografía usada en las etiquetas será Open Sans, una fuente de tipo *sans-serif*, universal y de amplio uso.

En relación con el *frontend*, Laravel 5.6 incorpora por defecto Bootstrap 4 como marco de trabajo CSS. Este, a su vez, requiere de las librerías javascript jQuery y Popper.js. Se diseñará la interfaz de la aplicación conjugando estos elementos, personalizando Bootstrap desde los archivos SASS, con un enfoque de diseño *responsive*. Además de Bootstrap, Laravel trae por defecto la librería axios, una librería javascript que nos facilitará manejar llamadas AJAX de manera elegante. Por último, se integrarán el conjunto de iconos Open Iconic y las tipografías Europa y Freight Text, en aplicación de la identidad corporativa de enlaz.art.

3. Objetivos

3.1 Principales

- Desarrollar una aplicación web para la generación de etiquetas identificativas de una exposición de obras y sus fichas técnicas, las cuales serán accesibles vía web mediante la lectura de un código QR en la etiqueta por parte del usuario.
- Ofrecer una primera versión de la aplicación en producción, abierta al público en general, al finalizar el TFG.
- Aprender nuevas tecnologías de desarrollo web, por ejemplo, el *framework MVC* Laravel.
- Implementar ambientes de desarrollo y producción basados en las tecnologías y mejores prácticas actuales de desarrollo web, por ejemplo, entorno de producción *cloud* y publicación de la aplicación mediante *webhooks*.

3.2 Secundarios

- La aplicación deberá permitir el registro y gestión de usuarios organizadores de exposiciones.
- La aplicación generará automáticamente los códigos QR únicos para cada obra.
- El usuario organizador podrá editar la información de cada obra.
- La aplicación generará las etiquetas identificativas de cada obra individualmente o por lotes.
- El usuario visitante, mediante un software lector de códigos QR, accederá a la ficha web detallada de la obra de cuya etiqueta lee el código QR.

4. Escenario

En palabras del profesor Manuel Castells, Internet es una "red de redes de ordenadores capaces de comunicarse entre ellos. No es otra cosa. Sin embargo, esa tecnología es mucho más que una tecnología. Es un medio de comunicación, de interacción y de organización social" (Castells, 2000).

El exponencial desarrollo de Internet en las últimas décadas está provocando profundas transformaciones en la sociedad. Manuel Castells denomina esta nueva sociedad como la *sociedad red*; al momento histórico lo define como la ***era informacional***. Una era en la que la información, el acceso a la red y la competencia digital son elementos clave para el desempeño de las personas y las organizaciones.

Castells reflexiona en su conferencia sobre la **divisoria digital**: mientras que la conectividad a la red va generalizándose en las sociedades económicamente más desarrolladas –aún no, por ejemplo, en el continente africano y otras zonas–, la brecha digital se sitúa sobre la competencia digital de la persona u organización. Por tanto, importa más el *saber hacer* y el *poder hacer*, que la capacidad de conexión.

Otro enfoque, a mi modo de ver complementario, es el del profesor Javier Echeverría, quien habla del *Tercer Entorno* o **entorno digital**, como "un nuevo marco espacio-temporal para las interrelaciones sociales y humanas" (Echeverría, 1999). Echeverría define el entorno digital desde un punto de vista matemático, contraponiendo parejas de conceptos característicos entre los entornos natural y urbano (Primer Entorno y Segundo Entorno, respectivamente) y el entorno digital (Tercer Entorno).

Una de las características, según la definición de Echeverría, del entorno digital, es el predominio del concepto de ***representación*** frente al de *presencialidad*: "casi ninguna de las acciones y experiencias que tienen lugar en él requieren la presencia física de los actores (…), sino que son llevadas a cabo mediante representaciones tecnológicamente construidas" (Echeverría, 1999).

En suma, enlaz.art se desarrolla como una herramienta que permitirá a las organizaciones con menos recursos ***poder hacer***. Se trata de aportar valor mediante el acceso a una solución sencilla a organizaciones, para un problema que, por su especificidad y actores implicados, presenta un nicho de mercado delimitado y no cubierto por la gama de aplicaciones disponibles en la actualidad.

También, el proyecto pivota sobre dos conceptos. Por un lado, la idea del **enlace** como concepto fundamental de la red, bajo la premisa de que en la red es más importante el enlace, la relación, la capacidad de llegar, que los propios nodos. Esto es lo que aportaría enlaz.art, capacidad de enlazar los entornos natural y urbano con el digital. De algún modo, conectar la experiencia física de la exposición o muestra de las obras, con una experiencia digital, más completa e hipervinculada.

Y, por otro lado, enlaz.art como **representación** de la propia organización en el entorno digital. Mediante la lectura de los códigos QR en las etiquetas de las obras, el usuario conecta con una extensión *transmedia* de la organización, es decir, generamos una representación digital de la organización que amplía y enriquece la experiencia del usuario.

5. Contenidos

enlaz.art es una plataforma de servicio para organizaciones que desean generar las etiquetas identificativas para sus obras a exponer. Adicionalmente, esta información que se mostrará en las etiquetas y otra añadida por la propia organización conformarán la base de las fichas técnicas de las obras en línea. Por tanto, la plataforma se nutrirá de los datos aportados por las organizaciones usuarias.

Se clasificarán las organizaciones usuarias en distintos niveles (o planes), con el objetivo de homogeneizar el consumo de recursos de la plataforma. Para la puesta en marcha, se ofrecerán dos planes de usuario: esencial y estándar. Todas las organizaciones o usuarios que se registren lo harán en el **plan esencial**, que impondrá una limitación de 25 obras, 1 colección y un periodo de publicación de la colección en la plataforma de 30 días. Transcurrido este plazo, la ficha pública de la colección dejará de ser accesible públicamente.

El **plan estándar** ampliará las capacidades de la plataforma para la organización a 50 obras, en 1 colección, con un periodo de publicación máximo de la ficha pública de la colección de 60 días. De nuevo, transcurrido este plazo, la ficha pública de la colección dejará de ser accesible. Para acceder a este plan, se requiere de la organización que solicite el cambio de plan.

En cualquiera de los planes, la funcionalidad de la plataforma respecto a las obras, las colecciones o su propia cuenta de usuario, será la misma. La organización podrá dar de alta, editar y eliminar obras, de acuerdo con las condiciones de su plan de usuario. Cada obra tendrá un identificador único, a través del cual se accederá a su ficha pública en la plataforma, siempre que esta esté visible. Para estas fichas visibles, la organización podrá generar la etiqueta identificativa, en formato imagen, la cual contendrá el código QR de enlace con la ficha pública visible en línea. Eliminar una obra será una acción irreversible, que requerirá la generación de una nueva etiqueta (el identificador único será de un solo uso, generado aleatoriamente).

En relación con las colecciones, la plataforma permitirá la generación de una página informativa de la exposición, muestra o colección, que resuma el acto expositivo de manera general, mostrando acceso a las fichas de las obras que componen dicha colección. En los planes esencial y estándar que se implementan en el desarrollo del presente proyecto, las organizaciones únicamente cuentan con 1 colección, por lo que se permitirá la edición de la información de esta, así como la publicación de la ficha pública de la colección.

6. Metodología

Como se ha introducido en el apartado 1.3, para el desarrollo del presente proyecto se ha seguido una metodología propia, primando la eficiencia y seguimiento de la gestión del proyecto, y obviando los aspectos relativos a la gestión de la coordinación que deberían tomarse en cuenta en un entorno real.

La metodología empleada se inspira en conceptos de *Kanban*[9] y el *Desarrollo Ágil*[10], organizando la temporalidad del proyecto en *sprints* o ciclos semanales. A partir del documento de planificación, que marca las directrices del proyecto en su totalidad y nos orienta respecto a la situación de avance o retraso en cada momento, se ha trabajado en ciclos semanales, adaptando las prioridades en cada plano —técnico o académico— en función de las fechas de entrega, facilidad/dificultad de las tareas y la situación personal.

El seguimiento de las tareas se ha realizado mediante Wunderlist[11], en el que se han configurado dos listas de tareas independientes, una para la parte técnica y otra para el ámbito académico. Entre las ventajas de usar una aplicación tan sencilla para un proyecto individual, se encuentran la copia de seguridad en la nube, la productividad en base a la escasa configuración y facilidad de uso, la adaptabilidad a la necesidad propia, la posibilidad de establecer y recibir avisos y notificaciones y, por último, que es multiplataforma[12].

De *Kanban* he aplicado el concepto de *tarjetas visuales*, siguiendo un convenio de especificación de tareas con un enfoque visual. Esta sintaxis autoimpuesta me ha ayudado a definir mejor las tareas, consiguiendo descripciones que, al ser revisadas en lista, me permitían orientarme sobre el estado actual del *sprint* mediante la mera inspección visual.

Del enfoque de *Desarrollo Ágil*, he organizado el trabajo en ciclos semanales. Por organización personal, los ciclos se han desarrollado de sábado a viernes, realizando la revisión parcial los martes y la revisión final los viernes. La revisión final, además de para constatar el grado de avance, se ha empleado sistemáticamente para hacer retrospectiva –otro concepto de las metodologías ágiles– del ciclo finalizado, planificando el ciclo siguiente en base al aprendizaje realizado

[9] Método visual para gestionar el trabajo, habitualmente empleado en metodologías ágiles de desarrollo de *software*.

[10] Metodologías de desarrollo de software basadas en el desarrollo iterativo e incremental.

[11] Página principal de la aplicación: https://www.wunderlist.com

[12] Existe versión de Wunderlist para ordenador, tableta, móvil y otros dispositivos conectados.

7. Arquitectura de la aplicación

7.1. Servidor

La capa de servidor consta de los siguientes elementos:

Elemento	Versión	Descripción
VPS	-	Servidor virtual privado, alojado en el proveedor Digital Ocean. Tiene instalada la distribución Linux Ubuntu 16.04.4 LTS y cuenta con 1 vCPU, 2 GB de RAM y 50 GB de almacenamiento. La gestión remota se realiza mediante ssh.
Ngnix	1.13.6	Servidor web ligero y de código abierto.
PHP	7.2.2	Lenguaje de programación de propósito general, de código abierto, ampliamente usado en programación web del lado del servidor.
MariaDB	10.2.13	Sistema de gestión de base de datos. Derivado de MySQL.
Laravel	5.6.12	*Framework* basado en la arquitectura MVC, desarrollado en PHP.
Simple QrCode	2.0	Librería PHP para la generación de códigos QR
Imagick	3.4	Librería PHP para la gestión y manipulación de imágenes
Intervention Image	2.4.1	Libraría PHP para la gestión y manipulación de imágenes

Tabla 1. Elementos de la capa de servidor

7.2. Cliente

La capa de cliente consta de los siguientes elementos:

Elemento	Versión[13]	Descripción
HTML	5	Lenguaje de marcado, para la elaboración de las páginas web.

[13] Versión o especificación

CSS	3	Hojas de estilos en cascada. Nos permitirán configurar la apariencia deseada en la aplicación.
Bootstrap	4	*Framework* CSS. Nos brinda una base de estilos y componentes sobre la que construir más rápidamente sitios web. Facilita la implementación del diseño *responsive*.
Javascript	-	Lenguaje de programación interpretado. Algunas de las librerías utilizadas en este proyecto -jQuery incluida- se desarrollan en este lenguaje: Axios, Popper.js, CKeditor y Dropzone.
jQuery	3.2+	Librería javascript. Nos facilita simplificar el código javascript y nos ofrece múltiples funcionalidades adicionales, como manipular el DOM o manejar eventos.
Axios	0.18	Librería javascript. Nos facilita la gestión de llamadas y respuestas AJAX.
Popper.js	1.12	Librería javascript requerida por Bootstrap, para la gestión de *tooltips*.
SVG Injector	1.1.3	Librería javascript que nos facilita la inserción de iconos SVG, en colaboración con la librería de iconos Open Iconic.
Dropzone	5.4.0	Librería javascript para la gestión de subida de archivos al servidor.
CKEditor	4.8.0	Librería javascript para la creación de editores de texto enriquecidos del lado del cliente.
Open Iconic	1.1	Conjunto de iconos en formatos SVG, webfont e imagen.
Europa	-	Tipografía web.
Freight Text	-	Tipografía web.
Open Sans	-	Tipografía para las etiquetas de las obras.

Tabla 2. Elementos de la capa de cliente

7.3. Base de datos

El modelo Entidad-Relación de la aplicación, implementado en un servidor de bases de datos MariaDB.

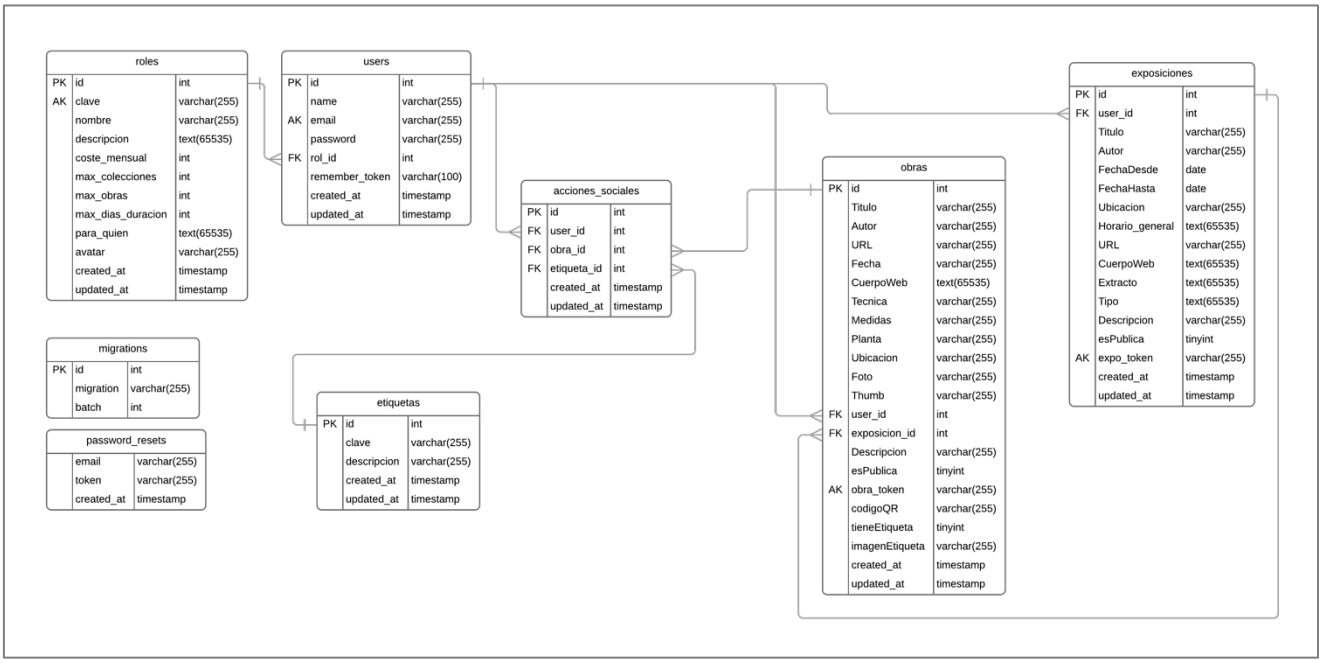

Figura 2. Modelo Entidad-Relación

8. Plataforma de desarrollo

8.1. *Software*

Entre los elementos de *software* más destacables para el desarrollo del proyecto están:

Software	Descripción
Microsoft Visual Studio Code	Entorno de desarrollo multiplataforma, desarrollado por Microsoft. Ampliable y configurable a través de extensiones, incluye múltiples herramientas de serie: emmet, GIT, terminal integrado, etc.
Sourcetree	Cliente con interfaz gráfica de GIT, desarrollado por Atlassian.
Terminal	Mayoritariamente BASH. La gestión del VPS, de la base de datos o la ejecución de distintos comandos durante el proceso de desarrollo, configuración y gestión se realizan mediante línea de comandos.
DBeaver	Interfaz gráfica para gestión de bases de datos y generación de diagramas.
Merlin Project Express	Gestión de proyectos y elaboración de diagramas de Gantt.
Dropbox	Copias de seguridad y almacenamiento en la nube.
Microsoft Word	Procesador de textos.
Wunderlist	Gestor de tareas
Apple Keynote	Diseño de presentaciones. Usado para el *wireframing* Lo-Fi.
Adobe XD CC	Diseño de experiencia de usuario, usado para el prototipado Hi-Fi.
Adobe Photoshop CC	Editor de gráficos y retoque fotográfico.
Adobe Illustrator CC	Gráficos vectoriales.

ScreenFlow	Grabación y edición de *screencasts*[14].

Tabla 3. Elementos de *software* para el desarrollo del proyecto

8.2. *Hardware*

El *hardware* empleado para el desarrollo del proyecto se indica a continuación:

Software	Descripción
Macbook Pro	13", 64 bits, 2,5 GHz Intel Core i7, 16 GB LPDDR3, macOS High Sierra 10.13.4

Tabla 4. Elementos de *hardware* para el desarrollo del proyecto

8.3. Web *apps*

Entre los elementos de software más destacables para el desarrollo del proyecto están:

Web *app*	Descripción
Bitbucket	Proveedor de repositorios GIT y herramientas relacionadas. https://bitbucket.org
Lucid Chart	Aplicación en línea para la realización de diagramas. Accesible en línea en: https://www.lucidchart.com

Tabla 5. Web *apps* empleadas para el desarrollo del proyecto

[14] Grabación digital de la salida por pantalla del ordenador o dispositivo. Puede contener grabación de audio u otros sonidos, como una narración.

9. Planificación

La siguiente planificación contempla el proceso íntegro de trabajo para el desarrollo completo del proyecto, desde el inicio del cuatrimestre hasta la finalización de este. Por ello, este marco temporal nos fija como fechas clave las entregas de las PECs previstas en el plan docente. Además, como hitos adicionales se incorporan puntos de control intermedios que, en colaboración con el *feedback* recibido por las entregas de las PECs previstas, contribuirán a generar una monitorización del estado del proyecto eficaz y proactiva.

9.1. Fechas clave

Fecha	Concepto	Entregables principales
6/marzo/2018	PEC 1	Memoria del proyecto, versión 1
4/abril/2018	PEC 2	Memoria del proyecto, versión 2 Diseños UML enlaz.art, versión alpha
14/mayo/2018	PEC 3	Memoria del proyecto, versión 3 enlaz.art, versión beta Vídeo demo 5 minutos
11/junio/2018	Entrega final	Memoria final del proyecto enlaz.art, versión en producción Presentación en formato libre Vídeo presentación 15 minutos

Tabla 6. Fechas clave en la planificación del proyecto

9.2. Hitos

Fecha	Concepto	Descripción
13/marzo/2018	Diseño formal finalizado	Diseño UML, diseño de base de datos y *wireframes* Lo-Fi y Hi-Fi

27/marzo/2018	enlaz.art, versión alpha	CRUD de las obras, usuarios y ficha web pública de la obra
25/abril/2018	enlaz.art, versión beta	Generación etiquetas con QR

Tabla 7. Hitos en la planificación del proyecto

9.3. Diagrama de Gantt

Nº	Título	Dur.	Inicio	Fin
0	**TFG**	**3,9m**	**21/2/18**	**11/6/18**
1	**Fase 1: Definición formal**	**1,8sem**	**21/2/18**	**5/3/18**
2	Propuesta de proyecto	4 días	21/2/18	26/2/18
3	Aceptación de proyecto		1/3/18	1/3/18
4	Definición de proyecto	2 días		5/3/18
5	Planificación de proyecto	2 días		5/3/18
6	Redactar memoria, versión 1	2 días		5/3/18
7	Entrega PEC 1			5/3/18
8	**Fase 2: Diseño**	**1 mes**	**7/3/18**	**4/4/18**
9	Procesar feedback PEC 1	2 días	12/3/18	13/3/18
10	**Etapa 1. Diseño formal**	**1sem**		**13/3/18**
11	Diseño UML	3 días		9/3/18
12	Diseño de base de datos	2 días		13/3/18
13	Diseño de wireframes Lo-Fi	2 días		13/3/18
14	Diseño de wireframes Hi-Fi	2 días		13/3/18
15	Preparar entregables: UML	2 días		13/3/18
16	Diseño formal			13/3/18
17	**Etapa 2. Desarrollo**	**3sem**		**4/4/18**
18	Crear ambientes desarrollo y producción	2 días		15/3/18
19	Desarrollo: autenticación de usuarios	2 días		19/3/18
20	Desarrollo: CRUD obras	4 días		23/3/18
21	Desarrollo: ficha pública obra	2 días		27/3/18
22	EnlazArt v. alpha: usuarios, CRUD obras y ficha pública obra			27/3/18
23	Redactar memoria, versión 2	1,6sem	23/3/18	3/4/18
24	Entrega PEC 2		4/4/18	4/4/18
25	**Fase 3: Implementación**	**1,2m**	**5/4/18**	**9/5/18**
26	Procesar feedback PEC 2	2 días	11/4/18	12/4/18
27	Desarrollo: integración QR en etiquetas	1sem		11/4/18
28	Diseño de logotipo	3 días		16/4/18
29	Diseño de identidad corporativa	1sem		23/4/18
30	Diseño gráfico web	2sem		25/4/18
31	Vídeo demo EnlazArt	2sem		25/4/18
32	Preparar entregables	1 día		26/4/18
33	Entregables preparados			26/4/18
34	EnlazArt: v. beta con diseño gráfico			25/4/18
35	Redactar memoria, versión 3	2sem	26/4/18	9/5/18
36	Entrega PEC 3		4/3/18	9/5/18
37	**Fase 4: Puesta en producción**	**3,8sem**	**15/5/18**	**11/6/18**
38	Procesar feedback PEC 3	2 días	21/5/18	22/5/18
39	EnlazArt v. final en producción	1sem		21/5/18
40	Elaborar presentación formato libre	1sem		21/5/18
41	Vídeo presentación 15 min. (Tribunal)	1sem		28/5/18
42	Preparar entregables finales	1 día		29/5/18
43	Redactar memoria final	1sem		5/6/18
44	Entrega final		11/6/18	11/6/18

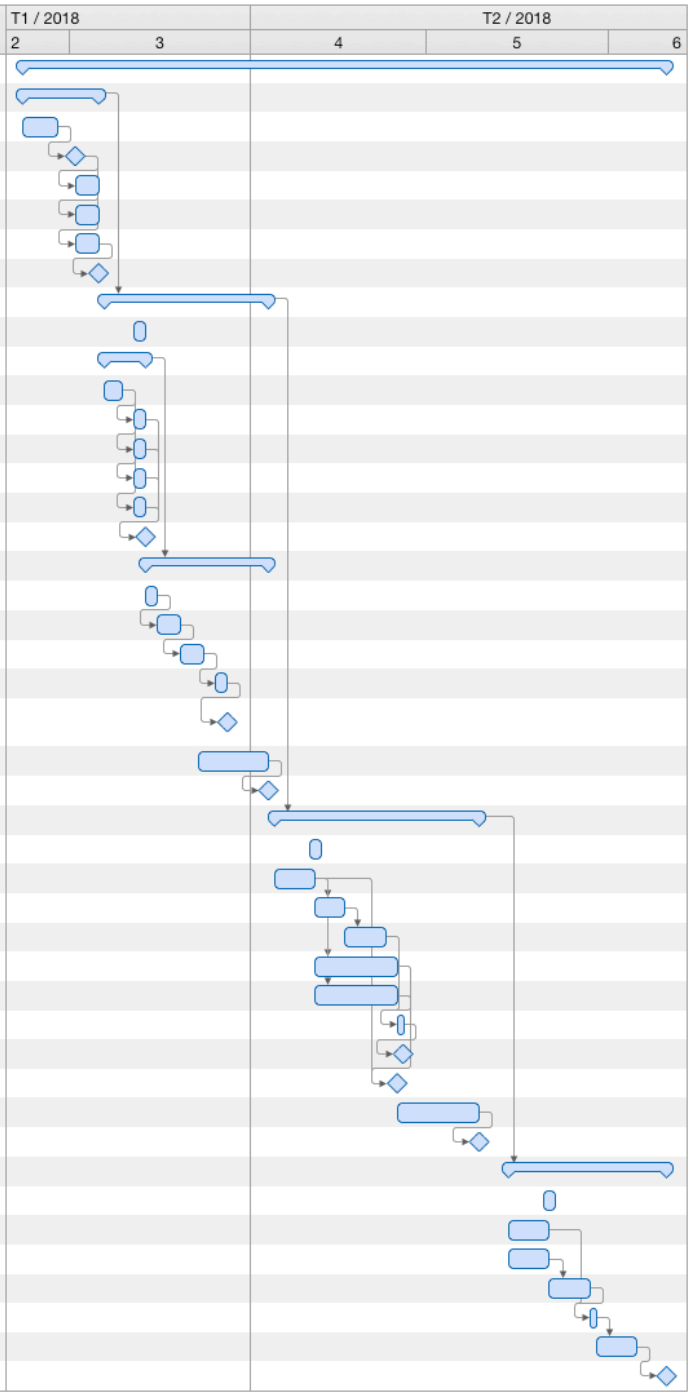

Figura 3. Diagrama de Gantt, planificación del proyecto

10. Proceso de trabajo

10.1. Entrega parcial 1

La primera entrega parcial tenía como objetivo definir y acotar el proyecto. Para ello, se consensuó con el consultor de la asignatura la idea a desarrollar –enlaz.art–, mediante el intercambio de varios correos electrónicos. También, se elaboró una primera propuesta, de carácter informal, descriptiva del proyecto y las tecnologías a emplear.

En paralelo, comencé el aprendizaje de las tecnologías necesarias para el desarrollo de la plataforma enlaz.art, en especial, el *framework* Laravel.

Finalmente, se comenzó a trabajar en la memoria, elaborando la primera versión del presente documento en la que se incluía una primera versión de *abstract*. Se entregó mediante el REC[15] en el plazo estipulado, el 5 de marzo.

10.2. Entrega parcial 2

Los objetivos de esta segunda entrega eran avanzar en una segunda versión de la memoria, con más contenido en la mayoría de los apartados, y realizar una primera entrega de código fuente desarrollado. De acuerdo con la planificación personal, debía además completar el diseño formal (prototipado, UML y base de datos), pero no pude finalizar este objetivo.

Por el contrario, opté por priorizar la parte de desarrollo de la aplicación, pudiendo completar una versión *alpha* en producción que incorporaba, de manera funcional, lo referente al registro y gestión de usuarios y a la creación/visualización/edición/eliminación de obras. Con el objetivo de entregar el proyecto desarrollado de una forma consistente, avancé así mismo en el desarrollo del logotipo de la aplicación y en la selección de la paleta de colores a aplicar en la identidad corporativa de enlaz.art. También, puse en marcha los ambientes de desarrollo y producción, así como el sistema de control de versiones del código fuente.

En relación con la memoria, además de comenzar a dotar de estilos alineados con el diseño de la plataforma y de incluir el *feedback* de la entrega anterior, ampliando los contenidos de la primera versión del documento, se trabajó en los apartados de arquitectura de la aplicación y de plataforma de desarrollo.

La entrega se realizó mediante el REC en el plazo estipulado.

[15] REC, Registro de Evaluación Continua. Funcionalidad de entrega de tareas del Campus Virtual de la UOC.

10.3. Entrega parcial 3

La tercera entrega tenía como objetivos entregar versiones muy avanzadas de la memoria y del proyecto desarrollado. Por tanto, es una fase en la que se encara buena parte del grueso del trabajo y en la que más temas en paralelo se trabajan.

Respecto al proyecto, se culminó el desarrollo de la funcionalidad prevista, realizando también añadidos de acuerdo con las propuestas del consultor (funcionalidad de "Favoritos/Me gusta"). Se trabajó también en la actualización del VPS de producción, así como en la instalación de las librerías para la generación de imágenes tanto en desarrollo, como en producción. En esta fase, se tuvo que trabajar especialmente en localizar y solucionar un *bug* detectado, referente a la encriptación (ver capítulo **19. Bugs**).

Respecto a la memoria, se trabajó en todos los apartados, generando una versión prácticamente definitiva.

Por último, en esta tercera entrega se realizó un breve vídeo descriptivo del funcionamiento de la plataforma. Este vídeo se entregó mediante la herramienta *Presenta* del campus virtual.

La entrega –memoria y código fuente– se realizó mediante el REC en el plazo estipulado.

10.4. Entrega final

La última entrega tenía como objetivo principal la entrega en su versión final de la memoria, el proyecto completo, así como los distintos archivos elaborados durante el proceso de desarrollo (prototipos, planificación, etc.). De acuerdo con la planificación propia del estudiante, los cambios en memoria y proyecto en esta fase han sido los mínimos correspondientes para realizar los últimos ajustes de cara a la última entrega de documentación.

Respecto al proyecto, el único desarrollo realizado ha sido la vista de los favoritos/me gusta propios del usuario, así como la aplicación de diseño y elaboración de contenidos para la parte pública del sitio web.

Como requisito de esta última entrega, se ha desarrollado una presentación escrita sobre el proyecto realizado, así como un vídeo de presentación del proyecto. El vídeo se entregó mediante la herramienta *Presenta* del campus virtual. También, se completó el informe de autoevaluación del estudiante.

La entrega –memoria, código fuente y todos los entregables solicitados (descritos en el anexo I)– se realizó mediante el REC en el plazo estipulado.

11. APIs utilizadas

El desarrollo actual de aplicaciones web requiere, habitualmente, la integración de diversos componentes, librerías y/o APIs. En consecuencia, el presente proyecto hace uso de diversas librerías, tanto del lado del cliente, como del lado del servidor, que nos ayudan en cuestiones específicas que, de otro modo, requerirían de una ingente cantidad de programación adicional. Más aún, el uso de librerías o componentes de referencia en sus distintos ámbitos nos aporta la seguridad de contar con soluciones probadas por la comunidad de desarrolladores, con documentación suficiente para la integración y con ejemplos de uso que, en la mayoría de las ocasiones, nos pueden servir como punto de inicio para nuestra solución.

A continuación, enumero varias de las librerías empleadas, describiendo su uso en la plataforma y la motivación de su uso y/o elección. No siendo una lista exhaustiva, las librerías citadas son las más relevantes para el presente proyecto.

11.1. Simple QrCode.

Simple QrCode es una interfaz sencilla para la **generación de códigos QR de manera dinámica**, del lado del servidor. Es un desarrollo que nos facilita el uso de la clase de PHP BaconQRCode[16], una interfaz muy potente, pero a su vez de mayor complejidad de uso que la anterior. La instalación de Simple QRCode, que se describe a continuación, es muy sencilla en el *framework* Laravel.

1. Instalamos la dependencia en nuestro archivo composer.json[17], editando directamente el array de dependencias requeridas:

```
"require": { (...)
    "simplesoftwareio/simple-qrcode": "~2"
},
```

Código fuente 1. Fragmento de composer.json. Registra las dependencias PHP del proyecto

2. Registramos la librería en el *Service Provider* (*array* **providers** en el archivo config/app.php).

```
'providers' => [ (...)
    SimpleSoftwareIO\QrCode\QrCodeServiceProvider::class, ],
```

Código fuente 2. Fragmento de config/app.php. *Array* de proveedores

[16] https://github.com/Bacon/BaconQrCode

[17] También puede hacerse mediante línea de comandos mediante el comando **composer require librería**Requerida.

3. Registramos el alias de la librería en el *array* **aliases**.

```
'aliases' => [ (...)
        'QrCode' => SimpleSoftwareIO\QrCode\Facades\QrCode::class, ],
```

Código fuente 3. Fragmento de config/app.php. *Array* de alias

Con la librería así disponible para su uso en nuestra aplicación, podremos generar los códigos QR que se adapten a nuestra necesidad. Un fragmento de ejemplo de generación de código QR, en las primeras pruebas de uso de la librería:

```
<img src="    // para poder ver el código QR en el navegador como archivo png,
                                    // hay que enviarlo codificado en base 64
        {!! base64_encode(QrCode::format('png')     // base64_encode codifica el QR generado en base 64
                                                    // QrCode instancia la clase Simple QrCode
                                                    // format() nos permite especificar el formato deseado
        ->color(54,52,52)   // color del código QR generado
        ->margin(.8)        // podemos aplicarle un margen al código generado
        ->errorCorrection('Q')      // tolerancia a errores. A mayor tolerancia, mayor complejidad tiene el código QR.
        ->size(300)         // tamaño del código generado
        ->generate( config('app.url', 'https://enlaz.art') . '/id/' .$obra->obra_token)) !!} ">
                // generate() genera el código QR de acuerdo a todas las especificaciones que le hemos dado
                // previamente. Recibe como parámetro la información que codificamos en el código, en este caso,
                // la dirección URL a la que llevará el código QR al ser leído por el dispositivo del usuario.
```

Código fuente 4. Ejemplo de uso de Simple QrCode

11.2. Intervention Image.

Intervention Image es una librería PHP de código abierto para la **manipulación de imágenes** del lado del servidor. En nuestro caso, las etiquetas generadas por la aplicación para las obras se envían al navegador en formato imagen, por lo que el uso de esta librería nos facilita la configuración personalizada de las etiquetas. La instalación en la aplicación es completamente análoga a la de Simple QrCode descrita en el apartado anterior.

No obstante, para poder hacer uso de todas las opciones de Intervention Image, deberemos hacer una configuración respecto al controlador de manejo de imágenes que se usará. Por defecto, PHP incorpora el controlador GD, que no nos permite seleccionar la tipografía de las etiquetas, salvo por un reducido abanico de

opciones. Por ello, deberemos instalar la extensión de PHP Imagick[18], para a continuación configurar Intervention Image para que use dicha opción.

1. En primer lugar, debemos publicar el archivo de configuración de Intervention Image para sobreescribirlo[19]. En línea de comandos: `$ php artisan vendor:publish`.
2. El comando anterior nos habrá generado una copia del archivo de configuración de Intervention Image en nuestra carpeta config, llamado image.php. Modificamos la línea del *driver* por el valor `'driver' => 'imagick'`.

A partir de esta configuración, Intervention Image usará el controlador Imagick. Un ejemplo de uso de la librería, para la generación de una etiqueta se incluirá en los Anexos.

11.3. Dropzone.

Dropzone es una librería javascript de código abierto para la **gestión de subidas de archivos** a nuestro servidor, del lado del cliente. Si bien podríamos implementar la funcionalidad de subidas mediante programación del lado del servidor, Dropzone nos provee de una interfaz sencilla para el desarrollador e intuitiva para el usuario. El uso de Dropzone en la plataforma enlaz.art se circunscribe a la subida de imágenes para las obras.

Su uso requiere importar la librería en la página o vista donde la vayamos a usar, así como sus archivos CSS de estilos asociados, incorporándola en nuestro sitio o directamente mediante el uso de algún CDN:

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/dropzone/5.4.0/min/dropzone.min.css">
// importación de estilos para Dropzone desde un CDN.
(...)
<script src="https://cdnjs.cloudflare.com/ajax/libs/dropzone/5.4.0/min/dropzone.min.js"></script>
// importación de la librería Dropzone desde un CDN.
```

Código fuente 5. Fragmento: importación de Dropzone en una vista

La librería permite que el usuario pueda seleccionar los archivos a subir de dos formas: pulsando sobre el área visible del control –su comportamiento es análogo al de un *input* de tipo *file* de HTML5– o, también, arrastrando y soltando (*drag and drop*) el archivo en dicha área visible. De cualquier modo, queda en manos del desarrollador gestionar dicha subida en el lado del servidor.

A continuación, configuración de Dropzone para la gestión de subidas de imágenes en la vista de creación de una nueva obra.

---

[18] http://www.php.net/manual/en/book.imagick.php

[19] Publicar en nuestra carpeta de configuraciones (config) el archivo de configuración de una clase para evitar hacer cambios en el código fuente de las librerías es una buena práctica de programación.

```
<script>
(...)
var miDropzone = new Dropzone('.dropzone', {
        // instanciación y parametrización del objeto Dropzone
        url: '/obra/{{ auth()->user()->id }}', // url en la que gestionaremos la subida en el servidor
        dictDefaultMessage: 'selecciona la imagen de la obra', // traducción del mensaje visible por defecto
        addRemoveLinks: true,
        dictRemoveFile: 'Borrar foto', // traducción de mensaje
        dictCancelUpload: 'Cancelar subida', // traducción de mensaje
        dictCancelUploadConfirmation: '¿Está seguro de que deseas cancelar la subida?', // traducción de mensaje
        paramName: 'fotoObra', // establecemos el nombre del parámetro que gestionaremos en la url de destino
        acceptedFiles: 'image/*', // filtramos el tipo de archivos que aceptaremos
        maxFilesize: 1, // tamaño máximo aceptado en Mb
        maxFiles: 1, // número máximo de archivos a subir
        headers: { 'X-CSRF-TOKEN': '{{ csrf_token() }}'} // cabecera con el token CSRF, requerida por Laravel
                // por seguridad (ver apartado 16.2. Seguridad)
});
miDropzone.on('error', function (file, res) {
        // gestión de errores en la subida de archivos. Capturamos los errores que retorna la url que gestiona las subidas
        var msg_error = res.errors.fotoObra[0];
        $('.dz-error-message:last > span').text(msg_error);
});
miDropzone.on('removedfile', function (file, res) {
        // una vez subido el archivo, el usuario puede decidir eliminarlo... La librería axios, y por tanto esta funcionalidad,
        // se detallan en el apartado siguiente.
(...) });
miDropzone.on('success', function (file, res) {
        // si la subida es exitosa, actualizamos la interfaz con feedback para el usuario
        $('#Thumb').val(res);
        $('#FotoThumb').attr("src", res);
});
(...)
</script>
```

Código fuente 6. Ejemplo de uso de Dropzone

11.4. Axios.

Axios es una librería javascript, del lado del cliente, que nos facilita realizar llamadas AJAX con una sintaxis elegante. Laravel incorpora esta librería por defecto, con lo cual no es necesario importar la librería explícitamente, y su uso en el presente proyecto se circunscribe a la funcionalidad de gestión de subidas de imágenes de las obras al servidor.

Se retoma el fragmento de código del apartado anterior, dedicado a Dropzone, para ilustrar el uso de axios.

```
miDropzone.on('removedfile', function (file, res) {
// en este punto, el usuario ha seleccionado la imagen de la obra y ha sido subida al servidor.
// Usamos axios para lanzar las llamadas necesarias para eliminar la imagen subida al servidor.
    axios({
        method: 'delete', // indicamos el método http de la petición
        url: '/obra/{{ auth()->user()->id }}', // ruta para la eliminación de la imagen
        data: { fotoURL: $('#FotoThumb').attr("src"), }, // previamente, hemos almacenado la ubicación en el
                // servidor de la imagen subida de la obra
        headers: { 'X-CSRF-TOKEN': '{{ csrf_token() }}' } // envíamos el token CSRF. Requisito de seguridad.
    }).then(res => {
        $('#FotoObra').addClass('d-none'); // gestionamos la respuesta a la llamada AJAX previamente lanzada
    });
});
```

Código fuente 7. Ejemplo de uso de axios

12. Diagramas UML

A continuación, se adjunta el diagrama de clases UML más relevantes implementado en la aplicación.

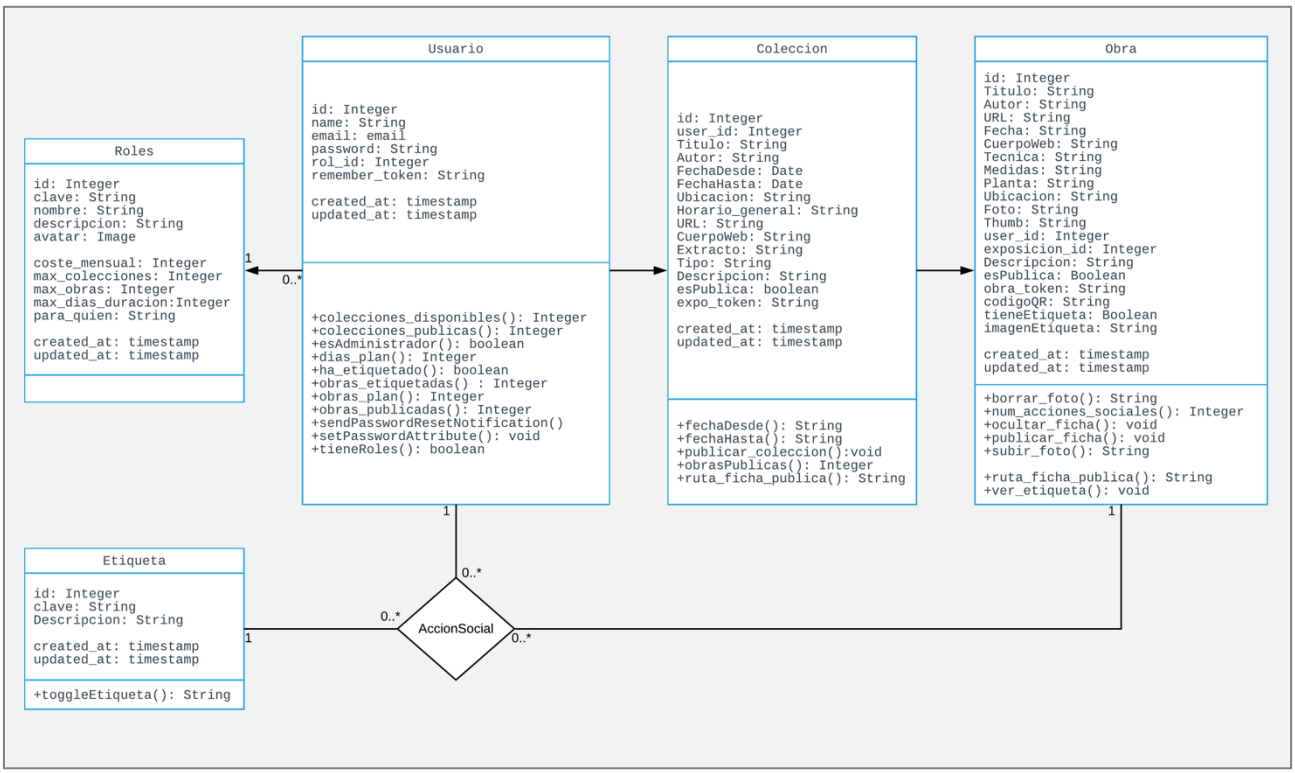

Figura 4. Diagrama de clases del proyecto

13. Prototipos

Se adjuntan a continuación los prototipos realizados en la fase de diseño. En ambos casos, prototipos de baja y de alta fidelidad, se diseñaron las pantallas de la aplicación de mayor relevancia: página inicial con y sin usuario autenticado, inicio de sesión, vista de índice de obras y colecciones, vista de cuenta de usuario, vista de creación/edición de obras, vista de edición de colección y cuenta de usuario. Adicionalmente, se realizó el prototipo de alta fidelidad de la etiqueta de la obra generada por la aplicación. Los prototipos se incluyen en resolución completa como anexos al presente documento (ver Anexo 1).

13.1. Lo-Fi

El prototipado de baja fidelidad se realizó usando el programa de presentaciones Apple Keynote.

Figura 5. Lo-Fi: Página inicial

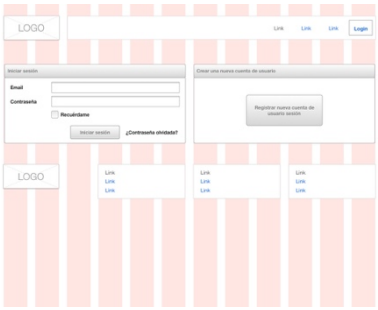

Figura 6. Lo-Fi: Inicio de sesión / registro de usuario

Figura 7. Lo-Fi: Página inicial con usuario autenticado

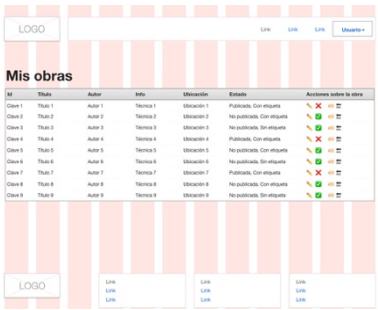

Figura 8. Lo-Fi: Mis obras

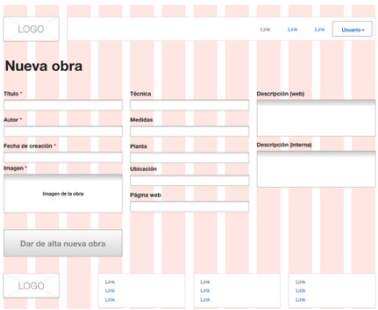

Figura 9. Lo-Fi: Nueva obra

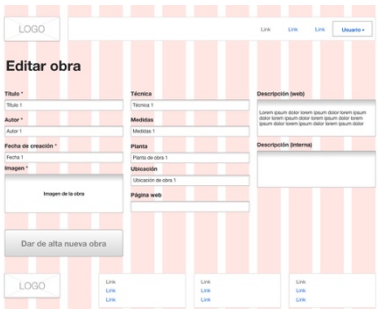

Figura 10. Lo-Fi: Editar obra

Figura 11. Lo-Fi: Ficha pública de la obra

Figura 12. Lo-Fi: Generar etiqueta

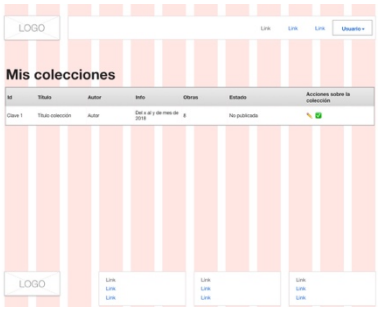

Figura 13. Lo-Fi: Mis colecciones

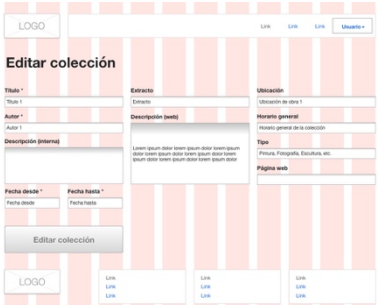

Figura 14. Lo-Fi: Editar colección

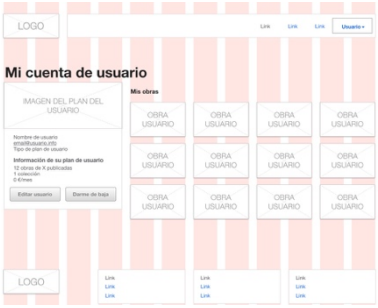

Figura 15. Lo-Fi: Mi cuenta de usuario

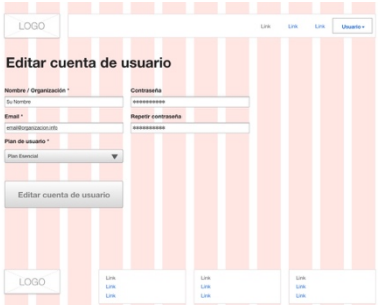

Figura 16. Lo-Fi: Editar cuenta de usuario

13.2. Hi-Fi

El prototipado de alta fidelidad se realizó usando el programa Adobe XD CC.

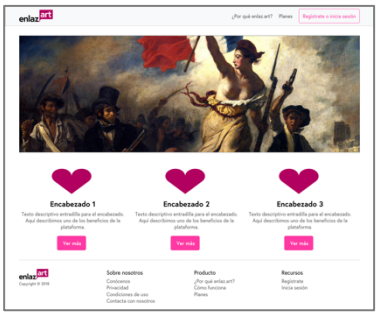

Figura 17. Hi-Fi: Página inicial

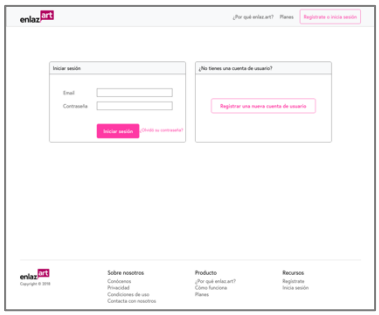

Figura 18. Hi-Fi: Inicio de sesión

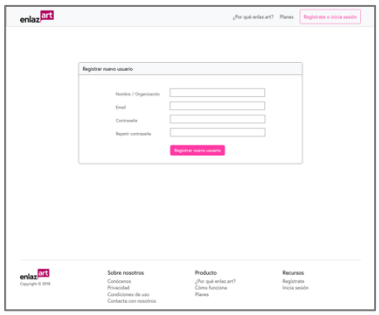

Figura 19. Hi-Fi: Registro de usuario

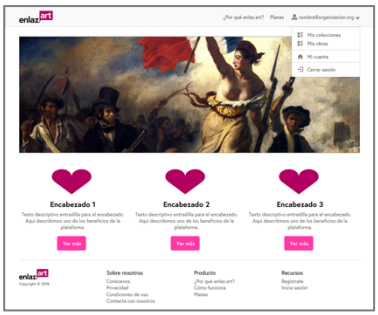

Figura 20. Hi-Fi: Página inicial con usuario autenticado

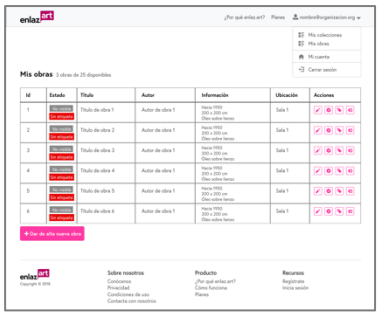

Figura 21. Hi-Fi: Mis obras

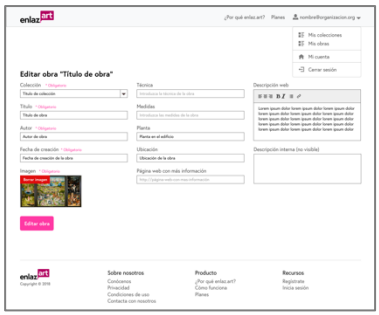

Figura 22. Hi-Fi: Editar obra

Figura 23. Hi-Fi: Generar etiqueta

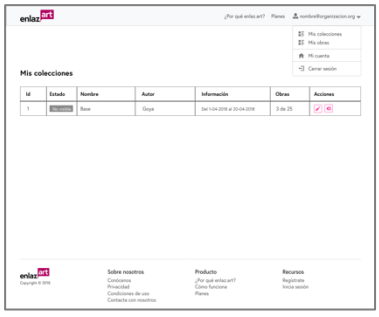

Figura 24. Hi-Fi: Mis colecciones

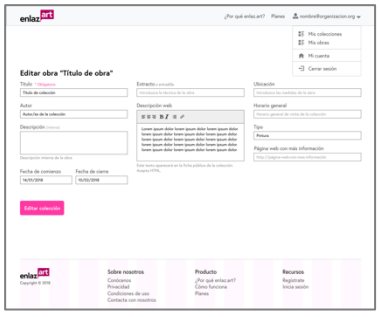

Figura 25. Hi-Fi: Editar colección

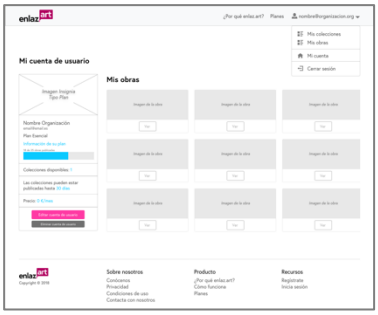

Figura 26. Hi-Fi: Mi cuenta de usuario

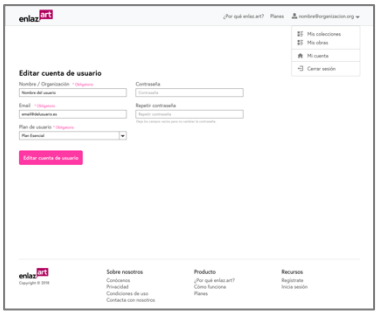

Figura 27. Hi-Fi: Editar cuenta de usuario

13.3. Etiqueta de obra

El prototipo de alta fidelidad de la etiqueta de obra se realizó usando el programa Adobe Illustrator. Para la puesta en marcha del proyecto se generarán todas las etiquetas en el mismo formato y tamaño.

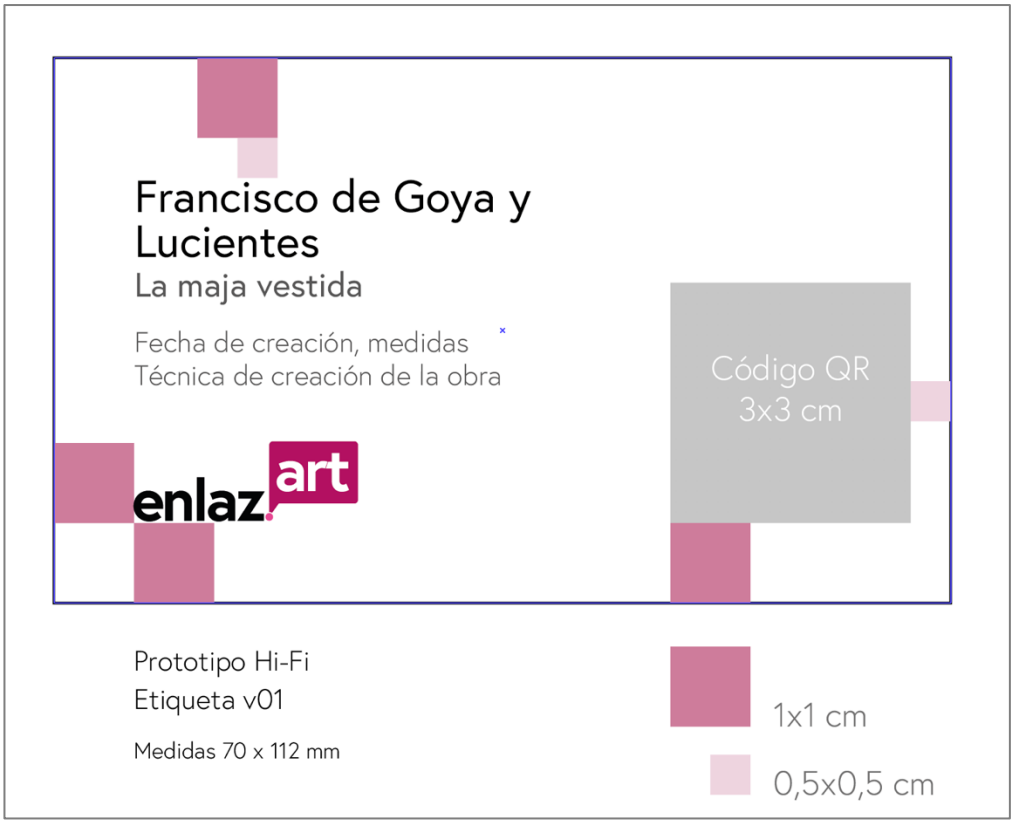

Figura 28. Hi-Fi: etiqueta generada para una obra

14. Perfiles de usuario

La plataforma **enlaz.art** contempla tres grandes tipos de usuarios para el momento de su puesta en producción: administradores del sistema, usuarios del plan Esencial y usuarios del plan Estándar. A continuación, se presentan dos personajes de usuario que corresponden a los tipos de usuario Esencial y Estándar.

14.1. Personajes de usuario

14.1.1. Usuario 1: Oleg, 33 años

Oleg es un fotógrafo amateur. Hace poco, le han ofrecido montar su primera exposición en una cafetería de su ciudad y está ilusionado por ello. Sin embargo, tiene que preparar las fotografías a exponer, hacer el montaje completo y el material publicitario. Ha encontrado enlaz.art en un buscador de Internet y, como su exposición es pequeña, decide darse de alta en el plan Esencial para crear las etiquetas de las fotografías. Así, además de solucionar de forma sencilla el etiquetado, obtendrá un sitio web para su exposición, lo que le ayudará a dar más difusión y mejorar su imagen de cara al futuro.

14.1.2. Usuario 2: Astrid, 39 años

Astrid es directora de un colegio de educación infantil y primaria. Todos los años organiza un concurso de dibujo, en colaboración con la Asociación de Madres y Padres de Alumnos. En el colegio se dispone de muy pocos recursos y el personal está saturado de trabajo, por lo que el montaje de los dibujos y su etiquetado es una de las tareas más tediosas. Astrid encuentra enlaz.art y constata que necesita un plan Estándar. Tras darse de alta y solicitar el plan Estándar, puede organizar la muestra de los dibujos del concurso de forma sencilla. También, al publicar la web de la colección, podrá enviar el enlace a las madres y padres para que, quienes no puedan acudir, puedan ver los dibujos a través de Internet.

15. Usabilidad

El presente proyecto se ha desarrollado bajo los principios de la metodología de Diseño Centrado en el Usuario. A partir de los perfiles de usuario anteriormente descritos, se desarrollaron los distintos prototipos. Si bien el estudiante entiende que lo ideal, más aún en el desarrollo de una plataforma con una funcionalidad nueva en el mercado, sería la realización de diversos ciclos de test con usuarios e iteración en el diseño, la limitación temporal y de recursos del proyecto, las diferentes pruebas del diseño y usabilidad de la plataforma se han restringido a la evaluación heurística por parte del propio estudiante.

Algunas de las **directrices aplicadas al diseño** de la plataforma son:

- Desarrollar una interfaz de usuario intuitiva y sencilla de usar y aprender.
- Aplicar un diseño consistente y coherente entre las diversas páginas del sitio.
- Aplicar la identidad corporativa de enlaz.art al sitio web, realizando un diseño plano, aplicando una paleta de colores sencilla y minimalista y utilizando una combinación tipográfica con elevada legibilidad.
- Implementar una composición visual o maquetación del sitio web fácilmente comprensible, para lo cual se ha empleado, en general, la estructura de rejilla de 12 columnas de Bootstrap 4[20].
- Componer las diferentes páginas del sitio de manera consistente, con una zona superior de cabecera común, personalizada para cada tipo de usuario, una zona de contenido central propia de cada página y una zona inferior de pie de página común a todos los usuarios.

En relación con la arquitectura de la información del sitio web, se distinguen 3 sistemas de navegación:

1. **Sistema de navegación principal**. Global, personalizado por tipo de usuario, se ubica en la cabecera de la página.
2. **Sistema de navegación secundario**. Global, personalizado por tipo de usuario, está ubicado en el pie de página de todas las páginas del sitio web.
3. **Sistema de navegación contextual**. En cada página del sitio, en función del contenido.

El **sistema de clasificación de la información** es ambiguo y orientado a la audiencia. Respecto al **etiquetado,** se ha seguido la premisa de disponer enlaces únicamente textuales para los enlaces en general, aplicando iconos únicamente a aquellos –enlaces o botones– específicos de usuarios autenticados.

[20] Bootstrap 4 implementa Flexbox y diversas funcionalidades para poder realizar el diseño de nuestro sitio web conforme a nuestras necesidades.

16. Seguridad

En el ámbito de la seguridad, constatado que la seguridad absoluta no existe, se ha puesto el foco en la adopción de medidas desde un punto de vista integral y por capas. A continuación, se enumeran algunas de las medidas desplegadas en diferentes aspectos que, si bien no componen lista exhaustiva, sí dan muestra suficiente del enfoque adoptado.

16.1. VPS

El servidor de producción se encuentra en un VPS alojado en el proveedor Digital Ocean. El acceso a la máquina virtual únicamente se puede realizar de dos formas:

1. **Acceso desde la web de Digital Ocean**. Para ello, se requiere usuario y contraseña y superar la autenticación de **doble factor**, implementada mediante la aplicación Google Authenticator[21] en el teléfono móvil del estudiante.
2. **Acceso mediante *ssh***, sin contraseña. Para ello, se ha creado una clave privada, protegida con contraseña, en el ordenador del estudiante. La clave pública relacionada se ubica en el VPS. La conexión sólo se permite a conexiones *ssh* (seguras) autenticadas con dicha clave privada.

Además, el proveedor implementa multitud de medidas de seguridad[22] física, de acceso y de infraestructura, cuyo alcance está fuera del ámbito de este proyecto. Entre ellas, nos permite desplegar balanceadores de carga o *firewalls* que se ajusten a nuestras necesidades. En este caso, se ha configurado, como se observa en la Figura 29, un ***firewall*** para permitir únicamente determinadas conexiones a la plataforma.

El acceso al servidor de **base de datos** es únicamente accesible desde el propio VPS, por lo que el acceso es, o bien desde línea de comandos –usuario autenticado–, o bien desde las propias aplicaciones alojadas en el servidor. En ambos casos, se requiere el uso de usuario y contraseña válidos.

Por último, la aplicación se sirve a través del **protocolo seguro *https***, produciéndose así la comunicación cliente-servidor de manera cifrada.

[21] Aplicación para la obtención de códigos de un solo uso de Google. https://itunes.apple.com/es/app/google-authenticator/id388497605?mt=8

[22] Información sobre seguridad en Digital Ocean: https://www.digitalocean.com/security/

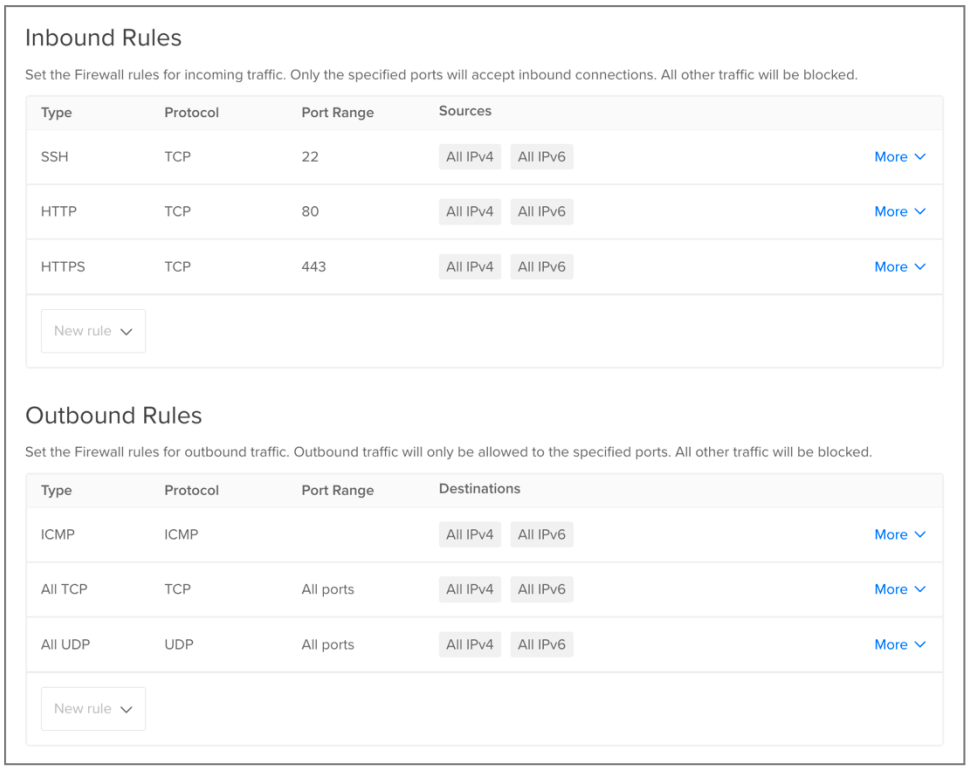

Figura 29. Configuración del *firewall* aplicado a la VPS en producción

16.2. Laravel

Laravel incorpora por defecto múltiples medidas de seguridad, por lo que se mencionarán algunas de las más relevantes empleadas para el presente proyecto.

Validación de peticiones de formularios.

Laravel nos permite crear clases específicas encargadas de acoger las diferentes reglas de validación de los formularios. Esta capa de validación en el servidor, además, nos permite centralizar —por tanto, mantener mejor— la lógica de nuestra aplicación.

```
(...)
public function rules() {
    /* En la función rules() del Request es donde especificamos las validaciones que debe verificar nuestro
    *          formulario.
    $rules = [
            'name' => 'required', // especificamos que el campo name sea requerido
            'email' => [ 'required', 'email', // el campo email es requerido y de tipo email
                    'unique:users,email,' . $this->route('usuario')->id
                    // En la línea anterior, validamos que el email sea único en el campo email
                    // de la tabla users para el usuario especificado por el id de la ruta
            ],
```

```
                'rol_id' => 'required', // especificamos que el campo rol_id sea requerido
        ];
(...)
}
```

Código fuente 8. Fragmento del *Request* ActualizarUsuarioRequest

Middlewares.

Los *middlewares* de Laravel son un mecanismo para el filtrado de peticiones *http* a nuestra aplicación. Por ejemplo, la funcionalidad por defecto que incorpora de autenticación de usuarios nos dota del *middleware auth* que, aplicado a una ruta, la protegerá del acceso de usuarios no autenticados.

A continuación, se muestra un fragmento de ejemplo de *middleware* creado *ad hoc*, para comprobar si el usuario que accede a una ruta dispone del rol necesario. En caso afirmativo, el *middleware* permite el acceso a la ruta. En caso negativo, redirigirá la petición a la ruta *index*.

```
class ComprobarRolUsuario {
        public function handle($request, Closure $next) {
                /**     handle() es la función que se activa por defecto al emplear el middleware   */
                $roles = array_slice( func_get_args(),2 );
                if (auth()->user()->tieneRoles($roles)) {
                        /** si el usuario autenticado tiene alguno de los roles especificado en el array $roles,
                        *       entonces se autoriza la petición */
                        return $next($request);
                }
                return redirect('/index');
        }
}
```

Código fuente 9. Fragmento del *middleware* ComprobarRolUsuario

Para aplicar el *middleware*, podemos hacerlo en el constructor de nuestro controlador. Previamente, ha sido registrado en nuestra aplicación (app\Http\kernel.php) con la palabra clave **roles**.

```
class UsuariosController extends Controller {
function __construct(){
$this->middleware('auth');
                $this->middleware('roles:admin',['except'=>['edit','update','show']]);
                // A todos los métodos de este controlador se les aplicará el middleware auth,
                // y, a continuación, el middleware roles (ComprobarRolUsuario).
```

```
            // El segundo no se aplicará a los métodos edit, update y show.
        }
(...)
```

Código fuente 10. Aplicación de *middlewares* en UsuariosController

CSRF.

Laravel nos provee de protección contra ataques de tipo CSRF [23] por defecto, aplicando el middleware `VerifyCsrfToken` a todas las rutas del grupo *middleware* web. En la práctica, esto obliga a insertar un código o *token* oculto en todas las peticiones de formularios para su procesamiento.

Inyección de SQL.

Laravel nos ofrece diversas formas de interactuar con nuestras bases de datos, entre ellas el Query Builder y Eloquent. Query Builder usa mapeo de parámetros mediante PDO, previniendo así ataques de inyección de SQL. Eloquent es un ORM[24], una abstracción, en la que interactuamos con las bases de datos a través de modelos. De nuevo, nos protege de serie de posibles inyecciones de SQL. Las posibles vulnerabilidades surgirían de un empleo incorrecto de sentencias SQL mediante el *facade* DB.

En el presente proyecto, el acceso a bases de datos se ha realizado íntegramente mediante el Query Builder o Eloquent.

Asignación masiva.

En Laravel, es posible asignar masivamente una petición a los campos de un registro de una base de datos. Para prevenir la modificación fraudulenta de datos sensibles en actualizaciones o inserciones de registros mediante estas asignaciones masivas se requiere, en la definición del modelo, activar o permitir explícitamente los campos para los que permitiremos realizar asignaciones masivas. Para ello, en el modelo podemos emplear dos atributos:

- *fillable*: en el que especificaremos los campos susceptibles de asignación masiva.
- *guarded*: en el que especificaremos los campos no permitidos para asignación masiva.

```
class Obra extends Model {
(...)
    protected $fillable = [
            /* Los siguientes campos se podrán insertar/actualizar mediante asignación masiva */
            'Titulo', 'Autor', 'URL', 'Fecha', 'CuerpoWeb', 'Tecnica', 'Medidas', 'Planta', 'Ubicacion', 'Foto',
```

[23] https://es.wikipedia.org/wiki/Cross-site_request_forgery

[24] https://es.wikipedia.org/wiki/Mapeo_objeto-relacional

```
                'Thumb', 'user_id', 'exposicion_id', 'Descripcion', 'obra_token'
        ];
(...)
```

Código fuente 11. Ejemplo de uso del atributo *fillable*

16.3. Copias de seguridad

La realización de copias de seguridad del código fuente, en sus distintas versiones, se apoya en el uso de un repositorio GIT que acoge íntegramente el código del proyecto. Este repositorio se publica en Bitbucket, de manera privada, y a su vez se despliega en el VPS de producción. Así, disponemos en todo momento de 3 copias de seguridad del código fuente, en sus diferentes versiones.

Quedan fuera del repositorio GIT los archivos de entorno de cada ambiente, lo cual nos permite ajustar el comportamiento de la aplicación de acuerdo con las necesidades de cada escenario (por ejemplo, habilitar la depuración y vista de errores en entornos de desarrollo y hacer lo contrario en entorno de producción). También, las imágenes y base de datos de cada entorno, en especial el entorno de producción.

De cara a la puesta en producción, el establecimiento y despliegue de la política de copias de seguridad de la plataforma sería una cuestión que valorar para el futuro, entendiendo el estudiante que queda fuera del ámbito del proyecto.

16.4. Ámbito de la aplicación

En el ámbito de la aplicación se ubican distintas medidas de seguridad de las que, por motivos de extensión, únicamente se mencionarán dos. En primer lugar, además de la validación de formularios mediante el mecanismo propio de Laravel, se ha incorporado validación de elementos de formulario en la capa del cliente, mediante el uso de atributos propios de HTML5 (por ejemplo, el atributo *required* en un campo de texto para indicar que dicho campo es obligatorio).

En segundo lugar, a la hora de implementar las diferentes vistas de la aplicación, Laravel nos ofrece dos formas de escribir código personalizado para enviar al cliente mediante su lenguaje de plantillas Blade. En el presente proyecto, se ha usado, en general, marcado del tipo {{ tuVariable }}[25], con el que le indicamos a Blade que escape los caracteres mediante la función de PHP **htmlspecialchars**, previniendo así ataques de tipo XSS[26].

[25] Con el marcado {!! tuVariable !!}, Blade omitiría el escapado de caracteres.

[26] https://es.wikipedia.org/wiki/Cross-site_scripting

17. Versiones

enlaz.art se trata de una aplicación web, por lo que el versionado de la aplicación difiere del que se seguiría de tratarse de una aplicación clásica compilada y/o empaquetada, hablando en este caso de diferentes *builds*. Por ello, se considerarán tres versiones (o *builds*) que corresponderán a las entregas segunda, tercera y final, siendo en este orden las versiones *alpha*, *beta* y *release*.

El desarrollo del proyecto se ha realizado mediante una filosofía de entrega continua[27], para lo cual se han empleado diversas herramientas: GIT, para el versionado del código fuente, webpack[28], para la automatización y generación de *builds*, composer[29], para la gestión de dependencias PHP, y npm[30], para la gestión de dependencias javascript.

La identificación única de las versiones entregadas se realizará mediante la cadena *hash* de identificación del *commit* aportado en cada una de las entregas[31].

17.1. Versión *alpha*

Incorpora la funcionalidad de gestión y autenticación de usuarios, una primera versión de la estructura de la página web y la creación-lectura-edición-borrado de obras. **Commit: 3b09875**.

17.2. Versión *beta*

Amplía la versión *alpha* con funcionalidad de gestión de las colecciones, más desarrollo de la gestión de usuarios y obras, y una primera versión de la generación etiquetas de obras con sus códigos QR embebidos. Incluye la funcionalidad de marcar como Favorito/Me gusta cada obra por parte de los usuarios de la plataforma. También, se ha trabajado en la parte pública del sitio web (*Cómo funciona*). **Commit**: **a58a965**.

17.3. Versión *release*

Amplía la versión *beta* con la versión para puesta en producción del proyecto, con toda la funcionalidad prevista inicialmente y las páginas estáticas (sin acceso autenticado) desarrolladas. **Commit**: **927eec3**.

---

[27] Entrega continua o *continuous delivery*, es un enfoque de desarrollo de *software* basado en la entrega de versiones en ciclos cortos. Relacionado con las metodologías de *Desarrollo Ágil*. https://en.m.wikipedia.org/wiki/Continuous_delivery

[28] Sistema de *bundling* para el desarrollo web. https://webpack.js.org

[29] https://getcomposer.org

[30] https://www.npmjs.com

[31] En un repositorio GIT, cada *commit* se identifica de forma única por una cadena *hash*. No existen dos *commits* con el mismo *hash* y, habitualmente, se hace referencia a ellos por los primeros caracteres del *hash*.

18. Instrucciones de uso

El uso de la plataforma **enlaz.art** por parte del usuario es muy sencillo, en línea con el resto de las aplicaciones y plataformas web en la actualidad. Por ello, se describe brevemente a continuación el funcionamiento de la plataforma, sin desarrollarse un manual de instrucciones al uso –una práctica en desuso en el entorno web.

En vez de esto, en la propia plataforma y accesibles públicamente, se ubicarán ayudas y explicaciones suficientes para los usuarios que lo requieran[32], ofreciendo además la posibilidad de solicitar ayuda adicional por medio del formulario de contacto en el sitio web.

18.1. Registro y acceso de usuarios

Creación de cuenta de usuario

El usuario accederá al formulario de registro mediante los enlaces habilitados en todas las páginas del sitio web ("Regístrate o inicia sesión"). Introducirá los datos solicitados (Nombre/organización, email y contraseña) y, una vez completados los datos, el sitio le mantendrá con la sesión iniciada. Todas las cuentas registradas serán inicialmente del tipo "Plan esencial".

Acceso a la plataforma

El usuario con cuenta registrada podrá acceder a la plataforma mediante los enlaces habilitados en todas las páginas del sitio web ("Regístrate o inicia sesión"). En el momento del acceso, el usuario puede seleccionar que se recuerde el inicio de sesión en su dispositivo, marcando la casilla de verificación "Recuérdame". En este caso, en futuras visitas en dicho dispositivo la sesión se mantendría iniciada, por lo que no sería necesario volver a introducir la dupla email/contraseña.

18.2. Obras

El elemento principal con el que interactúa el usuario son las obras. Un usuario podrá crear y publicar el número de obras disponibles, de acuerdo con su plan de usuario.

Vista "Mis obras": crear y borrar obras

El usuario autenticado podrá acceder a su listado de obras haciendo clic sobre el menú que muestra su email y, a continuación, haciendo clic en "Mis obras". Si tiene obras disponibles para publicar en su plan, un botón ("Dar de alta nueva obra") le permitirá hacerlo, dirigiéndole al formulario de creación de obra.

---

[32] https://enlaz.art/como-funciona

Para cada obra registrada, se mostrará una fila en una tabla que contendrá el listado completo de obras del usuario. En la columna ubicada a la derecha, se dispondrán las opciones disponibles para cada obra: editar la obra, publicar/ocultar la obra, generar la etiqueta y borrar la obra.

Editar una obra

El formulario de edición de la obra será accesible para el usuario autenticado desde "Mis obras", en el botón con el icono del lápiz ubicado en la columna "Acciones" para cada una de las obras. Mediante dicho formulario, el usuario podrá personalizar la información correspondiente a cada obra: título, autor, técnica, medidas…

Una información que el usuario puede aportar es la imagen de la obra, que ilustrará la ficha técnica de la obra en la web. Para aportar la imagen, el usuario tendrá dos mecanismos en el área de imagen: podrá hacer clic sobre ella, lo que abrirá en su ordenador la ventana de diálogo del sistema para requerirle que localice una imagen en su dispositivo, o bien *arrastrará* a dicha zona el archivo de la imagen. En caso de que el archivo no sea válido (medidas incorrectas, tamaño excesivo, etc.), el usuario recibirá información que le permitirá subsanar el error.

En el caso de que la obra ya tuviese una imagen y el usuario desee reemplazarla, deberá usar un botón "Borrar imagen" que se superpondrá sobre ella, para a continuación volver a aportar la imagen deseada.

Finalizada la edición, el usuario deberá guardar los cambios pulsando el botón "Editar obra".

Publicar una obra

Las obras introducidas en la plataforma serán por defecto no visibles. Esto quiere decir que, aún cuando se generen las etiquetas y un usuario externo desee acceder mediante el código QR de la etiqueta a la ficha web de la obra, esta ficha no será visible hasta su publicación por parte del usuario que ha aportado dicha obra.

Para publicar una obra en la plataforma, el usuario autenticado deberá pulsar, desde "Mis obras", en el botón con el icono de la marca de verificación ubicado en la columna "Acciones" para cada una de las obras.

Si el usuario deseara ocultar una obra en la plataforma, sin llegar a borrarla, deberá pulsar, desde "Mis obras", en el botón con el icono del aspa ubicado en la columna "Acciones" para cada una de las obras.

Los botones de publicación y borrado se ubican en la misma zona y se muestran alternativamente, en función del estado de visibilidad de la obra.

Etiquetas

Para generar la etiqueta identificativa de cada obra, el usuario autenticado deberá pulsar, desde "Mis obras", en el botón con el icono de la etiqueta ubicado en la columna "Acciones" para cada una de las obras. Al hacerlo, accederá a una página donde podrá descargar la etiqueta generada, en formato imagen (formato PNG), para lo cual deberá hacer clic sobre la imagen.

Para generar todas las etiquetas identificativas de las obras publicadas del usuario, desde "Mis obras", el usuario deberá pulsar el botón "Generar etiquetas". Al hacerlo, accederá a una página donde podrá descargar, en formato imagen (formato PNG) todas las etiquetas individuales de aquellas de sus obras que se encuentren publicadas.

18.3. Colecciones

Una colección en enlaz.art es una agrupación de obras. En los planes Esencial y Estándar —los únicos disponibles inicialmente en enlaz.art—, el usuario dispondrá únicamente de una colección, por lo que la gestión de esta se restringe a la edición de información de la colección y su publicación. El usuario podrá publicar la ficha de su colección en la plataforma durante un máximo de días que especificará su plan de usuario.

Vista "Mis colecciones"

El usuario autenticado podrá acceder a sus colecciones haciendo clic sobre el menú que muestra su email y, a continuación, haciendo clic en "Mis colecciones".

Para cada colección, se mostrará una fila en una tabla que contendrá el listado completo de colecciones del usuario. En la columna ubicada a la derecha, se dispondrán las opciones disponibles para cada colección: editar la colección y publicar la colección.

Editar una colección

El formulario de edición de la colección será accesible para el usuario autenticado desde "Mis colecciones", en el botón con el icono del lápiz ubicado en la columna "Acciones" para cada colección. Mediante dicho formulario, el usuario podrá personalizar la información correspondiente a cada colección: título, autor, extracto, cuerpo de la página…

Finalizada la edición, el usuario deberá guardar los cambios pulsando el botón "Editar colección".

En el caso de que la colección se encuentre publicada, los campos de fechas serán no editables.

Publicar una colección

Las colecciones del usuario serán por defecto no visibles. La ficha pública de la colección mostrará la información especificada por el usuario para dicha colección, así como ofrecerá acceso a las obras públicas de dicha colección.

Para publicar una colección en la plataforma, el usuario autenticado deberá pulsar, desde "Mis colecciones", en el botón con el icono de la marca de verificación ubicado en la columna "Acciones" para cada colección.

Una vez publicada la colección, esta no podrá ocultarse por parte del usuario. Una colección pública será accesible únicamente en el rango de fechas especificado por el usuario previamente a su publicación.

18.4. Favoritos

El usuario de la plataforma podrá marcar cada obra como *Favorita* o *Me gusta*. Conceptualmente, son acciones análogas, pero semánticamente albergan diferencias. Usar una u otra opción tiene que ver con la forma de consultar las obras favoritas o con *Me gusta* por parte del usuario.

Marcar como "Favorito" o "Me gusta"

El usuario autenticado que visite la ficha pública de cualquier obra dispondrá de una botonera con ambas opciones bajo la imagen de la obra. Haciendo clic en cada una de las opciones, activará o desactivará la opción en cuestión alternativamente: si la obra no es favorita y hace clic en *Favorito*, la marcará como favorita. Si la obra es favorita y hace clic en *Favorito*, la desmarcará como favorita. Análogamente ocurrirá con la opción *Me gusta*.

Ver mis "Favoritos" o mis obras con "Me gusta"

El usuario autenticado, mediante los enlaces en el menú principal de navegación y en el pie de página, podrá acceder a la página "Mis favoritos". En esta página, verá dos secciones. En la primera, se mostrarán las obras que el usuario ha marcado como "Favorito". En la segunda, se mostrarán las obras que el usuario ha marcado como "Me gusta".

Mediante el enlace "Ver" ubicado bajo cada obra, el usuario podrá acceder a dicha obra, para así visualizar e interactuar con ella, por ejemplo, eliminando una etiqueta aplicada previamente.

19. Bugs

Durante el desarrollo del proyecto se ha trabajado intentando minimizar los *bugs* (o errores de programación) de la aplicación. Se ha aplicado una política de corrección temprana, no encontrando *bugs* significativos en el momento de la entrega.

No obstante, a continuación, se explica un *bug* encontrado por casualidad y que, a pesar de resultar finalmente de fácil solución, requirió una ardua labor de investigación, prueba y error.

Doble encriptación de la contraseña de usuario

En el proyecto se ha implementado la funcionalidad de autenticación de usuarios por defecto de Laravel. Este mecanismo permite el registro de usuarios, inicio de sesión, regeneración de contraseñas... Como es lógico, la contraseña del usuario se almacena en la base de datos de manera cifrada.

Por motivos de estética e identidad visual, se sobrescribieron los emails de regeneración de contraseñas que de serie traía Laravel. Tras esto, se descubrió que se modificaba el almacenado en base de datos de la contraseña para hacerse en texto plano (no cifrado) en algunas ocasiones, por lo que era necesario implementar un mutador (ver **código fuente 12**, modelo *User*) en la base de datos que forzase el cifrado de la contraseña. Este cambio se incorporó en el commit **4f50ba9**[33], de 29 de marzo. A partir de aquí, se continuó trabajando con normalidad.

```
(...)
// Definiendo un mutador, forzamos a que todas las actualizaciones de dicho campo puedan ser tratadas antes
//   almacenarse en la base de datos.
// Así se configuró inicialmente en el commit 4f50ba9b
public function setPasswordAttribute($password) {
        $this->attributes['password'] = bcrypt($password); }
(...)
```

Código fuente 12. Mutador del campo *password* en el modelo *User*

Al cabo de cierto tiempo, se comenzó a observar un comportamiento extraño de la plataforma. Al intentar acceder a la aplicación, la contraseña del usuario no funcionaba. Se usaba el sistema de regeneración de contraseña y, una vez cambiada esta, se permanecía con la sesión iniciada en la plataforma, por lo que parecía que había sido un error puntual. Al acceder posteriormente, si se realizaba mediante *cookie* usando la opción "Recuérdame", el acceso era correcto. Por el contrario, al volver a introducir usuario y contraseña para acceder, el sistema informaba de que la combinación no era correcta.

[33] Accesible en el repositorio GIT del proyecto.

Tras realizar diversas búsquedas por Internet y multitud de pruebas, pude comprobar que, en algún momento del proceso de regeneración de la contraseña, el sistema por defecto realizaba una encriptación de la nueva contraseña del usuario. Esta encriptación, a su vez, volvía a encriptarse por el mutador referido anteriormente.

La solución a este error pasó por una implementación del mutador en la que se comprobase el valor a introducir en la base de datos: en caso de ser texto plano, debería cifrarse mientras que, si ya llegaba cifrado, se introduciría directamente en la base de datos.

La solución se implementó en el commit 088f2e5, de 30 de abril, y corresponde a la implementación definitiva.

```
(...)
public function setPasswordAttribute($password) {
        // Comprobar el password nos llega cifrado para almacenar en base de datos.
        // Usamos una expresión regular para comprobarlo.
        if ( preg_match('/^\$2y\$[0-9]*\$.{50,}$/', $password) ) {
                // Ya está cifrado -> no volvemos a encriptar
                $this->attributes['password'] = $password;
        } else {
                // No está cifrado -> encriptar
                $this->attributes['password'] = bcrypt($password);
        }
}
(...)
```

Código fuente 13. Mutador del campo *password* en el modelo *User*, versión final

20. Proyección a futuro

A partir de la puesta en producción de la plataforma enlaz.art, se contemplan diversas líneas de actuación para sucesivas versiones y/o ampliaciones del proyecto. A continuación, se enumeran y describen de manera breve.

20.1. Formatos de etiqueta

Una primera mejora que implementar en la plataforma sería la posibilidad de generar las etiquetas en diversos formatos. En la puesta en producción, se diseñó un formato único de etiqueta, por motivos de sencillez y alcance del proyecto. El horizonte ideal, sin embargo, sería que la plataforma ofreciera varios formatos predefinidos y, opcionalmente, para los planes de usuario más avanzados, se permitiera al usuario configurar su etiqueta personalizada con algún tipo de interfaz visual de composición WYSIWYG[34].

20.2. Planes de usuario

Inicialmente, enlaz.art se ha configurado con 2 tipos de planes de usuario con una capacidad básica. No obstante, en función de la aceptación en el mercado y la evolución de la adopción, deberían ampliarse los planes de usuario ofrecidos, planteando incluso establecer planes de pago con suscripción mensual. Ello requeriría, además de incorporar funcionalidades más avanzadas, como la personalización de etiquetas u otras descritas en el presente apartado, desarrollar o integrar algún tipo de pasarela o sistema de pagos recurrentes. Así mismo, especificar las condiciones de uso del sitio y revisar la normativa legal vigente para dar cumplimiento total a la misma.

Los planes de pago del usuario ampliarían la capacidad de obras para el usuario, el número de colecciones a manejar y el máximo de días de publicación de las colecciones, así como abrirían la posibilidad de acceder a las funcionalidades avanzadas de la plataforma.

20.3. Funcionalidad social

Otro tipo de funcionalidades que mejorarían la experiencia de uso de la plataforma son aquellas relacionadas con el enfoque social de enlaz.art. Desde el marcado de "Me gusta" o "Favoritos", que permiten a un usuario interaccionar con las obras publicadas, debería avanzarse hacia la interacción entre usuarios. Podría plantearse la posibilidad de que, además de lo anterior, el usuario pudiera hacer comentarios a las obras y/o colecciones y que estos comentarios pudieran recibir valoraciones por parte de otros usuarios. A partir de esto, es fácil observar cómo podríamos establecer la figura del usuario prescriptor o de referencia, que podría ayudar a otros usuarios a

[34] *What You See Is What You Get*. Se trata de herramientas en las que el usuario compone viendo visualmente el resultado que obtendrá.

descubrir más obras a través de su catálogo de recomendaciones. Esto podría derivar, finalmente, en un tipo de red social en torno al arte, que las galerías podrían usar para llegar a más usuarios, potencialmente compradores o coleccionistas.

20.4. Venta en línea

En línea con lo anterior, sería interesante implementar la opción de que el usuario organizador pudiese ofrecer y vender las obras de la plataforma en línea. Ello requeriría revisar la normativa vigente para realizar una adecuación completa a todos los requerimientos y obligaciones derivados, así como desarrollar/implementar el sistema de cobros/pagos necesarios para soportar las transacciones a realizar. Finalmente, habría de profundizarse en el análisis de mercado para determinar el porcentaje de la transacción que correspondería a la plataforma.

20.5. Etiqueta de colección

Resultaría de interés para el usuario disponer de la posibilidad de enlazar las obras con la ficha pública de la colección, así como de generar directamente una etiqueta que enlazara con la ficha pública de la colección. Este escenario se presenta en las organizaciones y/o usuarios de menor tamaño y con menos requisitos, y les permitiría reducir el número de etiquetas generadas para organizar su exposición o colección.

20.5. Geolocalización

En función de la adopción de la plataforma, podría plantearse introducir elementos de geolocalización en las etiquetas de las obras. Esto, alineado con el desarrollo de mayor funcionalidad social, permitiría a los usuarios descubrir exposiciones u obras en función de su ubicación.

20.6. Colecciones permanentes

En la puesta en marcha de enlaz.art, se ha planteado que las colecciones puedan publicarse por un máximo de vigencia de acuerdo con las condiciones del plan de cada usuario. No obstante, existen usuario u organizaciones con colecciones de carácter permanente. Para atender este escenario, deberían establecerse colecciones de carácter permanente, usando los campos de fecha de apertura y cierre como información de referencia, por ejemplo, para establecer puntualmente periodos de cierre de la colección (por ejemplo, vacaciones, etc.).

20.7. Etiquetas privadas

A partir de diversas conversaciones con usuarios potenciales y del análisis propio del escenario abordado, se detecta como área de mejora la posibilidad de incorporar a cada obra más información de gestión: número de inventario, referencia de catálogo, precio (ver apartado 20.4. Venta en línea) y otros. Se trataría de ampliar la información de gestión, pudiendo así adoptar la plataforma como herramienta de gestión de colecciones. Esto

implicaría analizar los perfiles de usuario de coleccionista y museos. Una primera funcionalidad que desarrollar sería la posibilidad de generar etiquetas privadas, con las que etiquetar, de manera no visible para el público visitante de la exposición, los cuadros (o elementos) de la colección: ocurre habitualmente, por ejemplo, que los cuadros en las exposiciones no van adecuadamente etiquetados y la organización dispone únicamente del catálogo para identificar cada obra. Adjuntando a cada obra su etiqueta identificativa se solucionaría de manera sencilla esta situación.

20.8. Linked Art[35]

En la fase de definición del proyecto, se valoró titularlo *LinkedArt*, en vez de enlaz.art. Una rápida prospección en Internet reveló la existencia previa de un proyecto llamado Linked Art, por lo que se redefinió el nombre del proyecto, observando la evolución de dicho proyecto. Tal como se describe en su página web, Linked Art es una comunidad dedicada a la creación de un modelo compartido para la descripción de Arte, basado en la iniciativa de Linked Open Data[36].

Así, el objetivo es avanzar en la descripción del Arte mediante el modelado a partir de datos estructurados, que posteriormente se enlazarían de acuerdo con las especificaciones del grupo dedicado a la iniciativa del W3C[37], incorporando así la información a la web semántica.

Si el proyecto *Linked Art* avanzase significativamente –no se han observado avances durante el plazo de desarrollo del proyecto– debería valorarse la adopción de esta estructura de datos, amplificando así el alcance potencial de las obras publicadas por los usuarios de la plataforma.

[35] https://linked.art

[36] https://es.wikipedia.org/wiki/Datos_enlazados

[37] https://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

21. Presupuesto

A continuación, se presenta una valoración económica del coste de desarrollo del proyecto enlaz.art.

Concepto	Perfil	Jornadas	Coste total[38]
Definición, planificación y gestión de proyecto	Jefe de proyecto	3	**1.440 €**
Diseño formal (UML y base de datos)	Jefe de proyecto	3	**1.440€**
Diseño formal (prototipado Lo-Fi, Hi-Fi, etiquetas)	Diseñador gráfico	2	**800€**
Configuración infraestructura (servidores, GIT, DNS, VPS)	Técnico de sistemas	1	**240€**
Diseño de logotipo e identidad corporativa	Diseñador gráfico	5	**2.000€**
Diseño gráfico web	Desarrollador *frontend*	5	**1.600€**
Desarrollo aplicación web	Desarrollador *backend*	15	**4.800€**
Puesta en producción	Jefe de proyecto	0,5	**240€**
	Desarrollador backend	0,5	**160€**
Documentación: memoria del proyecto	Jefe de proyecto	5	**2.400€**
Material audiovisual de presentación y promoción	Jefe de proyecto	1	**480€**
	Diseñador gráfico	2	**800€**
	Especialista multimedia	3	**1.200€**
Costes de infraestructura anuales (VPS, email)			**400€**
Total			18.000€

Tabla 8. Presupuesto económico

[38] Los precios no incluyen el IVA aplicable.

22. Análisis de mercado

22.1. Público objetivo

La plataforma **enlaz.art** se dirige, en su configuración de puesta en producción, a dos tipos fundamentales de usuarios.

En primer lugar, organizaciones y/o artistas amateurs que deseen organizar muestras de pequeño tamaño, con carácter ocasional. Son organizaciones y/o personas con muy poco o nulo presupuesto dedicado a la organización de las muestras y, con carácter general, se trata de eventos de acceso libre al público. Este tipo de organizaciones encajarían sus necesidades en el **Plan Esencial**, el plan de usuario que se asigna a todos lo nuevos usuarios y que ofrece capacidad de gestión de 1 colección (que correspondería con la exposición organizada) que podría publicarse durante 30 días y con un límite de 25 obras.

En segundo lugar, organizaciones y/o artistas amateurs que deseen organizar muestras de tamaño y duración media, con carácter ocasional. De nuevo, cuentan con muy poco o nulo presupuesto dedicado a la organización de las muestras, siendo estas en general de acceso libre al público. Este grupo de usuarios encajaría en el **Plan Estándar**, que ofrece capacidad de gestión de 1 colección que podría publicarse durante 60 días y con un límite de 50 obras.

22.2. Mercado actual de soluciones

El mercado actual de soluciones para el sector descrito se compone mayoritariamente de dos tipos de soluciones principales.

Por un lado, **soluciones específicas del sector**. Son soluciones fundamentalmente orientadas a galerías de arte, de tipo escritorio o SaaS[39], como por ejemplo ITGallery[40]. Por ello, tratan de cubrir de manera integral toda la posible funcionalidad requerida, por ejemplo, ofreciendo la posibilidad de integrarse con la página web del cliente, de gestionar el inventario o colección y los terminales de punto de venta o de realizar envíos de *emailings*. Suelen ser soluciones de pago y que conllevan una curva de aprendizaje elevada, por la elevada funcionalidad que aparejan. Ambos elementos hacen que el público objetivo en enlaz.art desconozcan y/o no tengan en cuenta este tipo de soluciones para su casuística.

---

[39] SaaS, *Software as a Service* o Software como servicio. Se realiza pago por suscripción y se accede a la solución habitualmente a través de Internet, pues se aloja y mantiene en la infraestructura del proveedor.

[40] https://www.itgalleryapp.com

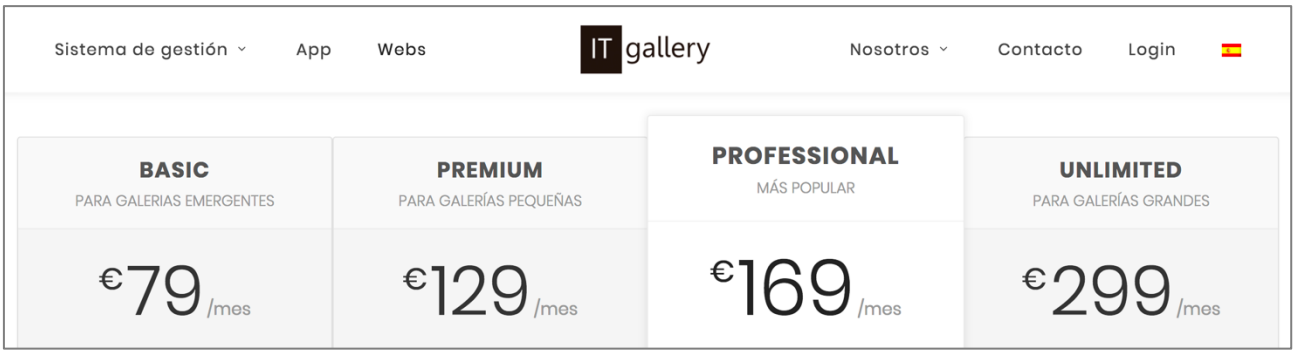

Figura 30. Captura de pantalla: precios de ITGallery (mayo, 2018)

Por otro lado, **soluciones generalistas** como, por ejemplo, Microsoft Word o Adobe InDesign. Son programas accesibles al público en general, con un coste asequible y documentación y comunidades de usuarios amplias. Son herramientas de uso más o menos general (Microsoft Word es un procesador de textos muy versátil, considerado un estándar *de facto*; Adobe InDesign es un software de maquetación muy conocido) que, sin embargo, no proveen de soluciones adaptadas al público objetivo de enlaz.art. Su adopción tiene que ver con la accesibilidad a la herramienta y conocimiento por parte del público objetivo, más que con una elección adaptada a la necesidad. Por tanto, suelen conllevar un coste de adaptación o personalización en dedicación que, finalmente, genera insatisfacción en los usuarios.

Adicionalmente, podríamos hablar de un tercer tipo de soluciones –con el objetivo de completar el análisis del mercado actual de manera exhaustiva–, siendo estas las aplicaciones gratuitas o no de **gestión de colecciones** (de elementos del tipo que sean: inventarios de libros, cuadros, fotos, muebles, etc.). Este tipo de aplicaciones, como GCStar[41] (gratuita y de código abierto) o Home Inventory[42] (software propietario y de pago), ofrecen funcionalidad orientada a la gestión interna de colecciones de elementos. No suelen permitir el etiquetado de dichos elementos y, de hacerlo, se trata de etiquetas identificativas de inventario. Tampoco ofrecen la posibilidad de generar un entorno o sitio web de dicha colección. De nuevo, se trata de soluciones que no cubren las necesidades del público objetivo de enlaz.art.

[41] http://www.gcstar.org

[42] https://binaryformations.com

23. Conclusiones

23.1. Conclusiones sobre el trabajo realizado

Las metodologías ágiles de desarrollo, como *SCRUM*, incorporan en cada *sprint* o ciclo de desarrollo una fase de retrospectiva, en la que el equipo reflexiona en común sobre los puntos más fuertes y más débiles de su funcionamiento como equipo. Al margen del producto desarrollado, se trata de buscar las áreas y los comportamientos que maximicen el rendimiento, el funcionamiento del equipo y de cada integrante del equipo. Adicionalmente, al final del proyecto, se realiza una retrospectiva general, con un alcance del proyecto completo.

Todo ello está encaminado al mejor desempeño en futuros proyectos y es, en ese momento, cuando ya no acosa el calendario, ni amenazan las urgencias por entregar material, cuando pueden realizarse aprendizajes reposados, reflexivos, centrados en lo importante y, en definitiva, de mayor valor.

En relación con el desarrollo general del proyecto y de la memoria y el resto del material entregado, destacaría tres cuestiones.

En primer lugar, la importancia de la **planificación**. En el plan docente –también el consultor desde el primer momento– se hace hincapié en elaborarla a nivel global desde el principio. Este ejercicio ayuda a dimensionar el proyecto y la cantidad de trabajo necesaria para culminarlo adecuadamente. No obstante, en perspectiva, creo que debería haber sido más pesimista –o realista– en el momento de su confección, pues he encontrado algún aspecto que, una vez que no pude realizarlo en el plazo previsto, retrasó en cadena alguna otra cuestión.

Otro factor clave es la **comunicación** con el consultor. Si en un entorno real es fundamental que la comunicación con el equipo y los *stakeholders* sea efectiva, en un proyecto académico lo es más, pues se trata de una empresa singular, en la que se va dando vida a una idea propia, habitualmente con el único punto de vista externo del consultor. Su aportación sobre el proyecto, sobre la idea y sobre la memoria y, en suma, la interlocución mantenida entre consultor y estudiante contribuye de manera decisiva al éxito final del proyecto.

Finamente, destacaría la importancia de la **selección de herramientas** utilizadas. La tecnología nos potencia o nos limita con su horizonte de potencialidad y, así, el *mix* de programas, sistemas, aplicaciones y/o soluciones que se empleen para el desarrollo del proyecto, contribuirán en mayor o menor medida a su éxito final.

23.2. Conclusiones sobre la aplicación

En relación con la plataforma enlaz.art desarrollada, hay varias cuestiones que considero relevantes.

Respecto a la **definición de la idea y el alcance**, como decía en el apartado anterior, se aborda un proyecto académico, pero con posibilidad, en este caso, de pasar en un futuro a mercado. Por tanto, a la hora de realizar el desarrollo, conviene realizar el análisis y la implementación con un nivel de abstracción que permita conjugar la

idea propia, con las aportaciones tanto del consultor, como de posibles interlocutores (por ejemplo, expertos en el área abordada) que nos brinden su opinión sobre otros enfoques, puntos de vista o funcionalidades no considerados inicialmente.

También resulta clave la **selección del *stack*** o conjunto de tecnologías sobre las que desarrollaremos la plataforma. Habrá que balancear la experiencia personal, el desarrollo laboral y académico del estudiante, así como el propio ciclo evolutivo del marco tecnológico seleccionado. En este caso, el conjunto de tecnologías seleccionado, totalmente nuevo para mí, me ha permitido realizar aprendizajes sobre nuevos enfoques en el desarrollo de proyectos web.

A partir de la selección del *stack* y de la definición del proyecto, se afrontó el **proceso de aprendizaje** del nuevo producto, Laravel. En este punto, a la hora de comenzar un nuevo desarrollo, incluso desde el punto de vista formal y de diseño, se abordó el aprendizaje con la premisa de priorizar el conocimiento *ancho* frente al conocimiento *profundo*, es decir, traté inicialmente de conocer el máximo de funcionalidades disponibles por Laravel para, posteriormente, profundizar en aquellas que optase por implementar. Un mayor conocimiento del potencial de posibilidades de la tecnología, al que antes hacía mención, tiene impacto *a posteriori* cuando, al implementar la solución, se decide optar por uno u otro enfoque, y redunda habitualmente en una mejor toma de decisiones.

No obstante, en la fase de implementación del proyecto, cuando debemos adoptar decisiones de enfoque, ni el conocimiento ancho, ni el profundo de la solución o producto seleccionados nos garantizarán –sí facilitarán– el acierto. Más allá del dominio de una tecnología concreta se encuentra la **experiencia**, fruto de haber interiorizado, reflexionado, madurado, probado y repetido los diferentes problemas, abstractos y concretos, a los que nos enfrentamos en todos los proyectos tecnológicos. Haciendo retrospectiva, cambiaría alguna decisión adoptada, fruto de la experiencia progresivamente adquirida, pero, en general, estoy muy satisfecho con el planteamiento y producto final realizados.

Por último, una cuestión que he realizado *de facto* y considero muy interesante, es definir desde las fases iniciales del proyecto las líneas generales del **libro de estilo**. Si bien el manual de identidad corporativa definitivo va madurando conforme avanza el proyecto en general, concretar los elementos mínimos (logotipo y paletas tipográficas y de colores) al principio del proyecto redundará en una fase de prototipado y desarrollos más cercanos al producto final, así como servirá de elemento de motivación y refuerzo positivo.

Anexo 1. Entregables del proyecto

La versión final del presente proyecto se compone de los siguientes archivos:

- **Memoria del proyecto**: 01-PEC_FINAL_mem_SimonLazaro_LuisJoaquin.pdf
- **Código fuente del proyecto**: 02-PEC_FINAL_prj_SimonLazaro_LuisJoaquin.zip
- **Prototipos Lo-Fi**: 03-lo-fi.zip
- **Prototipos Hi-Fi**: 04-hi-fi.zip
- **Prototipo Hi-Fi de la etiqueta de obra**: 05-hi-fi-etiqueta.png
- **Capturas de pantalla empleadas en el Anexo 03**: 06-Anexo03-CapturasPantalla.zip
- **Manual de identidad corporativa** (anexo 5): 07-Anexo05-IdentidadCorporativa.pdf
- **Documento de presentación del proyecto:** 08-PEC_FINAL_prs_SimonLazaro_LuisJoaquin.pdf

Anexo 2. Código fuente (extractos)

A continuación, se incorpora una selección de partes relevantes del código fuente de la aplicación. El código fuente completo está disponible en un archivo comprimido anexo, tal como se ha mencionado en el apartado anterior. Por motivos de claridad, se omite en el código fuente las partes de menor interés (se indica con puntos seguidos entre paréntesis), comentando los aspectos clave a juicio del estudiante. También, se ha reordenado el código comentado respecto al implementado, intentando agrupar secciones relacionadas semántica o lógicamente.

Anexo 2.1. Modelo *User*

Extracto del modelo *User* (usuario) implementado, comentado en sus partes más relevantes.

```
(...)
class User extends Authenticatable
{
    use Notifiable;
    /** Los siguientes atributos permiten asignación masiva */
    protected $fillable = ['name', 'email', 'rol_id', 'password',];
    /** Los siguientes atributos se ocultan en el request: por seguridad no serán visibles */
    protected $hidden = ['password', 'remember_token',];
    /** A continuación, se definen las relaciones del modelo User.
    *    con hasMany() definimos relaciones uno a muchos
    *    con belongsTo() definimos relaciones uno a uno. **/
    (...)
    public function exposiciones() {
            return $this->hasMany(Exposicion::class);  }
    public function rol() {
            return $this->belongsTo(Rol::class); }

    /** A continuación, se definen los métodos del modelo User. **/
    public function colecciones_disponibles() {
            /** Navegamos por el rol del usuario para obtener su máximo de colecciones en función del plan **/
            return $this->rol->max_colecciones; }
    public function colecciones_publicas() {
            /** Usando el Query Builder de Laravel, no es necesario usar SQL.
            * Más bien, construimos consultas encadenando condiciones. **/
            return $this->exposiciones->where('esPublica', true)->count(); }
```

```
(...)
        public function ha_etiquetado(Obra $obra, int $etiqueta) {
                /** Otro ejemplo de cómo encadenar condiciones para construir nuestra consulta a base de datos. **/
                return $this->acciones_sociales->where('etiqueta_id', $etiqueta)->where('obra_id', $obra->id)-
>count(); }
(...)
        public function sendPasswordResetNotification($token) {
                /** Para personalizar el email de regeneración de contraseña, se sobreescribe este método, original
                *  de la funcionalidad Auth por defecto de Laravel.
                *  Al comienzo de la definición del modelo, se ha importado el objeto Notification sobreescrito, en la
                *  línea use App\Notifications\ResetPasswordNotification;
                **/
                $this->notify(new ResetPasswordNotification($token)); }
        public function setPasswordAttribute($password) {
                /** Esta función es un mutador de Eloquent -> siempre que escribamos $password en base de datos
                *  se ejecutará esta función. Antes de almacenar el $password, comprobamos si está cifrado o en texto
                *  plano. Si va en texto plano, lo ciframos.
                */
                // Comprobar el password nos llega cifrado para almacenar en base de datos.
                // Usamos una expresión regular para comprobarlo.
                if ( preg_match('/^\$2y\$[0-9]*\$.{50,}$/', $password) ) {
                        // Ya está cifrado -> no volvemos a encriptar
                        $this->attributes['password'] = $password;
                } else {
                        // No está cifrado -> encriptar
                        $this->attributes['password'] = bcrypt($password);
                }
        }
(...)
 }
```

Código fuente 14. Extractos del modelo *User (app/User.php)*

Anexo 2.2. Controlador *ObrasController*

El controlador alberga la lógica relacionada con los métodos. Aquí es donde se define qué ocurre cuando se solicita un listado, una edición, una creación, un borrado, etc.

```
(...)
```

```
class ObrasController extends Controller
{
    function __construct(){
        /** El constructor se ejecuta siempre en primer lugar. Es un buen lugar para aplicar un middleware
        *  que nos ayudará a asegurar el funcionamiento de nuestra aplicación. En este caso, a todas las
        *  funciones del controlador se le requiere acceso autenticado del usuario.    **/
        $this->middleware('auth'); }

    /** En el controlador definimos los verbos/acciones de la implementación REST que necesitemos: index, create,
    *   edit, show, update, store, destroy. También, definimos otros métodos que sean necesarios. */
    public function index() {
        /**  Obtenemos las obras del usuario autenticado y a continuación retornamos la vista obras.index.
        *   A la vista, le pasamos el array de obras del usuario **/
        $obras = auth()->user()->obras;
        return view('obras.index', compact('obras'));
    }

    /** Vista de creación de nueva obra. Necesita el listado de colecciones del usuario */
    public function create() {
        /** Obtenemos las colecciones/exposiciones del usuario, creando un array asociativo de (Titulo,id)
        *   Retornamos la vista obras.create, a la que le pasamos dicho array. */
        $exposiciones = auth()->user()->exposiciones->pluck('Titulo','id');
        return view('obras.create', compact('exposiciones'));
    }

    /** Almacenamos la nueva obra en Base de Datos, con los datos que llegan del formulario. Los datos llegan ya
    *   validados por el objeto Request ValidarNuevaObra. */
    public function store(ValidarNuevaObra $request) {
        /** Creamos una nueva obra y le asignamos a cada campo el valor recibido */
        $obra = Obra::create([
            'Titulo' => $request->get('Titulo'),
            'Autor' => $request->get('Autor'),
            (...)
            'esPublica' => false, /** Por defecto, las obras publicadas no son visibles */
            'obra_token' => str_random(12), /** Generamos una cadena aleatoria única de 12 caracteres */
        ]);
        // Redireccionamos al índice de obras, con un mensaje de feedback para el usuario.
        return redirect()->route('obras.index')->with(['tipo' =>'success','info' =>'Nueva obra insertada']);
```

```
        }

        /** Ficha privada de la obra (con acceso autenticado) */
        public function show($id) {
                $obra = Obra::findOrFail($id);
                /** Además de que el user esté autenticado (por el middleware), comprobamos que el usuario
                    autenticado tenga acceso para realizar la acción en la obra. Esta verificación se hace mediante la
                    política de acceso (policy) ObraPolicy que veremos en otro apartado de la presente sección */
                $this->authorize('show', $obra);
                return view('obras.ficha-publica',compact('obra')); /** Si el user tiene permiso para show, seguimos */
        }
(...)
        /** Eliminación de obra */
        public function destroy($id)
        {
                $obraABorrar = Obra::findOrFail($id);
                $this->authorize('destroy', $obraABorrar); /** Validamos que el user tenga permiso de borrado */
                /** Al borrar la obra, hay que limpiar la imagen de la obra en almacenamiento, si existe */
                $fotoURL = str_replace('storage','public',$obraABorrar->Foto); /** obtenemos la ruta en storage */
                Storage::delete($fotoURL); /** Borramos la imagen de la obra */
                $obraABorrar->delete();
                return redirect()->route('obras.index')
                        ->with(['tipo' =>'success','info' =>'Obra eliminada correctamente']);
        }
        /** Servimos la página de generación de etiqueta. Antes, almacenamos en BBDD que la obra tiene etiqueta */
        public function ver_etiqueta($id) {
                $obra = Obra::findOrFail($id);  /** Localizamos la obra o mostramos un error */
                $obra->tieneEtiqueta = true;
                $obra->update();
                return view('obras.etiqueta',compact('obra')); /** La generación de la etiqueta se hace en la vista */
        }
        /** Gestionamos las subidas de imágenes de obras en servidor. Esta función no retorna ninguna vista,
        *    sino el resultado de la subida */
        public function subir_foto(User $usuario) {
                /** validamos la petición (request()) recibida */
                $this->validate(request(),[
                        'fotoObra' => 'required|image|max:2048|dimensions:min_width=600,min_height=600',
                        /** Validación requerida al campo fotoObra: archivo de tipo imagen, máx. 2048 kB,
```

```
                    *    mínimo 600x600 píxeles */
                ]);
                /** Si esto se ejecuta, ha pasado la validación. Almacenamos en servidor, y devolvemos la ruta del
                *    archivo subido */
                $fotoObra = request()->file('fotoObra')->store('public/enlazart/'.$usuario->id);
                $fotoURL = Storage::url($fotoObra);
                return $fotoURL;
        }
(...)
        /** Publicar la ficha pública de la obra */
        public function publicar_ficha($id) {
                /** Localizamos la obra y validamos que el usuario tenga permiso para publicar la ficha de la obra */
                $obra = Obra::findOrFail($id);
                $this->authorize('publicar_ficha', $obra);
                /** Publicaremos la ficha si la obra tiene imagen */
                if ($obra->Thumb != null) {
                        $obra->esPublica = true;
                        $obra->save();
                        // redireccionar
                        return redirect()->route('obras.index')
                                ->with(['tipo' =>'success','info' =>'Obra publicada correctamente']);
                }
                /** Si no hay imagen, no pubilcamos la ficha e informamos al usuario */
                return redirect()->route('obras.index')
                ->with(['tipo' => 'danger', 'info' => 'Atención: la obra necesita una imagen para poder publicarse' ]);
        }
(...)
}
```

Código fuente 15. Extracto del controlador ObrasController

Anexo 2.3. Request *ActualizarColeccionRequest*

Los objetos Request de Laravel nos facilitan centralizar la lógica de validación de peticiones. En este caso, se muestra un extracto de la validación aplicada a la actualización de colecciones del usuario.

```
(...)
class ActualizarColeccionRequest extends FormRequest
{
        /** En primer lugar, la función authorize nos permitiría incorporar alguna validación para determinar si el
```

```
        *   usuario está autorizado a realizar esta petición. He omitido su implementación, y siempre se autoriza.
        *   En mi caso, he autenticado el usuario mediante middlewares y policies */
        public function authorize()             {
                return true; }

        /** En rules() definimos las reglas de validación que debe cumplir nuestra petición */
        public function rules()
        {
                /** Preparamos valores para las fechas de la colección.
                *   fecha_tope será como máximo la fecha de comienzo más los días definidos por el plan del user */
                $fecha_inicial = Carbon::parse($this->get('FechaDesde'));
                $fecha_tope    = $fecha_inicial->addDays(auth()->user()->dias_plan())->toDateString();

                return [
                        'Titulo' => 'required', /** Titulo es obligatorio */
                        'Autor' => 'nullable|string',
                        'FechaDesde' => 'nullable|date', /** FechaDesde no es obligatorio. Si está tiene que ser date */
                        /** Con las reglas before y after podemos validar los posibles valores de fechas */
                        'FechaHasta' => 'nullable|date|before:' . $fecha_tope . '|after:FechaDesde',
                        'Ubicacion' => 'nullable|string',
                        'Horario_general' => 'nullable|string',
                        'URL' => 'nullable|URL', /** URL, si está, tiene que ser una URL válida */
                        (...)
                ];
        }
}
```

Código fuente 16. Extracto de la clase *Request* ActualizarColeccionRequest

Anexo 2.4. Política de acceso *ObraPolicy*

Mediante las políticas de acceso o *Policies*, Laravel nos permite agrupar y definir cómo funcionará nuestra aplicación, con un control fino sobre los permisos.

```
(...)
class ObraPolicy {
(...)
        /** La primera función nos permite definir condiciones para saltarnos la política de acceso.
        *   En mi caso, si el usuario es administrador, no hacemos más verificaciones y se le autoriza. */
        public function before($user, $ability) {
```

```
            if ($user->esAdministrador()) {
                return true;
            }
    }

    /** La lógica de la Policy es muy sencilla: definimos cada función del controller a la que queremos validar el acceso,
    *   teniendo que coincidir sus nombre en Policy-Controller. En cada función, se incorpora la lógica de control
    *   de acceso. En mi caso, es siempre la misma: que el usuario autenticado sea el organizador/propietario
    *   de dicha obra.
    *   Cuesta ver al principio que, al realizar la llamada en el controller, el método recibe un único parámetro,
    *   el segundo. El primero siempre es el usuario autenticado, y no se especifica en la llamada. */
    public function edit(User $user, Obra $obra) {
            return $user->id === $obra->organizador->id; }

    public function show(User $user, Obra $obra) { (...) }
    public function update(User $user, Obra $obra) { (...) }
    public function destroy(User $user, Obra $obra) { (...) }
    public function publicar_ficha(User $user, Obra $obra) { (...) }
    public function ocultar_ficha(User $user, Obra $obra) { (...) }
}
```

Código fuente 17. Extracto de la clase *Policy* ObraPolicy

Anexo 2.5. Vista ficha-publica.blade.php

La vista **/resources/views/obras/ficha-publica.blade.php** muestra la ficha de la obra, tanto en previsualización (acceso mediante usuario autenticado propietario de la obra o administrador de la aplicación), como en acceso público (acceso mediante usuario sin autenticación). Este ejemplo ilustra como se relacionan el controlador –en este caso, **PagesController**, al que no se le aplica el *middleware* **auth**– y la vista, así como el uso de *Blade*, el lenguaje de plantillas de Laravel.

En primer lugar, tenemos 2 formas de servir la vista: con o sin usuario autenticado. En el primer caso, la vista se sirve desde el controlador *ObrasController*, método **show**. En el segundo caso, la vista se servirá desde el controlador *PagesController*:

```
(...)
public function ficha_publica($obra_token){
    /** Acceso público a la ficha de la obra, mediante obra_token (identificador único y aleatorio) */
    $obra = Obra::where('obra_token',$obra_token)->firstOrFail();
    /** Comprobamos que la ficha pública está visible (publicada) por el usuario. Si no, lanzamos un error.  */
```

```
        if ( ! $obra->esPublica){
                abort(500, 'La obra no se encuentra publicada.');  }
        return view('obras.ficha-publica', compact('obra')); /** Si se llega aquí, se sirve la ficha pública */
                        // En la vista, tendremos a nuestra disposición la variable $obra.
}
(...)
```

Código fuente 18. Extracto del controlador PagesController

En la vista, personalizaremos la información mostrada en función de si se sirve como ficha pública o no.

```
@extends('layouts/layout')
        // La directiva de Blade @extends nos permite extender una plantilla: layouts/layout tiene el esqueleto de la web

@section('contenido')
        // Con la directiva @section, completamos una sección definida en la plantilla extendida
(...)
        // Con el marcado {{ ... }} personalizamos la vista, pudiendo acceder a variables como $obra o usar PHP
        <h1 class="display-5">{{ $obra->Titulo }} <small class="text-muted ml-1">de {{ $obra->Autor }}</small></h1>
        // Blade nos provee de multitud de directivas, por ejemplo, @if para implementar estructuras condicionales
                @if (request()->is('obras*'))
                        <p class="badge badge-primary mt-4">Página de previsualización</p>
                        <small>Esta página es privada. (...)</small>
                @endif
(...)
        <figure class="figure">
                // Si la obra no tiene imagen, se ha especificado una variable de entorno en la aplicación para mostrar
                // una imagen por defecto.
                <img (...) src="{{ ($obra->Thumb != null) ? $obra->Thumb : env('ENLAZART_THUMBNAIL') }}" >
                (...)

                // La directiva @guest nos permite implementar de manera muy sencilla un condicional para comprobar
                // si el usuario es invitado (guest) o ha iniciado sesión
                @guest
                        <div class="btn-group" role="group" aria-label="Marcar la obra como Favorito o Me gusta">
                        (...) // Aquí van los botones deshabilitados (para invitados) de Favorito/Me gusta
                        <small><a href="{{ route('login') }}">Inicia sesión</a> para poder interactuar</small>
                @else // si se ejecuta esta parte, el usuario tiene sesión iniciada
                        <div class="btn-group" (...)>
                                // Los botones Me gusta o Favorito implementan su funcionalidad en javascript,
```

```
                    // más abajo en el código, en el bloque scripts
                    <button id="2" type="button" class="etiquetar (...)">
                        <span class="oi oi-thumb-up"></span> Me gusta
                        // Indicamos el total de etiquetas fav o like de la obra
                        <span class="badge badge-secondary">
                            {{ $obra->num_acciones_sociales(2) }}
                        </span>
                    </button>
                    (...) // El botón de "Favorito" es similar
                </div>
                // Preparamos una zona para el feedback al usuario al usar los botones
                <div id="feedback_etiqueta" class='text-info'> <small></small></div>
            @endguest
        </figure>
(...)
// Para las tarjetas informativas de cada obra, se ha usado el componente card, de Bootstrap
// en combinación con los componentes (directiva @component de Blade).
        <div class="card">
            <div class="card-header">Obra</div>
            <div class="card-body"> (...) </div>
            // Un componente de Blade es como un plantilla de código a la que le podemos pasar valores
            // Este componente está accesible en resources/views/components/obra_card.blade.php
            @component('components.obra_card',
                ['Tecnica' => $obra->Tecnica, 'Fecha' => $obra->Fecha, (...)])
            @endcomponent
        </div>
(...)
@endsection
// Con la directiva @push podemos añadir contenido a nuestra plantilla extendida,
// sin sobreescribir lo existente en dicha sección en la plantilla
@push('scripts')
<script>
    // Implementamos la funcionalidad de los botones de Favorito/Me gusta.
    // Accedemos a los botones y eventos con jQuery y gestionamos las llamadas AJAX con Axios.
    $('.etiquetar').on('click', function()  {
        axios({ // configuramos la llamada AJAX
            method: 'post', // método de la llamada
            url: "/etiquetar/{{ $obra->id }}", // destino de la llamada
```

```
                    data: { // valores que enviamos en la llamada
                            etiqueta_id: $(this).attr("id"),
                            obra_id: '{{ $obra->id }}',
                    },
                    headers: { // por seguridad, hay que enviar el token CSRF en una cabecera
                            'X-CSRF-TOKEN': '{{ csrf_token() }}'
                    }
            }).then( res => { // gestión de la respuesta a la llamada AJAX: mostramos feedback con mostrar_mensaje()
                    mostrar_mensaje('#feedback_etiqueta small', res.data);
            });
    });

    function mostrar_mensaje($selector, $mensaje) {
            // Mostraremos un texto breve informativo al usuario, durante un breve lapso
            // Primero, recibimos la respuesta AJAX y la ponemos en un array
            $id = $mensaje.split("_"); // $id : [on|off, etiqueta_id, total_etiquetas, literal]
            // Actualizamos el aspecto del botón pulsado
            $boton = $('button.etiquetar[id='+$id[1]+']');
            if ($id[0] === 'on') { $boton.addClass('btn-primary').removeClass('btn-outline-primary');
            } else { $boton.addClass('btn-outline-primary').removeClass('btn-primary'); }
            // Actualizamos el contador de etiquetas (fav o like) aplicadas a la obra
            $boton.find('.badge').text($id[2]);
            // Creamos un span temporal para poner el mensaje informativo
            $($selector).append("<span class="+$id[1]+">"+$id[3]+" </span>");
            // Localizamos el span, le hacemos un fadeIn para mostrarlo, esperamos 1 segundo
            // aproximadamente y lo ocultamos con fadeOut
            $($selector).find('span.'+$id[1]).fadeIn( 200 ).delay( 1200 ).fadeOut( 400, function(){
                    // el segundo parámetro del fadeOut nos permite incorporar un callback que se
                    // ejecutará al finalizar la transición. Lo usamos para eliminar el span temporal creado.
                    $(this).remove();
            });
    }
</script>
@endpush
```

Código fuente 19. Extracto de la vista *resources/views/obras/ficha-publica.blade.php*

Anexo 3. Librerías externas utilizadas

Anexo 3. 1. *Backend*

- **Laravel**, versión 5.6. *Framework* MVC desarrollado en PHP. A partir de este, se ha desarrollado la plataforma, empleando buena parte de las funcionalidades de serie que incorpora: *middlewares*, *policies*[43], *requests*, funcionalidad de autenticación por defecto…
- **Simple QrCode**, versión 2.0. Librería para la generación de códigos QR, de acuerdo con nuestras necesidades.
- **Intervention Image**, versión 2.4. Gestión y manipulación de imágenes. Mediante esta librería, se pueden crear y/o editar imágenes mediante PHP, de manera sencilla y rápida. Puede configurarse para usar 2 controladores: GD, incluido de serie en PHP, o ImageMagick, una extensión de PHP no incluida por defecto. Para poder personalizar las etiquetas, por ejemplo, especificando el tipo de letra deseado, ha sido necesario instalar y configurar en el servidor esta segunda opción.
- **Imagick**, versión 3.4. Gestión y manipulación de imágenes en PHP. Es una interfaz para usar el controlador ImageMagick.

Anexo 3. 2. *Frontend*

- **Bootstrap**, versión 4. Framework CSS, nos ofrece una base de estilos y componentes sobre la que empezar a construir rápidamente nuestro sitio web.
- **jQuery**, versión 3.2+. Librería javascript que incrementa las opciones del lenguaje y nos permite usar una sintaxis más cómoda y resumida.
- **Axios**, versión 0.18. Librería javascript que nos permite realizar y escuchar llamadas AJAX de manera elegante.
- **Popper.js**, versión 1.12. Librería javascript requerida por Bootstrap para gestionar las *tooltips*, es decir, las etiquetas mostradas en la interfaz al pasar el puntero del ratón sobre el elemento en cuestión.
- **SVG Injector**, versión 1.1. Librería javascript para la inyección de SVG en la interfaz mediante una sintaxis elegante. Es una opción de uso del conjunto de iconos Open Iconic empleado en la interfaz.

[43] Mecanismo de Laravel para la gestión de accesos.

- **Dropzone**, versión 5.4. Librería javascript que nos facilita la gestión de subidas de archivos al servidor. Permite, de manera sencilla y agradable visualmente, arrastrar y soltar archivos al área del control.
- **CKEditor**, versión 4.8. Librería javascript que nos permite convertir un área de texto de un formulario en un área de texto con editor de texto enriquecido. Así, resulta más sencillo para el usuario poder editar texto y darle un formato básico: negrita, cursiva, listas numeradas y no numeradas, etc.

Anexo 4. Capturas de pantalla

Se incluyen a continuación algunas capturas de pantalla del ordenador del estudiante, con el objetivo de ilustrar el proceso de trabajo desarrollado.

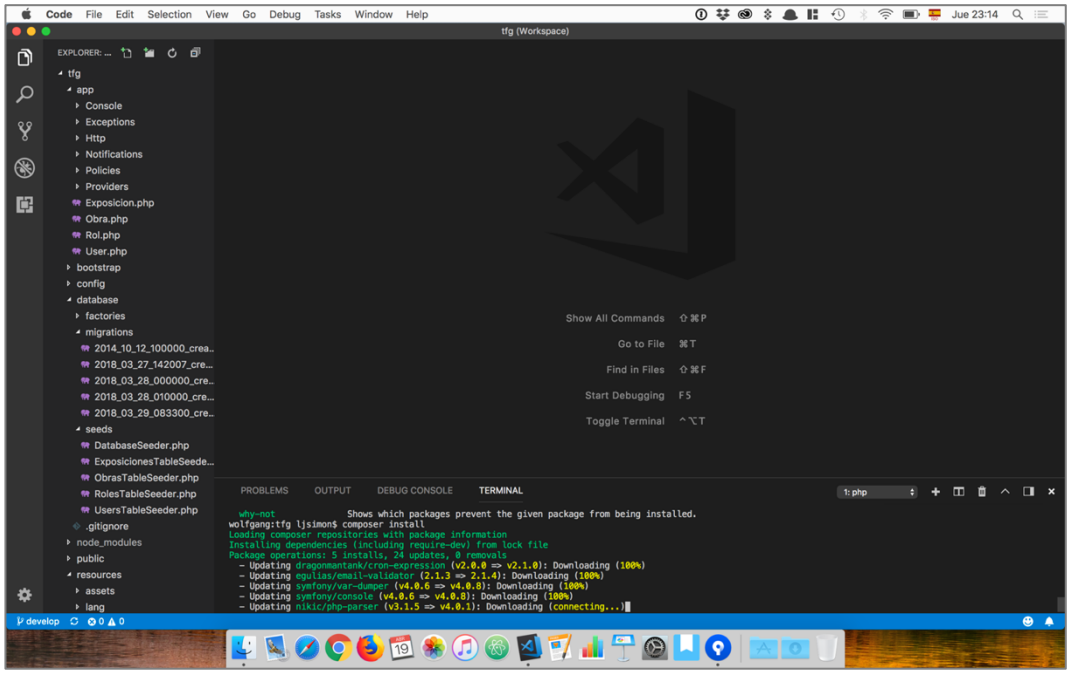

Figura 31. Captura de pantalla: instalando librerías con Composer en VisualStudio Code

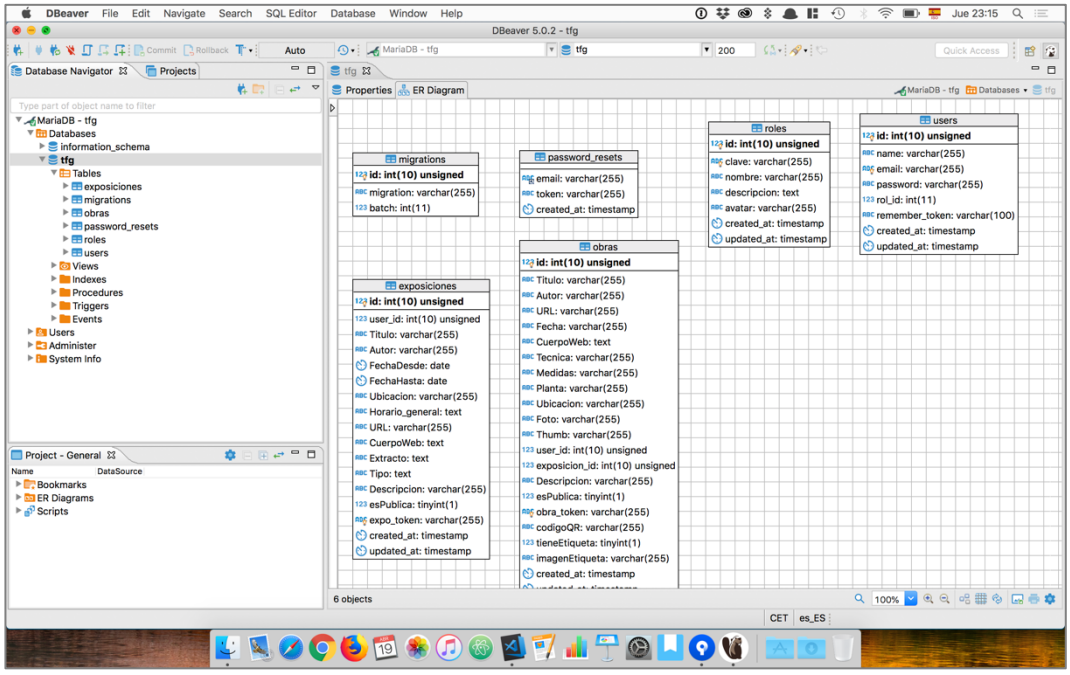

Figura 32. Captura de pantalla: primeras etapas de trabajo con la base de datos, con DBeaver

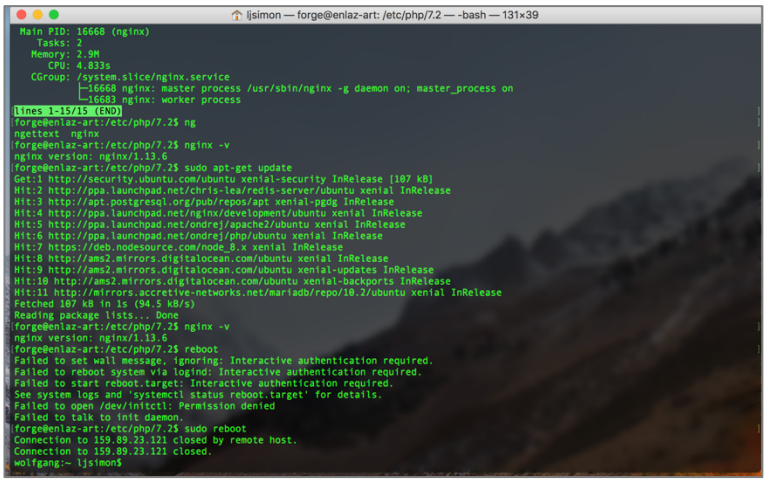

Figura 33. Captura de pantalla: actualizando mediante Terminal el VPS en producción

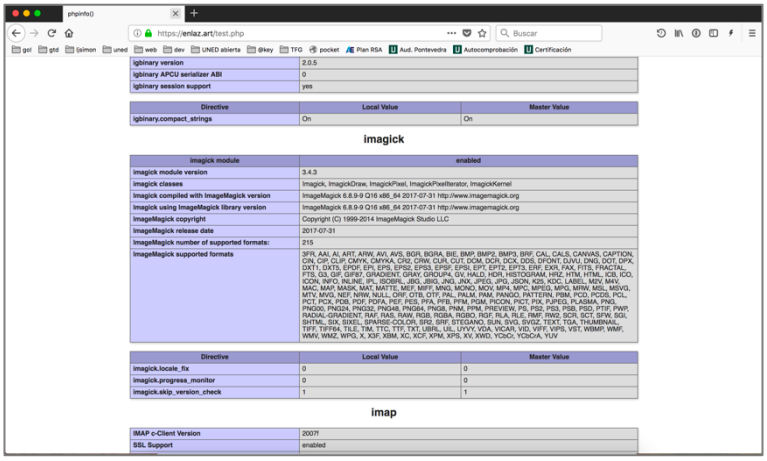

Figura 34. Captura de pantalla: comprobando la instalación de Imagick en el VPS de producción

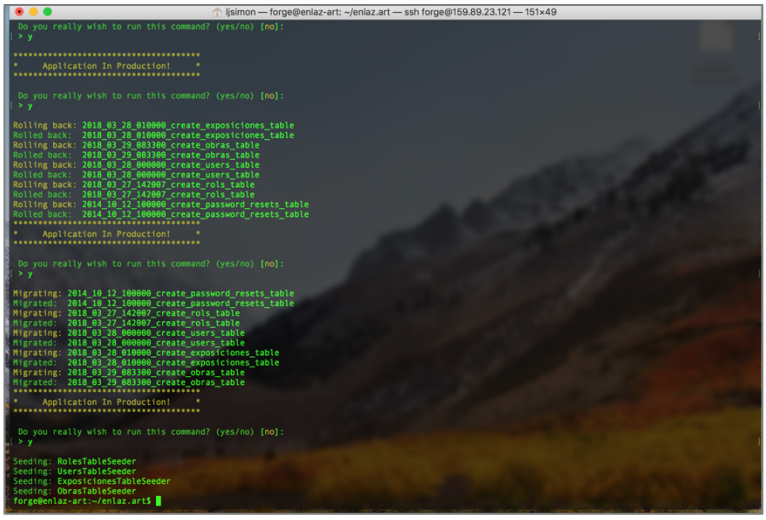

Figura 35. Captura de pantalla: ejecutando migraciones con *seeding* (datos de prueba) en producción

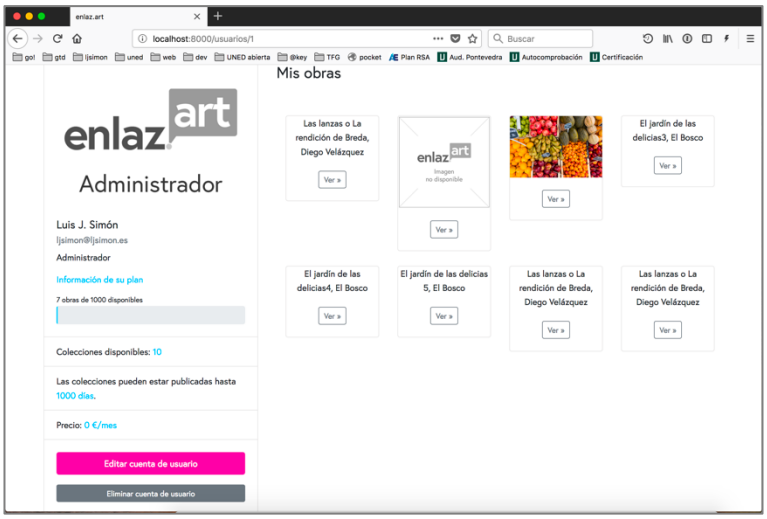

Figura 36. Captura de pantalla: trabajando en la ficha de usuario, entorno local de desarrollo

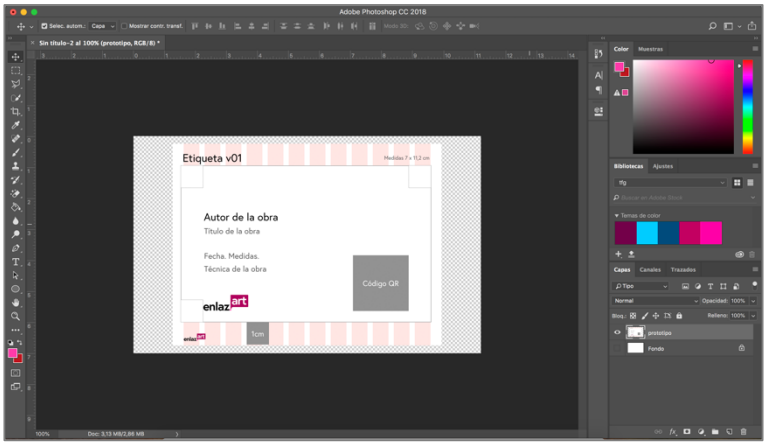

Figura 37. Captura de pantalla: fase inicial del prototipo de baja fidelidad de la etiqueta de obra, en Adobe Photoshop

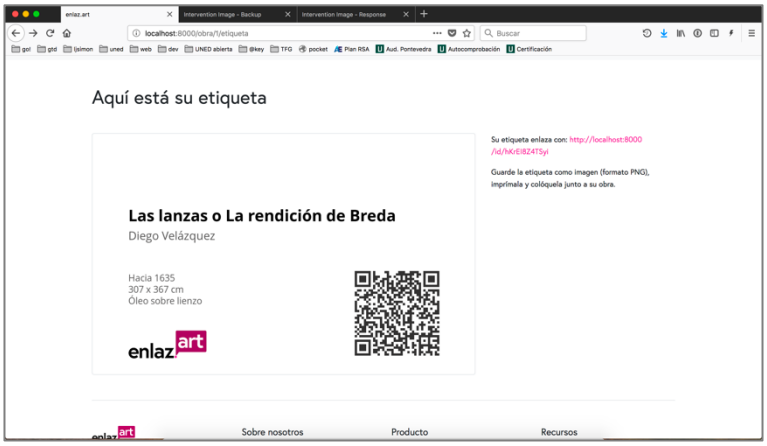

Figura 38. Captura de pantalla: entorno de desarrollo, implementado la etiqueta de obra

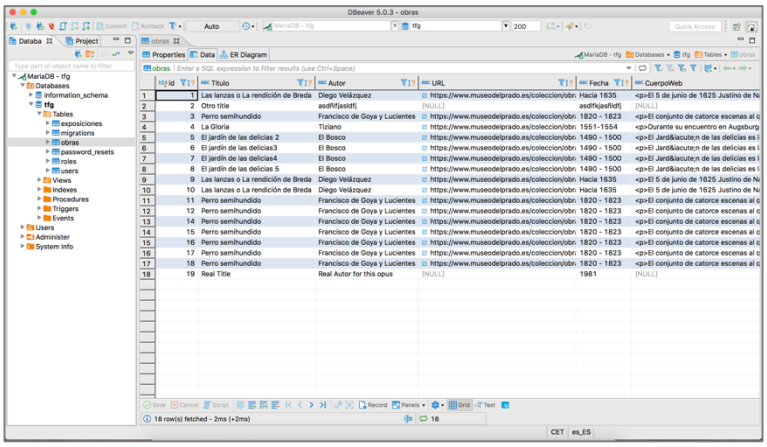

Figura 39. Captura de pantalla: trabajando en el entorno de desarrollo, con DBeaver, con datos de prueba

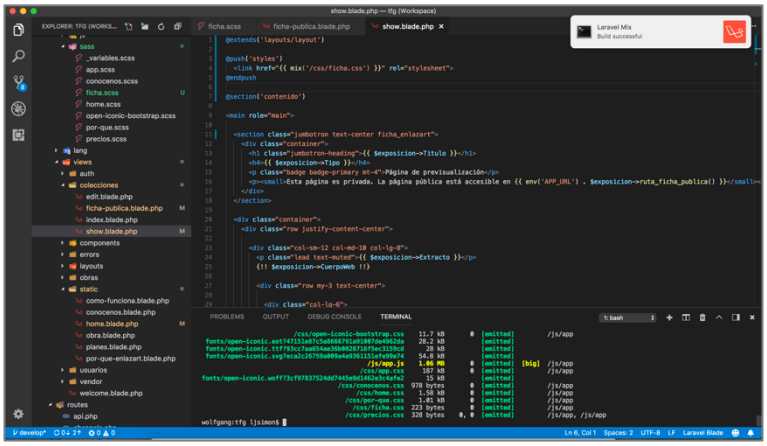

Figura 40. Captura de pantalla: compilando activos en VisualStudio Code, con Laravel Mix y el Terminal integrado

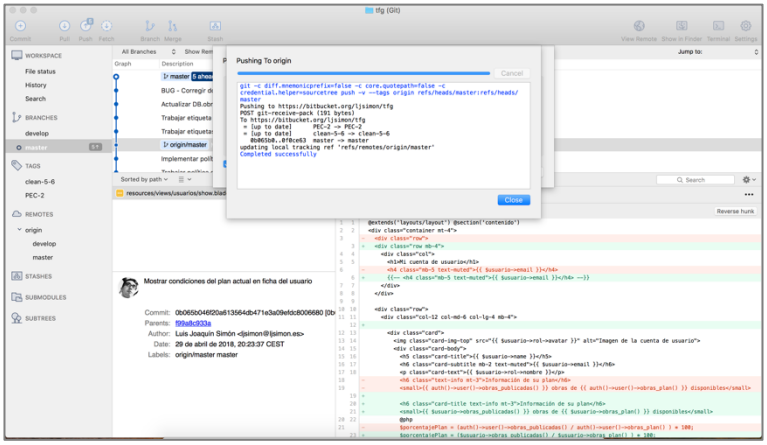

Figura 41. Captura de pantalla: *deploying* en producción desde Sourcetree, vía Bitbucket

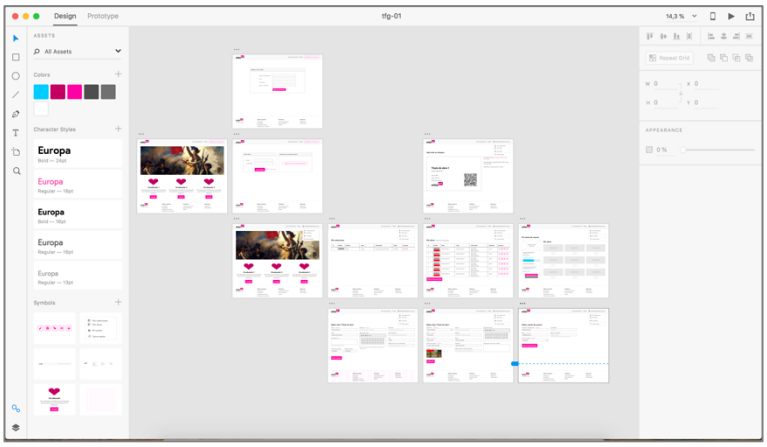

Figura 42. Captura de pantalla: elaboración de prototipos Hi-Fi, con Adobe XD

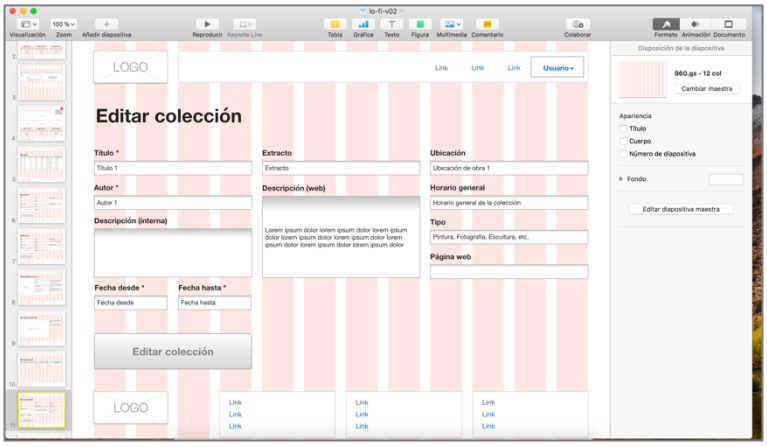

Figura 43. Captura de pantalla: elaboración de prototipos Lo-Fi, con Apple Keynote

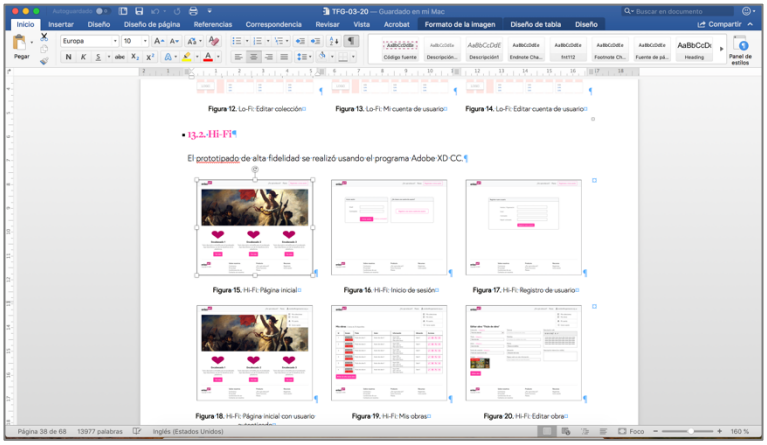

Figura 44. Captura de pantalla: integrando imágenes de los prototipos en la memoria, Microsoft Word

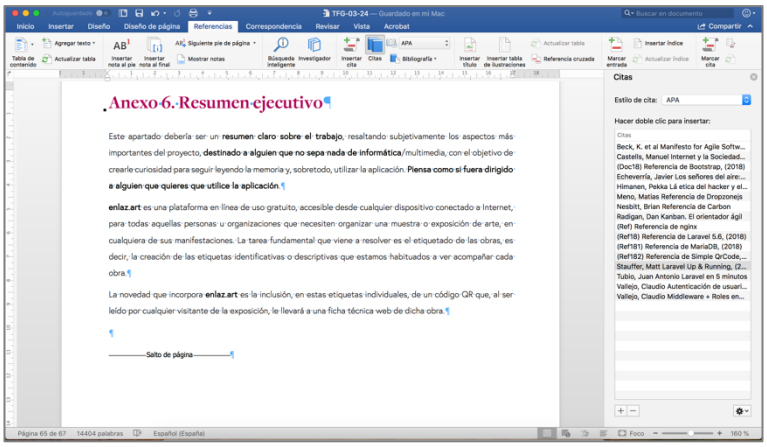

Figura 45. Captura de pantalla: trabajando en una versión avanzada de la memoria, Microsoft Word

Anexo 5. Libro de estilo

El manual de identidad corporativo completo se adjunta como anexo, según lo especificado en el anexo 1 (entregables del proyecto). A continuación, se ofrece una muestra de los elementos más importantes desarrollados en dicho manual.

Figura 46. Identidad corporativa: logotipo y variantes.

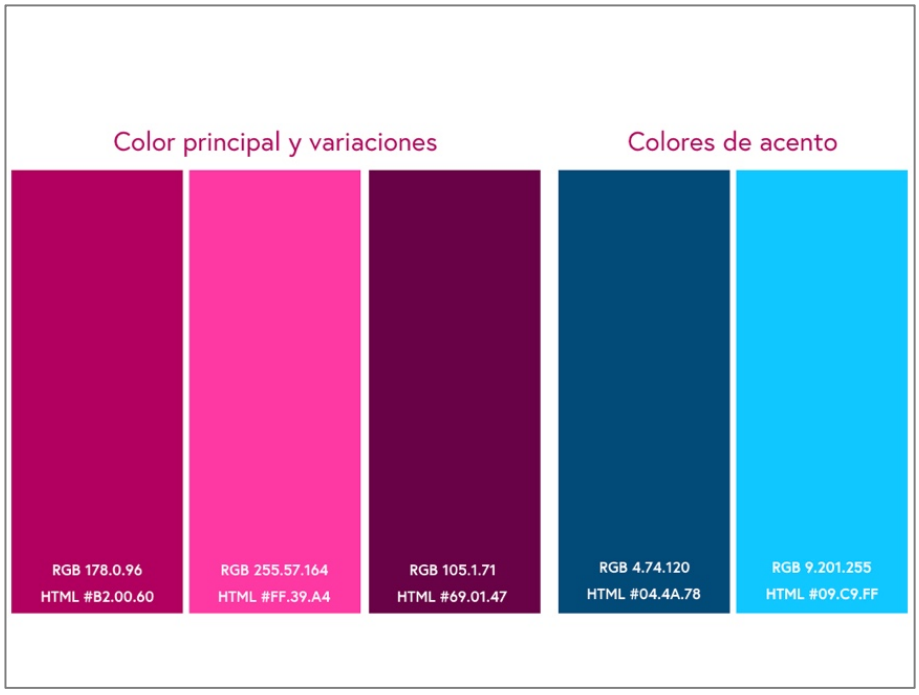

Figura 47. Identidad corporativa: paleta de colores

Figura 48. Identidad corporativa: tipografías

Anexo 6. Resumen ejecutivo

enlaz.art es una plataforma en línea de uso gratuito, accesible desde cualquier dispositivo conectado a Internet, para todas aquellas personas u organizaciones que necesiten organizar una muestra o exposición de arte, en cualquiera de sus manifestaciones. La tarea fundamental que viene a resolver es el etiquetado de las obras, es decir, la creación de las etiquetas identificativas o descriptivas que estamos habituados a ver acompañando a cada obra.

La novedad que incorpora **enlaz.art** es la inclusión, en estas etiquetas individuales, de un código QR que, al ser leído por cualquier visitante de la exposición, le llevará a una ficha técnica web de dicha obra. Esta ficha web se crea automáticamente con la información introducida por el usuario en el momento de creación de la etiqueta.

Otra ventaja de **enlaz.art** es la posibilidad de crear y publicar, de manera muy sencilla, una página web de la exposición. Introduciendo la información deseada, dispondremos de una página web que, además, nos mostrará una galería de todas las obras que la componen, pudiendo navegar y visualizar cada una de ellas.

Por último, la plataforma permite a cualquier usuario marcar sus obras como *Favorita* o *Me gusta*, facilitando así la creación de una colección personal de obras de arte.

Anexo 7. Bibliografía

Afmeti, A. (2016). *Optimize images using Intervention in Laravel*. Obtenido de albanafmeti.com: http://albanafmeti.com/2016/optimize-images-using-intervention-in-laravel/

Beck, K. et al. (2001). *Manifesto for Agile Software Development*. Obtenido de http://agilemanifesto.org

Castells, M. (2000). Internet y la Sociedad Red. *Conferencia de Presentación del Programa de Doctorado sobre la Sociedad de la Información y el Conocimiento*.

Echeverría, J. (1999). *Los señores del aire: Telépolis y el Tercer Entorno*. Ediciones Destino.

Himanen, P. (2004). *Lá etica del hacker y el espíritu de la era de la infomración*. Ediciones Destino.

Meno, M. (s.f.). *Referencia de Dropzone*. Obtenido de dropzonejs.com: http://dropzonejs.com

Nesbitt, B. (s.f.). *Referencia de Carbon*. Obtenido de http://carbon.nesbot.com/docs/

Radigan, D. (s.f.). *Kanban. El orientador ágil*. Obtenido de atlassian.com: https://es.atlassian.com/agile/kanban

Referencia de Bootstrap. (2018). Obtenido de getbootstrap.com: http://getbootstrap.com/docs/4.0/getting-started/introduction/

Referencia de Laravel 5.6. (2018). Obtenido de laravel.com: http://laravel.com/docs/5.6

Referencia de MariaDB. (2018). Obtenido de mariadb.org: http://mariadb.org/learn/

Referencia de nginx. (s.f.). Obtenido de nginx.org: http://nginx.org/en/docs/

Referencia de Simple QrCode. (2018). Obtenido de simplesoftware.io: http://simplesoftware.io/docs/simple-qrcode

Stauffer, M. (2017). *Laravel Up & Running*. O'Reilly Media.

Tubio, J. A. (s.f.). *Laravel en 5 minutos*. Obtenido de github.com/jatubio/5minutos_laravel

Vallejo, C. (10 de Septiembre de 2017). *Autenticación de usuarios y roles en Laravel 5.5*. Obtenido de medium.com: https://medium.com/@cvallejo/autenticaci%C3%B3n-de-usuarios-y-roles-en-laravel-5-5-97ab59552d91

Vallejo, C. (12 de Febrero de 2018). *Middleware + Roles en Laravel 5.6*. Obtenido de medium.com: https://medium.com/@cvallejo/middleware-roles-en-laravel-5-6-87541406426f

Anexo 8. Vita

Estudié Ingeniería Técnica Informática hasta que encontré el **Grado de Multimedia** de la UOC. Me apunté expectante de ver cómo encajaba el *mix* de áreas que se abordaban, lo he cursado con interés y finalizo agradecido por el enfoque multidisciplinar y práctico de los estudios.

Con más de 10 años de experiencia como responsable de Calidad e Identidad Digital, creo firmemente en la formación continua a lo largo de la vida, más cuanto más observo el contexto actual.

Me gusta aprender, la lectura, la ópera, la música clásica, practicar carrera por montaña, jugar con mis hijos y aprender con ellos.