Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Diseño e implementación de una aplicación web para la gestión de ficheros de impresión 3D de repuestos de la Armada

Grado en Ingeniería Mecánica

ALUMNO: Javier Hidalgo Sánchez

DIRECTORES: Belén Barragáns Martínez

Pablo Sendín Raña

CURSO ACADÉMICO: 2020-2021

Universidade Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Diseño e implementación de una aplicación web para la gestión de ficheros de impresión 3D de repuestos de la Armada

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

Universidade Vigo

RESUMEN

La transformación hacia la Armada 4.0 requiere la digitalización y un uso intensivo de las nuevas tecnologías. Una prueba de esta transformación es la incorporación de la fabricación aditiva en la construcción de las nuevas unidades de la Armada. Varias unidades contarán con impresoras 3D para agilizar la obtención de repuestos y mejorar la capacidad de abastecimiento.

Se hace visible la necesidad de compartir ficheros de impresión 3D de repuestos entre diferentes unidades. En consecuencia, este TFG propone el diseño y desarrollo de una aplicación web que gestione un repositorio de ficheros de impresión 3D. Para diseñar la aplicación se ha realizado una revisión de los repositorios online existentes y se ha estudiado el sistema de aprovisionamiento de la Armada.

La aplicación desarrollada permite compartir, descargar y gestionar los ficheros online. De esta forma, si un usuario necesitase un diseño para imprimir un repuesto, puede hacerlo, independientemente de si se encuentra desplegado o en su Arsenal de origen, permitiendo ahorrar tiempo y esfuerzo.

Todas las funcionalidades de la aplicación han sido validadas con éxito, sirviendo dicha prueba como manual de la aplicación.

PALABRAS CLAVE

Repuesto, Impresión 3D, Laravel, MVC, Aplicación web.

AGRADECIMIENTOS

Quisiera dirigir en primer lugar estas palabras de profundo agradecimiento a mi padre, a mi madre y a mi hermana, no solo por el apoyo que he recibido durante el desarrollo de este TFG, sino por haber estado siempre presentes en los mejores y peores momentos con su incesable apoyo, buenos consejos, continuos ánimos y por haber creído en mí a lo largo de estos cinco años de escuela.

A mis tutores, Doña Belén Barragáns Martínez y Don Pablo Sendín Raña, por haber aceptado y creído en esta idea desde el primer momento, por su dedicación, su constancia y su paciencia, gracias.

Al CF Touza Gil, por interesarse en la idea y ponerme en contacto con personal de Navantia para obtener información.

Por último pero no menos importante, a mis amigos y compañeros de promoción, por los ánimos recibidos y los buenos momentos.

CONTENIDO

Contenido 1
Índice de Figuras 3
Índice de Tablas 5
Índice de fragmentos de código 7
1 Introducción y objetivos 9
1.1 Motivación: Impresión 3D en la Armada 9
1.2 Objetivos del TFG 10
1.3 Organización de la memoria 10
2 Estado del arte 11
2.1 La Armada 4.0 11
2.2 Gestión de los repuestos en la Armada 12
2.2.1 Subdirección de aprovisionamiento y transporte SUBDAT 12
2.2.2 Proceso de asignación de un repuesto 14
2.2.3 Catalogación en las FAS 14
2.2.4 Fundamentos tecnológicos del apoyo logístico 4.0 15
2.2.5 Nuevo modelo de apoyo logístico 16
2.2.6 Planeamiento y gestión del sostenimiento 16
2.3 Impresión 3D 17
2.3.1 Funcionamiento de las impresoras 3D 18
2.3.2 Aplicaciones 19
2.3.3 Efectos de la impresión 3D en la cadena de suministros 19
2.4 Repositorios online 21
2.4.1 Introducción a los stocks 3D online 21
2.4.2 AMFG 21
2.4.3 GrabCAD 21
2.4.4 Thingverse 23
2.5 Herramientas software usadas en el TFG 23
2.5.1 Lenguajes de programación 24
2.5.2 Lenguaje de marcado 25
2.5.3 Framework de desarrollo 25
2.5.4 Laravel 25
2.5.5 MySQL 27
2.5.6 Composer 28

3 Desarrollo del TFG 29
3.1 Instalación y configuración del entorno de trabajo 29
3.1.1 Servidor Linux 29
3.1.2 Instalación de Composer y Laravel 29
3.2 Nuevo proyecto Laravel en remoto 30
3.3 Diseño preliminar de la página web 32
3.4 Registro y autenticación de usuarios 33
3.5 Gestión de piezas 36
3.5.1 Diseño de la tabla de datos 36
3.5.2 Controlador para la base de datos 37
3.5.3 Nuevo registro de una pieza 38
3.5.4 Búsqueda de piezas 40
3.5.5 Previsualización de una pieza y descarga 42
3.6 Perfil del usuario 43
3.6.1 Editar una pieza 44
3.6.2 Eliminar una pieza 45
3.7 Apariencia de la aplicación web 46
3.7.1 Menú 46
3.7.2 Vistas 47
4 Validación y prueba 49
4.1 Comprobaciones de funcionamiento de la web 49
4.1.1 Autenticación 49
4.1.2 Navegación 52
4.1.3 Búsqueda y filtrado de piezas 53
4.1.4 Previsualización y descarga de piezas 55
4.1.5 Subir una pieza 56
4.1.6 Gestión de las piezas desde el perfil 59
4.2 Comprobación de funcionamiento de la base de datos 62
5 Conclusiones y líneas futuras 65
5.1 Conclusiones 65
5.2 Líneas futuras 65
6 Bibliografía 67
Anexo I: Archivos php 73
Anexo II: Vistas 81

ÍNDICE DE FIGURAS

Figura 2-1 Estructura de la Jefatura de Apoyo Logístico (elaboración propia) 13
Figura 2-2 Estructura de la DISOS (elaboración propia) 13
Figura 2-3 Problemática del aprovisionamiento a nivel internacional (elaboración propia) 15
Figura 2-4 Estructura del NOC (elaboración propia) 15
Figura 2-5 Esquema de desarrollo PALI (elaboración propia) 17
Figura 2-6 Proceso de impresión 3D [20] 18
Figura 2-7 Suministro de repuestos mediante impresión 3D [20] 20
Figura 2-8 Flujo de información en GrabCAD [33] 22
Figura 2-9 Página principal de GrabCAD COMMUNITY [34] 22
Figura 2-10 Entrada de un diseño en GrabCAD COMMUNITY [34] 23
Figura 2-11 Lenguajes de programación, marcado y frameworks [36] 24
Figura 2-12 Flujo de información en el patrón Modelo-Vista-Controlador [52] 26
Figura 2-13 Diferencias entre paquetes de autenticación de Laravel [53] 27
Figura 2-14 Logo de Composer [56] 28
Figura 3-1 Comprobación de versiones de Apache2, PHP y MySQL (elaboración propia) 29
Figura 3-2 Descarga e instalación de paquetes Laravel (elaboración propia) 30
Figura 3-3 Operaciones para sincronizar el proyecto (elaboración propia) 31
Figura 3-4 Nuevo proyecto sincronizado con el servidor mediante ssh (elaboración propia) 32
Figura 3-5 Flujo de navegación entre diferentes funcionalidades de la aplicación web (elaboración propia) 33
Figura 3-6 Descarga de Breeze (elaboración propia) 34
Figura 3-7 Instalación necesaria de *npm* (elaboración propia) 35
Figura 3-8 Finaliza la instalación de Breeze con npm (elaboración propia) 35
Figura 3-9 Modificación de *.env* y migración de la base de datos (elaboración propia) 36
Figura 3-10 Esquema del empleo de Blade 47
Figura 4-1 Página de inicio (elaboración propia) 49
Figura 4-2 Registro de usuarios (elaboración propia) 50
Figura 4-3 Inicio de sesión por un usuario ya registrado (elaboración propia) 50
Figura 4-4 Recuperación de contraseña (elaboración propia) 51
Figura 4-5 Email de solicitud de cambio de contraseña (elaboración propia) 51
Figura 4-6 Actualización de contraseña (elaboración propia) 52
Figura 4-7 Página principal (elaboración propia) 53
Figura 4-8 Búsqueda de piezas (elaboración propia) 54

Figura 4-9 Búsqueda filtrada (elaboración propia) 54
Figura 4-10 Vista previa de una pieza (elaboración propia) 55
Figura 4-11 Previsualización del modelo descargado de la aplicación con Print 3D (elaboración propia) 56
Figura 4-12 Formulario para subir una pieza (elaboración propia).....56
Figura 4-13 Prueba con fichero de extensión incorrecta (elaboración propia) 57
Figura 4-14 Alerta de extensión no autorizada (elaboración propia).....57
Figura 4-15 Nuevo registro de una pieza (elaboración propia).....58
Figura 4-16 Pieza registrada (elaboración propia) 58
Figura 4-17 Perfil del usuario (elaboración propia).....59
Figura 4-18 Edición de una pieza (elaboración propia) 60
Figura 4-19 Soporte original (elaboración propia).....60
Figura 4-20 Soporte tras ser editado (elaboración propia).....61
Figura 4-21 Comprobación de eliminación de una pieza (elaboración propia) 61
Figura 4-22 Perfil del usuario con Pieza 4 eliminada (elaboración propia).....62
Figura 4-23 Tabla *users* (elaboración propia).....62
Figura 4-24 Tabla que gestiona las piezas (elaboración propia).....63

ÍNDICE DE TABLAS

Tabla 3-1 Diseño de la tabla *Pieza* 36
Tabla 3-2 Acciones manejadas por *FormularioController* 38

ÍNDICE DE FRAGMENTOS DE CÓDIGO

Fragmento de código 3-1 Definición de la tabla *piezas* 37
Fragmento de código 3-2 Definición del modelo 37
Fragmento de código 3-3 Función empleada para almacenar información en la base de datos 39
Fragmento de código 3-4 Función empleada por Laravel para cargar un archivo 40
Fragmento de código 3-5 Bucle *foreach* para mostrar registros almacenados 41
Fragmento de código 3-6 Función para realizar búsquedas en la base de datos 41
Fragmento de código 3-7 Envio de una pieza mediante un formulario 42
Fragmento de código 3-8 Recuperación de la información de una pieza en la base de datos 43
Fragmento de código 3-9 Método *index* del controlador de datos del usuario 43
Fragmento de código 3-10 Función *edit* del controlador *PiezaController* 44
Fragmento de código 3-11 Función definida para actualizar la información de una pieza 44
Fragmento de código 3-12 Funciones definidas para eliminar una pieza 46

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Motivación: Impresión 3D en la Armada

La transformación hacia la Armada 4.0 requiere de la digitalización y el uso intensivo de las nuevas tecnologías. La empresa pública de construcción naval Navantia, desarrolladora de las nuevas fragatas tipo F-110 y de los submarinos S-80 de la Armada, también apuesta por la implementación del concepto de Industria 4.0. Uno de los ámbitos en los que se puede ver la sólida apuesta de Navantia y la Armada por la Industria 4.0 es la fabricación aditiva.

La fabricación aditiva, unida al proceso de escaneado tridimensional, tiene un gran potencial en la reproducción de repuestos industriales. Las nuevas F-110 serán los primeros buques de la flota con una capacidad embarcada que le permita producir sus propios repuestos para equipos y sistemas. Además, cabe destacar que las fragatas han sido diseñadas con materiales de última generación y con elementos fabricados mediante impresión 3D [1], por lo que implementar este tipo de tecnología en las fragatas proporcionará una gran capacidad de autoabastecimiento en la mar.

El tipo de piezas que podrán fabricarse, aplicadas al ámbito naval y en concreto a la Armada, serán aquellas que se estimen como no críticas o esenciales para el funcionamiento del buque. Además, la gran mayoría de las piezas diseñadas por fabricación aditiva serán destinadas a sustituir piezas fabricadas en materiales poliméricos [2] con lo que sus características técnicas serán similares. Pero donde esta tecnología embarcada realmente alcanzará su mayor potencial será en la fabricación de repuestos que han quedado obsoletos, repuestos que ya no se fabrican y no se pueden adquirir por los métodos tradicionales. Por tanto, el uso de tecnologías de fabricación aditiva tiene sentido para dar solución a aquellos problemas donde dichos métodos tradicionales no llegan.

Otra de las ventajas que aporta la introducción de impresoras 3D en las unidades de la Armada es su alta disponibilidad. Estas impresoras pueden trabajar de forma autónoma en cualquier momento y sin supervisión, tanto en tierra como embarcadas, ya sea navegando o en puerto. Distintos estudios demuestran la capacidad de las impresoras 3D que funcionan mediante modelado por deposición fundida para fabricar piezas bajo condiciones de vibración similares a las que sufrirían en un buque durante la navegación.

En cuanto a la viabilidad económica, es conveniente realizar un estudio en cada caso para ver si es provechoso su empleo. Este estudio debe tener en cuenta no solo el coste de la pieza frente al coste del material de producción sino que debe considerar, además, el tiempo de espera hasta que se obtiene el repuesto, pues tener la capacidad de producir una pieza en cuestión de horas o minutos reduciría considerablemente la espera frente a tener que aguardar a que la pieza sea enviada a una unidad. Además,

ciertas piezas pueden retrasar el desarrollo de la actividad de una unidad, por lo que la ausencia de ésta podría implicar un coste añadido.

Para trabajar con las impresoras 3D hace falta pasar los diseños a formatos .stl (*Standard Triangle Language*). Existen gran cantidad de programas para trabajar con las impresoras 3D, muchos de ellos basados en software libre como Simplify3D [3] y Cura [4]. Gracias a este tipo de programas se pueden traducir los modelos 3D en instrucciones que una impresora pueda comprender.

Por lo tanto, la impresión 3D va intrínsecamente ligada a los diseños que habilitan a las impresoras para producir las piezas. Si se plantea integrar en los buques un sistema de impresión 3D que pueda ser usado por la Armada, es necesario implementar una herramienta estándar o catálogo que permita la gestión de estos diseños. Para hacer un catálogo digital de piezas, lo más adecuado será desarrollarlo con archivos tipo .stl, los cuales estarían listos para impreisón ya que almacenan todos los parámetros de altura de cada capa, velocidad de impresión y posición de cada elemento.

1.2 Objetivos del TFG

Dado que este TFG nace del empleo de técnicas de fabricación aditiva en la Armada, resulta lógico plantearse la necesidad que tendrá la Armada por compartir entre sus unidades los diseños que resulten válidos. Por tanto, se propone una herramienta web que sirva de almacén virtual o catálogo de repuestos que puedan ser desarrollados mediante fabricación aditiva y destinados a unidades de la Armada.

Esta herramienta web debe diseñarse de tal forma que permita almacenar diseños de extensión .stl, junto con la información propia de cada uno. Estos diseños deben ser accesibles por cualquier usuario para su descarga y, dado que el volumen de diseños que almacene la web puede ir creciendo, debe proporcionar al usuario un método de búsqueda y filtrado para que pueda diferenciar entre varias opciones.

Además, el usuario debe de tener control sobre sus propios diseños, permitiéndole modificar o eliminar uno si lo viese preciso. Por último, la web debe contar con un control de autenticación que permita gestionar el acceso de los usuarios.

1.3 Organización de la memoria

La presente memoria se compone de 5 capítulos junto con sus anexos.

Los capítulos uno y dos se corresponden con la introducción y el estado del arte, que servirán al lector para ubicarse y contextualizar el trabajo. Primero se plasma cómo la Armada gestiona los repuestos a nivel interno. A continuación, se estudia el desarrollo de la impresión 3D como una tecnología emergente, los beneficios que reporta a los usuarios y cómo la fabricación aditiva ha afectado a la industria y a la cadena de suministros. Por último, se realiza un estudio de los repositorios online existentes y se hace un repaso de las herramientas que se emplearán para el desarrollo de la aplicación web.

Los capítulos tres y cuatro de esta memoria se dedican a exponer el desarrollo de la aplicación. En ellos se afronta desde cero el diseño de la aplicación, comenzando por la instalación del entorno de trabajo y culminando con la validación de la web. Durante estos capítulos se afronta el problema de la gestión de piezas, el registro de usuarios y el diseño gráfico de la aplicación web.

El capítulo cinco plasma las conclusiones obtenidas y plantea las líneas futuras. El capítulo seis está compuesto por la bibliografía. Por último, figuran el anexo I y el anexo II. El primero contiene los archivos relacionados con el código en PHP y el segundo los archivos que definen las vistas en HTML.

2 ESTADO DEL ARTE

Este capítulo describe el empeño de la Armada por actualizarse y embarcarse en la Cuarta Revolución Industrial. Se estudiará cómo se gestionan los pedidos en las unidades de la Armada, así como el nuevo modelo de apoyo logístico que pretende alcanzar mediante el "Gemelo Digital". A continuación, se verá cómo las impresoras 3D han entrado a formar parte de muchas compañías como herramientas capaces de potenciar sus beneficios y cómo pueden afectar a la cadena de suministros tal y como la conocemos hoy en día. Por último, se revisarán los repositorios online más importantes y se hará un repaso de las herramientas software básicas que serán usadas en este TFG.

2.1 La Armada 4.0

A lo largo del tiempo la industria ha avanzado mano a mano con el desarrollo de las tecnologías y de los materiales, lo que se ha traducido en mejoras en la calidad de vida y nuevas oportunidades de desarrollo de productos y líneas de investigación.

A finales del siglo XVIII surge en Gran Bretaña la Primera Revolución Industrial con la invención de la máquina de vapor y su aplicación a la industria textil y al transporte. Más adelante, a mediados del siglo XIX, llegó la Segunda Revolución Industrial con el desarrollo de la electricidad y la producción en masa con líneas de montaje. La Tercera Revolución Industrial surgió a partir de 1920, época en la que la aviación, la energía atómica, la electrónica y el desarrollo de los medios de comunicación comenzaron a ganar mayor presencia permitiendo automatizar procesos industriales.

Queda demostrado que la industria avanza conforme avanzan la tecnología y los materiales, pero no son los únicos factores determinantes que afectan al avance de esta. Los intereses de los clientes han cambiado con el paso del tiempo, los nuevos mercados se basan en un enfoque al producto frente a la demanda, productos que requieren una mayor personalización, mayor calidad y mayor competitividad frente al resto.

Hoy nos encontramos en los inicios de lo que ya se denomina como Cuarta Revolución Industrial. Las tecnologías de fabricación sufren una transformación sin precedentes con el objetivo de dar respuesta a las exigencias del mercado. Esta transformación viene de la mano de una serie de tecnologías que, combinadas, dan paso a un nuevo modelo industrial.

La industria 4.0 se basa en la conexión entre personas y máquinas gracias al uso de sensores, manejo de gran cantidad de datos y procesamiento de los mismos, y novedosos sistemas de telecomunicaciones. Los avances de la industria se plasman gracias a los avances en robótica colaborativa, realidad aumentada, *Big Data*, impresión 3D, visión artificial, sistemas ciber-físicos o el avance en redes WiFi. La industria 4.0 supone la digitalización de los procesos industriales además de un nuevo modelo que nos permite nuevas formas de fabricar y hacer llegar los productos.

La transformación 4.0 es visible en todo tipo de organizaciones, públicas y privadas, civiles y militares. La transformación para realizar un uso más eficaz de sus tecnologías y redes es cada vez más notable y la Armada no se queda atrás en esta transformación. En este contexto, el desarrollo y puesta en práctica de una estrategia de transformación digital que permita explotar al máximo las ventajas competitivas que, en términos de eficacia, aportan este conjunto de nuevas tecnologías, resulta absolutamente crucial para alcanzar y mantener la superioridad sobre potenciales adversarios.

La Armada, apoyada por el CESTIC (Centro de Sistemas y Tecnologías de Información y Comunicaciones de la Secretaría de Estado de Defensa) [5], ha iniciado un proceso de transformación digital que le permita adaptarse a las nuevas tecnologías sacando el máximo provecho de estas, con el objetivo de maximizar la eficacia en el uso de los recursos materiales, económicos y humanos de los que dispone.

El enfoque que pretende la Armada para esta transformación tiene como pieza angular el recurso humano. La persona será siempre el centro sobre el que se trabajará debido a dos razones. La primera, la transformación debe permitir concentrar la utilización del personal en aquellas tareas que sean de mayor importancia y en las que el factor humano pueda aportar mayor beneficio que un proceso automatizado. La segunda, la introducción de cualquier nueva tecnología estará siempre dirigida a facilitar el trabajo del personal, permitiéndole ser más eficiente [6].

En un coloquio online entre Navantia y la Armada, el entonces Almirante Jefe de Estado Mayor de la Armada (AJEMA), Teodoro López Calderón, afirmó que se podrían considerar algunas tecnologías más importantes que otras, por ejemplo, las de "sensorización y tratamiento de datos para optimizar el uso de la fuerza naval con el objetivo último de la superioridad en el uso de la información y, por tanto, en el combate" [7].

La Armada ha comenzado esta transformación centrando su apuesta en el ámbito logístico, iniciando estudios para el desarrollo de un Sistema Integrado de Sostenimiento (SIS) [8]. Como un derivado de este programa, y a iniciativa de la Jefatura de Apoyo Logístico, la Dirección General de Armamento y Material ha iniciado recientemente el programa *Soprene* [9], cuyo objetivo es el uso de la inteligencia artificial para el tratamiento de los datos obtenidos de cada buque con el fin de realizar un mantenimiento predictivo de las unidades.

Los nuevos submarinos S-80 y las futuras F-110 ofrecen una oportunidad única para dar el salto a la Armada 4.0. Mediante la explotación de los datos proporcionados por maquetas y gemelos digitales se consigue un mayor desarrollo y mejor gestión de los programas.

Pero la Armada no se encuentra sola en este cambio hacia la transformación digital. Está apoyada por empresas como BABEL, un equipo internacional especializado en tecnologías de vanguardia, cuya misión principal es asistir a grandes clientes en sus retos y procesos de digitalización. Desde hace más de 30 años BABEL forma un pilar fundamental para la Jefatura de Apoyo Logístico (JAL), la Jefatura de Personal (JEPER) y la Dirección de Asuntos Económicos (DAE), manteniendo el sistema integrado de gestión de materiales de la Armada que engloba las operaciones logísticas relativas a repuestos, pertrechos, pintura, material de oficina, combustibles, grasas, subsistemas, munición y motores.

2.2 Gestión de los repuestos en la Armada

2.2.1 Subdirección de aprovisionamiento y transporte SUBDAT

La Armada se basa en tres pilares fundamentales: Cuartel General de la Armada, la Fuerza y el Apoyo Logístico. El Apoyo Logístico lo lleva a cabo la Jefatura de Apoyo Logístico (JAL), de la que cuelgan la Dirección de Ingeniería y Construcciones Navales (DIC), la Dirección de Sostenimiento (DISOS), la Dirección de Infraestructura (DIN) y la Dirección de Gestión Económica (DIGEC) [10] tal y como se puede ver en la Figura 2-1.

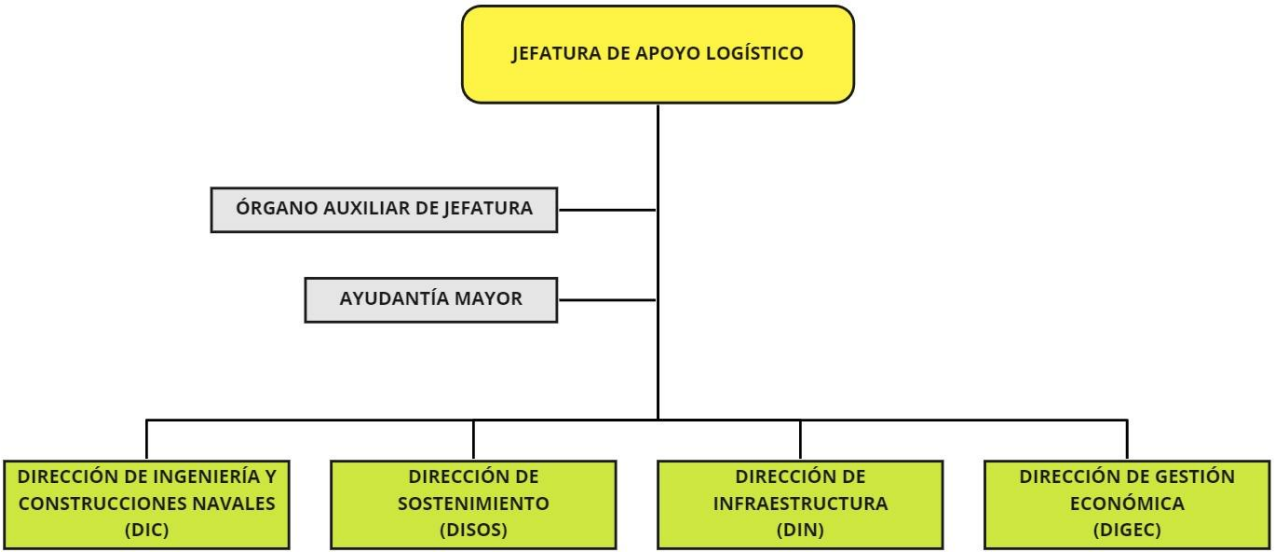


Figura 2-1 Estructura de la Jefatura de Apoyo Logístico (elaboración propia)

Nos centraremos en la DISOS [11] pues de ésta cuelga la Subdirección de Aprovisionamiento y Transportes (SUBDAT). La SUBDAT es la encargada de llevar a cabo las actividades relacionadas con la gestión, administración y análisis referentes a aprovisionamiento, almacenamiento y distribución de suministros, repuestos y pertrechos, así como los transportes y gestiones necesarios para asegurar el apoyo a la Fuerza [12].

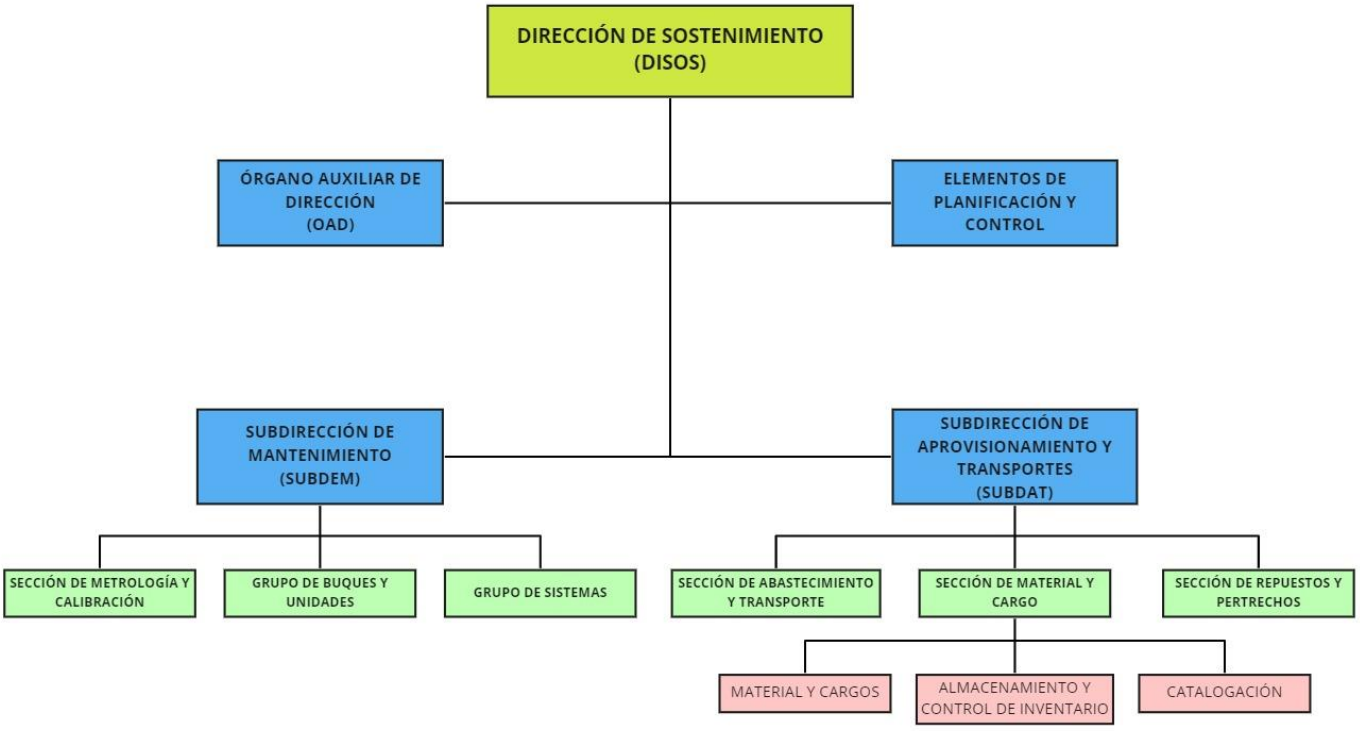


Figura 2-2 Estructura de la DISOS (elaboración propia)

Como se puede ver en la Figura 2-2, la SUBDAT se estructura en: Sección de Abastecimiento y Transporte, Sección de Material y Cargos y Sección de Repuestos y Pertrechos.

La Sección de Material y Cargos tiene como finalidad la gestión de catalogación, almacenamiento y control de inventario de material de la Armada. Es la encargada de dirigir y controlar el proceso de elaboración y actualización de los libros de cargo de las unidades, donde viene reflejado todo el material

que deben tener disponible en todo momento. Además, actúa como segundo escalón de catalogación al encargarse de establecer los criterios técnicos para los primeros escalones de catalogación de la Armada, los Arsenales. La Secretaría de Material y Cargos trabaja en coordinación con el Servicio de Catalogación del Ministerio de Defensa (SACADE), que conformaría el tercer escalón de catalogación.

Como se puede ver en la Figura 2-2, la Sección de Material y Cargos también se encarga de proponer, controlar y gestionar la obtención y adquisición de los repuestos y pertrechos de la Armada.

2.2.2 Proceso de asignación de un repuesto

Para poder entender el proceso de asignación de un repuesto es necesario conocer en qué consisten los escalones de aprovisionamiento de la Armada.

El Primer Escalón se corresponde con el material de aprovisionamiento que se almacena en los buques, unidades o instalaciones (BUIs) para apoyar sus cometidos durante un tiempo determinado. Es el material que almacena cada entidad dentro de sus pañoles de aprovisionamiento para tener disponible en cualquier momento. El mantenimiento de los niveles de este escalón será responsabilidad de sus Jefes o Comandantes.

El Segundo Escalón es el que se corresponde con todo el material de aprovisionamiento que se encuentra en los almacenes de la Armada o en almacenes ajenos a ella para apoyar a los BUIs durante un tiempo determinado. La responsabilidad de mantener los niveles de este escalón cuelga del Almirante Jefe de Apoyo Logístico (AJAL).

El proceso de adquisición de un repuesto en un barco involucra a ambos escalones. Cuando es necesario obtener un repuesto, el Jefe de Servicio emite un vale de material a través de la aplicación SIGAPEA [13] (Sistema Integrado de Gestión de Aprovisionamiento de Primer Escalón de la Armada) al servicio de aprovisionamineto de la unidad. En caso de no encontrarse el repuesto a bordo, el servicio de aprovisionamiento emite otro vale, o pedido SIGMA, al segundo escalón. Al recibir este vale, los Arsenales gestionan y localizan la pieza mediante otra aplicación de uso interno, SIGMA WEB (Sistema Integrado de Gestión de Material de la Armada) [13]. Es preciso aclarar que para la obtención de material se puede recurrir a distintas fuentes como otros barcos de la Armada, buques extranjeros, fabricación a bordo o unidades militares españolas ajenas a la Armada [13].

Es importante tener en cuenta que el concepto de autonomía de un buque puede venir determinado por diversos factores: combustible, resistencia de la dotación, víveres o existencias. Por ello, la autonomía de un buque es la resistencia del mismo para permanecer durante un período prolongado de tiempo en la mar sin que sus capacidades de actuación en el cumplimiento de una misión se vean disminuidas. Por lo tanto, el material a bordo es necesario para su permanencia en la mar.

2.2.3 Catalogación en las FAS

El apoyo logístico en una unidad tiene como objetivo prever, obtener, almacenar y distribuir piezas para satisfacer las necesidades de dichas unidades. Las actividades de aprovisionamiento requieren la identificación, el control de inventario, la obtención y la distribución de los artículos necesarios. Es posible que un mismo elemento, que sea idéntico física y funcionalmente, sea fabricado por distintas empresas en distintos países.

Así pues, varias empresas pueden generar artículos idénticos pero con números de referencia diferentes. Esto se convierte en un problema a la hora de aprovisionar a las Fuerzas Armadas, no solo a nivel nacional sino también a nivel internacional, pues muchos sistemas son compartidos por diferentes países. Por ello, surge la necesidad de desarrollar un sistema propio de catalogación que asigne un número identificador único para cada artículo. La Figura 2-3 Problemática del aprovisionamiento a nivel internacionalmuestra un ejemplo del problema que supone tener distintas nomenclaturas.

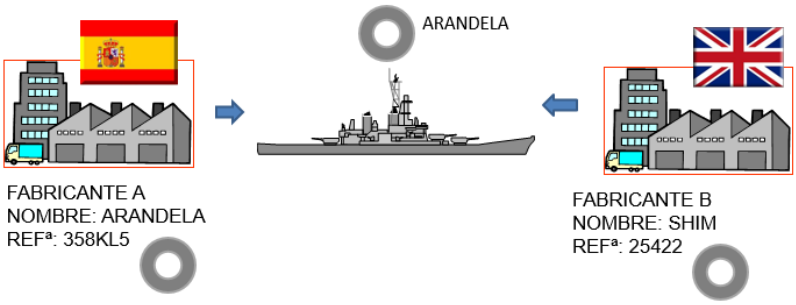


Figura 2-3 Problemática del aprovisionamiento a nivel internacional (elaboración propia)

De esta forma, mediante una nomenclatura común y uniforme para la identificación de artículos se busca aumentar la eficacia de los sistemas logísticos y facilitar las gestiones de los usuarios, simplificar mucho las comunicaciones y reducir los gastos. Para solventar este problema de catalogación, la OTAN y países apadrinados (países que no son miembros pero interactúan de forma activa) emplean el Sistema OTAN de Catalogación (SOC) [14].

El SOC es un procedimiento uniforme y común, usado para la denominación, clasificación, identificación y numeración de los artículos de abastecimiento. Para ello, cada artículo recibe un Número OTAN de Catalogación o NOC, un número que, como podemos ver en la Figura 2-4, está compuesto por 13 dígitos divididos en tres campos en los que se identifican el grupo o la clase del artículo, la oficina nacional de catalogación y un número no significativo asignado por la oficina que lo cataloga como referencia. A nivel nacional, se emplea el Número de Identificación Nacional (NIN), que es similar al NOC suprimiendo los 4 primeros dígitos.

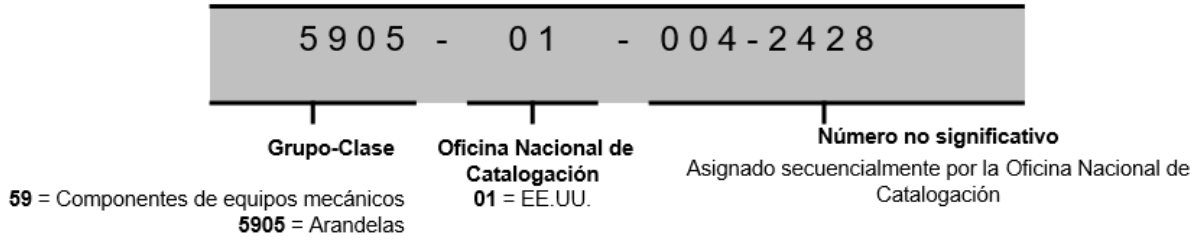


Figura 2-4 Estructura del NOC (elaboración propia)

De esta forma, todo lo que se encuentre a bordo de un buque queda perfectamente catalogado. Mediante las aplicaciones SIGAPEA y SIGMA WEB es posible introducir estos números identificadores para ver el tipo de artículo del que se trata, siendo posible comprobar las características de cada artículo consultando la información que aparecerá en cada campo. Volviendo al ejemplo de la Figura 2-3, al introducir el NOC obtendríamos que corresponde con las arandelas que estamos buscando, y en las aplicaciones se podrían consultar las dimensiones, el material del que están hechas, la cantidad de arandelas que suministra el proveedor, si se suministra de forma individual o incluso su vida útil.

2.2.4 Fundamentos tecnológicos del apoyo logístico 4.0

Como se ha visto en el apartado 2.1, la Armada se embarca en la denominada Cuarta Revolución Industrial con el fin de obtener en tiempo real patrones de comportamiento, condiciones de uso y datos sobre el estado de sus mecanismos que le permitan hacer un mantenimiento efectivo de sus buques. Los nuevos programas de obtención de la Armada, las futuras F-110, los S-80 y el nuevo BAM-IS, suponen una oportunidad única para impulsar un nuevo concepto de apoyo logístico.

Como viene recogido por el AJEMA en [15], la evolución hacia un nuevo concepto de apoyo logístico en la Armada se basará en los siguientes principios:

- **Interoperabilidad:** El nuevo modelo debe permitir la conexión entre las máquinas y los usuarios a través de una plataforma digital que siga un mismo patrón. De esta forma, los usuarios podrán operar en distintas unidades sin necesidad de tener que aprender de nuevo el funcionamiento de otra plataforma distinta.
- **Virtualización:** Se implantará de forma digital un modelo que permita a las nuevas unidades el sostenimiento mediante un mantenimiento predictivo, el control de la configuración, la enseñanza y el adiestramiento de la dotación a través de la implementación de un gemelo digital [16] que se base en la simulación única de los buques, capaz de realizar el análisis masivo de datos.
- **Automatización:** Se detectarán aquellas tareas que puedan ser automatizadas con el fin de liberar al personal de estas funciones, quedando disponibles para otros trabajos.
- **Capacidad de trabajo en tiempo real:** Logrando esta característica, los usuarios podrán analizar los datos de las plataformas en tiempo real, permitiendo así una mayor capacidad de reacción ante fallos que se produzcan en el proceso.
- **Fiabilidad:** El modelo de mantenimiento se centrará en la fiabilidad, combinando un mantenimiento preventivo, predictivo y correctivo de las unidades.
- **Modularidad:** Los sistemas estarán formados por módulos para facilitar el trabajo por componentes, sin necesidad de cambiar el sistema principal cuando se desee reparar o actualizar.
- **Orientación:** Los sistemas de trabajo se orientarán a mejorar la operatividad de las unidades, buscando un menor coste y la confianza de los contratistas y promotores externos.

2.2.5 Nuevo modelo de apoyo logístico

La finalidad será mantener la mayor operatividad de los buques y sistemas con un coste menor. Las nuevas unidades que se incorporarán a la Armada deben beneficiarse de este nuevo modelo, mientras que las unidades en servicio adoptarán este modelo de forma progresiva y bajo un criterio adecuado según eficacia-coste.

Como se ha comentado anteriormente en el apartado 2.2.4, la configuración de los buques pasará a fundamentarse sobre una única base de datos para cada barco que se denominará “Maqueta Digital” o “Gemelo Digital”. El nuevo modelo de gestión logística proporcionará más servicios a la vez que demandará mayor gestión de datos y una mayor colaboración con la industria. La Dirección General de Armamento y Material (DGAM) [17] elaborará para cada unidad un Concepto de Apoyo Logístico (CAL), mientras que la Armada elaborará un Concepto de Apoyo Logístico Integrado (CALI) de acuerdo con el concepto desarrollado por la DGAM, como puede verse en la Figura 2-5 Esquema de desarrollo PALI . A continuación, a partir del CALI se redactará un Plan de Apoyo Logístico Integrado, el cual se aplicará con la Maqueta Digital para trabajar con distintos programas tanto civiles como militares [15].

2.2.6 Planeamiento y gestión del sostenimiento

El sostenimiento, regulado por el Secretario de Defensa (SEDEF) en su Instrucción 5/2008 [18], es el conjunto de actividades logísticas desarrolladas por una unidad para mantener y reparar los sistemas de armas de forma que se garantice la correcta operación de los mismos cuando y donde sea necesario. El sostenimiento de las unidades se desarrolla mediante tres funciones logísticas: Mantenimiento, Aprovisionamiento e Ingeniería del Ciclo de Vida.

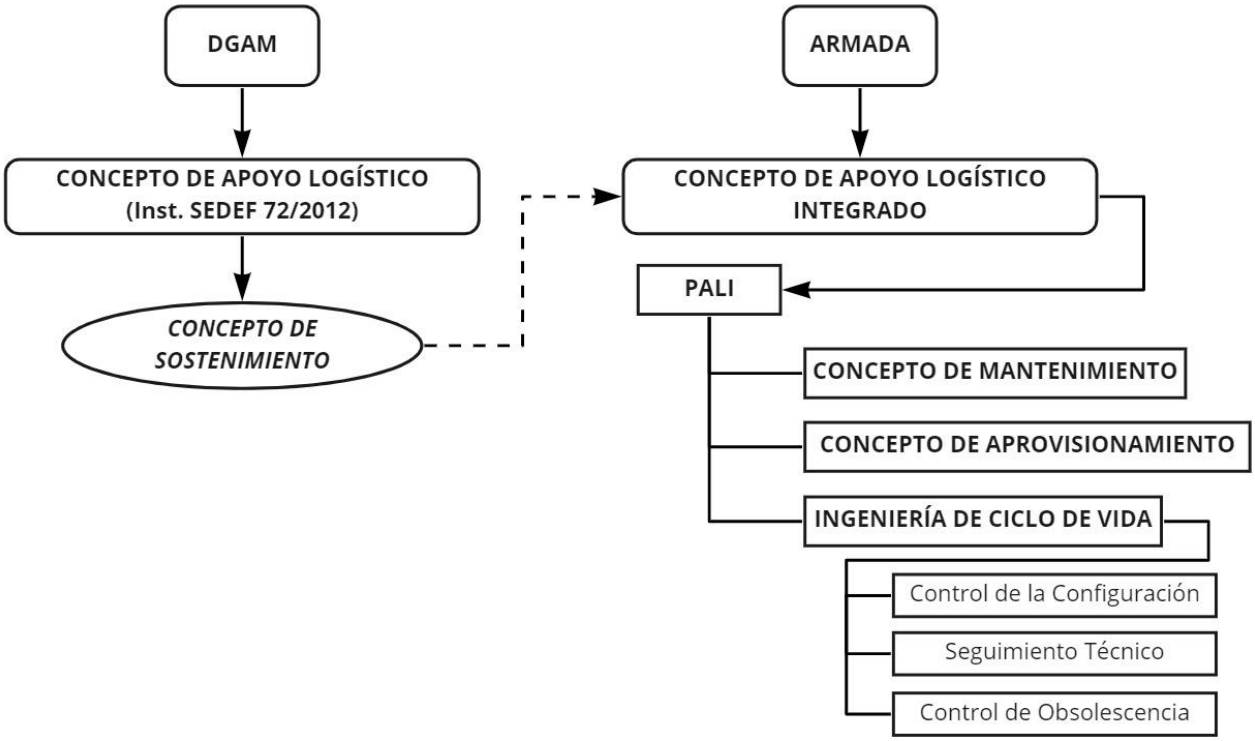


Figura 2-5 Esquema de desarrollo PALI (elaboración propia)

Las unidades dispondrán de un plan de mantenimiento orientado a la prevención de fallos que puedan afectar a la operatividad y poner en compromiso su disponibilidad. Este plan está basado en la fiabilidad, usando para cada caso el modelo de mantenimiento que resulte más adecuado: correctivo, preventivo o predictivo.

En el aprovisionamiento se buscará maximizar el empleo de las tecnologías de la información mediante el concepto de "Almacén Virtual Único de la Armada" [15]. Este concepto busca potenciar el uso de las tecnologías para optimizar y agilizar las gestiones y los niveles de stock, facilitando la toma de decisiones en lo referente al abastecimiento de las unidades de la Armada.

La ingeniería de ciclo de vida se corresponde con el conjunto de actividades necesarias para adecuar los sistemas de armas a sus requisitos operativos. Dependiendo del sistema o elemento, se realizarán trabajos distintos para asegurar el ciclo de vida atendiendo al control de la configuración o a la gestión de la obsolescencia.

2.3 Impresión 3D

Como se ha comentado en el apartado 2.1, la Cuarta Revolución Industrial se compone de una serie de tecnologías revolucionarias que, combinadas, dan lugar a un nuevo concepto de fabricación y manejo de la información. La impresión 3D comienza en 1992 con la estereolitografía, técnica para crear prototipos 3D mediante la solidificación de un fotopolímero. Los avances en este campo han sido notables: hemos pasado de imprimir pequeños prototipos de figuras de baja calidad y de carácter muy general, a la impresión de órganos humanos válidos gracias a los avances producidos en las impresoras.

La impresión 3D ha aportado una nueva forma en la que construimos objetos [19] mediante la aplicación de material por capas es posible la obtención de nuevas formas que antes requerían mayor trabajo o dificultad. Estas nuevas geometrías y formas de construcción traen consigo la ventaja de que los objetos obtenidos pueden ser más ligeros, más resistentes y más fuertes que sus homólogos. Además, los ensamblajes se ven simplificados dado que los objetos necesitan menos piezas para realizar el ensamblado.

Durante los últimos años, la impresión 3D ha pasado de ser una tecnología emergente a ocupar los puestos más altos de innovación. Este crecimiento del uso de la impresión aditiva se debe al aumento de compañías que se han decidido a invertir en esta tecnología debido a que ofrece la posibilidad de crear nuevos modelos y servicios, que antes eran imposibles o más complicados, y a la ventaja económica que ofrece la impresión 3D en comparación con la producción convencional.

2.3.1 Funcionamiento de las impresoras 3D

La idea principal de funcionamiento de la impresión 3D no difiere mucho de la impresión tradicional en 2D. Ambas impresoras obtienen la información en formato digital, la procesan e imprimen el resultado. La diferencia reside en que, mientras que la impresora tradicional imprime la tinta sobre el papel u otro tipo de soporte, la impresora 3D aplica material de diferentes formas para crear un objeto sólido. El material por lo general se aplica capa a capa, aunque esto podría variar según el tipo de técnica de impresión que se utilice.

Por lo tanto, como se puede ver en la Figura 2-6, para la impresión 3D son necesarios tres pasos básicos:

1. Modelo digital: Primero se realizar el modelo digital de la pieza a imprimir. Los modelos digitales se pueden crear a partir de programas CAD como AutoCAD, SolidWorks o Siemens NX; o mediante el empleo de escáneres 3D que obtengan la información tridimensional del objeto original. Una vez que se crea el modelo 3D, el software descompone cada pieza en planos horizontales, de tal forma que cada punto del objeto se corresponde con una posición en un plano determinado. El formato estándar adoptado por la industria 3D para la transformación de datos son los ficheros .stl (*Standard Triangle Language*).
2. Material de impresión: La selección del material de impresión es importante pues juega un papel crucial en las propiedades finales del objeto tales como la dureza, la resistencia o la flexibilidad. Hoy en día se trabaja con una gran cantidad de materiales, destacando los polímeros y los metales, pero también se realizan investigaciones con cerámicas, hormigón e incluso materiales comestibles.
3. Impresora 3D: La impresora 3D es el hardware final que se emplea para la obtención del objeto. Existen varios modelos de impresoras atendiendo al uso que se le dará, el material que se desea emplear o el volumen de piezas que se desea producir.

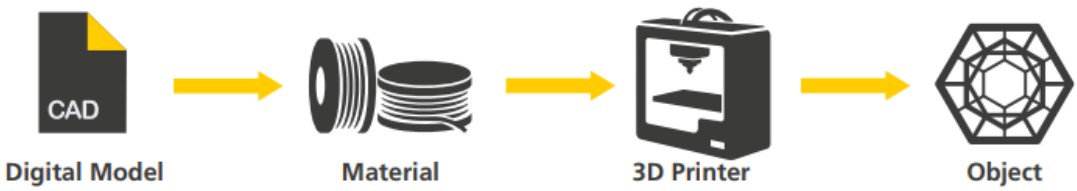


Figura 2-6 Proceso de impresión 3D [20]

Existen muchas técnicas diferentes de impresión 3D [21], de entre ellas cabe destacar las siguientes:

- Sinterizado selectivo por láser: Técnica de prototipado funcional basada en la unión de material por sinterización de partículas en polvo mediante un láser de CO_2 de potencia media.
- Modelado por deposición fundida: Técnica de prototipado formal basada en materiales sólidos. Hace pasar polímeros o ceras a través de un extrusor a alta temperatura, estos se enfrían y se endurecen tras fluir por el extrusor y se adhieren a las capas anteriores.
- Estereolitografía: Técnica de prototipado formal que se basa en el uso de resinas líquidas, las cuales se endurecen tras su exposición a un láser de luz ultravioleta.

2.3.2 Aplicaciones

Este apartado se dedica a revisar las principales aplicaciones de la impresión 3D y el potencial que tiene. Se presentan ejemplos de empresas que hoy en día ya trabajan con esta tecnología y veremos el resultado y la experiencia que se ha obtenido.

- Educación: Las impresoras 3D son utilizadas en colegios y universidades como apoyo a la enseñanza. ThingMaker [22] es una herramienta desarrollada por Autodesk que permite a los niños crear juguetes mediante un software intuitivo.
- Salud: El uso de impresoras 3D en salud busca proporcionar al cliente una experiencia individualizada. NextDent [23] es una empresa que produce coronas dentales que se ajustan perfectamente a los pacientes. Para ello usa el empleo de un escáner 3D y luego imprime el modelo. Otro ejemplo es SOLS [24], una empresa que produce plantillas personalizadas. Los clientes solo tienen que escanear la planta del pie mediante un smartphone o a través de un ortopeda y SOLS produce las plantillas. Marcas como Nike o Adidas también plantean sacar provecho a las impresoras y ya están estudiando el modo de crear modelos totalmente personalizados.
- Automóvil: Local Motors ha sido la primera en diseñar un coche eléctrico con un chasis construido mediante una impresora 3D. El grupo BMW incorpora más de 10.000 elementos impresos en el Rolls-Royce Phantom y afirma que mediante el uso de la impresión 3D se reducirán tiempos de producción en el modelo [20].
- Aeronáutica: La multinacional General Electric [25] ha diseñado toberas para la inyección de combustible en turbinas un 25% más ligeras y cinco veces más duraderas. Los modelos anteriores de toberas se componían de 20 piezas procedentes de distintos distribuidores que debían ser ensambladas. Airbus también ha incorporado esta tecnología, solo en el modelo A350 incluye cerca de 1.000 piezas [26].
- Militar: Las impresoras 3D pueden llevar piezas a lugares donde es más difícil acceder como buques o bases desplegadas en el extranjero. En 2014 la US Navy incorporó impresoras 3D en los portaaviones de la clase Essex con el fin de entrenar a los marineros en el diseño de repuestos y componentes de distintas armas, con el objeto de reducir el tiempo de aprovisionamiento [27]. Otro proyecto de la Navy es el diseño de drones bajo demanda. A la salida de puerto se embarcarán los componentes mínimos del dron como los motores, hélices y cableado, y mediante las impresoras 3D se diseñará el cuerpo del dron atendiendo a las necesidades de la misión, inteligencia o reconocimiento [28].

2.3.3 Efectos de la impresión 3D en la cadena de suministros

Como hemos visto, una de las características principales que se espera que puedan aportar las impresoras 3D es la capacidad de producir figuras muy complejas o con un alto grado de especificación en el diseño de una forma muy sencilla. En cambio, plantearse la producción de grandes volúmenes de piezas mediante impresión 3D sería un error debido a que el tiempo que se tarda en imprimir una pieza es relativamente alto. Sin embargo, si es aplicada de forma correcta, la impresión 3D podría llegar a tener un fuerte impacto en la cadena de suministros de algunas compañías.

Hoy en día, se almacenan miles de repuestos en almacenes distribuidos por todo el mundo. Existen todo tipo de almacenes atendiendo a los repuestos: algunos almacenes más genéricos, como por ejemplo aquellos enfocados al sector automovilístico, y otros más específicos, como por ejemplo aquellos destinados a los repuestos de material sanitario o prótesis.

El objetivo de una compañía a la hora de crear una red de almacenes es poder hacer llegar estos repuestos de forma rápida adonde sea necesario, ahorrando en transporte y manteniendo siempre una

alta disponibilidad. Pero la realidad es que el almacenamiento de repuestos y el mantenimiento de un stock es costoso, depende de una previsión y puede variar según las necesidades del cliente. Se estima que aproximadamente el 25% [29] del inventario de un stock se corresponde con piezas que no son usadas y que no tienen garantías de ser usadas en el futuro, piezas que le generan un coste añadido a la compañía que regula el almacén.

Mediante la implementación de impresoras 3D, las compañías podrían reducir la cantidad de piezas almacenadas en grandes almacenes, llegando incluso a conseguir mayor alcance y eficiencia a la hora de distribuir las piezas o los repuestos demandados. Como se puede ver en la Figura 2-7, es posible realizar el suministro de repuestos trasladando el almacén físico a un "almacén virtual".

Figura 2-7 Suministro de repuestos mediante impresión 3D [20]

Pero para conseguir estos objetivos es necesario crear una red de impresoras 3D capaz de imprimir de forma instantánea cualquier objeto. Los archivos de impresión, archivos CAD o stl, deben almacenarse de forma segura en bases de datos que funcionen como un "almacén virtual", de esta forma podrían ser accesibles desde cualquier punto de la red. Una organización que ya ha puesto en práctica este concepto de almacén virtual es Kazzata [30].

Kazzata es una web que funciona como un mercado online público de repuestos en donde los usuarios suben sus diseños 3D para promocionarlos o venderlos. Otra de las opciones que incorpora Kazzata es la posibilidad de que ellos mismos diseñen el repuesto; es decir, en el caso de que el usuario no sea capaz de diseñar su pieza, manda toda la información necesaria a Kazzata y ellos mismos diseñan el fichero del repuesto. Una vez obtenido el diseño, los usuarios solo tienen que buscar el punto de impresión 3D más cercano.

Muchas compañías comienzan a apostar por la impresión de repuestos, como la conocida marca Mercedes-Benz. En un esfuerzo por resolver su falta de suministros, Mercedes ha diseñado una red propia destinada a los repuestos de piezas para sus camiones [31]. La compañía ha anunciado que pondrá a disposición de los clientes piezas de repuesto para los camiones mediante impresión metálica. De esta forma, los clientes no tendrán que esperar largos periodos de tiempo desde que la marca les envía la pieza desde la fábrica hasta que les llega.

Otra ventaja que aportan los almacenes virtuales de repuestos es que son escalables, es decir, las empresas pueden aumentar su capacidad, aumentar el alcance de la red o redefinir la base de datos para una mejor gestión, sin necesidad de incurrir en grandes costes.

2.4 Repositorios online

2.4.1 Introducción a los stocks 3D online

Si buscamos en Internet podemos encontrar distintos repositorios online para descargar ficheros de piezas muy diversas. Por lo general, todos los repositorios permiten navegar entre la larga variedad de piezas y descargarlas de forma gratuita. Estos repositorios funcionan como una comunidad que tiene dos perfiles: el usuario que busca y descarga la pieza; y el desarrollador que sube la pieza a la plataforma.

Debido a la cantidad de aplicaciones a las que se puede destinar la impresión 3D, los repositorios online son muy diferentes, algunos enfocados a piezas destinadas a educación, otros enfocados a prototipos para ingenierías que requieren un mayor grado de complejidad o, incluso, aquellos que sirven como repositorios de intercambio para promocionar a los autores de los ficheros.

Los sectores automovilístico, aeronáutico y militar requieren de piezas de alta calidad y, como se ha visto en el apartado 2.3.2, hay compañías que ya incorporan repuestos 3D en su flota. Otras, en cambio, no tienen esta capacidad. Por ello, en los últimos años, a la vez que comenzaba a despegar la impresión 3D, las empresas dedicadas a la manufactura de piezas han ido incorporando estas impresoras para aumentar sus capacidades y ofrecer nuevos servicios a sus clientes. A continuación se expondrán diversas plataformas que relacionan la impresión 3D y los usuarios con distintos enfoques.

2.4.2 AMFG

La empresa Autonomous Manufacturing AMFG [32] es una empresa dedicada a apoyar a empresas en el proceso de incorporar la impresión 3D en sus negocios. Con su sede en Londres, AMFG cuenta con más de 1200 máquinas y produce más de 21 millones de piezas al año en 20 paises distintos. La empresa se especializa en automoción, aeronáutica y defensa, maquinaria industrial y diseños específicos de clientes.

Para ello, pone a disposición de sus clientes un software al que se le pueden añadir módulos enfocados a:

- Gestión de los pedidos.
- Portal de pedidos.
- Conectividad de impresoras.
- Control de calidad y post-procesado.
- Gestión de la cadena de suministros.
- Inventario digital.
- Planeamiento de producción.
- Análisis e informes.

Mediante el software de AMFG, los clientes pueden realizar un prototipado rápido de la pieza, controlar la producción en serie de piezas y gestionar las impresiones. Mediante los módulos, el programa se vuelve más específico según las necesidades de cada empresa.

2.4.3 GrabCAD

GrabCAD Shop [33] es un software de gestión de tiendas de impresión 3D enfocado a desarrollar el comercio, simplificando el flujo de información entre ingenieros, desarrolladores y dependientes en las tiendas. Para los ingenieros y los desarrolladores, el software simplifica la comunicación con los clientes a la hora de exportar ficheros CAD y .stl. Para las tiendas, el software simplifica y ahorra tiempo en la gestión de los procesos de producción y la administración de los pedidos. En la Figura 2-8 se muestra el flujo de trabajo para una empresa que integre GrabCAD.

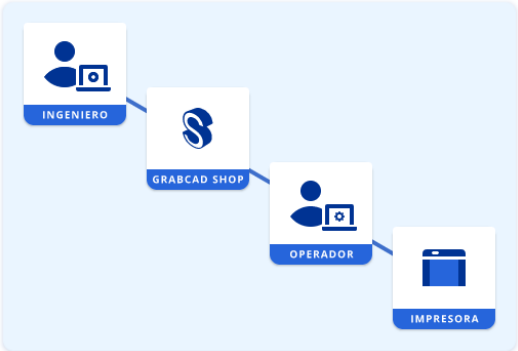


Figura 2-8 Flujo de información en GrabCAD [33]

El software permite gestionar solicitudes de impresión 3D de manera que los operadores de las tiendas pueden dar un presupuesto del coste a los clientes, actualizar el estado y prever la entrega. Permite conectar varias impresoras con el fin de crear una red para reducir tiempos o trabajar en varias piezas a la vez desde la misma plataforma.

Además, GrabCAD pone a disposición de quien lo desee GrabCAD COMMUNITY [34], un repositorio online de piezas para descargar de forma gratuita. Esta plataforma cuenta con cerca de 4.530.000 de archivos y 8.430.000 de usuarios registrados. El repositorio cuenta con un buscador y permite aplicar filtros según las interacciones de los usuarios (más recientes, más descargadas, más populares, etc) según la categoría de la pieza (agricultura, aeronáutica, educación, mecánica, etc) o según el software de diseño de la pieza (AutoCAD, Inventor, NX, SolidWorks, etc). Cada diseño 3D cuenta con una descripción, imágenes, perfil del autor y permite la opción de descargar, en caso de estar compuesto por varios ficheros, una carpeta con todos los ficheros que forman el diseño o únicamente un fichero de una parte específica del diseño. Podemos ver en la Figura 2-9 y en la Figura 2-10, respectivamente, como GrabCAD estructura su una página principal y un ejemplo de un diseño.

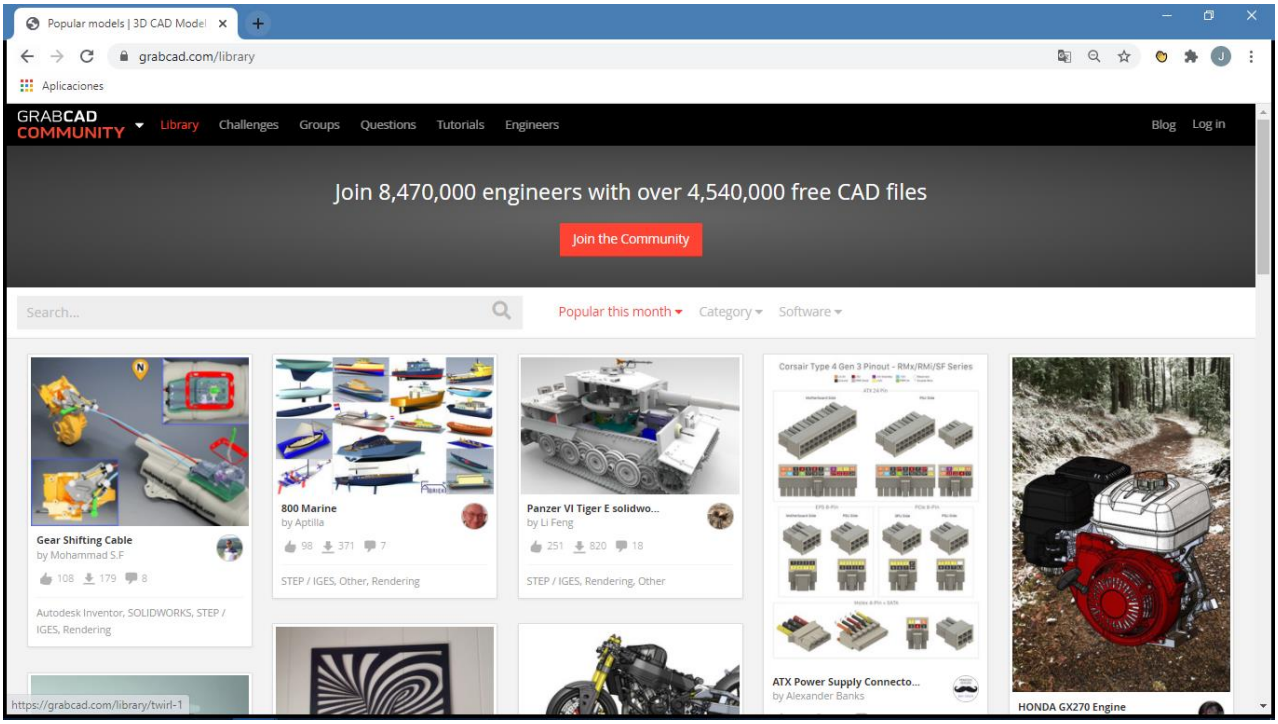


Figura 2-9 Página principal de GrabCAD COMMUNITY [34]

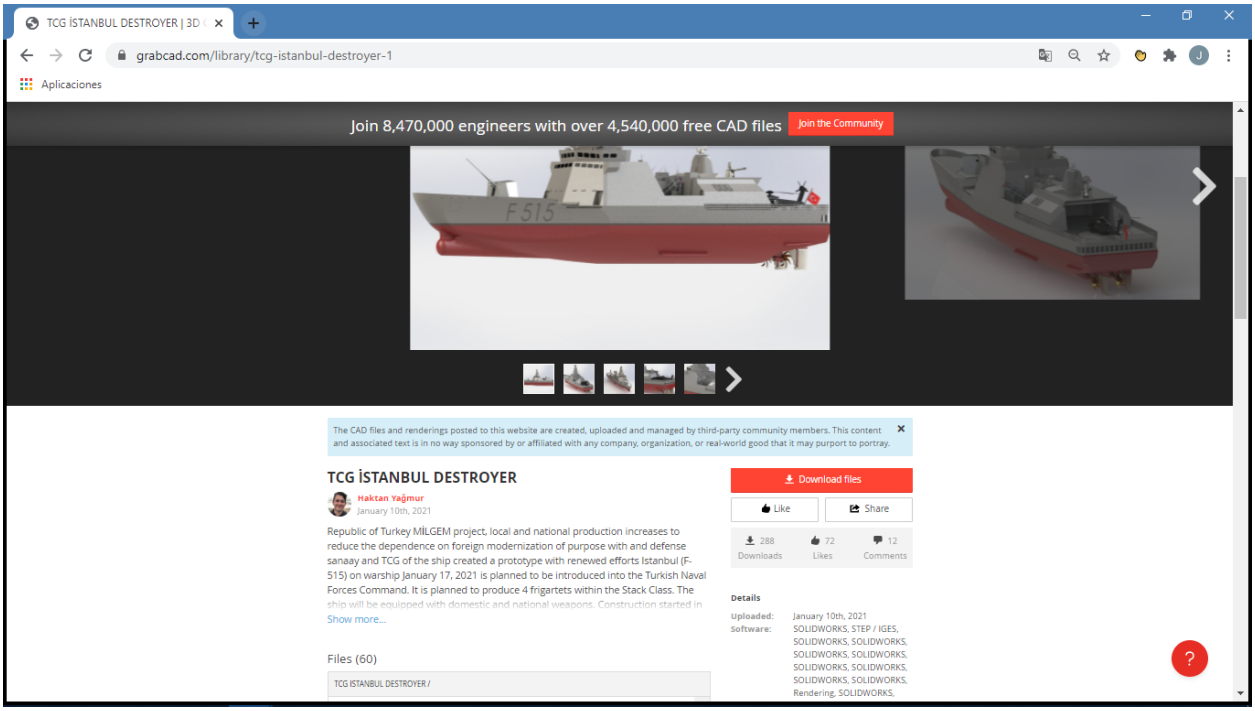


Figura 2-10 Entrada de un diseño en GrabCAD COMMUNITY [34]

2.4.4 Thingverse

MakerBot's Thingverse [35] es la comunidad más grande actualmente destinada a descubrir, crear y compartir diseños 3D. Thingverse apuesta por una plataforma a la que se puedan subir todo tipo de piezas sin necesidad de requisitos previos o especificaciones técnicas en el diseño, de tal forma que cualquiera puede subir, descargar o modificar un diseño de forma totalmente gratuita.

Aunque el diseño de la plataforma es muy similar al de GrabCAD, Thingverse cuenta con algunas diferencias. La primera es que permite hacer una búsqueda más específica mediante un filtro más desarrollado que incluye usuarios, colecciones y tipos de licencia. Otra diferencia con GrabCAD es que, aunque cuenta con diseños de todo tipo, Thingverse se centra en educación. Pero probablemente la diferencia más importante es que Thingverse da la opción de personalizar un diseño ya existente en la web, de tal forma que los usuarios pueden adaptar los diseños como quieran.

2.5 Herramientas software usadas en el TFG

En este TFG se plantea el diseño e implementación de una herramienta web para la gestión de ficheros de impresión 3D de repuestos de la Armada. Esta herramienta debe ser accesible a todas las unidades y debe funcionar como repositorio de ficheros, de tal forma que una unidad pueda buscar el modelo de la pieza deseada y descargarlo. Del mismo modo, debe poder ofrecer un control de usuarios. Por ello, antes de comenzar es preciso conocer una serie de conceptos relativos a la programación y a motores de bases de datos.

Figura 2-11 Lenguajes de programación, marcado y frameworks [36]

2.5.1 Lenguajes de programación

Para poder desarrollar la aplicación web es conveniente realizar una selección adecuada del lenguaje de programación que mejor se adapte a los objetivos que queremos lograr. Hoy en día existen una gran cantidad de lenguajes, por ello, en los siguientes subapartados se procederá a revisar de acuerdo con [37] y [38] y de manera breve, los más destacados, analizando sus ventajas e inconvenientes.

2.5.1.1 Java

Creado en 1995, Java [39] es un lenguaje de programación orientado a objetos que permite crear programas en una gran variedad de dispositivos, permitiendo ejecutar la misma aplicación en diversos sistemas operativos. Java usa una sintaxis muy similar a C++ [40], ofrece una curva de aprendizaje sencilla y un código robusto. Al ser uno de los lenguajes de programación principales hoy en día es fácil encontrar gran cantidad de recursos y soporte en línea.

2.5.1.2 PHP

Lenguaje enfocado a la creación de webs dinámicas. Hereda su sintaxis de C, Java y Perl. PHP permite además la conexión con gran cantidad de base de datos: MySQL, PostgreSQL, Oracle o MS SQL Server. Requiere Apache con librerías de PHP. Una de las características clave que hace que sea tan popular es su sencillez, pues se trata de un lenguaje de programación web apto para inexpertos compatible con HTML. Una crítica a este lenguaje es la falta de opciones dentro de la gestión de errores, aunque el hecho de que muchas páginas web populares hoy en día sigan usando PHP es una prueba de su popularidad y buen funcionamiento.

2.5.1.3 Python

Lenguaje de programación de alto nivel multiplataforma, basado en un código compacto de sintaxis sencilla y fácil de escribir. Muchos servicios web, algunos tan conocidos como Youtube o Google, emplean Python. Además la industria de los videojuegos o el ámbito científico se ha beneficiado de las características de este lenguaje. No obstante, la velocidad de ejecución es relativamente baja y sus métodos pueden llegar a ser liosos o ambiguos.

2.5.1.4 Ruby

Lenguaje que hereda su sintaxis de Python pero, a diferencia de este, Ruby hace un enfoque únicamente a objeto. La sintaxis de Ruby es flexible y fácil de leer aunque difícil de dominar. Una de las principales críticas se centra en que los errores tipográficos pueden ser difíciles de detectar y pueden generar largos procesos de revisión para solventarlos.

2.5.2 Lenguaje de marcado

Hypertext Markup Language o HTML [41] es el lenguaje de marcado principal usado hoy en día. Es conveniente apuntar que HTML no es un lenguaje de programación, sino un lenguaje de marcado, pues su función principal es la de estructurar una página web. Un lenguaje de marcado es aquel que permite codificar un texto mediante etiquetas o marcas, de tal forma que este se presente como el desarrollador quiere.

El lenguaje HTML está diseñado para la elaboración de páginas web. Especificando cada elemento como tal, HTML permite definir titulares, párrafos, listas, tablas o gráficos en una página web de tal forma que cualquier navegador puede interpretar el código y mostrar cada elemento. No obstante, HTML no incluye el diseño gráfico de la página web (en la actualidad esto se hace utilizando CSS, *Cascading Style Sheets*), pues solo sirve para indicar al navegador cómo ver el contenido mediante el uso de marcas de hipertexto.

Por tanto, HTML utiliza marcas para definir el contenido que se muestra en el navegador [42]. Las marcas HTML definen "elementos", y cada elemento se define mediante etiquetas, que se compone por el nombre del elemento rodeado por los símbolos "<" y ">". Por ejemplo, para definir el elemento que formaría el "título" de la web debemos colocar la etiqueta <title> al inicio y al final para definirlo.

2.5.3 Framework de desarrollo

Los *frameworks* o entornos de trabajo son herramientas de desarrollo web basadas en un conjunto de reglas, funciones y convenios estandarizados que trabajan juntos para el desarrollo de código y proyectos. Permiten el uso de librerías, implementan funciones y permiten ahorrar tiempo y ser más eficiente a la hora de escribir código.

El empleo de *frameworks* a la hora de comenzar un proyecto web se recomienda debido a las ventajas [43] que puede aportar su uso:

1) Mediante el uso de *frameworks* el código está organizado desde el primer momento ya que muchos *frameworks* incorporan una pre-estructura ya definida.
2) Proporcionan soluciones a la mayoría de problemas que pueden surgir en el desarrollo de aplicaciones web. Por ejemplo: la autenticación de usuarios, niveles de acceso, sesiones, la estructura de directorios y archivos, o el manejo de peticiones y respuestas, entre otros, son problemas que pueden ser afrontados de forma más sencilla mediante las herramientas incorporadas en un *framework*.
3) El empleo de una arquitectura MVC (*Model - View – Controller*) para aplicaciones web.
4) La seguridad en aplicaciones web de entrada/salida de datos se puede ver en ocasiones comprometida, pero esta queda cubierta de forma automática mediante funciones implementadas en los *frameworks*, sin necesidad de tener que realizar la protección de la web de forma manual.
5) Los *frameworks* están respaldados por grandes comunidades de desarrolladores, por lo que es más fácil conseguir ayuda a la hora de hacer consulta.
6) Fomentan el trabajo en equipo.

Hoy en día existen muchos *frameworks* disponibles para el diseño de webs. Destacan Django [44], Foundation [45], Spring [46], Laravel [47] o CodeIgniter [48]. Es importante seleccionar un *framework* adecuado según el lenguaje de programación que vayamos a utilizar.

2.5.4 Laravel

Para el diseño y desarrollo de la herramienta web que se pretende llevar a cabo en este TFG, se empleará PHP por ser un lenguaje de programación enfocado al desarrollo de páginas web, por lo que consecuentemente se ha decidido el empleo de Laravel como *framework*, frente a otros como

CodeIgniter, debido principalmente a la cantidad de paquetes y herramientas que ofrece, que serán explicados más adelante.

Laravel recibe influencia de *frameworks* anteriores [49] como Ruby on Rails, el cual desde su lanzamiento en 2004 por David Heinemeier, sentó las bases para el desarrollo de aplicaciones web popularizando herramientas y conceptos como el empleo de la arquitectura MVC, uso de APIs o formalizando un convenio de normas para el desarrollo de aplicaciones web.

Fueron muchos los *frameworks* de PHP que surgieron tras el lanzamiento de Ruby on Rails. CakePHP fue el primero en 2005, seguido por Symfony, CodeIgniter, Zend y Kohana [50]. Pero no fue hasta 2011 que aparecieron FuelPHP y Laravel como alternativas a CodeIgniter. De estos, algunos seguían los pasos de Rails y eran *frameworks* enfocados al desarrollo rápido de aplicaciones web, uso de bases de datos o estructuras MVC. A finales de 2010, CodeIgniter se convirtió en el *framework* más popular para el diseño de web con PHP. Otros, como Symfony o Zend, se centraban en el diseño de webs para comercio electrónico o multinacionales.

Taylor Otwell, creador de Laravel, lanzó la primera beta del *framework* en 2011 tras ver que otros *frameworks* como CodeIgniter se quedaban atrás al no ser capaces de actualizarse a la velocidad que PHP y otras tecnologías avanzaban.

Taylor Otwell crea así un *framework* que busca proporcionar una forma clara, simple y bonita de desarrollar código, proporcionando herramientas sencillas que ayuden al desarrollador con un aprendizaje rápido [49].

La filosofía que sigue Laravel es sencilla: "*Happy developers make the best code*" [51]. Conseguir la felicidad del programador es uno de los objetivos principales de Laravel y, para ello, se apoya en dos estrategias:

- Primera: se centra en una curva de aprendizaje sencilla. Las tareas más comunes a la hora de desarrollar una aplicación web, como pueden ser las conexiones con bases de datos, los registros e inicios de sesión o los buzones de correos, se convierten en tareas más sencillas gracias a las funciones que Laravel ofrece.
- Segunda: Laravel busca la simplicidad, por lo que ofrece un enfoque global para la resolución de problemas antes que funciones más específicas y, por tanto, más complejas.

2.5.4.1 Conceptos y funciones de Laravel

- **MVC:** El patrón de arquitectura que sigue Laravel es el Modelo-Vista-Controlador, que diferencia entre la lógica, la aplicación de la lógica y la vista en una aplicación. Tener esta separación nos permite hacer cambios únicamente en partes específicas del proyecto sin necesidad de alterar todo el proyecto para ello. A continuación en la Figura 2-12 se ejemplifica el flujo de información que sigue el patrón MVC.

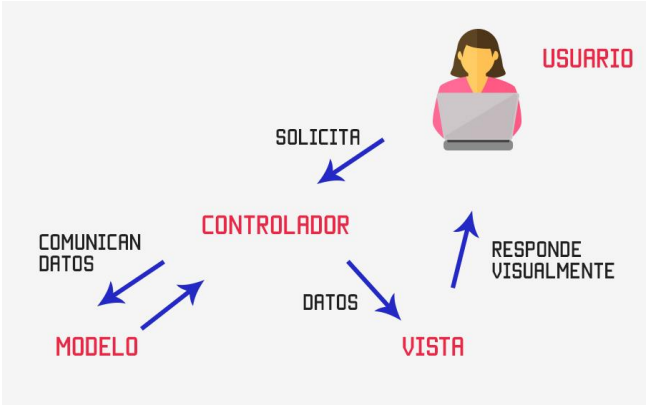


Figura 2-12 Flujo de información en el patrón Modelo-Vista-Controlador [52]

- **Blade:** Para diseñar las vistas, Laravel cuenta con un gestor de plantillas denominado Blade. A diferencia de otros gestores de plantillas, Blade nos permite insertar estructuras de PHP y trabajar con datos procedentes del controlador.
- **Eloquent:** Para trabajar con bases de datos, Laravel incluye Eloquent, un ORM (*Object Relational Mapping*) que permite trabajar con la base de datos tratando la información como objetos. Eloquent asigna a cada tabla de la base de datos un modelo para interactuar con el controlador.
- **Artisan:** Otra de las herramientas que ofrece Laravel es Artisan, un interfaz de línea de comandos que se encuentra en el directorio raíz de la aplicación que se va a desarrollar y que proporciona una serie de comandos que ayudan a realizar diferentes tareas durante el desarrollo del proyecto. Artisan favorece al desarrollador automatizando tareas como migraciones, creación de nuevos elementos, optimización de formatos o incluso poner en modo mantenimiento una aplicación.
- **Técnicas de autenticación:** Laravel pone a disposición de los desarrolladores distintos paquetes de código abierto para realizar la autenticación de usuarios de forma automática. Los principales paquetes son Breeze, Jetstream, Fortify y LaravelUI [53]. Como podemos ver en la Figura 2-13, cada uno tiene unas características distintas. Para el desarrollo de la aplicación web, se ha decidido el empleo de Breeze debido a que es un paquete que, aunque ofrece menos prestaciones que otros como Jetstream, presenta menos complejidad y cumple con las funciones que se buscan para la autenticación de usuarios.

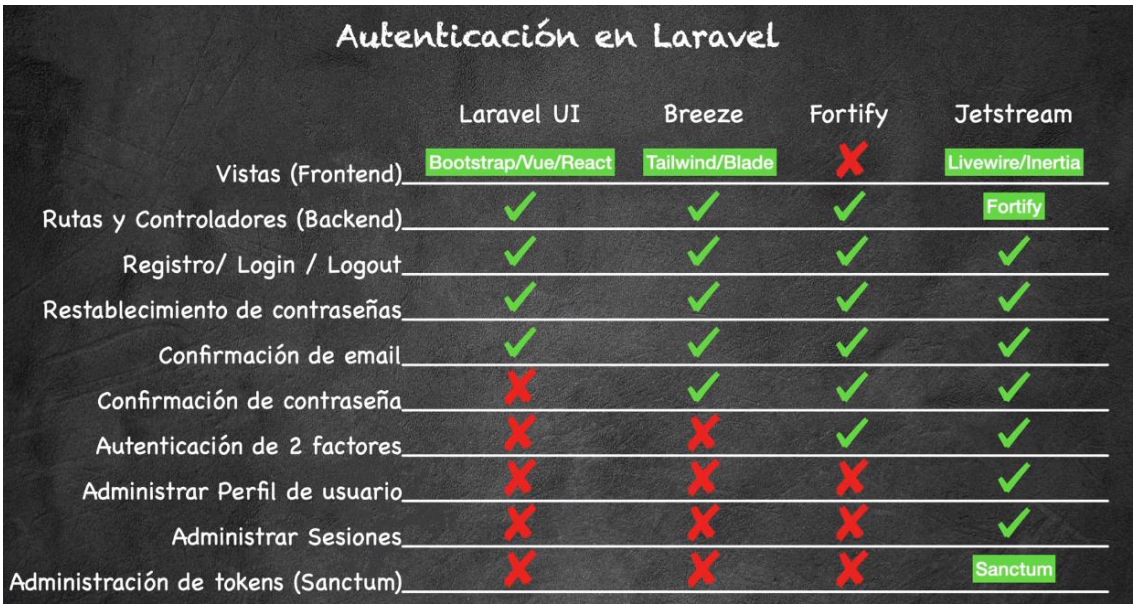


Figura 2-13 Diferencias entre paquetes de autenticación de Laravel [53]

2.5.5 MySQL

MySQL [54] es un sistema de gestión de bases de datos relacionales de código abierto que permite crear y administrar bases de datos. Una base de datos permite almacenar y organizar la información. El término relacional se refiere a que los datos se almacenan en tablas y cada tabla se relaciona con otras de alguna manera. MySQL es compatible con Microsoft Windows, macOS y Linux.

El funcionamiento de MySQL se basa en un sistema de petición y respuesta. Los clientes, a través de sus dispositivos, pueden realizar solicitudes de información obteniendo una respuesta del servidor o pueden también realizar operaciones de insertar, modificar y eliminar información.

MySQL es flexible y fácil de usar. Al ser de código abierto permite modificar el código fuente para personalizarlo. Ha demostrado un alto rendimiento gracias al compendio de servidores que lo respalda y durante años ha sido usado por la industria para el desarrollo de sus aplicaciones.

2.5.6 Composer

Composer [55] es un gestor de paquetes que proporciona un estándar para descargar e instalar dependencias y librerías en PHP [56]. Composer facilita el trabajo cuando se requiere descargar varias librerías o dependencias de distintas fuentes, pues en vez de hacer manualmente el trabajo de visitar la página de cada programa, descargarlo y copiarlo en la carpeta de nuestro proyecto, Composer centraliza todo ese trabajo y lo hace de manera automática. No solo eso, también permite actualizar librerías que estén en uso en un proyecto sin tener que volver a descargarlas de nuevo y actualizar archivos.

A la hora de trabajar con *frameworks*, Composer resulta de gran utilidad dado que estos se componen de varias librerías que, de otra forma, deberían instalarse y mantenerse actualizadas manualmente.

El método de trabajo de Composer es muy sencillo. Primero se solicita una librería o dependencia a Composer. Acto seguido, debemos instalarla, acción que también podemos realizar mediante Composer, que accederá a los repositorios necesarios y descargará las dependencias en la carpeta del proyecto. Por último, se asocia la dependencia descargada a nuestro proyecto.

Figura 2-14 Logo de Composer [56]

3 DESARROLLO DEL TFG

En este capítulo describiremos las fases del desarrollo de la aplicación web, desde la configuración del entorno de trabajo, diseño de la aplicación web y de la estructura de almacenamiento de información, a la implementación de las distintas funcionalidades.

3.1 Instalación y configuración del entorno de trabajo

Para el desarrollo del TFG se ha facilitado un servidor virtual, por lo que el primer paso será instalar y configurar nuestro entorno de trabajo con los programas necesarios para comenzar a programar. A continuación definiremos la estructura del programa y su modo de funcionamiento.

3.1.1 Servidor Linux

Como se ha comentado en el apartado 2.5, para el desarrollo de la herramienta web se va a utilizar el *framework* de Laravel. El servidor virtual proporcionado trabaja sobre Linux (Ubuntu 20.04 server) por lo que, para conectarnos desde nuestro terminal con nuestro usuario y contraseña, emplearemos el comando ssh desde un terminal cmd de Windows 10:

***C:\Users\JAVIER >ssh* hidalgo@193.146.212.191**

Una vez conectados al servidor, comprobamos que se encuentran ya instalados Apache2, PHP y MySQL, tal y como se muestra en la Figura 3-1.

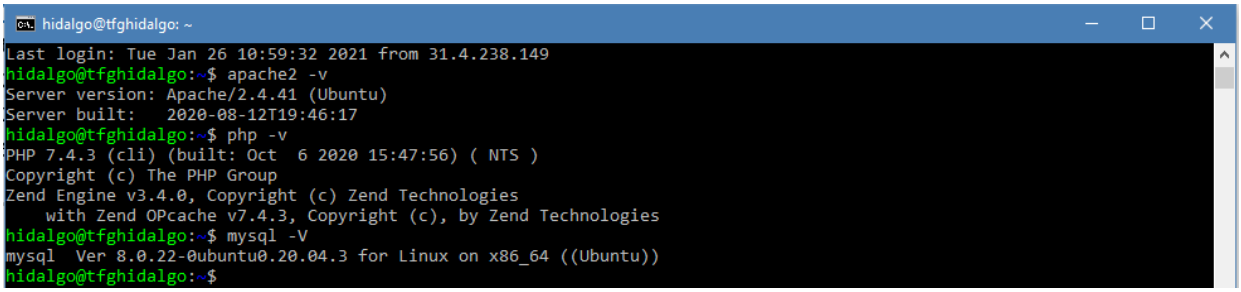


Figura 3-1 Comprobación de versiones de Apache2, PHP y MySQL (elaboración propia)

3.1.2 Instalación de Composer y Laravel

A continuación, procedemos a la instalación de Laravel. Para ello, primero debemos instalar Composer.

$ curl -sS https://getcomposer.org/installer | php
$ sudo mv composer.phar /usr/local/bin/composer
$ sudo chmod +x /usr/local/bin/composer

Con el primer comando se descarga Composer, con el segundo comando lo movemos a un directorio donde Ubuntu ubica los ficheros ejecutables, y con el tercero se define como fichero ejecutable.

3.2 Nuevo proyecto Laravel en remoto

A través de una conexión ssh con el servidor, se accede al directorio de Apache: ***$ cd /var/www.*** Una vez en dicho directorio, se crea el proyecto Laravel. Existen dos formas de crear el proyecto, el cual denominaremos ***hidalgotfg***:

- Mediante laravel: ***$ laravel new hidalgotfg***
- Mediante la función de crear proyectos con Composer: ***$ composer create-project laravel/laravel hidalgotfg***

Dado que en [49] se recomienda la primera (crear el proyecto mediante el comando ***laravel***) se ejecuta el comando ***$ laravel new hidalgotfg***. En nuestro caso el servidor no reconoce laravel como un comando, por lo que procedemos a indicar el camino del ejecutable desde Composer ejecutando: ***$ sudo /var/www/home/user/.config/composer/vendor/bin/laravel new hidalgotfg.***

Como se puede ver en la Figura 3-2, para crear el proyecto de Laravel deben descargarse e instalarse varios paquetes. Este proceso puede llevar un tiempo, una vez finalice, el terminal nos avisará con un mensaje en el terminal.

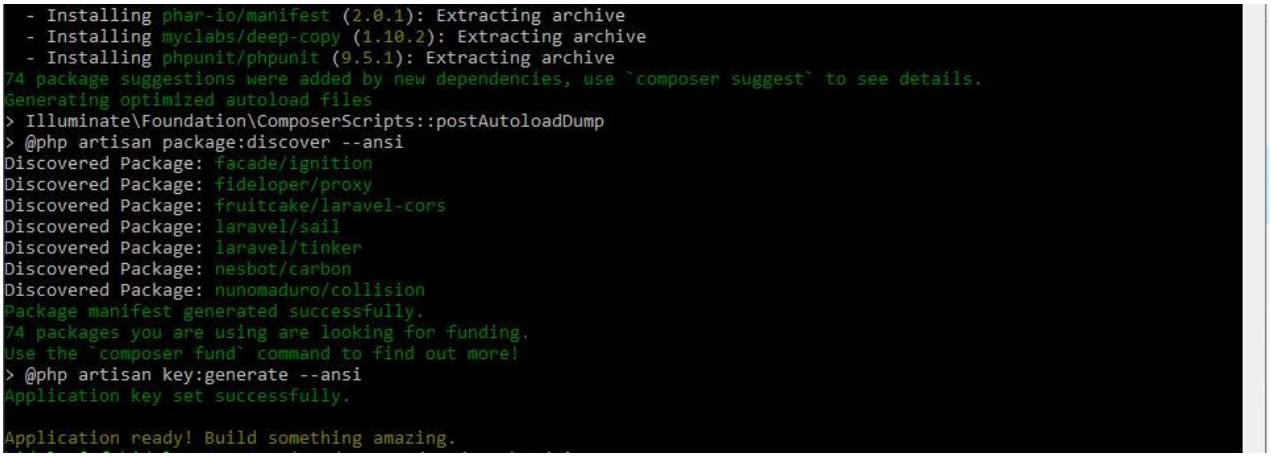


Figura 3-2 Descarga e instalación de paquetes Laravel (elaboración propia)

Ahora que hemos creado en nuestra carpeta del servidor un nuevo directorio con la estructura de Laravel, ***hidalgotfg***, se comprueba adonde apunta el directorio de Apache y hacemos que apunte al directorio Laravel que acabamos de crear. De esta forma, cuando Apache atienda a peticiones HTTP acudirá al directorio ***hidalgotfg***. Para ello, mediante un editor de texto cambiamos el directorio donde Apache buscará los recursos: ***$ sudo nano /etc/apache2/sites-available/000-default.conf***.

Acto seguido reiniciamos el servidor con ***$ sudo service apache2 restart*** y comprobamos los permisos del directorio con ***$ ls –la*** para que nos muestre todos los ficheros y directorios en formato extendido. Así vemos que ***hidalgotfg*** pertenece al ***root***, cuestión que debemos modificar para poder operar con el proyecto que hemos creado. Este paso es necesario para poder sincronizar el proyecto con Eclipse, como veremos más adelante.

Para cambiar el propietario del directorio ***hidalgotfg*** usamos el comando ***$ sudo chown –R www-data:www-data hidalgotfg*** [57]. De esta forma, el nuevo propietario es el usuario asociado al servidor web.

Por último, dado que vamos a trabajar en remoto, se debe crear un link simbólico desde donde Apache busca el directorio de Laravel hasta la carpeta de usuario en el servidor. Para ello haremos: ***$ ln –s /var/www/hidalgotfg /home/hidalgo/hidalgotfg***. Todos estos pasos los podemos ver ejecutados en la Figura 3-3.

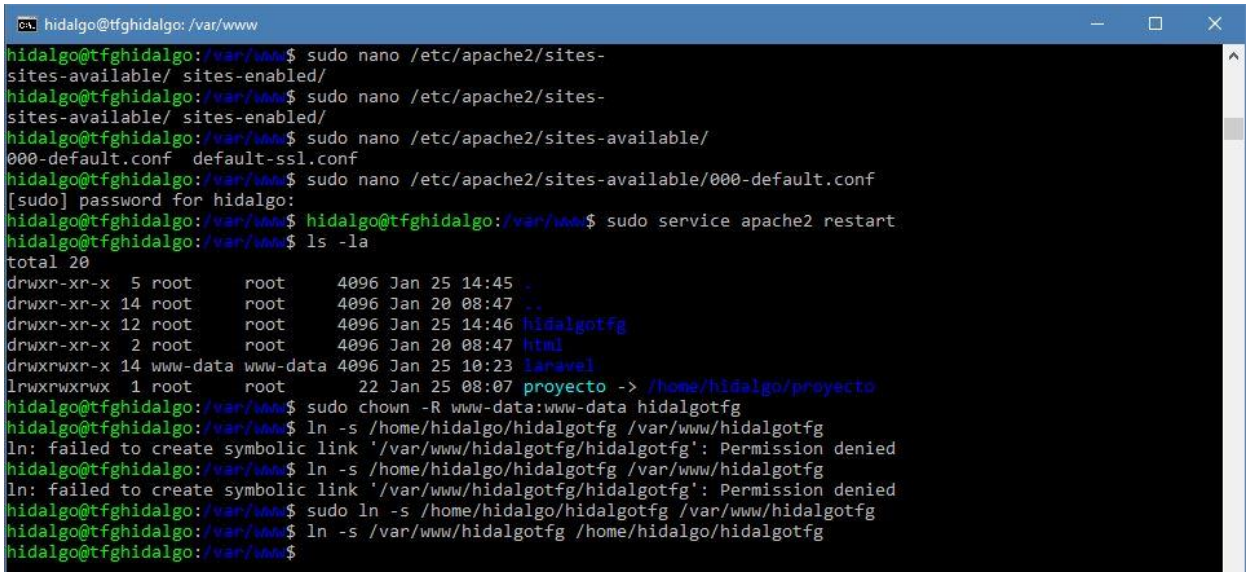


Figura 3-3 Operaciones para sincronizar el proyecto (elaboración propia)

Una vez hemos instalado Laravel con éxito, procedemos a descargar un entorno de desarrollo para trabajar más cómodos desde un equipo local con Windows 10 como sistema operativo. En este caso, se ha optado por Eclipse IDE [58] para Windows. Dado que Eclipse necesita Java [59] y el ordenador no contaba con Java, también se ha descargado Java. A continuación, se procederá a crear un nuevo proyecto en Eclipse desde el terminal local pero sincronizado con el servidor Linux asignado para este TFG en el que se encuentra Laravel.

De esta forma, podremos trabajar en remoto de una forma más cómoda. Para ello, se procede como se muestra en los pasos de la Figura 3-4. Primero, se define el nombre del proyecto PHP sincronizado, que para este caso será el mismo que el de nuestro directorio. Después, se crea una nueva conexión y se especifican las características: nombramos la conexión, añadimos la IP (193.146.212.191), el usuario y la contraseña. Por último, se indica el directorio del servidor donde se ubica el proyecto Laravel que creamos anteriormente.

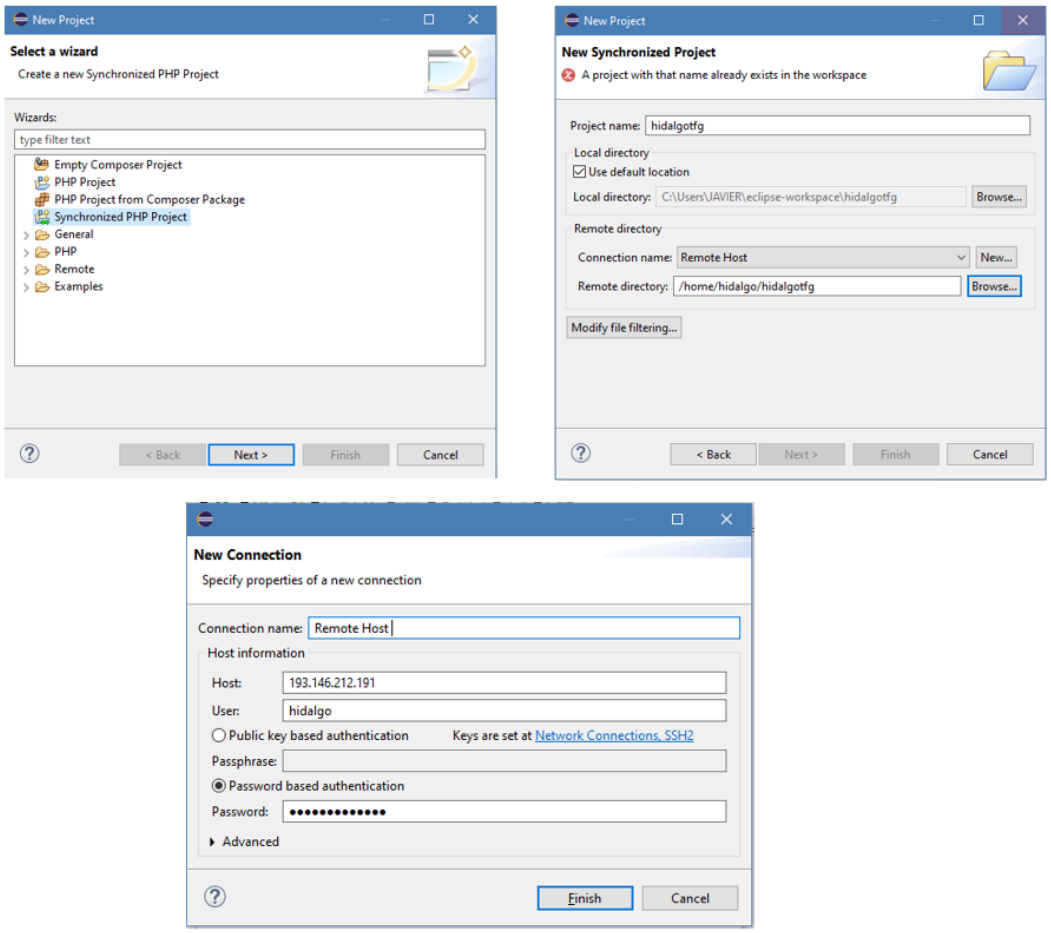


Figura 3-4 Nuevo proyecto sincronizado con el servidor mediante ssh (elaboración propia)

Al finalizar, se obtiene la misma estructura de directorios y archivos que emplea Laravel en el servidor remoto.

3.3 Diseño preliminar de la página web

Para el diseño de la página web, se propone la estructura de la Figura 3-5. La web contará con un sistema de registro de usuarios y un menú principal desde el que se gestionará la navegación por la misma.

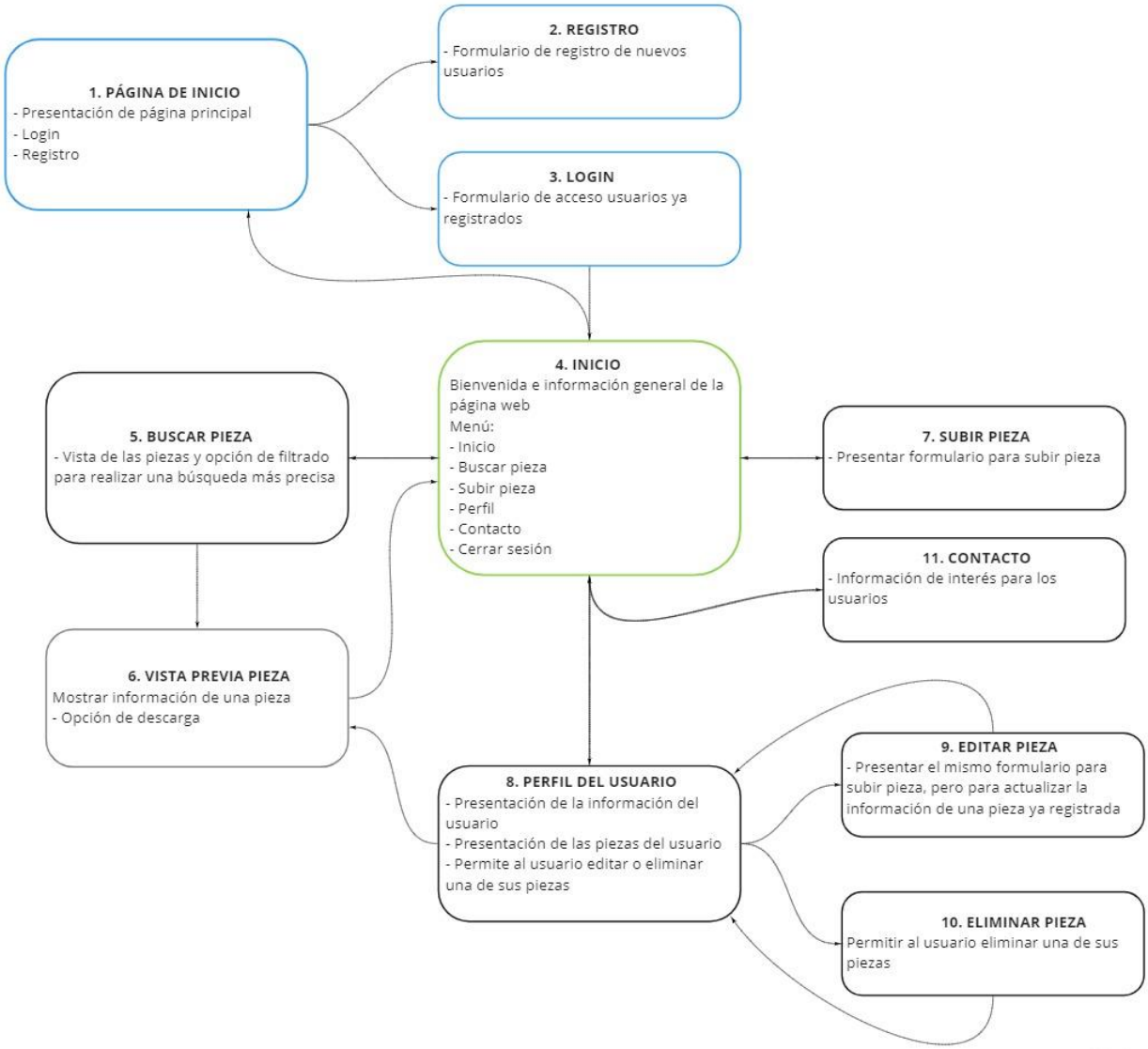


Figura 3-5 Flujo de navegación entre diferentes funcionalidades de la aplicación web (elaboración propia)

3.4 Registro y autenticación de usuarios

Una de las ventajas de emplear un *framework* como Laravel es la cantidad de documentación y paquetes que los desarrolladores y la comunidad de usuarios mantienen actualizados. En cuanto a la autenticación y el registro de usuarios en una página web, Laravel ofrece diferentes paquetes para hacerle el código más sencillo y fluido al programador. Por supuesto, estos paquetes son de código abierto, por lo que si el programador quisiese modificar una ruta, vista, función o valor, es libre de hacerlo.

Es conveniente destacar que trabajamos con la versión de Laravel 8.24.0, pues de ahora en adelante se hará referencia a documentación oficial publicada en la página de Laravel [60] y esta documentación se organiza según la versión de Laravel con la que trabajemos. Además, dependiendo de la versión, algunos comandos y procedimientos pueden variar.

Son varias las razones por las que se ha seleccionado Breeze para las funcionalidades de registro, autenticación de usuarios y gestión de contraseñas. La primera es que es un paquete desarrollado por Taylor Otwell, lo cual siempre añade cierta confianza pues, como vimos en el capítulo 2, Otwell es el desarrollador de Laravel. La siguiente es que es un paquete que nos va a proporcionar las rutas y vistas para realizar las funciones de: registro, login, logout, restablecimiento de contraseña, confirmación de

email y autenticación de dos factores. Lo bueno de ser de código abierto es que todas estas funciones se pueden modificar o suprimir si no nos interesan o cargan mucho el código.

Recurriendo a la documentación de Laravel [61], obtendremos los pasos a seguir en la instalación de Breeze.

El primer paso es conectarse al servidor mediante ssh y acceder al directorio donde se encuentra el proyecto, en nuestro caso: ***$ cd /var/www/hidalgotfg***.

Ahora se descargan las dependencias de Laravel Breeze usando Composer. Para ello se ejecuta: ***$ sudo composer require laravel/breeze –dev***

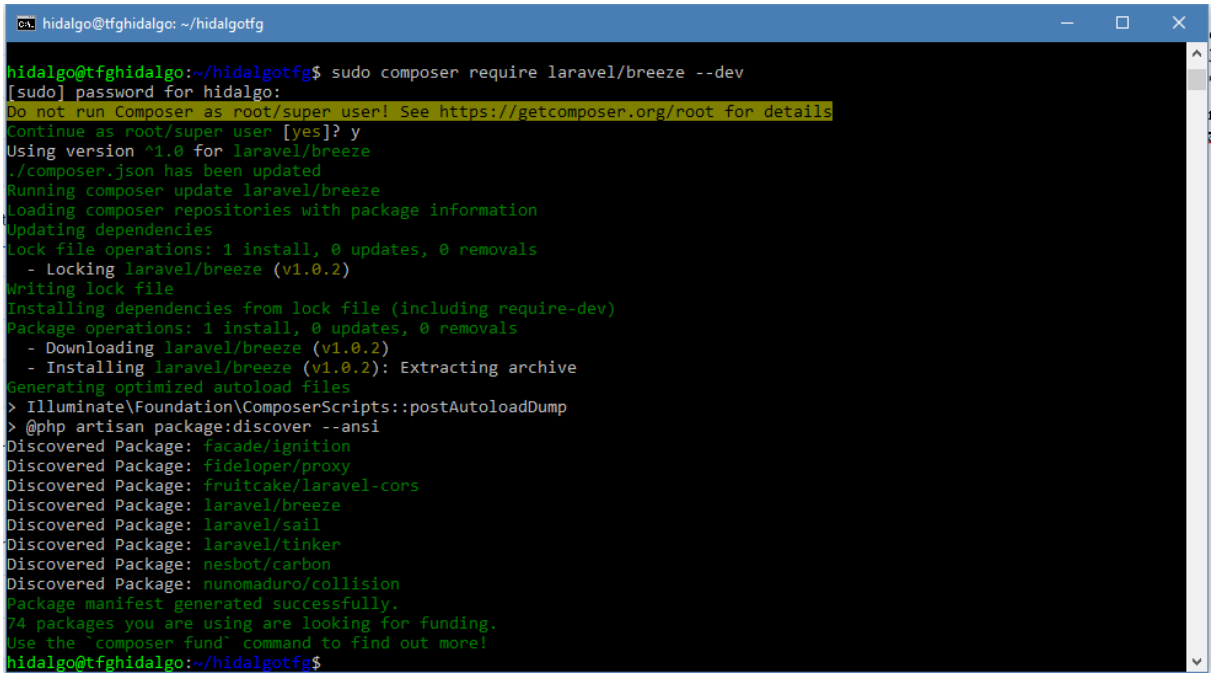


Figura 3-6 Descarga de Breeze (elaboración propia)

Como se puede ver en la Figura 3-6, Composer ha descargado el paquete Laravel Breeze. Ahora mediante el comando ***$ php artisan breeze:install***, se instalarán todas las dependencias de este paquete en el proyecto para que se generen las autorizaciones, vistas, rutas y controles en el proyecto ***hidalgotfg***.

Tras la instalación, el sistema sugiere ejecutar los comandos ***$ npm install && npm run dev*** para construir los recursos en el proyecto. Al ejecutar el comando ***npm,*** se advierte que el comando no se puede encontrar, pero se puede instalar, por lo que se realiza dicha instalación ejecutando ***$ sudo apt install npm***. Estos comandos los podemos ver ejecutados en el terminal de comandos en la Figura 3-7.

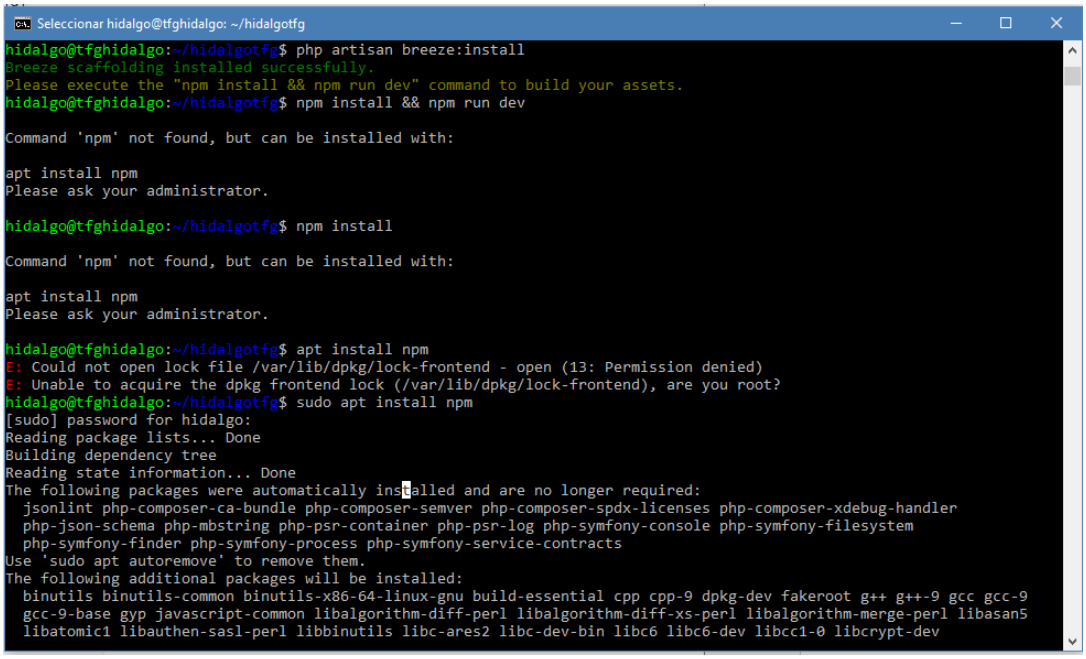


Figura 3-7 Instalación necesaria de *npm* (elaboración propia)

Una vez instalado ***npm***, volvemos a ejecutar ***$ php artisan breeze:install*** , acto seguido ***$ npm install*** y seguidamente ***npm run dev***, tal y como se muestra en la Figura 3-8.

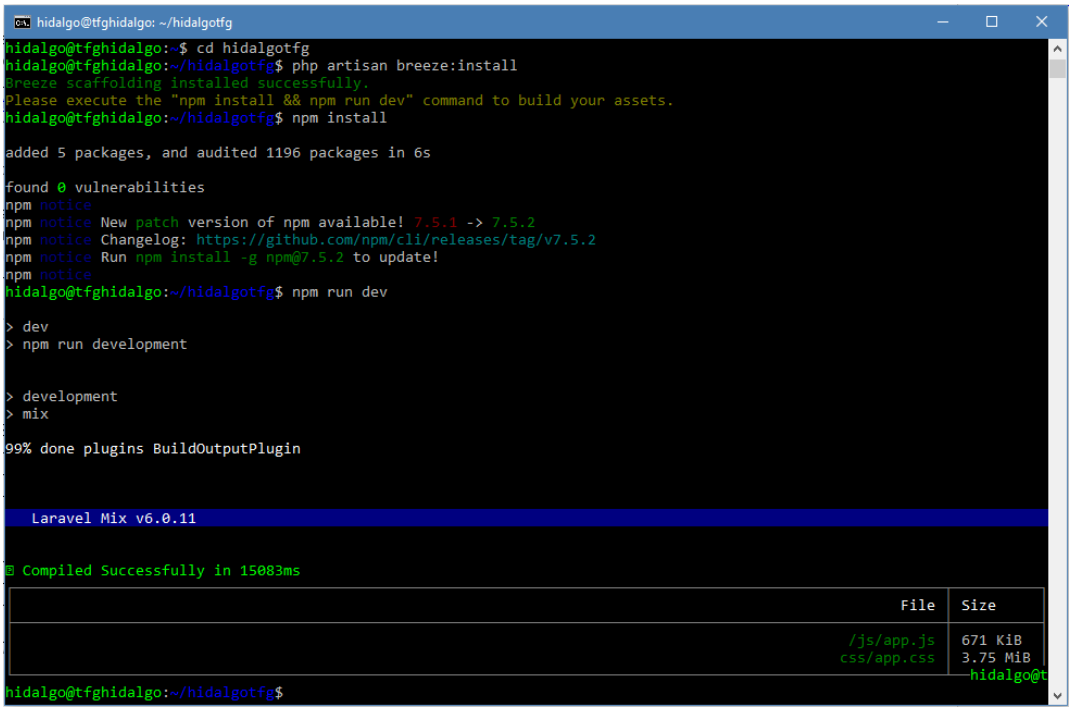


Figura 3-8 Finaliza la instalación de Breeze con npm (elaboración propia)

En el servidor ya existe la base de datos con el nombre ***tfg***. El siguiente paso, como se ve en la Figura 3-9, es migrar la base de datos ***tfg*** creada con MySQL. Para ello, primero debemos modificar los parametros del archivo *.env* ejecutando ***sudo nano .env*** en el directorio ***hidalgotfg***. En este archivo cambiaremos los siguientes parámetros: ***DB_DATABES= tfg*** y ***DB_USERNAME= hidalgo*** y ***DB_PASSWORD=*** la que corresponda. Una vez realizados los cambios, estaremos listos para realizar

la migración mediante ***$ php artisan migrate***. Con este comando, se crean las tablas y los modelos en MySQL necesarios para trabajar con Eloquent.

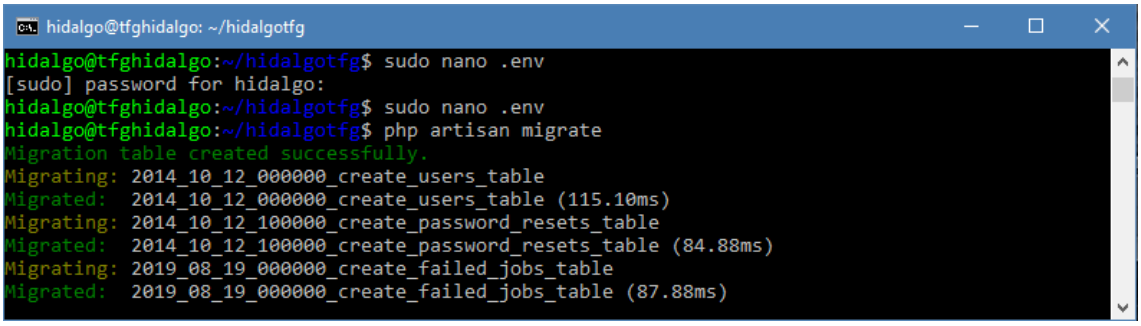


Figura 3-9 Modificación de *.env* y migración de la base de datos (elaboración propia)

A continuación, se guarda y se sincronizan los datos con el proyecto en local desde Eclipse. Así se sincroniza el trabajo realizado para instalar Breeze en el proyecto que se encuentra en el servidor, con el proyecto que se encuentra en local, actualizándose con las nuevas rutas, vistas y controladores que aporta el paquete Breeze.

3.5 Gestión de piezas

3.5.1 Diseño de la tabla de datos

En este apartado se aborda el problema relacionado con el almacén de ficheros .stl, que es el principal objetivo de este TFG. La solución que se propone para este problema es el desarrollo de una base de datos con una tabla en la que se pueda almacenar la información de los ficheros o, en su defecto, una dirección de descarga dentro de nuestro servidor. En la Tabla 3-1 se muestran los campos que el usuario deberá completar para realizar el registro de una pieza en la web, así como el tipo de información que contendrá cada atributo.

Pieza	
Atributo	Tipo de variable
$ id	Numérico (clave primaria)
NOC	Numérico
Nombre de la pieza	Texto
Autor	Texto
Categoría	Texto
Descripción	Texto
Software de diseño	Texto
Foto	Texto

Tabla 3-1 Diseño de la tabla *Pieza*

Dado que anteriormente relacionamos la base de datos ***tfg*** con nuestro proyecto para realizar los registros de los usuarios, no será necesario repetir este paso. Por lo tanto, el siguiente paso consistirá en crear una tabla para almacenar los ficheros de los diseños. A esta tabla la llamaremos *Pieza*. Para ello, desde el servidor nos colocaremos en la carpeta de nuestro proyecto ***hidalgotfg***. Ahora ejecutaremos ***$ php artisan make:model Pieza -m***. Este comando realiza la migración del modelo a la vez que crea la tabla, por lo tanto ahora en el proyecto existen dos archivos nuevos: en la carpeta *app/Models* se creará el modelo y en la carpeta *database/migrations*, se creará el archivo en el que configura el diseño de la tabla, donde le indicaremos las propiedades de la tabla.

Es conveniente aclarar que, al ejecutar el comando *artisan make:model*, nosotros introducimos el nombre como *Pieza*, y Laravel crea el modelo con el nombre de *Pieza*, pero el fichero donde se diseña la tabla se genera como *create_piezas_table.php*, con *piezas* en plural y minúscula.

La tabla almacenará la mayor parte de la información, tal y como podemos ver en la Tabla 3-1. Al analizar el contenido del fichero *create_piezas_table.php* podemos ver que tenemos una estructura predefinida para la tabla a través de un método *up( )*, por lo que procederemos a añadir los campos que se consideran necesarios, como se puede observar en el Fragmento de código 3-1.

Fragmento de código 3-1 Definición de la tabla *piezas*

```
public function up()
{
    Schema::create('piezas', function (Blueprint $table) {
        $table->id();
        $table->bigInteger('NOC');
        $table->string('NombrePieza');
        $table->string('Autor');
        $table->string('Categoria');
        $table->mediumText('Descripcion');
        $table->string('Software');
        $table->string('Foto')->nullable();
        $table->string('Pieza');
        $table->timestamps();
    });
}
```

Asociado a esta tabla se define el modelo *Pieza*, generado anteriormente, donde establecemos el nombre de la tabla de donde tomamos los datos, la clave primaria y el resto de las columnas de la tabla. Dado que cada pieza tiene que ser semejante a la tabla *piezas*, se debe indicar en el modelo, tal y como se muestra en el Fragmento de código 3-2.

Fragmento de código 3-2 Definición del modelo

```
class Pieza extends Model
{
use HasFactory;
protected $table = 'piezas';
protected $primarykey = 'id';

protected $fillable = [  'NOC',
                         'NombrePieza',
                         'Autor',
                         'Categoria',
                         'Descripcion',
                         'Software',
                         'Foto',
                         'Pieza'
                         ];
}
```

3.5.2 Controlador para la base de datos

Tras haber creado la tabla, el siguiente paso consiste en llenarla de datos y poder hacer consultas en ella. Laravel ofrece una herramienta para poblar las tablas: los *Seeders*. Los *Seeders* permiten poblar las bases de datos con entradas falsas y poner a prueba la estructura de la base de datos, pero no se realizará esta tarea mediante el empleo de *Seeders*.

Lo que haremos para probar el funcionamiento de la base de datos será introducir nosotros a través de la web los registros mediante operaciones CRUD: *Create, Retrieve, Update, Delete*. Para ello, el primer paso será crear un controlador para gestionar estas funciones. El comando para crear cualquier controlador es *php artisan make:controller* "nombre del controlador", pero dado que Laravel ofrece

algunas funciones ya definidas, podemos generar el controlador con el comando anterior para administrar las operaciones tipo CRUD. Para ello le pasamos el parámetro ***–resource*** para indicarle que queremos que cree el controlador para operaciones CRUD. Por tanto, en el terminal se ejecutará el siguiente comando: ***$ php artisan make:controller PiezaController –resource.***

Como se ha comentado, al ejecutar ***–resource*** se crea un controlador con una estructura ya definida para operaciones CRUD. La Tabla 3-2 muestra las acciones que el controlador que acabamos de crear manejará de forma predefinida.

Método	URL	Acción	Nombre de la ruta
GET	/ pieza	*Index*	*pieza.index*
GET	/ pieza /create	*Create*	*pieza.create*
POST	/ pieza	*Store*	*pieza.store*
GET	/ pieza /{ id }	*Show*	*pieza.show*
GET	/ pieza /{ id }/edit	*Edit*	*pieza.edit*
PUT/PATCH	/ pieza /{ id }	*Update*	*pieza.update*
DELETE	/ pieza /{ id }	*Destroy*	*pieza.destroy*

Tabla 3-2 Acciones manejadas por *PiezaController*

En este controlador se definirá el código que se seguirá para aplicar la lógica y, según las peticiones que le lleguen por parte de los usuarios, retornará distintas vistas. El controlador también se relacionará con el modelo *Pieza*.

Laravel ofrece un fichero para definir todas las rutas que forman un proyecto. Este archivo se denomina *web.php*. Para acceder al controlador se define la siguiente ruta en dicho fichero: `Route::resource('/pieza', PiezaController::class);.`

3.5.3 Nuevo registro de una pieza

Para crear un nuevo registro en la base de datos, el usuario que desee subir una pieza deberá rellenar los campos de la tabla *pieza*. Para ello, se define un formulario en una plantilla que se mostrará cuando acceda a la URL */pieza/create*. Por ello, el primer paso será crear una plantilla, a la que denominaremos *form.blade.php*. En ella el usuario podrá introducir los siguientes datos: NOC, el nombre de la pieza, la categoría, la descripción de la pieza, el software de diseño empleado, la foto y el archivo.

Para evitar que un usuario avance y envíe un formulario incompleto, se añade a los campos del formulario la etiqueta de *required.* Así, cada vez que un usuario trate de enviar el formulario con algún campo vacío, este le mandará un mensaje requiriendo que complete el campo correspondiente. Además, para evitar errores a la hora de introducir el NOC, se han añadido dos etiquetas de mínimo y máximo valor permitido, de tal forma que cada vez que el usuario introduzca el NOC, el sistema compruebe que se ha introducido correctamente con un mínimo de trece dígitos. Finalmente, cuando el usuario pulse sobre el botón de guardar, el formulario enviará a través de la ruta */guardar*, toda esta información al controlador *PiezaController* mediante el método POST.

La información debe trasladarse a la función *store* del controlador *PiezaController* para ser almacenada. Por ello, en el archivo de rutas *web.php* le indicaremos que, cuando reciba la ruta */guardar*, ejecute la función *store*:

```
Route::post ('/guardar',[PiezaController::class,'store']);
```

A continuación, el controlador ejecutará la función *store*, que recibe los datos procedentes del formulario mediante un *request*, luego los asignará a una variable con los mismos parámetros que nuestro modelo *Pieza* y, por último, almacenará en la base de datos este modelo. En el Fragmento de código 3-3 se puede ver de forma detallada el proceso.

Fragmento de código 3-3 Función empleada para almacenar información en la base de datos

```
    public function store(Request $request)
    {
       $pieza=new Pieza();
       $user=auth()->user();

       $pieza->NOC=$request->get('noc');
       $pieza->NombrePieza=$request->get('nombre');
       $pieza->Autor=$user->email;
       $pieza->Categoria=$request->get('categoria');
       $pieza->Descripcion=$request->get('descripcion');
       $pieza->Software=$request->get('software');

       $pathfoto=$request->file('photo')->store('public/fotopiezas');
       $pieza->Foto=$pathfoto;

       $extension = $request->file('file')->getClientOriginalExtension();
       if($extension != "stl"){
           echo "formato con extensi&oacute;n no autorizado, solo ficheros con
extensi&oacute;n .stl";
           return view("form");
       }
       else{
           $filename = uniqid().'.'.$extension;
           $pathfichero=$request->file('file')->storeAs('public/piezas',$filename);

           $pieza->Pieza=$pathfichero;
           $pieza->save();

           return redirect()->route('perfil');
       }
    }
```

Primero se define un objeto de tipo *Pieza*, llamado *$pieza*, que tenga los mismos atributos que el modelo con el que estamos trabajando. A continuación, se asignará a cada atributo de la variable *$pieza* aquella información que el controlador haya recibido y almacenado en la variable *$request*. Por tanto, es importante ser precisos en la asignación, se debe asignar a cada atributo de *$pieza* lo que la variable *$request* obtenga de cada campo definido en el formulario que el usuario rellena al subir una pieza.

Merecen especial mención los campos de autor, foto y fichero. Para evitar malas prácticas en Internet (no repudio de las piezas subidas) y facilitar la búsqueda de los usuarios introduciendo el nombre del autor, al subir una pieza el nombre del autor debe completarse en la base de datos de forma automática. De esta forma, se impide a un usuario subir una pieza bajo el nombre de otra persona. Dado que cada usuario debe autenticarse para acceder a la web, la sesión queda registrada en el sistema. Laravel almacena la información del usuario en *auth( )-> user*, por lo que creando una variable y asignándole la información del usuario, se puede registrar el email del usuario que sube la pieza en la base de datos sin ser necesario introducir al usuario en el formulario de forma manual.

La imagen y el fichero .stl son dos archivos pero, como se mencionó anteriormente, en la base de datos se almacena para cada uno una cadena de caracteres que contiene la ruta en el sistema de ficheros donde se ubica el archivo. Para la imagen, se crea una variable en la que se almacena la dirección donde

el archivo (en este caso la imagen que se recibe) se guarda. Cabe destacar que al ser un recurso al que los usuarios podrán acceder, la carpeta debe ser de acceso público. A continuación se indica a la variable *$pieza* la ruta.

Mediante una estructura *if/else* se asegura que el usuario únicamente sube a la web archivos con la extensión .stl. Es conveniente aclarar que Laravel, cuando sube un archivo, emplea la función que se presenta en el Fragmento de código 3-4. Sin embargo, este procedimiento no se puede utilizar para el fichero que almacena la pieza. Esto es debido a que Laravel por defecto intenta identificar el tipo MIME del fichero que se está subiendo al servidor, pero los ficheros de tipos no reconocidos (como es el caso) los almacena sin extensión.

Fragmento de código 3-4 Función empleada por Laravel para cargar un archivo

```
public function hashName($path = null)
{
   if ($path) {
   $path = rtrim($path, '/').'/';
   }
   $hash = $this->hashName ?: $this->hashName = Str::random(40);

return $path.$hash.'.'.$this->guessExtension();
// $this->guessExtension()..... this is what returns .txt  on your text file
}
```

Como se puede ver en el Fragmento de código 3-4, perteneciente a las librerías de Laravel para subir ficheros, se asigna un nombre aleatorio de 40 caracteres a los archivos y, en el caso de la extensión, si no es capaz de reconocer el tipo, asigna por defecto una extensión tipo .txt.

Dado que los formatos de diseño 3D y fabricación aditiva pueden ser desconocidos para Laravel, el modo de atajar este problema será obtener la extensión original del archivo y almacenarlo en una variable. Acto seguido se crea un nombre compuesto del fichero que se guarda en el servidor. De esta forma nos aseguramos que en la descarga se mantenga la extensión original del fichero. Por último, al igual que con las fotos, se asigna a la variable *$pieza* la ruta hasta el archivo.

3.5.4 Búsqueda de piezas

La aplicación web debe permitir a los usuarios navegar entre los registros existentes, acotar la búsqueda mediante un filtro y permitir la descarga de ficheros. A continuación, se expondrán los pasos seguidos para: listar las entradas existentes de cada pieza, realizar una búsqueda más específica aplicando filtros y, por último, la descarga de los archivos.

Para desarrollar estas funciones trabajaremos con el archivo *web.php*, donde se definen las rutas que usará la aplicación; el archivo controlador *PiezaController;* y, adicionalmente, generaremos dos plantillas nuevas: *display.blade.php* y *ver.blade.php*.

El primer paso consistirá en elaborar la plantilla a la que accederá el usuario cuando desee realizar una búsqueda. Esta plantilla debe contener un formulario que le permita al usuario mediante diferentes campos introducir los parámetros para realizar la búsqueda. Se ha decidido permitir la búsqueda por el NOC, por ser un número identificativo único para cada pieza, el cual debería ser conocido por un servicio de aprovisionamiento en un barco. Adicionalmente, y por si se desconociese dicho NOC, se permite la búsqueda según el destino, el nombre asignado para la pieza y el autor. Debajo del formulario, se listarán en orden descendiente los archivos almacenados.

Cuando se realiza una petición a la URL */pieza*, se llama al método *index()* del controlador *PiezaController*, como queda reflejado en la Tabla 3-2. Este método debe ser el encargado de listar todas las piezas existentes en la base de datos. Para ello, se crea una variable en la que se almacenará la información de las piezas recuperadas de la base de datos. Después se debe indicar que la variable *pieza* debe quedar ordenada en orden descendiente según el *id* identificativo de cada una en la base de datos.

Adicionalmente podemos indicarle el número de registros que se mostrarán. Para pasar la información de los usuarios (nombre y correo electrónico) que han registrado las piezas, se crea la variable *$users*, a la que se le asignan todos los usuarios registrados en la aplicación web. Por último, esta información se debe pasar a la vista para que el usuario pueda visualizar el contenido de la tabla *piezas*.

En la plantilla *display* se mostrarán los registros existentes justo debajo del formulario. Como se ha comentado anteriormente, una de las ventajas que ofrece Laravel a la hora del diseño de plantillas es la posibilidad de incluir sentencias de PHP. Para mostrar los registros, se emplea un bucle *foreach* como se muestra en el Fragmento de código 3-5.

Fragmento de código 3-5 Bucle *foreach* para mostrar registros almacenados

```
@foreach ($pieza as $piezas)
       <tr>
       <td>{{$piezas->NOC}}</td>
       <td>{{$piezas->NombrePieza}}</td>
       <td>
              @foreach ($users as $user)
                 @if ($user->email == $piezas->Autor)
                      {{$user->name}}
                  @endif
              @endforeach
```

En Fragmento de código 3-5 se observa como el bucle recibe la varaible *$pieza* desde el controlador y muestra los campos en pantalla. Pero además, recibe la variable *$users* con todos los usuarios registrados, y mediante Eloquent, se busca donde el email de un usuario registrado sea igual al que existe almacenado en el campo *Autor* de la variable *$piezas* que se autocompletó al registrar la pieza. Cuando encuentre una coincidencia, muestra por pantalla el nombre del usuario. Este bucle se repite para todas las piezas.

El siguiente paso consiste en generar el código que le permita al usuario navegar a través de estos registros haciendo uso de un buscador que le permita filtrar y hacer una búsqueda más específica. Como se comentó anteriormente, en la plantilla se incluye un formulario que recoge del usuario el NOC, el destino, el nombre de la pieza y el autor. Cuando el usuario envía este formulario, se manda mediante *GET* la información al controlador, que ejecuta la lógica para realizar las consultas en la base de datos. Para esta operación definimos en el fichero de rutas *web.php* la ruta `Route::get('/search', [PiezaController::class, 'filtro'])`. Esta ruta se ejecuta cuando el formulario envía los datos, llamando al método *filtro* del controlador, el cual debemos definir. Podemos ver en el Fragmento de código 3-6 cómo se gestionan las búsquedas desde dicho método.

Fragmento de código 3-6 Función para realizar búsquedas en la base de datos

```
public function filtro(request $id){
   $NOC = $id->get('noc');
   $Categoria = $id->get('categoria');
   $Nombre = $id->get('nombre');
      if ($id->autor!="") {
        $user= User::where('name','LIKE', '%'.$id->get('autor').'%')->first();
        $Autor = $user->email;
      }
      else {
        $Autor = "";
      }
       $pieza=Pieza::where('NOC', 'LIKE', '%'.$NOC.'%')->where('Categoria', 'LIKE',
       '%'.$Categoria.'%')->where('NombrePieza', 'LIKE', '%'.$Nombre.'%')
       ->where('Autor', 'LIKE', '%'.$Autor.'%')->paginate(20);
       $users = User::all();
     return view ("display", compact ('pieza','users') );    }
```

El controlador recibe la información y mediante Eloquent y consultas filtradas con *where* recupera el listado filtrado de piezas. Para el caso del autor, antes de emplear *where* en las piezas registradas, se emplea una estructura *if/else.* Si recibe la órden de buscar mediante el nombre de un usuario, busca mediante Eloquent en los usuarios registrados la primera coincidencia y la almacena en la variable *$Autor*. Si no recibe nada, no hace nada. El resultado queda guardado en la variable *$piezas* que se le pasa a la vista *display.blade.php*, donde se retornan los resultados mediante el mismo bucle (Fragmento de código 3-5) que se usaba para listar las piezas existentes en la base de datos.

3.5.5 Previsualización de una pieza y descarga

Tras realizar el filtrado y obtener las que, a priori, son las piezas que el usuario busca, este debe poder acceder a más información. Por ello, el siguiente paso será agregar al listado un botón que se muestre junto a la entrada de cada pieza, cuya función será la de mostrar al usuario toda la información complementaria. De esta forma, el usuario podrá contrastar si está accediendo a la pieza que busca, guiándose por los campos que el autor completó al subir dicha pieza.

Se comenzará añadiendo el botón que reenviará al usuario a la ventana donde se mostrará toda la información de la pieza. Al igual que se implementó en el apartado anterior, para mostrar los registros se emplea un bucle tipo *foreach* que accede a la base de datos y presenta todos los resultados que se relacionen con la búsqueda. En el Fragmento de código 3-7 se muestra la forma en la que se incorpora este botón al bucle mediante un formulario.

Fragmento de código 3-7 Envio de una pieza mediante un formulario

```
<td><form action="/ver" method= 'GET'>
   <input type="hidden" value="{{$pieza->id}}" name="id">
   <input type="submit" value="Ver">
</form></td>
```

Este formulario se incorpora dentro del bucle *foreach*, de tal forma que cada vez que se ejecute un ciclo del bucle, la etiqueta de tipo input se corresponderán con una única entrada de la tabla *piezas.* Como se puede ver, existe únicamente un campo oculto correspondiente a cada pieza. Este campo identifica cada pieza por el *$id*, que se enviará al controlador mediante el método *GET*.

Para que el formulario pueda enviar esta información, debemos definir en *web.php* la ruta y para que se muestre, debe llamar al método *show* del controlador que gestiona las operaciones con las piezas. Por ello definiremos la ruta como `Route::get('/ver', [PiezaController:: class, 'show']);`

El siguiente paso es definir la lógica de la función *show* en el controlador. La función recibe la información mediante un *request*. En el método *show* se crea una variable con el nombre *$pieza* y para que pueda trabajar con los registros de la base de datos se asigna esta al modelo *Pieza( )*. Después se indica a Eloquent que busque en la base de datos la primera coincidencia con el *$id* que ha recibido del formulario mediante el *request*.

En el caso de la foto y el archivo, es conveniente recordar que lo que se almacenó en la base de datos no fueron los archivos propiamente. Estos archivos se almacenaron en una carpeta pública, de tal forma que en la base de datos lo que se guardaba era la ruta hasta el archivo ubicado en esas carpetas. Para hacer accesibles estos archivos para su posterior descarga, primero debemos crear un enlace simbólico que nos permita acceder desde la carpeta *public*, a la carpeta *storage*. Para ello, Laravel proporciona el siguiente comando: ***$ php artisan storage:link***, el cual se ejecutará desde el directorio en el que se encuentra el proyecto en el servidor. El siguiente paso será recuperar la dirección donde se almacenan los archivos y cambiar el nombre del directorio para que apunte a la ruta creada por el enlace simbólico.

Adicionalmente, para que un usuario pueda contactar con el autor de una de las piezas se muestra en esta vista el correo con el que se registró el usuario. Para ello, en el método *show* se define la variable *$user* y mediante Eloquent se busca el email en la tabla de usuarios que coincida con el almacenado en el campo *Autor* de la pieza.

Por último, se le pasa mediante un *return* la información de la pieza y el email del autor a una nueva vista que se denominará *ver*, donde se mostrará toda la información. Podemos ver la lógica que sigue el controlador en el Fragmento de código 3-8.

Fragmento de código 3-8 Recuperación de la información de una pieza en la base de datos

```
    public function show(request $request)
{
        $pieza=Pieza::where('id',$request->id)->first();
        $pieza->Foto=str_replace("public","storage",$pieza->Foto);
        $pieza->Pieza=str_replace("public","storage",$pieza->Pieza);
        $user= User::where('email','LIKE', '%'.$pieza->Autor.'%')->first();

     return view ("ver", compact ('pieza', 'user'));

}
```

En la vista *ver* se define una tabla para mostrar toda la información de la pieza y, en la parte inferior, un enlace para descargar la pieza mediante una etiqueta HTML: `<a href="{{$pieza->Pieza}}" download>Descargar</a>`.

3.6 Perfil del usuario

La página web debe contar con una vista que permita al usuario comprobar tanto su información como las piezas cuyos modelos ha publicado en la web. Por ello, en el menú, se ha definido un apartado reservado al perfil. Cuando el usuario accede a esta vista, se retorna la información que ingresa el usuario al registarse (nombre y correo electrónico, así como el número de usuario registrado en la plataforma) y las piezas que son de su propiedad y sobre las que tiene permiso de edición.

Para que el usuario pueda ver sus datos se ha generado un controlador con el nombre *UsuarioDatosController*. Este controlador es el encargado de recuperar los datos del usuario que está registrado, y a su vez, mostrar las piezas que le pertenecen. Para ello se ejecuta en el servidor el comando ***$ php artisan make:controller UsuarioDatosController –resource***. De esta forma se genera un nuevo controlador con todos los métodos necesarios para realizar operaciones de listado, registro, actualizar, etc.

Para listar la información del usuario, se trabajará en el método *index* del controlador. El primer paso es definir una variable a la que se le asignará la información contenida en la variable de sesión del usuario que se encuentra autenticado en la web en ese momento.

El siguiente paso consiste en crear una variable, *$piezas*, la cual contenga todas las piezas registradas. Dado que solo interesan las piezas del usuario, mediante Eloquent se filtran las piezas que pertenezcan únicamente al usuario. Eloquent rescata las piezas en las que coincida el autor mediante la búsqueda por email mediante el empleo del método *where*. De esta forma nos aseguramos que si existiesen dos usuarios con el mismo nombre, solo se muestren las piezas que realmente pertenecen al usuario que actualmente se encuentra haciendo uso de la aplicación web, ya que el campo email identifica de forma unívoca a los autores.

Por último, se pasa la información del usuario y las piezas que le corresponden a la vista *perfil.blade.php* para que el usuario pueda visualizar la información. El código empleado se puede observar en el Fragmento de código 3-9.

Fragmento de código 3-9 Método *index* del controlador de datos del usuario

```
public function index()
{
    $piezas= new Pieza();
    $user=auth()->user();
    $piezas=Pieza::orderBy('id', 'ASC')->paginate(10);
```

```
        $piezas= Pieza::where('Autor', 'LIKE', '%'.$user->email.'%')->get();
        return view ('perfil', compact ('user', 'piezas'));
    }
```

Tras pasar esta información a la vista denominada *perfil.blade.php*, el usuario podrá ver el número de usuario registrado que ocupa, el nombre bajo el que se registró y el correo con el que accede. Dado que es información que se muestra en la información de cada pieza de forma automática, es importante que el usuario sea consciente de esta información.

También aparecerá un listado con las piezas que le pertenecen. El usuario, como propietario, debe poder tener control sobre sus piezas, es decir, si quiere modificar algún campo de la información, volver a subir un archivo mejorado de una pieza existente o simplemente eliminar esa pieza de la página web, debe poder hacerlo.

Para ello, se ha introducido en la tabla que presenta las piezas del usuario ambas opciones: editar y eliminar una pieza. La forma en la que se van a gestionar estas operaciones será descrita en los siguientes apartados.

3.6.1 Editar una pieza

Para editar una pieza se han empleado los métodos *edit* y *update* del controlador *PiezaController* y se ha creado otra vista denominada *edit.blade.php*.

Como se puede ver en el Fragmento de código 3-10, cuando un usuario decide editar una pieza, el controlador recibe en el método *edit* el *id* de la pieza a través de la URL. Después, Eloquent compara en la base de datos el valor que acaba de recibir con el *id* de cada pieza y, cuando obtiene una coincidencia, recupera toda la información de la pieza. Después se le indica a la variable la ruta donde se encuentran almacenadas la imagen de la pieza y el archivo .stl. Por último, se le envía toda la información a la vista *edit*.

Fragmento de código 3-10 Función *edit* del controlador *PiezaController*

```
public function edit($id)
{
   $pieza= Pieza::find($id);
   $pieza->Foto=str_replace("public","storage",$pieza->Foto);
   $pieza->Pieza=str_replace("public","storage",$pieza->Pieza);
   return view ("edit", compact ('pieza'));
}
```

En la vista *edit.blade.php* se muestra un formulario similar al empleado para subir una pieza. La única diferencia es que este formulario se encuentra parcialmente cubierto con la información que ya existe sobre esa pieza, a excepción de la foto y del archivo .stl. Esto se debe a que, por razones de seguridad, HTML no permite dejar estos campos rellenos con anterioridad. Por último, este formulario llama al método *update* del controlador y envía la información mediante el método POST.

Tras completar todos los campos del formulario y guardar, el controlador recibe en el método *update* dos parámetros: uno con los datos de la pieza y otro con los datos que se van a modificar. La lógica que sigue el controlador para editar una pieza se puede ver en el Fragmento de código 3-11.

Fragmento de código 3-11 Función definida para actualizar la información de una pieza

```
public function update(Request $request, Pieza $pieza)
{
   $pieza->NOC=$request->noc;
   $pieza->NombrePieza=$request->nombre;
   $pieza->Categoria=$request->categoria;
   $pieza->Descripcion=$request->descripcion;
   $pieza->Software=$request->software;
```

```
        if ($request->file('photo')!=NULL) {
            $pieza->Foto=$request->file('photo')->store('public/fotopiezas');
        }


        if ($request->file('file')!=NULL) {
            $extension = $request->file('file')->getClientOriginalExtension();
            if($extension != "stl"){
              echo "<font color=red> Formato con extensi&oacute;n no autorizado, solo
ficheros con extensi&oacute;n .stl</font>";
              return view ("edit", compact ('pieza'));
             }
             else{
               $filename = uniqid().'.'.$extension;
               $pathfichero=$request->file('file')->storeAs('public/piezas',$filename);
               $pieza->Pieza=$pathfichero;
             }
         }
          $pieza->save();
          return redirect()->route('perfil');
    }
```

El método *update* recibe con un *request* lo que se encuentre cubierto en el formulario. Como se ha pasado al formulario la variable antes de invocar este método (ver Fragmento de código 3-10), en el código HTML se indicó que se debía presentar el formulario con los datos precubiertos (mediante la etiqueta *value*), por lo que, cuando el usuario guarde los cambios, estos datos precubiertos se enviarán al controlador que asignará esta información a la pieza y la guardará en la base de datos. Esto es posible dado que el *id* de la pieza se mantiene intacto.

Merecen especial mención la imagen y el archivo .stl, pues como se menciona anteriormente, estos datos no aparecen precubiertos por razones de seguridad de HTML, pero el usuario debe poder enviar el formulario sin necesidad de tener que volver a cargar estos datos. De lo contrario, sería un inconveniente, pues un usuario podría necesitar realizar una actualización y no poder disponer de la imagen o del fichero .stl.

Para solventar esta situación, se hace uso de una estructura *if/else*. Para la imagen, en el caso de que se reciba un archivo nuevo, se procede a guardarlo. Si no recibe nada, la variable ya cuenta con la imagen en el servidor y la ruta almacenada en la base de datos, por lo que no se realiza ninguna acción. Para el fichero .stl se realiza la misma acción que con la imagen, pero adicionalmente se añade otra estructura *if/else* que compruebe la validez de la extensión del archivo.

3.6.2 Eliminar una pieza

Para eliminar una pieza se ha hecho empleo de los métodos *eliminar* y *destroy* del controlador *PiezaController* y, adicionalmente, se ha creado una vista que servirá como control para asegurar que el usuario desea eliminar una pieza.

En el controlador se ha definido el Fragmento de código 3-12. Primero se crea una variable que contenga los campos del modelo *Pieza( )* y le ordenamos a Eloquent que busque el registro en la base de datos con el *id* que recibe de la vista y que identifica a la pieza que se va a destruir. Acto seguido, se le indica a la pieza la ruta hasta la imagen y el archivo .stl. Por último, se le pasa la variable *$pieza* a la vista que se ha creado como forma de control: *eliminar.blade.php*.

Fragmento de código 3-12 Funciones definidas para eliminar una pieza

```
public function eliminar($id){
   $pieza= new Pieza();
   $pieza= Pieza::find($id);
   $pieza->Foto=str_replace("public","storage",$pieza->Foto);
   $pieza->Pieza=str_replace("public","storage",$pieza->Pieza);
   return view ("eliminar", compact ('pieza'));
}

public function destroy(Pieza $pieza)
{
   $pieza->delete();
   return redirect()->route('perfil');
}
```

En la vista de control se crean dos formularios, cada uno compuesto únicamente por un botón: *Eliminar* y *Cancelar*. Cuando se selecciona *Eliminar*, se activa la función de eliminar una pieza descrita en el método *destroy* del controlador. En el Fragmento de código 3-12 se puede ver cómo el controlador recibe toda la información de la pieza y la eliminna de la base de datos. Cuando, por el contrario, el usuario selecciona *Cancelar*, se redirige al usuario de vuelta al perfil.

3.7 Apariencia de la aplicación web

Para dar la misma apariencia a todo el sitio web, se ha empleado el mecanismo que Laravel pone a disposición de los usuarios para gestionar las vistas. El motor de plantillas de Laravel, Blade, permite diseñar una plantilla inicial y usarla como modelo o base para el resto de vistas. Trabajar con Blade puede resultar cómodo especialmente si un componente de la plantilla se va a repetir en varias vistas, pues, de esta forma, el usuario no necesita integrar los mismos componentes en cada plantilla una y otra vez.

Para trabajar con Blade, es necesario conocer ciertas etiquetas propias como: *@yield*, *@extend* y *@section.* Mediante estas etiquetas, Blade llama a las vistas y las inserta según convenga en cada caso. A continuación, se procederá a definir el funcionamiento de las etiquetas empleadas y la forma en las que se aplican en el desarrollo de esta aplicación.

3.7.1 Menú

Dado que el menú y el encabezado se mostrarán en la mayor parte de nuestras vistas, se definirá la vista *menu.blade.php* como la plantilla sobre la que se trabajará. Para ello, en los lugares donde se quiera incluir algún elemento que haga referencia a una acción o muestre una vista diferente, se empleará la etiqueta *@yield(‘ ’)*, debiéndose introducir dentro del paréntesis el nombre que identifica a esa vista.

La forma de implementar la plantilla principal (o vista *menu.blade.php*) en el resto de vistas será mediante el empleo de la etiqueta *@extends(‘ ’)*, donde se debe indicar el nombre de la vista que se quiere emplear. De esta forma, Blade incorpora en la vista actual todo lo que estuviese definido en la vista que se usa como plantilla. Si bien es verdad que el estilo puede quedar definido en la plantilla principal y que puede definir el resto de plantillas, esto no debe presentar ningún problema, pues en cada vista se puede heredarse en el estilo propio y hacer que prevalezca sobre la plantilla principal que se está empleando.

Por último, en cada vista que emplee otra vista como plantilla, se debe delimitar el código que define cada vista y que se insertará en la etiqueta *@yield(‘ ’)* de la plantilla principal. Esto se realiza mediante las etiquetas *@section(‘ ’)* y *@endsection(‘ ’)*. Es decir, lo que quede definido dentro de estas etiquetas es lo que Blade mostrará al cargar cada vista, pero manteniendo intacto lo que se defina dentro de la plantilla principal. Se muestra en la Figura 3-10 un esquema de cómo funciona este sistema.

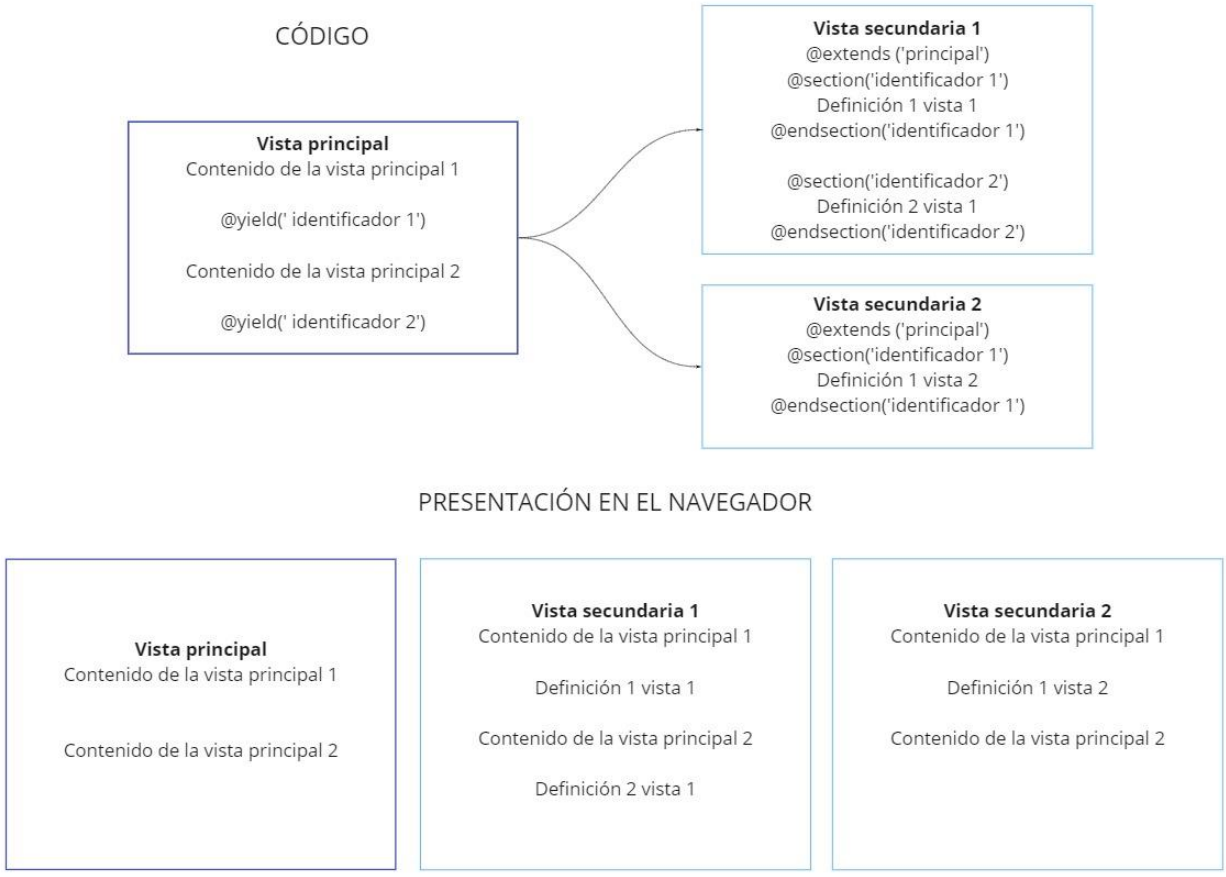


Figura 3-10 Esquema del empleo de Blade

De esta forma, en *menu.blade.php* se emplea la etiqueta *@yield('contenido')* para insertar cada vista. Y del mismo modo, en cada vista se emplean las etiquetas *@extends('menu')*, *@section('contenido')* y *@endsection('contenido')* para definir cada vista.

3.7.2 Vistas

Dado que cada vista se ha empleado para fines diferentes: buscar piezas, subir piezas, mostrar piezas y autenticación, se debe definir en cada una la presentación mediante el empleo de tablas, formularios y enlaces. Además, como se comenta en el apartado anterior, puede ser necesario el empleo de un estilo propio en cada página para diferenciar botones, enlaces, celdas, etc.

El estilo específico en cada vista se define antes del cuerpo de la página HTML, en la etiqueta *style* de cada documento. Ahí se definen clases mediante comandos CSS, se indica la posición, la alineación y el diseño gráfico de algunos elementos. De esta forma, cuando en el cuerpo se llama a una clase específica definida previamente, esta se muestra con un formato ya predefinido.

Tanto el diseño de las páginas como el CSS quedan recogidos en los anexos de este documento.

4 VALIDACIÓN Y PRUEBA

En este capítulo se expondrán los métodos de validación y prueba empleados para comprobar que la herramienta web gestiona de forma adecuada las piezas. En la validación, se evaluará que la aplicación realiza de forma correcta las acciones para las que se ha programado. Después, se comprobará que todas las acciones quedan registradas en la base de datos con la que trabaja esta aplicación.

4.1 Comprobaciones de funcionamiento de la web

4.1.1 Autenticación

Cuando cualquier usuario ingresa en la web accediendo a la dirección http://193.146.212.191/, accederá a la página de inicio mostrada en la Figura 4-1 donde encontrará los botones para acceder y para registrarse como nuevo usuario.

Figura 4-1 Página de inicio (elaboración propia)

Un usuario que quiera registrarse deberá completar los campos solicitados en la Figura 4-2. Estos datos quedan registrados en la base de datos como se verá más adelante. Los datos que el usuario proporciona serán los que luego se empleen para identificar al usuario como autor de una pieza.

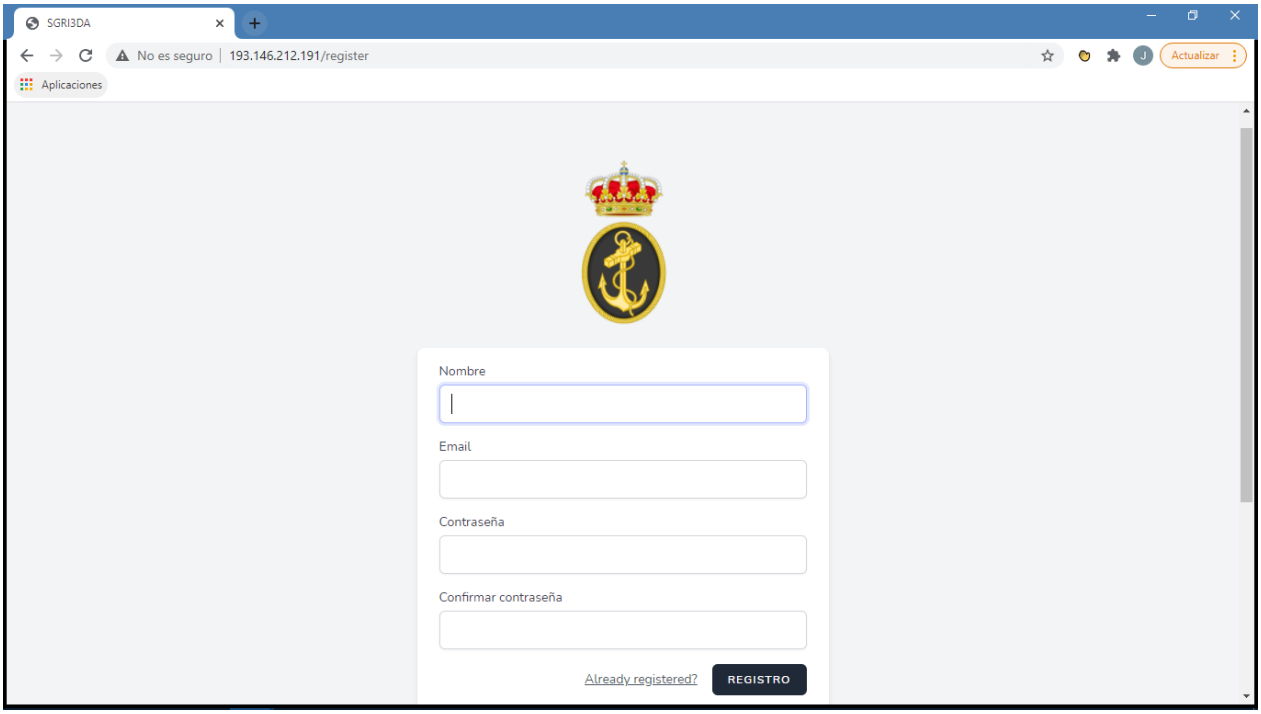


Figura 4-2 Registro de usuarios (elaboración propia)

Un usuario ya registrado accederá a la página mediante el botón de *Acceso*. Simplemente necesitará introducir los campos mostrados en la Figura 4-3: el correo con el que se registró y la contraseña. Laravel comprobará los datos introducidos y, de ser correctos, conducirá al usuario a la página principal de la aplicación web.

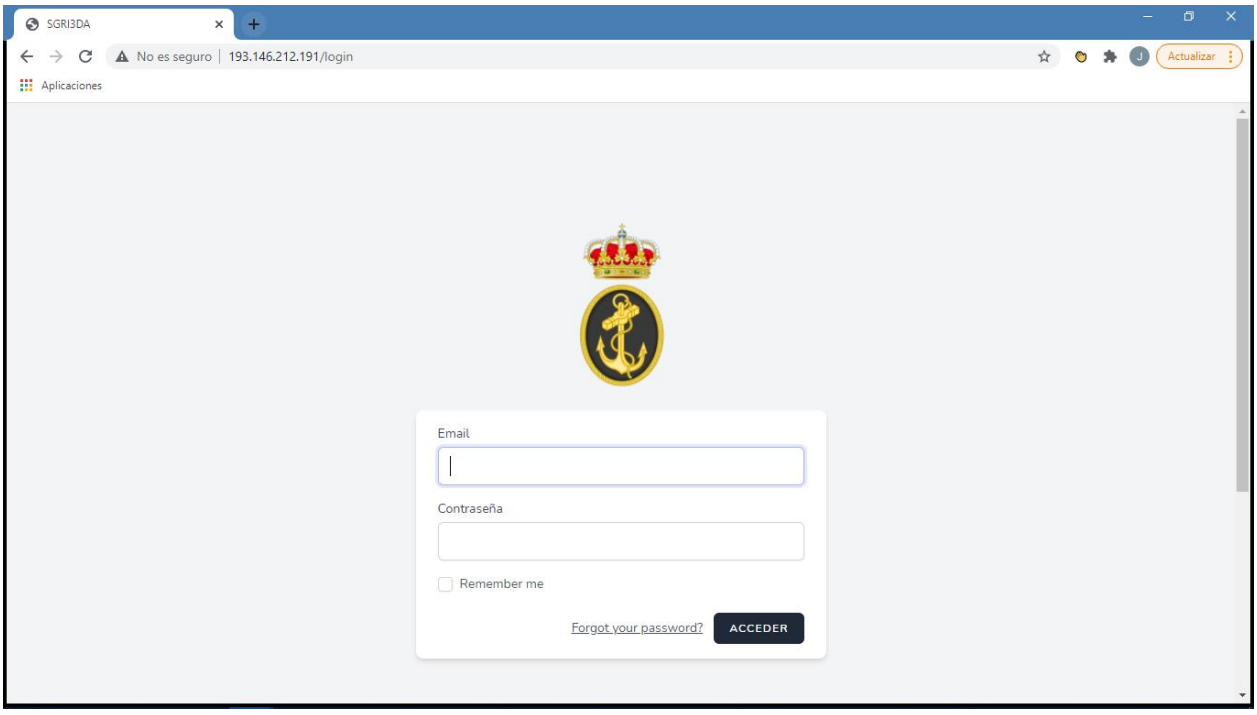


Figura 4-3 Inicio de sesión por un usuario ya registrado (elaboración propia)

Es importante mencionar que, debido a que esta página está destinada a compartir ficheros de repuestos para la Armada, alguna información podría ser sensible. Por ello, mediante el empleo de *middlewares* de autenticación, resulta imposible acceder a cualquier URL que emplea sin antes haberse identificado.

Por último, se puede dar el caso de que un usuario olvide su contraseña. Ante esta situación, el usuario deberá cambiarla, haciendo uso de la herramienta *"Forgot your password?"*. Como se ve en la Figura 4-4, se solicitará un correo electrónico para realizar el cambio de contraseña.

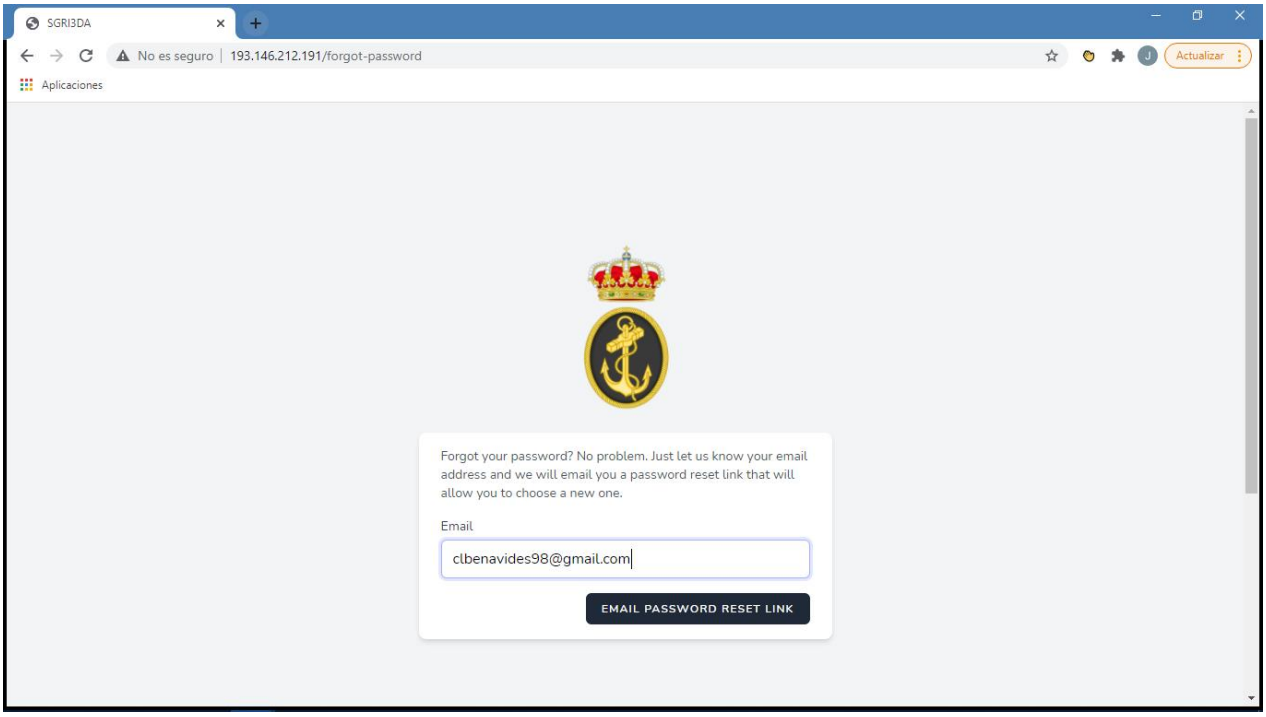


Figura 4-4 Recuperación de contraseña (elaboración propia)

Cuando el usuario acceda al correo que ha proporcionado, tendrá un email como el que se muestra en la Figura 4-5.

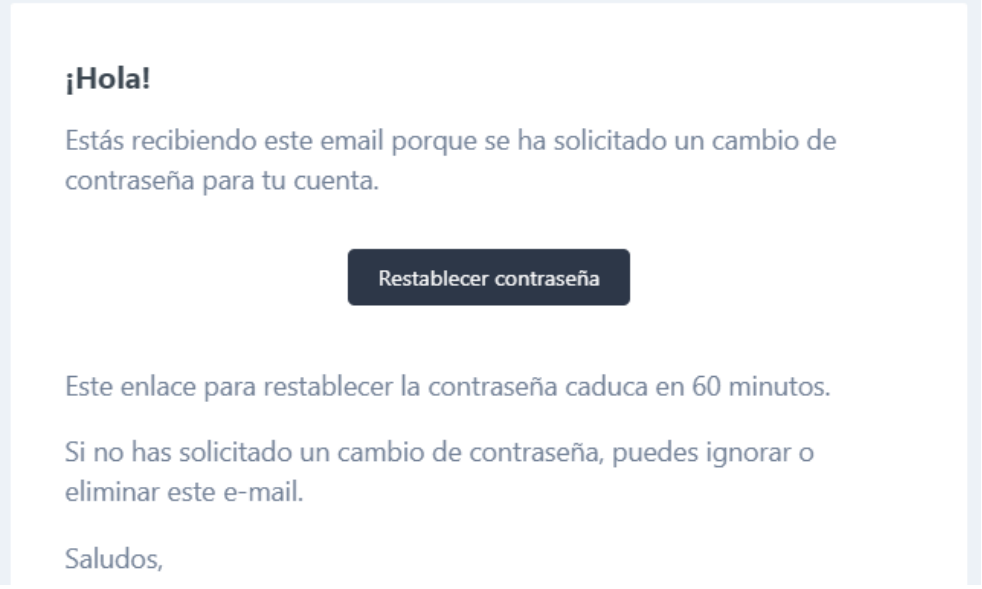


Figura 4-5 Email de solicitud de cambio de contraseña (elaboración propia)

Accediendo al botón de *Restablecer contraseña* que figura en el correo, la web permite cambiar la contraseña del usuario, tal y como se muestra en la Figura 4-6. Tras realizar el cambio de contraseña, la web devolverá al usuario a la página de inicio, donde deberá acceder como un usuario ya registrado empleando la nueva contraseña.

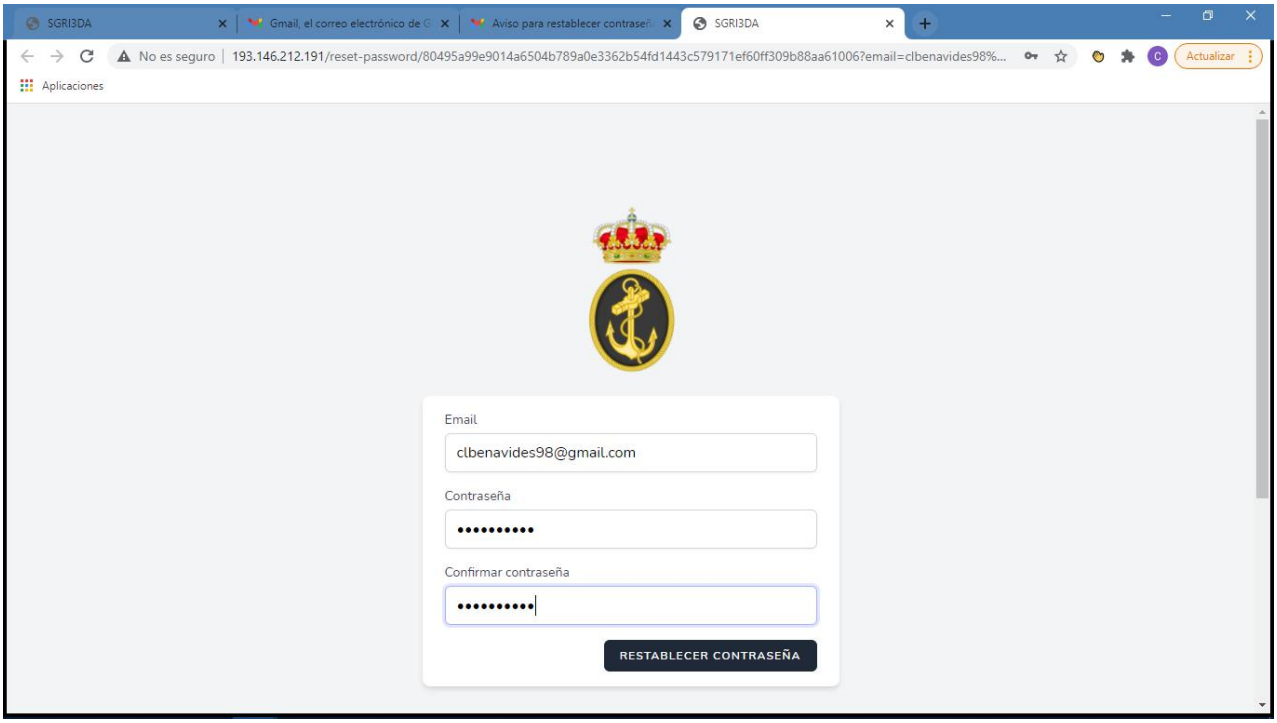


Figura 4-6 Actualización de contraseña (elaboración propia)

4.1.2 Navegación

Una vez registrado, ya en la página principal de la aplicación, el usuario tendrá acceso al menú para navegar por las distintas opciones que la web ofrece. Como se puede ver en la Figura 4-7, en esta página principal, además del menú, se le presenta al usuario un texto de bienvenida con información referente a la página web y dos vídeos relacionados con la funcionalidad de la impresión 3D.

Figura 4-7 Página principal (elaboración propia)

Dado que las funciones principales de gestión de piezas se verán más adelante, cabe destacar que accediendo a *Contacto* a través del menú, el usuario accede a información que le puede ser útil acerca de la catalogación, con quién debe ponerse en contacto en caso de dudas a la hora de catalogar una pieza. Se ofrece también información del Cuerpo de Ingenieros de la Armada por si se requiriesen diseños de mayor complejidad o escáneres 3D y, además, se ofrece la página de referencia de las impresoras 3D empleadas. Así mismo, también se ofrece información sobre el desarrollador de la página web.

Por último, cabe señalar que el botón para cerrar sesión se encuentra ubicado al final de este menú. Cuando el usuario cierre sesión, volverá a la página de inicio.

4.1.3 Búsqueda y filtrado de piezas

Cuando un usuario necesite buscar una pieza, tendrá a su disposición un listado con todas las piezas que existen registradas en la página, como se puede ver en la Figura 4-8. Para facilitar la búsqueda, se proporciona un filtro. Este filtro se compone por el NOC, que de conocerse debería mostrar todos los resultados de una única pieza de forma inequívoca; el nombre de la pieza que el autor le asignó al registrarla en la aplicación; la categoría, por si se quisiese hacer un filtrado por los destinos de un barco; y el autor, que es el usuario que registró la pieza en la aplicación.

Figura 4-8 Búsqueda de piezas (elaboración propia)

El sistema de filtrado permite varias opciones. Se pueden realizar búsquedas muy específicas, completando todos los campos. O se pueden completar los campos de forma más ambigua, por ejemplo, introduciendo un único dígito en el NOC o usando solo un nombre en el autor. Ante estas situaciones, la aplicación muestra todos los resultados que coincidan con el filtro de forma ordenada. Si solo se introduce un dígito en el NOC, mostrará todos los registros que contengan ese dígito; si solo se introduce un nombre, mostrará todas las piezas de los autores que respondan a ese nombre. La Figura 4-9 se corresponde con un filtrado introduciendo en el NOC el valor "1" y en el Autor el valor de "Javier". Dado que en la base de datos existen piezas previas a que se realizase el control de errores, la web nos muestra la Pieza 1, que tiene un NOC inferior a 13 dígitos y el usuario se corresponde con Javier. Esto no es incorrecto pues, como se comenta, el filtro está diseñado para que muestre todas las coincidencias.

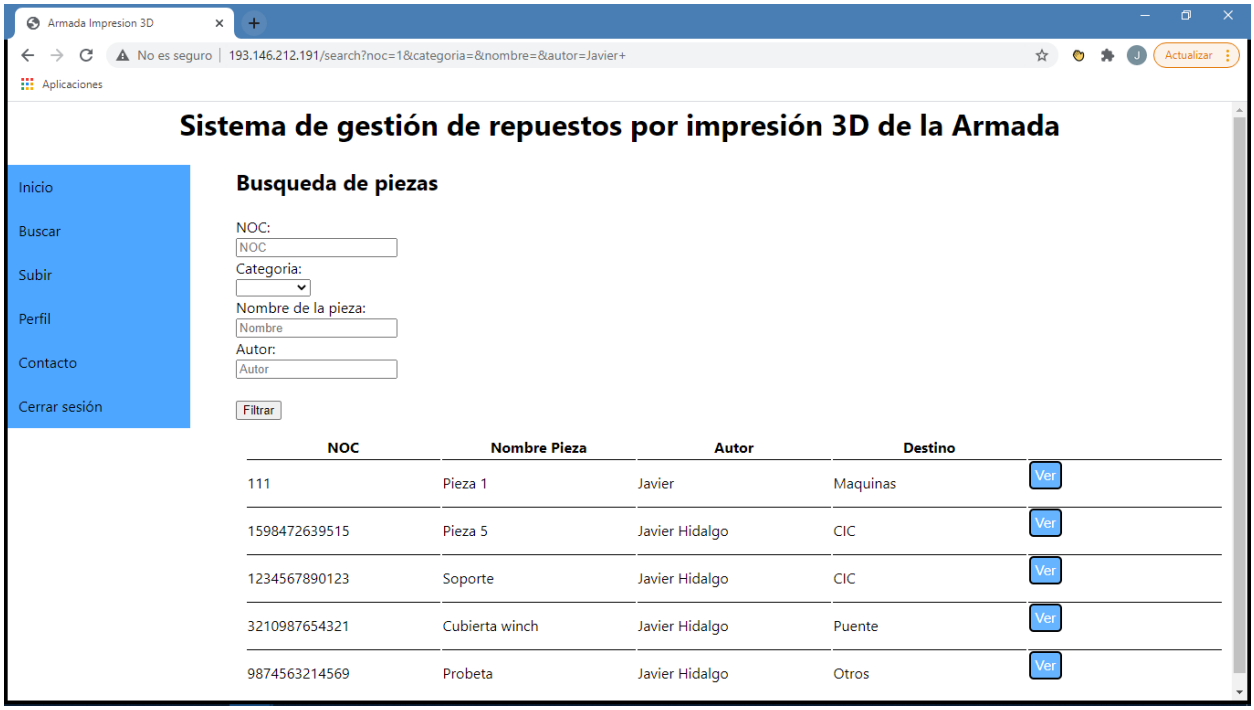


Figura 4-9 Búsqueda filtrada (elaboración propia)

4.1.4 Previsualización y descarga de piezas

Dado que la página irá creciendo y almacenará cada vez más piezas, es necesario que los usuarios puedan acceder a toda la información completa y no tengan que decidir qué pieza descargar guándose únicamente por los campos de la tabla mostrada en la Figura 4-8.

Cuando el usuario seleccione la opción de *Ver*, accederá a una vista en la que se le muestra toda la información relativa a una pieza a través de una tabla, como se puede apreciar en la Figura 4-10. Si la pieza fuese la que está buscando, podrá descargar el diseño seleccionando la opción de descarga situada en la parte inferior derecha de la tabla.

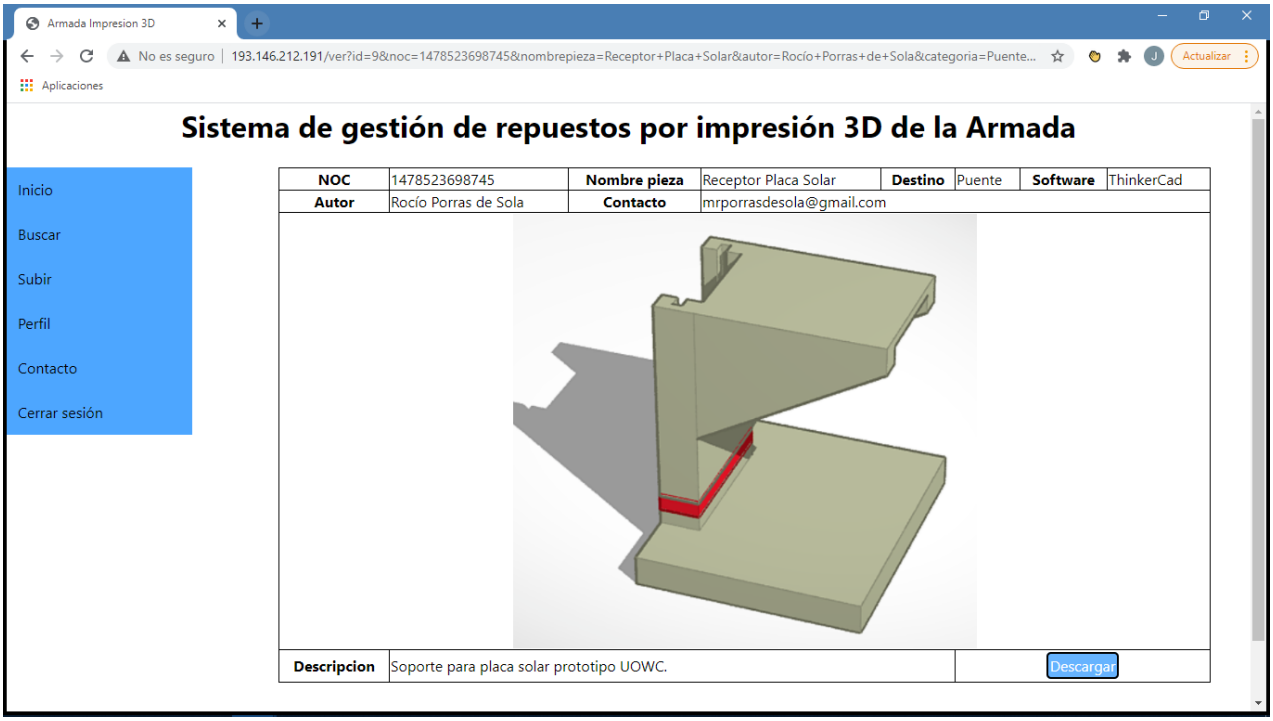


Figura 4-10 Vista previa de una pieza (elaboración propia)

Esta opción activará automáticamente una ventana para que el usuario seleccione dónde quiere guardar la descarga. Como se muestra en la Figura 4-11, cuando finalice la descarga, el usuario tendrá el diseño 3D listo en su dispositivo para imprimir. Windows cuenta con un programa instalado llamado *Print 3D* [62] que permite abrir diseños 3D y que, en este caso, nos permite comprobar el fichero que acabamos de descargar.

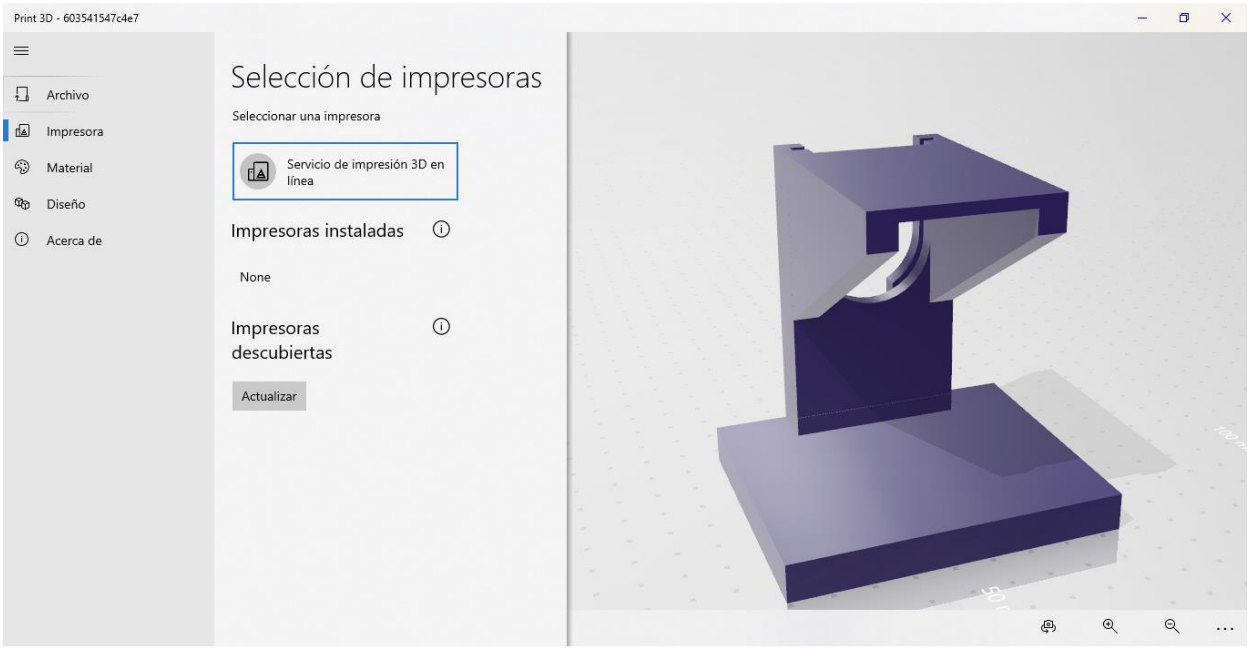


Figura 4-11 Previsualización del modelo descargado de la aplicación con Print 3D (elaboración propia)

4.1.5 Subir una pieza

Accediendo a la opción *Subir* desde el menú, cualquier usuario puede registrar una pieza en la web. Para ello, debe completar y enviar el formulario que se mostrará, como se puede apreciar en la Figura 4-12. Es importante que se rellenen los campos de manera adecuada para mantener el mismo criterio en todas las piezas.

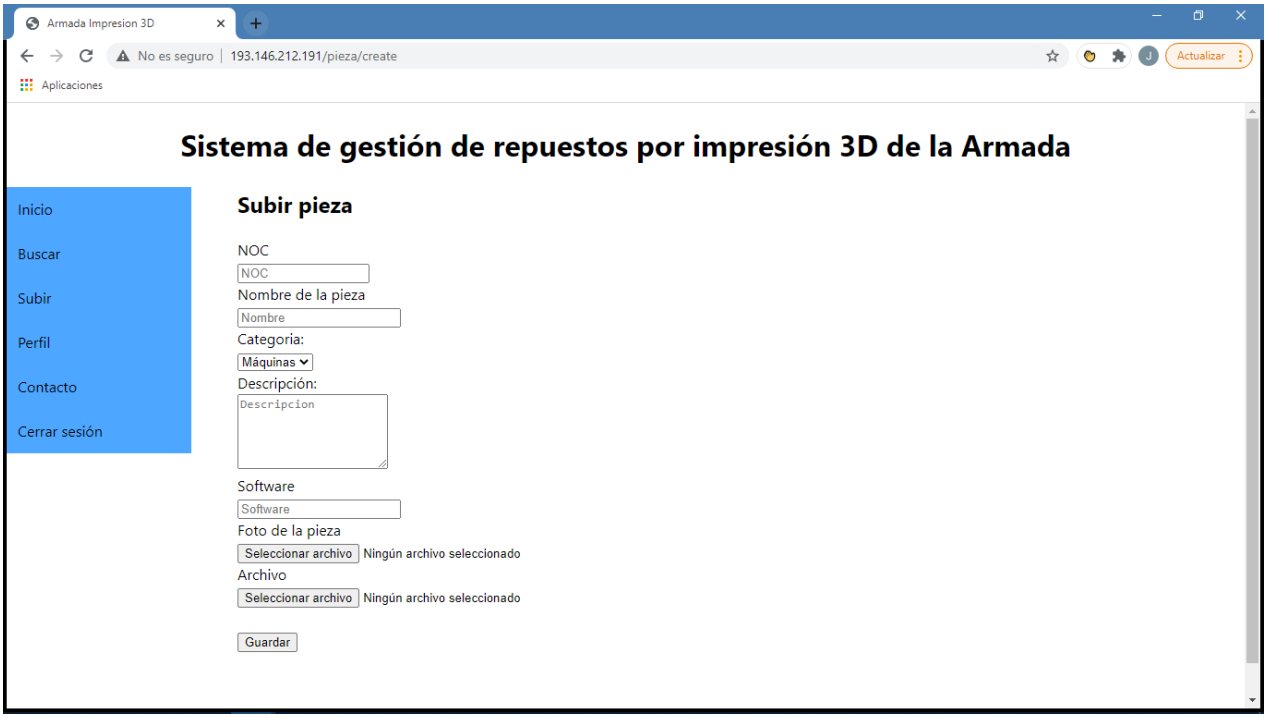


Figura 4-12 Formulario para subir una pieza (elaboración propia)

Para evitar errores y que los usuarios puedan subir piezas de forma aleatoria o de forma descuidada, se han restringido algunas acciones. Si el usuario intenta subir una pieza con un NOC sin los 13 dígitos requeridos, no podrá continuar. Para que en las imágenes no se produzcan errores, se ha restringido la

selección de los usuarios a la hora de cargar estas, pudiendo seleccionar únicamente archivos que el servidor reconozca como imágenes.

Como se expuso en el apartado 3.5.3, para evitar que un usuario pueda subir un archivo con una extensión distinta a .stl, se incluyó un código específico que evitará esta acción por parte del usuario. En la Figura 4-14 se ha probado a subir un archivo Excel para validar el código, obteniéndose satisfactoriamente en la Figura 4-14 un mensaje que nos indica el error en la extensión del fichero.

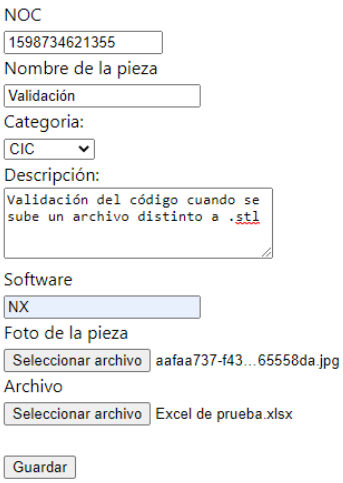


Figura 4-13 Prueba con fichero de extensión incorrecta (elaboración propia)

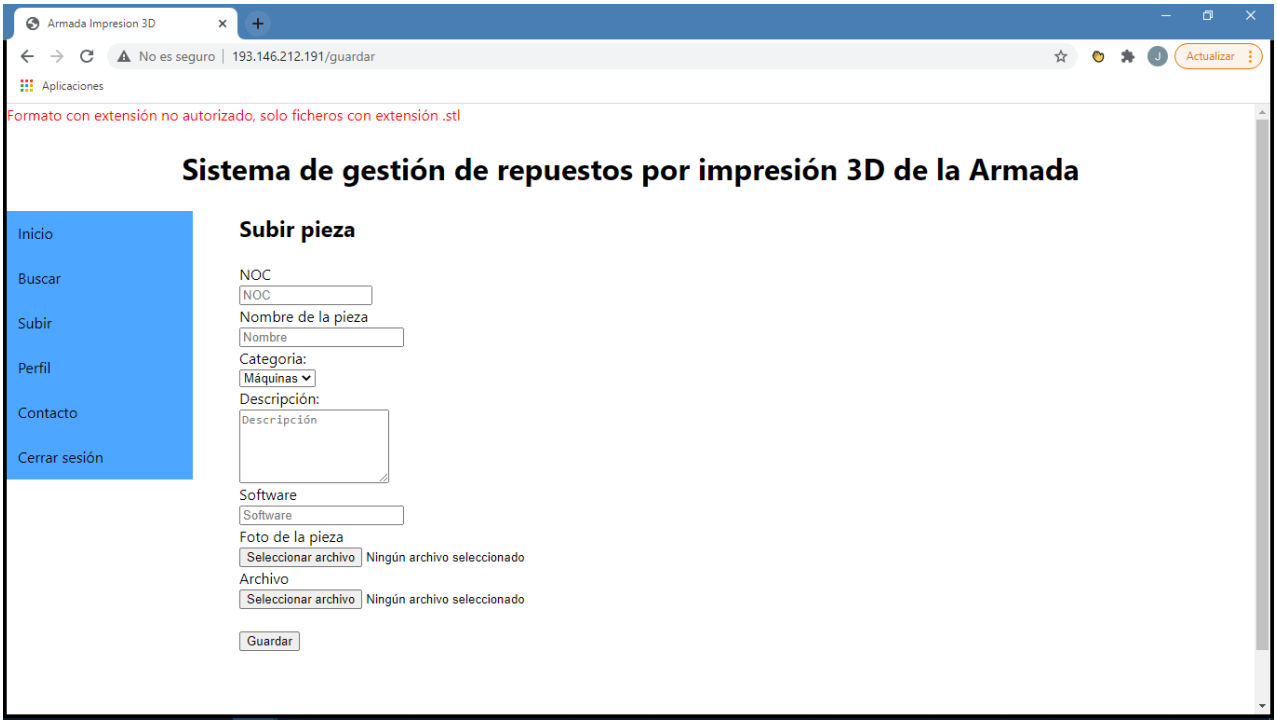


Figura 4-14 Alerta de extensión no autorizada (elaboración propia)

Por último, dado que el usuario debe completar toda la información de una pieza, se ha probado a hacer un registro con campos incompletos. Si el usuario envía un formulario incompleto, el navegador nos advierte de que, para continuar, es preciso completar los campos que falten.

A continuación, probaremos a realizar el registro de una pieza de forma correcta y comprobaremos que queda almacenada correctamente en la aplicación. Para ello introduciremos en el formulario la información de la Figura 4-15. Es conveniente destacar, que, en esta ocasión, se ha subido el modelo de

una pieza obtenida mediante escáner, por lo que su peso (de 15Mb) es mayor de lo habitual. El servidor está preparado para gestionar ficheros de hasta 50Mb.

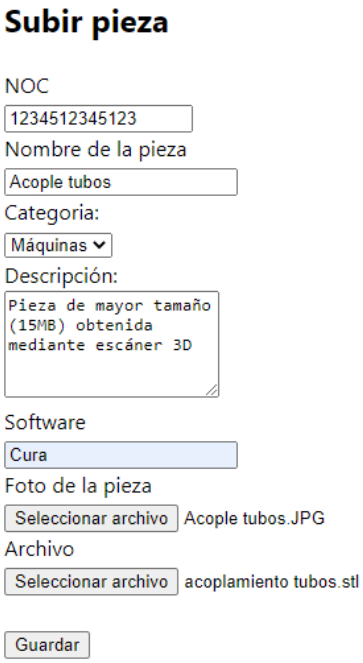


Figura 4-15 Nuevo registro de una pieza (elaboración propia)

Una vez que el usuario registra una pieza de forma correcta, la página le redirige a su perfil, donde podrá ver que esa pieza se ha añadido de forma correcta al listado de las piezas que haya subido. Como podemos comprobar en la Figura 4-16, la pieza ha quedado registrada en la aplicación web.

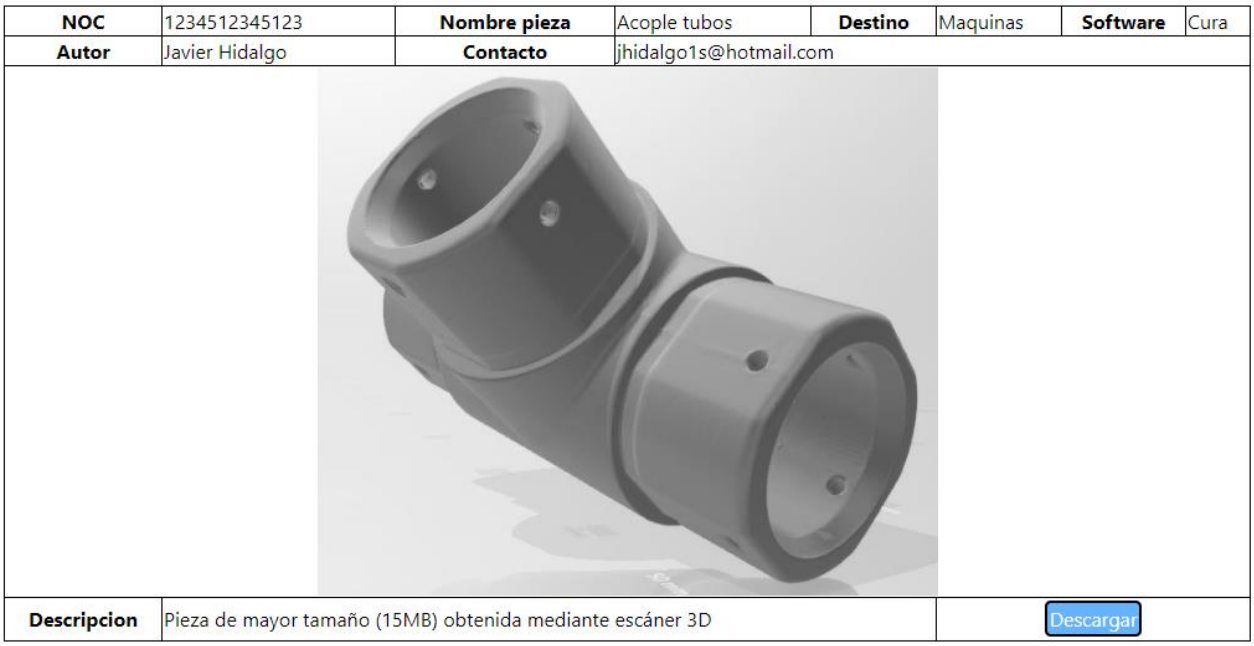


Figura 4-16 Pieza registrada (elaboración propia)

4.1.6 Gestión de las piezas desde el perfil

En el perfil se muestra toda la información del usuario: el número de usuario registrado en la web, el nombre con el que se ha registrado y el email, ambos datos coinciden con el autor y el email que se mostrará en la información de cada pieza.

Pero, además, se muestra una tabla únicamente con las piezas que el usuario ha ido subiendo. Como se puede ver en la Figura 4-17, esta tabla tiene el mismo diseño que la tabla que se muestra al realizar una búsqueda, con la salvedad de que se elimina al autor y se añaden las opciones de editar y eliminar una pieza . A continuación, se pondrán a prueba ambas herramientas, disponibles únicamente para el usuario sobre sus propias piezas, y se comprobará que ambas cumplen de forma correcta con sus cometidos.

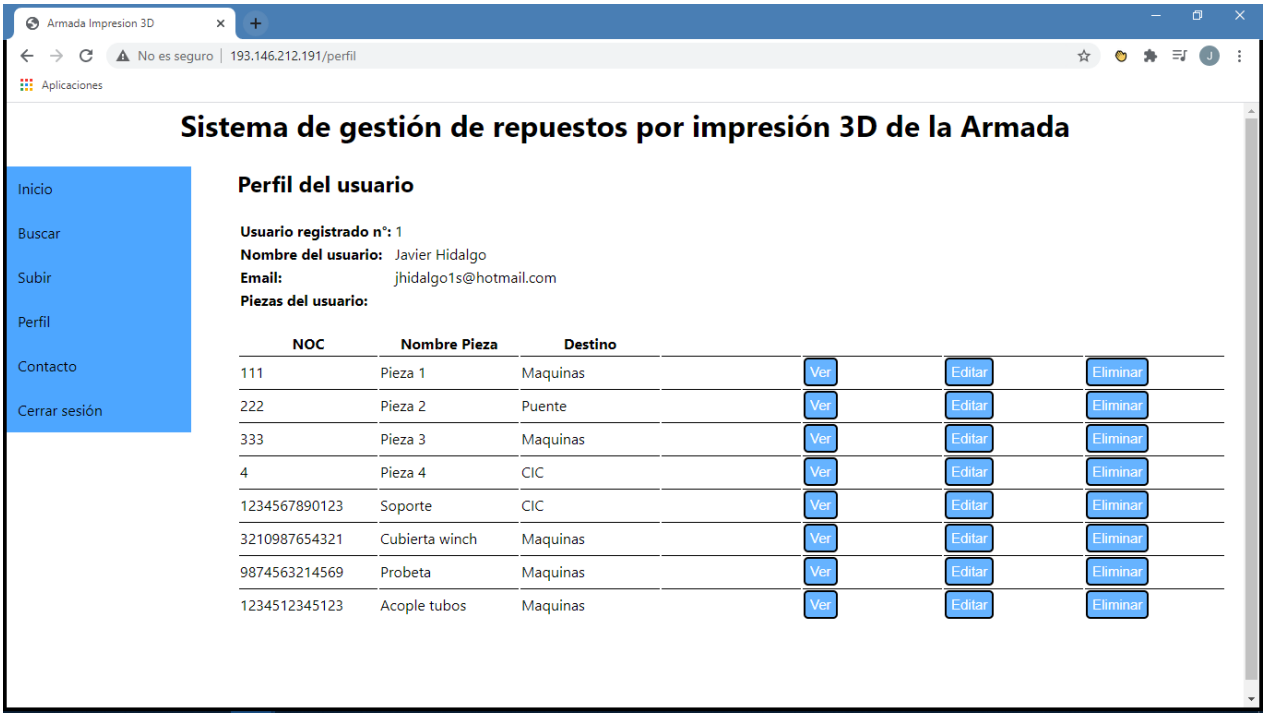


Figura 4-17 Perfil del usuario (elaboración propia)

Cuando un usuario ejecute *Editar* accederá a una vista similar a la vista empleada para registrar una pieza en la web, pero con la diferencia de que el formulario se encontrará con los datos de la pieza ya cumplimentados. Los únicos datos que no aparecerán cumplimentados de forma previa son el fichero correspondiente a la imagen y el fichero del diseño .stl.

En la Figura 4-18, tomada como ejemplo, si el usuario quisiese cambiar cualquiera de los datos le valdría con modificarlo y guardar los cambios. Para el caso de los ficheros, si se quisiesen mantener intactos, bastaría con no añadir ningún fichero (tal y como se indica).

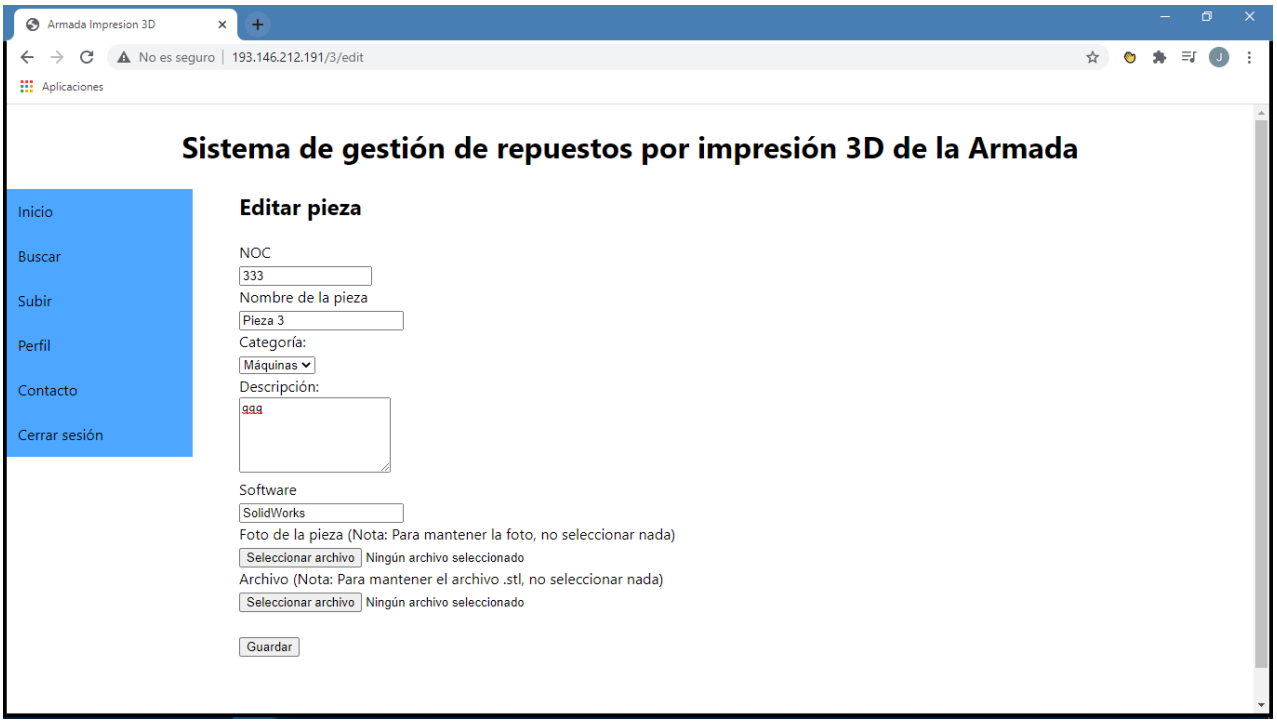


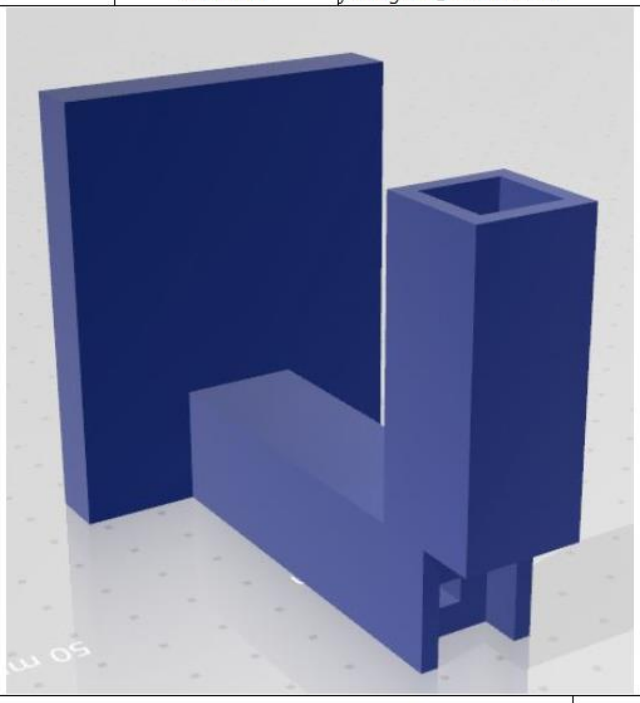
Figura 4-18 Edición de una pieza (elaboración propia)

Para este ejemplo se va a modificar el NOC, la categoría de la pieza por CIC, la descripción y la imagen. Al guardar los cambios, la aplicación web nos redirecciona al perfil. Para la prueba se va a emplear la pieza de la Figura 4-19.

Figura 4-19 Soporte original (elaboración propia)

A continuación, se modifican los campos que corresponda en el formulario. Para este caso, se modificarán el destino, la descripción y la foto.

NOC	1236547895874	**Nombre pieza**	Pieza 3	**Destino**	CIC	**Software**	SolidWorks
Autor	Javier Hidalgo	**Contacto**	jhidalgo1s@hotmail.com				



Descripcion	Validación cambios edición de una pieza	Descargar

Figura 4-20 Soporte tras ser editado (elaboración propia)

Como podemos ver en la Figura 4-20, se ha modificado correctamente el destino, la descripción y la foto, manteniéndose el NOC, el nombre de la pieza, el software, el autor y su correo de contacto, y el botón de descarga.

Para comprobar el funcionamiento de la aplicación web a la hora de eliminar una pieza, se ha decidido eliminar la Pieza 4. Para ello, se debe seleccionar *Eliminar* desde la tabla de piezas del perfil del usuario. Tras seleccionar esta opción, vemos en la Figura 4-21 que la web devolverá un mensaje para comprobar que realmente se quiere eliminar la pieza y no se trata de un error.

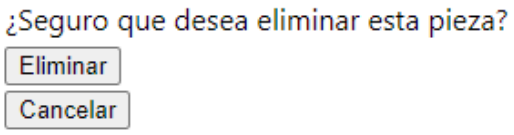


Figura 4-21 Comprobación de eliminación de una pieza (elaboración propia)

Haciendo click en *Eliminar* podemos ver en la Figura 4-22 que la página redirecciona al usuario a su perfil y que la pieza ha desaparecido de la lista de piezas.

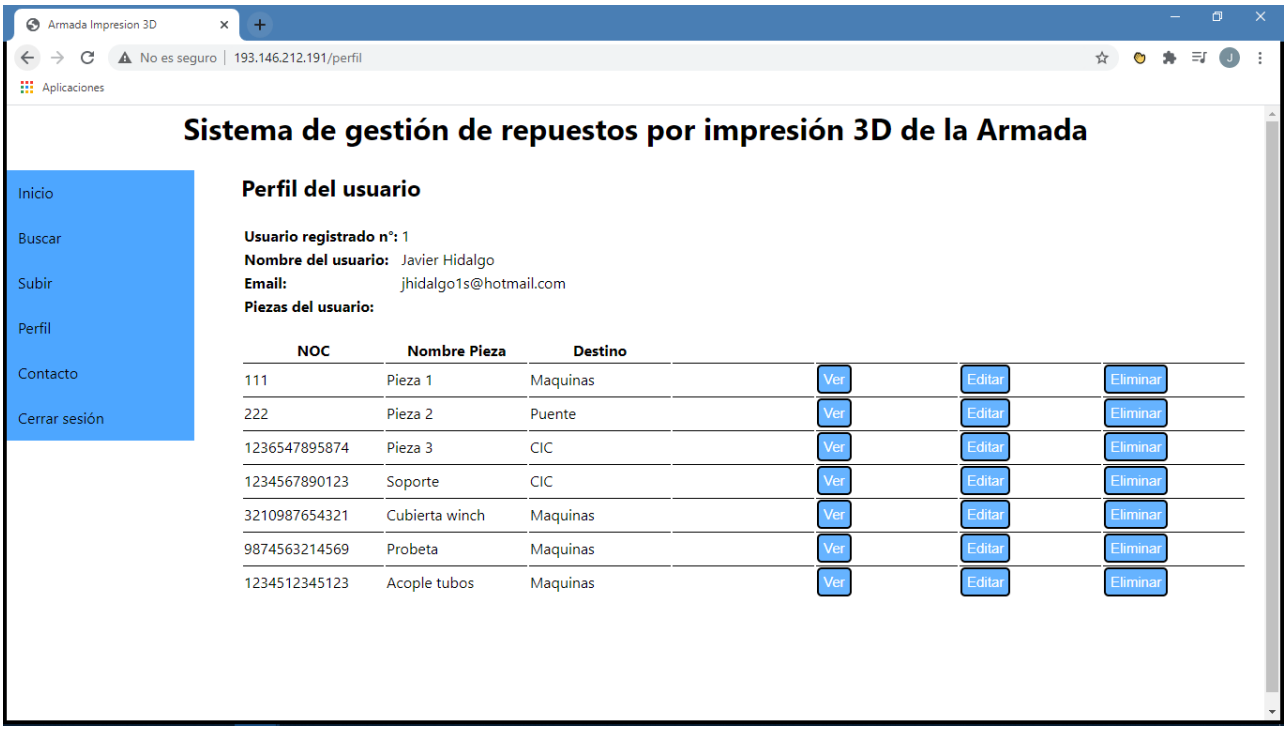


Figura 4-22 Perfil del usuario con Pieza 4 eliminada (elaboración propia)

4.2 Comprobación de funcionamiento de la base de datos

Una vez comprobada la funcionalidad de la aplicación web, queda comprobar que las operaciones realizadas anteriormente se han registrado correctamente en la base de datos.

Comenzando por la tabla que gestiona la información de los usuarios, podemos ver en la Figura 4-23 que el funcionamiento es correcto ya que alberga varios usuarios ya registrados. Sin esta tabla, la función de autenticación de la web no sería posible.

```
mysql> select * from users;
+----+-----------------------------+-------------------------+-----------------+--------------------------------------------------------------+
--------------+
| id | name                        | email                   | email_verified_at | password                                                     |
ed_at          |
+----+-----------------------------+-------------------------+-----------------+--------------------------------------------------------------+
--------------+
|  1 | Javier Hidalgo              | jhidalgo1s@hotmail.com  | NULL              | $2y$10$gfEvZUe5eDl0ybcM7iU14O7QUk5SKVQ5lhcjFgNq7klzrkRRQEUxa |
02-19 13:35:16 |
|  2 | Pablo                       | pablo@cud.uvigo.es      | NULL              | $2y$10$UtBMA52/dzW9V6BCg2RYheeYp6Yf1x5Gf.2Ymj0o5jyF6.vz1EAx2 |
02-19 19:04:29 |
|  3 | AF Pardo                    | mikebro39@gmail.com     | NULL              | $2y$10$.SXDx5bhnDj/Zm9cTlYine9ZA1VKx8cfNA2j8zaK4uLlGUlp/L4fq |
02-20 17:28:59 |
|  4 | Laura Isabel Cervera Pelayo | lauracervera98@gmail.com | NULL              | $2y$10$fpEu5YOaoQH7aUNGGRZPQOx8w6QWH0jg7UB6JCjkro9QxRvydhfHS |
02-20 18:45:12 |
|  5 | Belén Barragáns Martínez    | belen@cud.uvigo.es      | NULL              | $2y$10$E/oKsNTf4H1ET.lEqiau8eQAU6.jTeWxscaZVZ.IrRL3Ccw.3dwqm |
02-21 00:08:20 |
|  6 | Carlos Lázaro Benavides     | clbenavides98@gmail.com | NULL              | $2y$10$zVnLVZMKgqtjO4NtzKt18u4BrtRrQHGkp3gKhAoQ3tSEjL4iBqNYS |
03-02 21:37:06 |
|  7 | Rocío Porras de Sola        | mrporrasdesola@gmail.com | NULL              | $2y$10$uckSXHpqTr5K2eXnZklw5eMO4jy4aFukxqZ8CNqNBV71cO9nXmnUO |
02-23 17:49:43 |
+----+-----------------------------+-------------------------+-----------------+--------------------------------------------------------------+
```

Figura 4-23 Tabla *users* (elaboración propia)

La columna de la tabla que hace referencia a la verificación de email se encuentra como *NULL*, esto se debe a que ningún usuario ha verificado su email, principalmente, porque esta función no se ha llegado a desarrollar. Se puede ver también que la tabla guarda correctamente la contraseña de forma cifrada.

A continuación, veremos en la Figura 4-24 la gestión de las piezas en la aplicación web. En el apartado 4.1 se ha registrado una nueva pieza con el nombre de *Acople tubos*, que se puede apreciar como el último registro de la tabla, al que le corresponde el *id* 17.

Después se editó la pieza *Soporte*, con *id* 7, a la que se le cambió la categoría y la descripción. Los cambios en la foto no se podrán ver, pues la foto y el archivo de la pieza no se pueden ver a simple vista en la tabla, ya que lo que se almacena en esta tabla es una cadena de caracteres que indican el nombre

del archivo que se debe buscar en la carpeta pública donde se almacenan los ficheros cuando la pieza sea llamada. Para simplificar la vista de la base de datos, no se han añadido las columnas de Foto y Pieza.

Por último, se eliminó la *Pieza 5* y como se puede ver en la Figura 4-24, esta pieza ya no existe en la base de datos. Adicionalmente, se puede ver que la tabla almacena información temporal de cuándo se realiza cada registro y de cuándo fue la última vez que se actualizó, en huso horario Zulú.

id	NOC	NombrePieza	Autor	Categoria	Descripcion	Software	created_at	updated_at	
1	111	Pieza 1	jhidalgo1s@hotmail.com	Maquinas	qqq	SolidWorks	2021-02-19 13:37:05	2021-02-19 13:37:05	s/UUqx4x1n1OSUG2FWb97HOeTBhVZnAp7OxRZifHPQ.bin
2	222	Pieza 2	jhidalgo1s@hotmail.com	Puente	jjj	.stl	2021-02-19 21:23:18	2021-02-19 21:23:18	s/Jsyuv6jWlklP86tHjleBF8scU2YH3RAYThuozPfZ.bin
3	1236547895874	Pieza 3	jhidalgo1s@hotmail.com	CIC	Validación cambios edición de una pieza	SolidWorks	2021-02-20 10:30:07	2021-03-10 18:27:08	s/60490f7c6de17.stl
6	8888121212121	test pablo2222	pablo@cud.uvigo.es	Maquinas	test	stl	2021-02-20 16:59:11	2021-03-03 08:17:14	s/603f45a7cf44a.stl
7	1234567890123	Soporte	jhidalgo1s@hotmail.com	CIC	Validación cambio edición campos de una pieza	NX	2021-02-21 11:12:37	2021-03-02 21:03:38	s/603240253e69a.stl
8	3210987654321	Cubierta winch	jhidalgo1s@hotmail.com	Maquinas	Cubierta de un winche de vela	Inventor	2021-02-21 16:56:54	2021-03-02 20:50:01	s/603290d62f8bd.ipt
9	1478523698745	Receptor Placa Solar	mrporrasdesola@gmail.com	Puente	Soporte para placa solar prototipo UOWC.	ThinkerCad	2021-02-23 17:54:28	2021-02-23 17:54:28	s/603541547c4e7.stl
10	7685665403271	Soporte plano	belen@cud.uvigo.es	Maquinas	Pieza soporte plano peso < 5 kg	Inventor	2021-02-24 00:19:06	2021-02-24 00:29:13	s/60359dd9dc12f.stl
11	8844222232942	Placa R3	belen@cud.uvigo.es	Otros	Placa R3 grandes dimensiones	Inventor	2021-02-24 00:24:16	2021-02-24 00:24:16	s/60359cb0d0abf.stl
12	8852935654823	Placa R6	belen@cud.uvigo.es	Maquinas	Placa R6 (diseño alta calidad)	Inventor	2021-02-24 00:25:11	2021-02-24 00:25:11	s/60359ce7c4057.stl
13	9874563214569	Probeta	jhidalgo1s@hotmail.com	Maquinas	Probeta de pruebas de tracción	SolidWorks	2021-02-24 21:44:42	2021-03-02 17:46:01	s/603e7039c5fd4.stl
16	1234567812312	TEST	pablo@cud.uvigo.es	CIC	Test	STL	2021-03-02 09:38:27	2021-03-02 09:38:27	s/603e07939253e.stl
17	1234512345123	Acople tubos	jhidalgo1s@hotmail.com	Maquinas	Pieza de mayor tamaño (15MB) obtenida mediante escáner 3D	Cura	2021-03-02 16:15:27	2021-03-02 16:15:27	s/603e649fac024.stl
19	1234567891231	test	pablo@cud.uvigo.es	Maquinas	qqw	stv	2021-03-10 08:47:05	2021-03-10 08:47:05	s/60488789992a2.stl

Figura 4-24 Tabla que gestiona las piezas (elaboración propia)

5 CONCLUSIONES Y LÍNEAS FUTURAS

En este apartado se procede a plasmar las conclusiones obtenidas durante el desarrollo de la aplicación y especialmente tras la finalización de la misma. También se plantearán las líneas futuras que se considera que podrían complementar y mejorar la aplicación.

5.1 Conclusiones

Si se revisan los objetivos establecidos en el capítulo 1. Se puede considerar que, efectivamente, se ha cumplido con los objetivos de desarrollar una herramienta web capaz de poner al alcance de todas las unidades diseños de impresión 3D. Esta herramienta permitirá a sus usuarios ahorrar tiempo y dinero en el caso de necesitar un producto que no se pueda obtener por medios tradicionales y que esté disponible para su impresión mediante fabricación aditiva.

La aplicación web permite almacenar y descargar diseños de hasta 50 MB, realizar búsquedas por distintos campos, consultar más información sobre cada pieza, editar información y eliminar una pieza, por lo que el control y la gestión de este inventario virtual diseñado para la Armada queda cubierto. Además, se han implementado estructuras y etiquetas en el código destinadas al control de errores y a evitar que los usuarios puedan realizar malas prácticas en la web, tales como registros no válidos de ficheros o, suplantar la identidad de otro usuario impidiendo registrar una pieza bajo el nombre de otro miembro de la web.

Esta última caracterísitca no hubiese sido posible si la web no se hubiese diseñado con un sistema de autenticación y registro. Esto permite dotar de seguridad a la página y permite a cada usuario mayor control sobre sus piezas. Además, permite al resto de usuarios una dirección para ponerse en contacto con los autores de cada diseño si fuese preciso.

A título personal, este TFG me ha permitido profundizar en el sistema de aprovisionamiento y gestión de repuestos de la Armada, el cual, desconocía hasta la fecha. También me ha permitido ahondar en el nuevo concepto de Armada 4.0 y cómo definirá el nuevo horizonte en las futuras adquisiciones de la Armada. Pero además, este TFG me ha permitido familiarizarme con el diseño de aplicaciones web, herramientas que utilizamos diariamente pero de las que a penas conocía su funcionamiento.

5.2 Líneas futuras

Esta aplicación supone el inicio de la que podría ser una nueva plataforma diseñada para trabajar como una herramienta usada en los servicios de aprovisionamiento. Dado que el desarrollo de esta aplicación se ha realizado en un tiempo limitado, aun quedan áreas en las que se podría profundizar más y que quedarán como líneas futuras.

Las líneas futuras que a continuación se describen, están relacionadas con la funcionalidad de la aplicación web y con su escalabilidad. El diseño gráfico de la aplicación web queda abierto y podría ser modificado según las exigencias o necesidades del programador.

En cuanto a la funcionalidad de la aplicación web, se proponen como líneas de trabajo futuro:

- Diseñar un sistema de roles que permita asignar diferentes permisos a los usuarios. Se proponen tres perfiles de usuarios diferentes: usuario, ingeniero y administrador. El perfil usuario únicamente podrá acceder a la aplicación web para buscar y descargar, quedando restringida la capacidad de subir ficheros, editar y eliminar. El perfil ingeniero, por el contrario, podrá subir, editar y eliminar ficheros a la web. Por último, el administrador tendrá todos los permisos y gestionará tanto los diseños registrados como los usuarios. El fin de este sistema de roles será añadir control sobre la web, de tal forma que un usuario de perfil ingeniero (aquel que cumpla con unas cualidades que le certifiquen para el diseño de piezas válidas) sea el encargado de subir a la web los diseños y no cualquier usuario que pueda carecer de experiencia en este ámbito y, por lo tanto, llene de registros no válidos o defectuosos la web lo que generará pérdida de valor y confianza en la aplicación.
- Diseño de un foro y proporcionar a los usuarios la funcionalidad de hacer comentarios y proporcionar valoraciones. De esta forma se podrá crear una comunidad en la que si algún usuario tuviese algún problema, ya sea con la web o a la hora de imprimir un diseño, pueda recibir ayuda de otros usuarios.
- Debido al gran número de registros que esta aplicación podría llegar a albergar, se propone la posibilidad de ordenar las listas para realizar la búsqueda de forma más sencilla. El orden podría ser por NOC, nombre, categoría, autor, valoración, descargas o fecha de registro.
- Incluir un sistema de mensajería interna en la aplicación que permita a los usuarios comunicarse sin necesidad de hacer uso de otros medios tales como correo o SACOMAR.

En cuanto a la escalabilidad:

- Dado que el NOC es un sistema compartido OTAN, se propone estudiar hasta qué punto se podría extender esta aplicación, tanto en ámbito nacional con otros ejércitos, como internacional con otras marinas.
- Al tratarse de una aplicación cuyo fin principal es el aprovisionamiento, se propone estudiar la posibilidad de introducir y/o relacionar la aplicación con SIGAPEA y SIGMAWEB. Esta relación podría realizarse incluyendo esta aplicación web como un apartado más de las mismas o como un enlace desde estas hasta la aplicación web desarrollada en este Trabajo de Fin de Grado, lo que complementaría la funcionalidad de dichas aplicaciones SIGAPEA y SIGMAWEB.

6 BIBLIOGRAFÍA

[1] Beatriz Couce, «Navantia da a las F-110 capacidad para producir sus propios repuestos,» La Voz de Galicia, 03 junio 2019. [En línea]. Available: https://www.lavozdegalicia.es/noticia/ferrol/2019/06/03/navantia-da-f-110-capacidad-producir-propios-repuestos/0003_201906F3C1995.htm. [Último acceso: 22 febrero 2021].

[2] Victor Casal, «Informe 4.0,» Navantia, Ferrol, 2021.

[3] Simplify3D, «Simplify3D,» Simplify3D, 2021. [En línea]. Available: https://www.simplify3d.com/. [Último acceso: 22 febrero 2021].

[4] Ultimaker Cura, «Ultimaker Cura,» Ultimaker BV, 2020. [En línea]. Available: https://ultimaker.com/es/software/ultimaker-cura. [Último acceso: 22 febrero 2021].

[5] Ministerio de Defensa, «Centro de Sistemas y Tecnologías de la Información y las Comunicaciones,» 2020. [En línea]. Available: https://www.defensa.gob.es/ministerio/organigrama/sedef/cestic/. [Último acceso: 14 enero 2021].

[6] Almirante José Luis Urcelay, «La transformación digital y el Sostenimiento 4.0 en la Armada» 04 abril 2019. [En línea]. Available: https://www.infodefensa.com/es/2019/04/04/opinion-transformacion-digital-sostenimiento-armada.php. [Último acceso: 27 diciembre 2020].

[7] Jose Mª Navarro García, «La Armada y Navantia trabajan en el proceso de transformación digital» 21 junio 2020. [En línea]. Available: https://www.defensa.com/espana/armada-navantia-trabajan-proceso-transformacion-digital. [Último acceso: 27 diciembre 2020].

[8] Diego Carriazo Hernández, «Sistema Integrado de Sostenimiento,» Armada, Madrid, 2018.

[9] Carrasco B., «Soprene, punta de lanza de la revolución tecnológica en la Armada,» infodefensa.com, 04 abril 2019. [En línea]. Available: https://www.infodefensa.com/es/2019/04/04/noticia-soprene-punta-lanza-revolucion-tecnologica-armada.html. [Último acceso: 27 diciembre 2020].

[10] Ministerio de Defensa, «Jefatura de Apoyo Logístico,» 2020. [En línea]. Available: https://armada.defensa.gob.es/ArmadaPortal/page/Portal/ArmadaEspannola/conocenosorganizacion/prefLang-es/04Apoyofuerza--01jal. [Último acceso: 21 enero 2021].

[11] Ministerio de Defensa, «Dirección de Sostenimiento,» Gobierno de España, 2020. [En línea]. Available: https://armada.defensa.gob.es/ArmadaPortal/page/Portal/ArmadaEspannola/conocenosorganizacion/prefLang-es/04Apoyofuerza--01jal--02disos. [Último acceso: 21 enero 2021].

[12] Capitán Giménez Guitart, «Organización,» Control de material de aprovisionamiento, Organización de la JAL, Marín, 2020.

[13] Ministerio de Defensa, «Manual de Aprovisionamiento del Primer Escalón de la Armada,» Jefatura de Apoyo Logístico, Dirección de Sostenimiento, Madrid, 2012.

[14] Capitán Giménez Guitart, «Tema 1 Introducción a la catalogación,» Marín, 2021.

[15] Almirante Jefe del Estado Mayor de la Armada Teodoro Esteban López Calderón, «Concepto del apoyo logístico,» Ministerio de Defensa, Madrid, 2017.

[16] Díaz de los Ríos, Juan Durán, «Futuro Gemelo Digital (GD) de la F-110,» Archivo Armada, Madrid, 2020.

[17] Ministerio de Defensa, «Dirección General de Armamento y Material,» Gobierno de España, 2020. [En línea]. Available: https://www.defensa.gob.es/ministerio/organigrama/sedef/dgam/. [Último acceso: 16 enero 2021].

[18] Secretaría de Estado de Defensa, «Instrucción 5/2008, regulación del sostenimiento del armamento y material,» Madrid, 2008.

[19] M. Annunziata, «*2020: The year ahead in 3D (printing)*,» Forbes, 17 diciembre 2019. [En línea]. Available: https://www.forbes.com/sites/marcoannunziata/2019/12/17/2020-the-year-ahead-in-3d-printing/?sh=400a1172407f. [Último acceso: 29 diciembre 2020].

[20] D. M. Kückelhaus, «*3D Printing and the future*» DHL Trend Research, Alemania, Noviembre 2016.

[21] Lara Febrero Garrido, «Tema 3: Análisis, implantación y optimización,» Ingeniería de fabricación y calidad dimensional, Centro Universitario de la Defensa, Marín, 2020.

[22] «Thingmaker,» Mattel, 2021. [En línea]. Available: http://www.thingmaker.com/design/. [Último acceso: 27 diciembre 2020].

[23] 3D Systems, «Next Dent,» 3D Systems, [En línea]. Available: https://nextdent.com/. [Último acceso: 04 enero 2021].

[24] Sarah Perez, «*SOOLS let you buy 3D-Printed Insoles*,» Tech Crunch, 05 octubre 2015. [En línea]. Available: https://techcrunch.com/2015/10/05/sols-lets-you-buy-3d-printed-insoles-customized-to-your-feet-right-from-an-iphone-app/. [Último acceso: 04 enero 2021].

[25] General Electric, «General Electric,» General Electric, 2021. [En línea]. Available: https://www.ge.com/. [Último acceso: 04 enero 2021].

[26] B. Krassenstein, «*New Airbus A350 XWB Aircraft Contains Over 1,000 3D Printed Parts*,» 3Dprinting.com, 6 mayo 2015. [En línea]. Available: https://3dprint.com/63169/airbus-a350-xwb-3d-print/#:~:text=New%20Airbus%20A350%20XWB%20Aircraft%20Contains%20Over%201%2C000%203D%20Printed%20Parts,-May%206%2C%202015&text=When%20it%20comes%20to%20manufacturing,its%20way%20into%20the%20process.. [Último acceso: 28 diciembre 2020].

[27] Cunningham, Victor, «*Navy Additive Manufacturing: Adding parts, subtractig steps*,» Naval Postgraduate School, Monterey, California, Junio 2015.

[28] R. Uppal, «*US Navy 3D printing entire fleet from UAVS, and submarines to weapons systems and their AMMO*,» IDST International Defence, Security & Technology, 30 septiembre 2019. [En línea]. Available: https://idstch.com/military/navy/darpa-perfecting-3d-printing-for-military-through-its-open-manufacturing-program/. [Último acceso: 28 diciembre 2020].

[29] T. D. Board, «*The value of excess and obsolete inventory*,» Industrial Supply, [En línea]. Available: https://industrialsupplymagazine.com/pages/Management---Value-of-excess-inventory.php. [Último acceso: 27 diciembre 2020].

[30] Michael Molitch-Hou, «*KAZZATA: The first marketplace for 3D printed spare parts*,» 3D Printing Industry, 21 mayo 2014. [En línea]. Available: https://3dprintingindustry.com/news/kazzata-first-marketplace-3d-printed-spare-parts-27567/. [Último acceso: 06 enero 2021].

[31] Mecedes-Benz, «*Mercedes-Benz replacement parts from the 3D printer*,» Mercedes-Benz, [En línea]. Available: https://www.mercedes-benz.com/en/classic/classic-service-parts/next-generation-mercedes-benz-replacement-parts-from-3d-printer/. [Último acceso: 06 enero 2021].

[32] «*Autonomous Manufacturing*,» [En línea]. Available: https://amfg.ai/. [Último acceso: 18 enero 2021].

[33] GrabCAD, «*GrabCAD Shop*,» STRATSYS solution , 2021. [En línea]. Available: https://grabcad.com/shop. [Último acceso: 21 enero 2021].

[34] GrabCAD, «*GrabCAD COMMUNITY*,» STRATSYS solution, 2021. [En línea]. Available: https://grabcad.com/library?page=1&time=this_month&sort=popular. [Último acceso: 21 enero 2021].

[35] M. Thingverse, «Thingverse,» MakerBot Industries, LLC, 2021. [En línea]. Available: https://www.thingiverse.com/about. [Último acceso: 22 enero 2021].

[36] Denia Chavarria, «Taller de Hardware UTD,» [En línea]. Available: https://tallerdehardwareutd.wordpress.com/software-de-programacion/. [Último acceso: 01 enero 2021].

[37] «Lenguajes de programación web: los más usados en Internet,» Digital Guide IONOS, 07 julio 2019. [En línea]. Available: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/lenguajes-de-programacion-web/. [Último acceso: 02 enero 2021].

[38] «Principales lenguajes de programación para el desarrollo web,» PiensaSolutions, 19 octubre 2017. [En línea]. Available: https://www.piensasolutions.com/blog/principales-lenguajes-programacion-web/. [Último acceso: 02 enero 2021].

[39] «Conoce el lenguaje de programación Java,» Blog SEAS, 19 julio 2019. [En línea]. Available: https://www.seas.es/blog/informatica/conoce-el-lenguaje-de-programacion-java/. [Último acceso: 28 enero 2021].

[40] «Welcome to Cplusplus.com,» 2020. [En línea]. Available: https://www.cplusplus.com/. [Último acceso: 28 enero 2021].

[41] «Tutorial básico de HTML: crea tu propio proyecto web,» Digital Guide IONOS, 15 noviembre 2019. [En línea]. Available: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/aprende-html-tutorial-para-principiantes/. [Último acceso: 01 enero 2021].

[42] «HTML: Lenguaje de etiquetas de hipertexto,» MDN Web Docs, 10 diciembre 2020. [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTML. [Último acceso: 02 enero 2021].

[43] E. Tébar, «Frameworks en el desarrollo web: las mejores prácticas para tu negocio online,» We Are Marketing Global Growth Agents, 13 febrero 2020. [En línea]. Available: https://www.wearemarketing.com/es/blog/frameworks-en-el-desarrollo-web-las-mejores-practicas-para-tu-negocio-online.html. [Último acceso: 04 enero 2021].

[44] «Django,» Django Software Foundation, 2021. [En línea]. Available: https://www.djangoproject.com/. [Último acceso: 17 enero 2021].

[45] «Foundation,» ZURB Inc, 2021. [En línea]. Available: https://get.foundation/. [Último acceso: 17 enero 2021].

[46] «Spring,» VMware Inc, 2021. [En línea]. Available: https://spring.io/. [Último acceso: 17 enero 2021].

[47] «Laravel,» Laravel, 2021. [En línea]. Available: https://laravel.com/. [Último acceso: 21 enero 2021].

[48] «Codelgniter,» EllisLab, 2021. [En línea]. Available: https://www.codeigniter.com/. [Último acceso: 17 enero 2021].

[49] M. Stauffer, Laravel Up & Running, O'Reilly, 2020.

[50] Matt Stauffer, «Why Laravel?,» de *Laravel Up & Running*, O'Reilly, 2020, p. 3.

[51] M. Stauffer, «Why Laravel?,» de *Laravel Up & Running*, O'Reilly, 2019, pp. 1-9.

[52] Uriel Hernández, «MVC (*Model, View, Controller*) Explicado.,» Códigofacilito, 22 Febrero 2015. [En línea]. Available: https://codigofacilito.com/articulos/mvc-model-view-controller-explicado. [Último acceso: 14 Febrero 2021].

[53] Aprendible, «Mastering Authentication,» Aprendible, 2021. [En línea]. Available: https://aprendible.com/series/autenticacion. [Último acceso: 14 febrero 2021].

[54] Gustavo B, «¿Qué es MySQL?,» Hostinger tutoriales, 03 diciembre 2020. [En línea]. Available: https://www.hostinger.es/tutoriales/que-es-mysql/. [Último acceso: 28 enero 2021].

[55] «Composer,» Private Packagist, 2021. [En línea]. Available: https://getcomposer.org/. [Último acceso: 28 enero 2021].

[56] «Composer, gestor de dependencias para PHP,» desarrolloweb.com, 24 enero 2020. [En línea]. Available: https://desarrolloweb.com/articulos/composer-gestor-dependencias-para-php.html. [Último acceso: 28 enero 2021].

[57] «*Linux manual pages*,» Chown, [En línea]. Available: https://manpages.courier-mta.org/htmlman1/chown.1.html. [Último acceso: 25 enero 2021].

[58] Eclipse, «Eclipse.org,» Eclipse foundation, 2021. [En línea]. Available: https://www.eclipse.org/ide/. [Último acceso: 24 enero 2021].

[59] Java, «Java,» [En línea]. Available: https://www.java.com/es/. [Último acceso: 24 enero 2021].

[60] Laravel, «Documentación Laravel,» Laravel, 2021. [En línea]. Available: https://laravel.com/docs/8.x. [Último acceso: 07 02 2021].

[61] Laravel, «Laravel Breeze,» Laravel, 2021. [En línea]. Available: https://laravel.com/docs/8.x/starter-kits#laravel-breeze. [Último acceso: 07 febrero 2021].

[62] Microsoft, «Print 3D,» Microsoft Corporation, 2021. [En línea]. Available: https://www.microsoft.com/es-es/p/print-3d/9pbpch085s3s?activetab=pivot:overviewtab. [Último acceso: 03 marzo 2021].

ANEXO I: ARCHIVOS PHP

Web.php

```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\FormularioController;
use App\Http\Controllers\PiezaController;
use App\Http\Controllers\UsuarioDatosController;
/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('/', function () {
    return view('inicio');
});

Route::get('/dashboard', function () {
    return view('principal'); //
})->middleware(['auth'])->name('dashboard'); // seguramente habrá que estudiarse los middlewares

require __DIR__.'/auth.php';


Route::resource('/pieza', PiezaController::class)->middleware(['auth']);
Route::post('/guardar',[PiezaController::class,'store'])->middleware(['auth']);
Route::get('/search', [PiezaController::class, 'filtro']);
Route::get('/ver', [PiezaController::class, 'show'])->name('mostrar');

Route::get('/perfil', [UsuarioDatosController::class, 'index'])->name('perfil');
Route::get('/{id}/edit', [PiezaController::class, 'edit'])->name('editar');//ruta a la que nos direcciona el boton "editar"
Route::put('/{pieza}', [PiezaController::class, 'update'])->name('actualizar');//ruta con la que actualizamos la pieza, Laravel recomienda el empleo de PUT, mandamos por la url la información de la pieza al controlador
//ruta que muestra una cuadro para asegurar que el usuario quiere eliminar esa pieza
Route::get('{id}/eliminar', [PiezaController::class, 'eliminar'])->name('eliminarControl');
//ruta para eliminar una pieza, laravel accede a PiezaController y a continuación al método delete
Route::delete('/{pieza}', [PiezaController::class, 'destroy'])->name('eliminar');
Route::get('/contacto', function() {
    return view ('contacto');
});
```

Pieza.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Builder;

class Pieza extends Model
{
    use HasFactory;
    protected $table = 'nubes';
    protected $primarykey = 'id';

   protected $fillable = [      'NOC',
                                'NombrePieza',
                                'Autor',
                                'Categoria',
                                'Descripcion',
                                'Software',
                                'Foto',
                                'Pieza'
                          ];
}
```

UsuarioDatosController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Pieza;

class UsuarioDatosController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {

        $piezas= new Pieza();
        $user=auth()->user();
        $piezas=Pieza::orderBy('id', 'ASC')->paginate(10);
        $piezas= Pieza::where('Autor', 'LIKE', '%'.$user->email.'%')->get();

        return view ('perfil', compact ('user', 'piezas'));
    }
```

PiezaController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Pieza;
use App\Models\User;
use Illuminate\Support\Facades\DB;

class PiezaController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */


    public function index()
    {

        $pieza= Pieza::orderBy('id', 'DESC')->paginate(10);
        $users = User::all();
        return view ("display", compact('pieza','users'));
    }

    public function filtro(request $id){

           $NOC = $id->get('noc');
           $Categoria = $id->get('categoria');
           $Nombre = $id->get('nombre');
         if ($id->autor!="") {
                 $user= User::where('name','LIKE', '%'.$id->get('autor').'%')->first();
                 $Autor = $user->email;
         } else {
                $Autor = "";
         }
           $pieza=Pieza::where('NOC', 'LIKE', '%'.$NOC.'%')->where('Categoria', 'LIKE',
'%'.$Categoria.'%')->where('NombrePieza', 'LIKE', '%'.$Nombre.'%')->where('Autor',
'LIKE', '%'.$Autor.'%')->paginate(20);
          $users = User::all();
         return view ("display", compact ('pieza','users') );
    }


    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */



    public function create()
    {

      return view("form");

    }
```

```
    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */



    public function store(Request $request)
    {
       $pieza=new Pieza();
       $user=auth()->user();

       $pieza->NOC=$request->get('noc');
       $pieza->NombrePieza=$request->get('nombre');
       $pieza->Autor=$user->email;
       $pieza->Categoria=$request->get('categoria');//Lo que venga del form lo metemos
aquí, como otro campo cualquiera
       $pieza->Descripcion=$request->get('descripcion');
       $pieza->Software=$request->get('software');
       $pathfoto=$request->file('photo')->store('public/fotopiezas');
       $pieza->Foto=$pathfoto;

       $extension = $request->file('file')->getClientOriginalExtension();
       //permitir únicamente ficheros .stl
       if($extension != "stl"){
           echo  "<font color=red> Formato con extensi&oacute;n no autorizado, solo
ficheros con extensi&oacute;n .stl </font>";
           return view("form");
       }
       else{
           $filename = uniqid().'.'.$extension;
           $pathfichero=$request->file('file')->storeAs('public/piezas',$filename);

           $pieza->Pieza=$pathfichero;
           $pieza->save();

           return redirect()->route('perfil');
       }
       //fin control de extensión

    }



    public function guardar(Request $request) {
    echo "sss";
    }
    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
```

```
    public function show(request $request)
    {
          $pieza=Pieza::where('id',$request->id)->first();
          $pieza->Foto=str_replace("public","storage",$pieza->Foto);
          $pieza->Pieza=str_replace("public","storage",$pieza->Pieza);
       $email= User::where('email','LIKE', '%'.$pieza->Autor.'%')->first();
         $user->email=$email;

         return view ("ver", compact ('pieza', 'user'));

    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */



    public function edit($id)
    {
        $pieza= new Pieza();
        $pieza= Pieza::find($id); //Creo una nueva variable del tipo pieza y le digo a
Eloquent que busque el registro con el id recibido
        $pieza->Foto=str_replace("public","storage",$pieza->Foto);
        $pieza->Pieza=str_replace("public","storage",$pieza->Pieza);
        return view ("edit", compact ('pieza'));//le paso a edit la varaible pieza
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */



    public function update(Request $request, Pieza $pieza) //si defino en el parentesis
la clase de la variable no hace falta que haga $pieza= new Pieza(); //Defino también una
metodo request para recibir todo los cambios del formulario
    {

        //Comienzo a cambiar, por un lado recibo la pieza y por otro los cambios (con
$request)
        $pieza->NOC=$request->noc;
        $pieza->NombrePieza=$request->nombre;
        $pieza->Categoria=$request->categoria;
        $pieza->Descripcion=$request->descripcion;
        $pieza->Software=$request->software;
           if ($request->file('photo')!=NULL) {
                  $pieza->Foto=$request->file('photo')->store('public/fotopiezas');
      }
          if ($request->file('file')!=NULL) {
                  $extension = $request->file('file')->getClientOriginalExtension();
                if($extension != "stl"){
```

```
                        echo "<font color=red> Formato con extensi&oacute;n no
autorizado, solo ficheros con extensi&oacute;n .stl</font>";
                        return view ("edit", compact ('pieza'));
            }
             else{
                 $filename = uniqid().'.'.$extension;
                 $pathfichero=$request->file('file')->storeAs('public/piezas',$filename);
                 $pieza->Pieza=$pathfichero;
             }
            }
        $pieza->save();
           return redirect()->route('perfil');

    }




    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */

    public function eliminar($id){
        $pieza= new Pieza();
        $pieza= Pieza::find($id); //Creo una nueva variable del tipo pieza y le digo a
Eloquent que busque el registro con el id recibido
        $pieza->Foto=str_replace("public","storage",$pieza->Foto);
        $pieza->Pieza=str_replace("public","storage",$pieza->Pieza);
        return view ("eliminar", compact ('pieza'));//le paso a la view eliminar la
varaible pieza
    }


    public function destroy(Pieza $pieza) //Le indico que lo que va a recibir es una
variable de la clase pieza.
    {
        $pieza->delete();//mediante este código le indico que debe eliminar de la BBDD
el registro que le llegue en pieza
        return redirect()->route('perfil');
    }
}
```

ANEXO II: VISTAS

inicio.blade.php

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <title> Armada 3D </title>

        <!-- Fonts -->
        <link
href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;600;700&display=swap"
rel="stylesheet">

        <!-- Styles -->
        <style>

          /*! normalize.css v8.0.1 | MIT License | github.com/necolas/normalize.css
*/html{line-height:1.15;-webkit-text-size-adjust:100%}body{margin:0}a{background-
color:transparent}[hidden]{display:none}html{font-family:system-ui,-apple-
system,BlinkMacSystemFont,Segoe UI,Roboto,Helvetica Neue,Arial,Noto Sans,sans-serif,Apple
Color Emoji,Segoe UI Emoji,Segoe UI Symbol,Noto Color Emoji;line-
height:1.5}*,:after,:before{box-sizing:border-box;border:0 solid
#e2e8f0}a{color:inherit;text-decoration:inherit}svg,video{display:block;vertical-
align:middle}video{max-width:100%;height:auto}.bg-white{--bg-opacity:1;background-
color:#fff;background-color:rgba(255,255,255,var(--bg-opacity))}.bg-gray-100{--bg-
opacity:1;background-color:#f7fafc;background-color:rgba(247,250,252,var(--bg-
opacity))}.border-gray-200{--border-opacity:1;border-color:#edf2f7;border-
color:rgba(237,242,247,var(--border-opacity))}.border-t{border-top-
width:1px}.flex{display:flex}.grid{display:grid}.hidden{display:none}.items-center{align-
items:center}.justify-center{justify-content:center}.font-semibold{font-weight:600}.h-
5{height:1.25rem}.h-8{height:2rem}.h-16{height:4rem}.text-sm{font-size:.875rem}.text-
lg{font-size:1.125rem}.leading-7{line-height:1.75rem}.mx-auto{margin-left:auto;margin-
right:auto}.ml-1{margin-left:.25rem}.mt-2{margin-top:.5rem}.mr-2{margin-right:.5rem}.ml-
2{margin-left:.5rem}.mt-4{margin-top:1rem}.ml-4{margin-left:1rem}.mt-8{margin-
top:2rem}.ml-12{margin-left:3rem}.-mt-px{margin-top:-1px}.max-w-6xl{max-width:72rem}.min-
h-screen{min-height:100vh}.overflow-hidden{overflow:hidden}.p-6{padding:1.5rem}.py-
4{padding-top:1rem;padding-bottom:1rem}.px-6{padding-left:1.5rem;padding-right:1.5rem}.pt-
8{padding-top:2rem}.fixed{position:fixed}.relative{position:relative}.top-0{top:0}.right-
0{right:0}.shadow{box-shadow:0 1px 3px 0 rgba(0,0,0,.1),0 1px 2px 0 rgba(0,0,0,.06)}.text-
center{text-align:center}.text-gray-200{--text-
opacity:1;color:#edf2f7;color:rgba(237,242,247,var(--text-opacity))}.text-gray-300{--text-
opacity:1;color:#e2e8f0;color:rgba(226,232,240,var(--text-opacity))}.text-gray-400{--text-
opacity:1;color:#cbd5e0;color:rgba(203,213,224,var(--text-opacity))}.text-gray-500{--text-
opacity:1;color:#a0aec0;color:rgba(160,174,192,var(--text-opacity))}.text-gray-600{--text-
opacity:1;color:#718096;color:rgba(113,128,150,var(--text-opacity))}.text-gray-700{--text-
opacity:1;color:#4a5568;color:rgba(74,85,104,var(--text-opacity))}.text-gray-900{--text-
opacity:1;color:#1a202c;color:rgba(26,32,44,var(--text-opacity))}.underline{text-
decoration:underline}.antialiased{-webkit-font-smoothing:antialiased;-moz-osx-font-
smoothing:grayscale}.w-5{width:1.25rem}.w-8{width:2rem}.w-auto{width:auto}.grid-cols-
1{grid-template-columns:repeat(1,minmax(0,1fr))}@media (min-width:640px){.sm\:rounded-
lg{border-radius:.5rem}.sm\:block{display:block}.sm\:items-center{align-
items:center}.sm\:justify-start{justify-content:flex-start}.sm\:justify-between{justify-
content:space-between}.sm\:h-20{height:5rem}.sm\:ml-0{margin-left:0}.sm\:px-6{padding-
left:1.5rem;padding-right:1.5rem}.sm\:pt-0{padding-top:0}.sm\:text-left{text-
align:left}.sm\:text-right{text-align:right}}@media (min-width:768px){.md\:border-t-
```

```
0{border-top-width:0}.md\:border-l{border-left-width:1px}.md\:grid-cols-2{grid-template-
columns:repeat(2,minmax(0,1fr))}}@media (min-width:1024px){.lg\:px-8{padding-
left:2rem;padding-right:2rem}}@media (prefers-color-scheme:dark){.dark\:bg-gray-800{--bg-
opacity:1;background-color:#2d3748;background-color:rgba(45,55,72,var(--bg-
opacity))}.dark\:bg-gray-900{--bg-opacity:1;background-color:#1a202c;background-
color:rgba(26,32,44,var(--bg-opacity))}.dark\:border-gray-700{--border-opacity:1;border-
color:#4a5568;border-color:rgba(74,85,104,var(--border-opacity))}.dark\:text-white{--text-
opacity:1;color:#fff;color:rgba(255,255,255,var(--text-opacity))}.dark\:text-gray-400{--
text-opacity:1;color:#cbd5e0;color:rgba(203,213,224,var(--text-opacity))}}

            body {
                font-family: 'Nunito';
                background-image: url
('/fotopiezas/f0yrUZ4ODxIr9WwiP5klcd87mwhTsydyflQ1qhLJ.jpg');

            }

            .fondo {
                background-image: url("https://pbs.twimg.com/media/DgiJsKxXcAAqIKG.jpg");
                  height: 1000px; /* You must set a specified height */
                  background-position: center; /* Center the image */
                  background-repeat: no-repeat; /* Do not repeat the image */
                  background-size: cover; /* Resize the background image to cover the
entire container */
            }

            button{
                border-color: black !important;
                border-radius: 5px;
                border: 2px solid;
                padding: 15px;
                font-size: 20px;
                text-decoration-style: solid;
                background-color:#66b3ff !important;
                color: #ffffff;
            }
           </style>
    </head>
    <body>
    <div class="principal">
      <div class="header" align="center">
            <h1>Sistema de gesti&oacute;n de repuestos por impresi&oacute;n 3D de la
Armada</h1>
        </div>
            <div class="fondo">
            <div class="flex items-top justify-center min-h-screen sm:items-center sm:pt-
0">
                @if (Route::has('login'))
                    <div class="container md justify-center">
                        @auth
                        <button type="submit" class="buttom"><a href="{{ url('/dashboard')
}}" class="buttom">Gesti&oacute;n de repuestos</a></button>
                        @else
                        <button type="submit" class="buttom"><a href="{{ route('login')
}}" class="buttom">Acceso</a></button>

                            @if (Route::has('register'))
                        <button type="submit" class="buttom"><a href="{{ route('register')
}}" class="buttom">Registro</a></button>
                            @endif
                        @endauth
```

```
                    </div>
                @endif

            </div>
            </div>
        </div>
    </body>
</html>
```

menu.blade.php

```
<!DOCTYPE html>
<html lang="Es">
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <!-- Fonts -->
        <link
href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;600;700&display=swap"
rel="stylesheet">

        <!-- Styles -->
        <style>

        .navegar {
          width: 200px;
          height: 600px;
          float:left;
          margin-right:50px;
       }

        .navegar a {
          background-color: #4da6ff;
          color: black;
          display: block;
          padding: 12px;
          text-decoration: none;
        }

        .navegar a:hover {
          background-color: #ccc;
        }
        .escudo{
          margin-left: 20px;
        }

           /*! normalize.css v8.0.1 | MIT License | github.com/necolas/normalize.css
*/html{line-height:1.15;-webkit-text-size-adjust:100%}body{margin:0}a{background-
color:transparent}[hidden]{display:none}html{font-family:system-ui,-apple-
system,BlinkMacSystemFont,Segoe UI,Roboto,Helvetica Neue,Arial,Noto Sans,sans-serif,Apple
Color Emoji,Segoe UI Emoji,Segoe UI Symbol,Noto Color Emoji;line-
height:1.5}*,:after,:before{box-sizing:border-box;border:1 solid
#e2e8f0}a{color:inherit;text-decoration:inherit}svg,video{display:block;vertical-
align:middle}video{max-width:100%;height:auto}.bg-white{--bg-opacity:1;background-
color:#fff;background-color:rgba(255,255,255,var(--bg-opacity))}.bg-gray-100{--bg-
opacity:1;background-color:#f7fafc;background-color:rgba(247,250,252,var(--bg-
opacity))}.border-gray-200{--border-opacity:1;border-color:#edf2f7;border-
color:rgba(237,242,247,var(--border-opacity))}.border-t{border-top-
width:1px}.flex{display:flex}.grid{display:grid}.hidden{display:none}.items-center{align-
items:center}.justify-center{justify-content:center}.font-semibold{font-weight:600}.h-
5{height:1.25rem}.h-8{height:2rem}.h-16{height:4rem}.text-sm{font-size:.875rem}.text-
lg{font-size:1.125rem}.leading-7{line-height:1.75rem}.mx-auto{margin-left:auto;margin-
right:auto}.ml-1{margin-left:.25rem}.mt-2{margin-top:.5rem}.mr-2{margin-right:.5rem}.ml-
2{margin-left:.5rem}.mt-4{margin-top:1rem}.ml-4{margin-left:1rem}.mt-8{margin-
top:2rem}.ml-12{margin-left:3rem}.-mt-px{margin-top:-1px}.max-w-6xl{max-width:72rem}.min-
h-screen{min-height:100vh}.overflow-hidden{overflow:hidden}.p-6{padding:1.5rem}.py-
4{padding-top:1rem;padding-bottom:1rem}.px-6{padding-left:1.5rem;padding-right:1.5rem}.pt-
8{padding-top:2rem}.fixed{position:fixed}.relative{position:relative}.top-0{top:0}.right-
0{right:0}.shadow{box-shadow:0 1px 3px 0 rgba(0,0,0,.1),0 1px 2px 0 rgba(0,0,0,.06)}.text-
center{text-align:center}.text-gray-200{--text-
```

```
opacity:1;color:#edf2f7;color:rgba(237,242,247,var(--text-opacity))}.text-gray-300{--text-
opacity:1;color:#e2e8f0;color:rgba(226,232,240,var(--text-opacity))}.text-gray-400{--text-
opacity:1;color:#cbd5e0;color:rgba(203,213,224,var(--text-opacity))}.text-gray-500{--text-
opacity:1;color:#a0aec0;color:rgba(160,174,192,var(--text-opacity))}.text-gray-600{--text-
opacity:1;color:#718096;color:rgba(113,128,150,var(--text-opacity))}.text-gray-700{--text-
opacity:1;color:#4a5568;color:rgba(74,85,104,var(--text-opacity))}.text-gray-900{--text-
opacity:1;color:#1a202c;color:rgba(26,32,44,var(--text-opacity))}.underline{text-
decoration:underline}.antialiased{-webkit-font-smoothing:antialiased;-moz-osx-font-
smoothing:grayscale}.w-5{width:1.25rem}.w-8{width:2rem}.w-auto{width:auto}.grid-cols-
1{grid-template-columns:repeat(1,minmax(0,1fr))}@media (min-width:640px){.sm\:rounded-
lg{border-radius:.5rem}.sm\:block{display:block}.sm\:items-center{align-
items:center}.sm\:justify-start{justify-content:flex-start}.sm\:justify-between{justify-
content:space-between}.sm\:h-20{height:5rem}.sm\:ml-0{margin-left:0}.sm\:px-6{padding-
left:1.5rem;padding-right:1.5rem}.sm\:pt-0{padding-top:0}.sm\:text-left{text-
align:left}.sm\:text-right{text-align:right}}@media (min-width:768px){.md\:border-t-
0{border-top-width:0}.md\:border-l{border-left-width:1px}.md\:grid-cols-2{grid-template-
columns:repeat(2,minmax(0,1fr))}}@media (min-width:1024px){.lg\:px-8{padding-
left:2rem;padding-right:2rem}}@media (prefers-color-scheme:dark){.dark\:bg-gray-800{--bg-
opacity:1;background-color:#2d3748;background-color:rgba(45,55,72,var(--bg-
opacity))}.dark\:bg-gray-900{--bg-opacity:1;background-color:#1a202c;background-
color:rgba(26,32,44,var(--bg-opacity))}.dark\:border-gray-700{--border-opacity:1;border-
color:#4a5568;border-color:rgba(74,85,104,var(--border-opacity))}.dark\:text-white{--text-
opacity:1;color:#fff;color:rgba(255,255,255,var(--text-opacity))}.dark\:text-gray-400{--
text-opacity:1;color:#cbd5e0;color:rgba(203,213,224,var(--text-opacity))}} */

        </style>
        <meta charset="UTF-8">
        <title>Armada Impresion 3D</title>
</head>
<body>
        <div class="header" align="center">
        <h1>Sistema de gesti&oacute;n de repuestos por impresi&oacute;n 3D de la
Armada</h1>
        </div>

        <div class="contenedor">
        <div align="left" id="navegador" class="navegar">
                    <a href="/dashboard">Inicio</a>
                    <a href="/pieza">Buscar</a>
                    <a href="/pieza/create">Subir</a>
                    <a href="/perfil">Perfil</a>
                    <a href="/contacto">Contacto</a>
                    <form method="POST" action="{{ route('logout') }}">
                        @csrf
                        <x-responsive-nav-link :href="route('logout')"
                              onclick="event.preventDefault();
                                       this.closest('form').submit();">
                            {{ __('Logout') }}
                        </x-responsive-nav-link>
                    </form></li>
        </div>

        @yield('contenido')

        </div>
</body>
</html>
```

principal.blade.php

```
@extends ('menu')
@section ('contenido')

<section>
<article>
<p>Bienvenido al sistema de gesti&oacute;n de repuestos por impresi&oacute;n 3D de la Armada. Esta aplicaci&oacute;n web le va a permitir descargar y compartir los dise&ntilde;os para impresi&oacute;n 3D de piezas de elementos ubicados en buques y unidades de la Armada. </p>
<p>Esta herramienta pone al alcance de todas las unidades los modelos de piezas generados con software especializado para impresi&oacute;n 3D, de tal forma que, si resulta imposible conseguir un repuesto mediante los m&eacute;todos tradicionales, se pueda recurrir a esta herramienta para comprobar si existe su modelo an&aacute;logo para ser impreso. En esta aplicaci&oacute;n web, los usuarios podr&aacute;n navegar entre una amplia colecci&oacute;n de modelos de repuestos y piezas desarrollados para la impresi&oacute;n 3D.</p>
<p>Cualquier usuario puede publicar sus dise&ntilde;os y compartirlos con el resto de unidades que pudiesen demandar dise&ntilde;os similares. &Uacute;nicamente se requiere publicar los dise&ntilde;os de las piezas en formato .stl, dando a cada una el NOC que le corresponde para ser identificada en SIGAPEA y en SIGMAWEB. Adem&aacute;s, cada vez que se a&ntilde;ada una pieza se deber&aacute;n completar una serie de datos como son el nombre de la pieza, el destino al que pertenece, el software empleado para el dise&ntilde;o del modelo, una descripci&oacute;n as&iacute; como suministrar una imagen de la pieza.</p>
<p>Adicionalmente, el propietario de cada modelo de pieza (el usuario que lo publica) puede gestionar sus propios modelos de piezas, editarlos y eliminar cada registro seg&uacute;n convenga.</p>
<br>
</article>
<article>
<b>Sobre la impresi&oacute;n 3D y la Armada 4.0</b>
<br>

<table>
</tr>
<td>&iquest;C&oacute;mo afecta la impresi&oacute;n 3D a la cadena log&iacute;stica?</td>
<td>&iquest;Qu&eacute; es la impresi&oacute;n 3D?</td>
<tr>
<td><iframe width="480" height="320"
src="https://www.youtube.com/embed/AUmml1CUusA?start=26" frameborder="0"
allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>
<td><iframe width="480" height="320" src="https://www.youtube.com/embed/AYRDVMZ3TbY"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media;
gyroscope; picture-in-picture" allowfullscreen></iframe>
</tr>

</table>
</article>
</section>

@endsection ('contenido')
```

display.blade.php

```
@extends ('menu')

<style>
    .tablaGeneral{
    table-layout:fixed;
    width: 80%;
    }
    .tablaGeneral td{
    border-top: 1px solid black;
    border-collapse: collapse;
    }

    .boton{
    border-color: black !important;
    border-radius: 5px;
    border: 2px solid;
    padding: 5px;
    font-size: 15px;
    text-decoration-style: solid;
    background-color:#66b3ff !important;
    color: #ffffff;
    }

    .marcapaginas{
    margin-left: 250px;  /* asi se me centra*/

    }

</style>


@section ('contenido')

<body>
      <div class="container">
            <div class="row">
                  <div class="col-md-12">
                        <div class="page-header">
                        <h2>Busqueda de piezas</h2>
                              <form action="/search" method='GET' class='form-inline
pull-right'>
                              <div class="form-group">
                        <div class="form-group row">
                          <label for="lNOC" class="col-sm-2 col-form-label">NOC:
</label>
                                          <div class="col-sm-10">
                             <input type="number" class="form-control" placeholder="NOC"
name="noc" id="noc">
                          </div>
                        </div>

                        <div class="form-group row">
                          <label for="lcategoria" class="col-sm-2 col-form-
label">Categor&iacute;a: </label>
                                          <div class="col-sm-10">
                             <select placeholder="Categoria" name='categoria'
id='categoria'>
                             <option value=""></option>
                             <option value="Maquinas">M&aacute;quinas</option>
```

```
                                <option value="Puente">Puente</option>
                                <option value="CIC">CIC</option>
                                <option value="Otros">Otros</option>

                                </select>
                              </div>
                            </div>

                            <div class="form-group row">
                              <label for="lnombre" class="col-sm-2 col-form-label">Nombre de
la pieza: </label>
                                                  <div class="col-sm-10">
                              <input type="text" class="form-control" placeholder="Nombre"
name="nombre" id="nombre">
                              </div>
                            </div>

                            <div class="form-group row">
                              <label for="lautor" class="col-sm-2 col-form-label">Autor:
</label>
                                                  <div class="col-sm-10">
                                <input type="text" class="form-control" placeholder="Autor"
name="autor" id="autor">
                              </div>
                            </div>

                            <br>
                            <div class="form-group row">
                              <button type="submit" class="btn btn-primary">Filtrar</button>
                            </div>
                          </div>
                                    </form>
                                </div>
                        </div>
                        <div align="center" class="col-md-12">
<!--                            <table class="table table-hover table-striped"> -->
                            <table class="tablaGeneral">
                                  <tbody>
                                  <tr>
                                        <th>NOC</th>
                                        <th>Nombre Pieza</th>
                                        <th>Autor</th>
                                        <th>Destino</th>
                                  </tr>
                                  @foreach ($pieza as $piezas)
                                  <tr>
<!--                                      <td>{{$piezas->id}}</td> -->
                                        <td>{{$piezas->NOC}}</td>
                                        <td>{{$piezas->NombrePieza}}</td>
                                        <td>{{$piezas->Autor}}</td>
                                        <td>{{$piezas->Categoria}}</td>
                                        <td><form action="/ver" method= 'GET'>
                                              <input type="hidden" value="{{$piezas->id}}"
name="id">

                                              <input type="hidden" value="{{$piezas->NOC}}"
name="noc">

                                              <input type="hidden" value="{{$piezas-
>NombrePieza}}" name="nombrepieza">

                                              <input type="hidden" value="{{$piezas-
>Autor}}" name="autor">
```

```
                                        <input type="hidden" value="{{$piezas-
>Categoria}}" name="categoria">
                                        <input type="hidden" value="{{$piezas-
>Descripcion}}" name="descripcion">
                                        <input type="hidden" value="{{$piezas-
>Software}}" name="software">
                                        <input type="hidden" value="{{$piezas-
>Foto}}" name="foto">

                                        <input type="hidden" value="{{$piezas-
>Pieza}}" name="pieza">
                                        <input type="submit" class="boton"
value="Ver"></form></td> <!--así mando el id a /ver que a su vez lo manda al controlador
show*/ -->
                                </tr>
                                @endforeach
                                <tr><div class= "marcapaginas">{{$pieza-
>render()}}</div></tr>

                                </tbody>

                            </table>
<!--                            <div class= "marcapaginas">{{$pieza->render()}}</div> -->
                    </div>
                </div>
        </div>

</body>

@endsection ('contenido')
```

form.blade.php

```
@extends ('menu')

@section ('contenido')

<h2>Subir pieza</h2>
<form action="/guardar"   method="post" enctype="multipart/form-data">
@csrf
  <div class="form-group row">
    <label for="lNOC" class="col-sm-2 col-form-label">NOC</label>
    <div class="col-sm-10">
      <input type="number" class="form-control" name="noc" id="noc" placeholder="NOC"
min="1000000000000" max="9999999999999" required>
    </div>
  </div>

  <div class="form-group row">
    <label for="lnombre" class="col-sm-2 col-form-label">Nombre de la pieza</label>
    <div class="col-sm-10">
      <input type="text" class="form-control" name="nombre" id="nombre"
placeholder="Nombre" required>
    </div>
  </div>

   <div class="form-group row">
    <label for="lcategoria" class="col-sm-2 col-form-label">Categor&iacute;a:</label>
    <div class="col-sm-10">
      <select name='categoria' id='categoria' required>
      <option value="Maquinas">M&aacute;quinas</option>
      <option value="Puente">Puente</option>
      <option value="CIC">CIC</option>
      <option value="Otros">Otros</option>
      </select>
    </div>
  </div>


  <div class="form-group row">
    <label for="ldescripcion" class="col-sm-2 col-form-label">Descripci&oacute;n:</label>
    <div class="col-sm-10">
      <textarea rows="5" class="form-control" name="descripcion" id="descripcion"
placeholder="Descripci&oacute;n" required></textarea>
    </div>
  </div>

  <div class="form-group row">
    <label for="lautor" class="col-sm-2 col-form-label">Software</label>
    <div class="col-sm-10">
      <input type="text" class="form-control" name="software" id="software"
placeholder="Software" required>
    </div>
  </div>

  <div class="form-group row">
    <label for="lfoto" class="col-sm-2 col-form-label">Foto de la pieza</label>
    <div class="col-sm-10">
             <input type="file" accept="image/*" name="photo" id="photo" required>
    </div>
  </div>

  <div class="form-group row">
```

```
    <label for="lsoftware" class="col-sm-2 col-form-label">Archivo</label>
    <div class="col-sm-10">
            <input type="file" name="file" placeholder="Escoge fichero" id="file"
required>
    </div>
  </div>

<br>
  <div class="form-group row">
    <div class="col-sm-10">
      <button type="submit" class="btn btn-primary">Guardar</button>
    </div>
  </div>
  <br>
  <br>
</form>

@endsection ('contenido')
```

ver.blade.php

```
@extends ('menu')

        <style>
        .descargar{
            border-color: black !important;
            border-radius: 5px;
            border: 2px solid;
            padding: 2px;
            text-decoration-style: solid;
            background-color:#66b3ff !important;
            color: #ffffff;
        }
        </style>

<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link
href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;600;700&display=swap"
rel="stylesheet">

        <style>


           table, td, th{

                margin-left: auto;  /* asi se me centra*/
                margin-right: auto;
                margin-bottom: 100px;
                border: 1px solid black;
                border-collapse: collapse;
                pading:30px;
                pading-right:30px;
                pading-top:10px;
                pading-bottom:10px;
            }

            .centrado{
            text-align: center;
            }

        </style>
</head>
@section ('contenido')
<body>

      <table style="width: 75%">
            <tr>
                  <th>NOC</th>
                  <td>{{$pieza->NOC}}</td>
                  <th>Nombre pieza</th>
                  <td>{{$pieza->NombrePieza}}</td>
                  <th>Destino</th>
                  <td>{{$pieza->Categoria}}</td>
                  <th>Software</th>
                  <td>{{$pieza->Software}}</td>
            </tr>
            <tr>
                  <th>Autor</th>
```

```
                        <td>{{$user->name}}</td>
                        <th>Contacto</th>
                        <td colspan="5">{{$user->email}}</td>
                </tr>
                <tr>
                        <td class="centrado" colspan="10"><img alt="No hay foto disponible"
src="{{$pieza->Foto}}" align="center" width="500px" height="auto"></td>
                </tr>
                <tr height="35px">
                        <th>Descripcion</th>
                        <td colspan="4">{{$pieza->Descripcion}}</td>
                        <td colspan="6" class="centrado"><a class="descargar" href="{{$pieza-
>Pieza}}" download>Descargar</a> </td>
                </tr>

        </table>
</body>
@endsection ('contenido')
```

perfil.blade.php

```
@extends ('menu')
<head>
       <style>
       .usuario{
        margin-right: auto;
        pading:10px;
       }

    .tablaPiezasUsuario{
    padding-top:20px;
    table-layout:fixed;
    width: 80%;
    }
    .tablaPiezasUsuario td{
    border-top: 1px solid black;
    border-collapse: collapse;
    }
    .boton{
    border-color: black !important;
    border-radius: 5px;
    border: 2px solid;
    padding: 5px;
    font-size: 15px;
    text-decoration-style: solid;
    background-color:#66b3ff !important;
    color: #ffffff;
    }

    .boton2{
    margin-top: 1px;
    border-color: black !important;
    border-radius: 5px;
    border: 2px solid;
    padding: 5px;
    font-size: 15px;
    text-decoration-style: solid;
    background-color:#66b3ff !important;
    color: #ffffff;
    }

</style>
       </style>
</head>
@section ('contenido')
<body>
    <h2>Perfil del usuario</h2>
    <table class="usuario">
      <tbody>
      <tr>
            <td><b>Usuario registrado n&deg;: </td>
            <td>{{$user->id}}</td>
      </tr>
      <tr>
            <td><b>Nombre del usuario: </td>
            <td>{{$user->name}}</td>
      </tr>
      <tr>
            <td><b>Email: </td>
            <td>{{$user->email}}</td>
      </tr>
```

```
      <tr>
            <td><b>Piezas del usuario:</b></td>
      </tr>
      </tbody>
    </table>

      <table class="tablaPiezasUsuario">
            <tbody>
                  <tr>
                        <th>NOC</th>
                        <th>Nombre Pieza</th>
                        <th>Destino</th>
                  </tr>
                        @foreach ($piezas as $pieza)
                        <tr>
                              <td>{{$pieza->NOC}}</td>
                              <td>{{$pieza->NombrePieza}}</td>
                              <td>{{$pieza->Categoria}}</td>
                              <td><form action="/ver" method= 'GET'>
                                    <input type="hidden" value="{{$pieza->id}}"
name="id">
                                    <input type="hidden" value="{{$pieza->NOC}}"
name="noc">
                                    <input type="hidden" value="{{$pieza-
>NombrePieza}}" name="nombrepieza">
                                    <input type="hidden" value="{{$pieza->Autor}}"
name="autor">
                                    <input type="hidden" value="{{$pieza->Categoria}}"
name="categoria">
                                    <input type="hidden" value="{{$pieza-
>Descripcion}}" name="descripcion">
                                    <input type="hidden" value="{{$pieza->Software}}"
name="software">
                                    <input type="hidden" value="{{$pieza->Foto}}"
name="foto">
                                    <input type="hidden" value="{{$pieza->Pieza}}"
name="pieza"></td>
                              <td><input type="submit" value="Ver"
class="boton"></form></td>
                              <td><button type="submit" class="boton"><a
href="{{route('editar', $pieza->id)}}">Editar</a></button> <!-- Mando el id de cada pieza
a la ruta al pinchar en el enlace Editar, se recibe en el controlador edit -->
                              <td><button type="submit" class="boton"><a
href="{{route('eliminarControl', $pieza->id)}}">Eliminar</a></button> <!-- Mando el id de
la pieza a una vista de control para eliminar la pieza -->
                        </tr>
                        @endforeach

            </tbody>

      </table>
</body>
@endsection ('contenido')
```

edit.blade.php

```
@extends ('menu')

@section ('contenido')

<h2>Editar pieza</h2>
<form action="{{route('actualizar', $pieza)}}"   method="post" enctype="multipart/form-data">

@csrf

@method('put') <!-- html no entiende el método put, por ello se usa la directiva method para indicarle que la ruta que queremos usar es aquella definida en web.php con el metodo put -->

  <div class="form-group row">
    <label for="lNOC" class="col-sm-2 col-form-label">NOC</label>
    <div class="col-sm-10">
      <input type="number" class="form-control" name="noc" id="noc" value="{{$pieza->NOC}}" min="1000000000000" max="9999999999999" required>
    </div>
  </div>

  <div class="form-group row">
    <label for="lnombre" class="col-sm-2 col-form-label">Nombre de la pieza</label>
    <div class="col-sm-10">
      <input type="text" class="form-control" name="nombre" id="nombre" value="{{$pieza->NombrePieza}}" required>
    </div>
  </div>


   <div class="form-group row">
    <label for="lcategoria" class="col-sm-2 col-form-label">Categor&iacute;a:</label>
    <div class="col-sm-10">
      <select name='categoria' id='categoria' value="{{$pieza->Categoria}}"required>
      <option value="Maquinas">M&aacute;quinas</option>
      <option value="Puente">Puente</option>
      <option value="CIC">CIC</option>
      <option value="Otros">Otros</option>
      </select>
    </div>
  </div>


  <div class="form-group row">
    <label for="ldescripcion" class="col-sm-2 col-form-label">Descripci&oacute;n:</label>
    <div class="col-sm-10">
      <textarea rows="5" class="form-control" name="descripcion" id="descripcion" required>{{$pieza->Descripcion}}</textarea>
    </div>
  </div>

  <div class="form-group row">
    <label for="lautor" class="col-sm-2 col-form-label">Software</label>
    <div class="col-sm-10">
      <input type="text" class="form-control" name="software" id="software" value="{{$pieza->Software}}"  required>
    </div>
  </div>
```

```
  <div class="form-group row">
    <label for="lfoto" class="col-sm-2 col-form-label">Foto de la pieza (Nota: Para
mantener la foto, no seleccionar nada)</label>
    <div class="col-sm-10">
            <input type="file" name="photo" id="photo" accept="image/*" value="{{$pieza-
>Foto}}">
    </div>
  </div>

  <div class="form-group row">
    <label for="lsoftware" class="col-sm-2 col-form-label">Archivo (Nota: Para mantener el
archivo .stl, no seleccionar nada)</label>
    <div class="col-sm-10">
            <input type="file" name="file" placeholder="Escoge fichero" id="file"
value="{{$pieza->Pieza}}">
    </div>
  </div>

<br>
  <div class="form-group row">
    <div class="col-sm-10">
      <button type="submit" class="btn btn-primary">Guardar</button>
    </div>
  </div>
  <br>
  <br>
</form>

@endsection ('contenido')
```

eliminar.blade.php

```
@extends ('menu')

@section('contenido')

<body>
&iquest;Seguro que desea eliminar esta pieza?
<form action="{{route('eliminar', $pieza)}}" method="POST">
@csrf
@method('delete')
<button type="submit">Eliminar</button>
</form>
<form action="{{route('perfil')}}" method="GET">
@csrf
<button type="submit">Cancelar</button>
</form>

</body>



@endsection('contenido')
```