TRABAJO FIN DE GRADO

INGENIERÍA EN INFORMÁTICA

Sistema de librerías de componentes para ALTIUM en Docker

Autor
Juan Carlos Hermoso Quesada

Directores
Andrés Roldán Aranda

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 17 de noviembre de 2021

Sistema de librerías de componentes para ALTIUM en Docker

Autor
Juan Carlos Hermoso Quesada

Directores
Andrés Roldán Aranda

Sistema de librerías de componentes para ALTIUM en Docker

Juan Carlos Hermoso Quesada

Palabras clave: PartKeepr, Altium, Docker, contenedor, imagen, PCB, MySQL, MariaDB, Github, Gitlab, Linux, servidor, GranaSAT, Open Source, Ubuntu, migración, HTTP, PHP, Symfony2, Javascript

Resumen

Proyecto software que consiste en levantar una aplicación de código abierto mediante Docker la cual va a consistir en un sistema de librerías de componentes que permita controlar el stock de componentes electrónicos en el diseño de placas de circuito y que permita su acceso remoto a partir de su instalación en un servidor

Inventory management software for Altium developed in Docker

Juan Carlos Hermoso Quesada

Keywords: PartKeepr, Altium, Docker, container, image, PCB, MySQL, MariaDB, Github, Gitlab, Linux, servidor, GranaSAT, Open Source, Ubuntu, migration, HTTP, PHP, Symfony2, Javascript

Abstract

Software project which consists in starting an open source application through Docker which consists in an inventory management software which allows to control the stock of electronic components for PCB design and which allows remote access due to its installation on a server.

Yo, **Juan Carlos Hermoso Quesada**, alumno de la titulación INGENIERÍA INFORMÁTICA de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 41512475M, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Juan Carlos Hermoso Quesada

Granada a 17 de noviembre de noviembre de 2021

D. **Andrés Roldan Aranda** (**tutor1**), Profesor del Departamento Electrónica y Tecnología de Computadores de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado **Sistema de librerías de componentes para ALTIUM en Docker** ha sido realizado bajo su supervisión por Juan Carlos Hermoso Quesada (alumno), y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda

Y para que conste, expiden y firman el presente informe en Granada a 17 de noviembre de 2021

Los directores:

Andrés Roldán Aranda (tutor1)

Agradecimientos

A mis padres, a mi tutor y a mis compañeros M. y A.

ÍNDICE

GLOSARIO

- PartKeepr: sistema de librerías de componentes en código abierto y escrito en PHP, MySQL y Javascript.
- Docker: herramienta de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores.
- Docker Composer: herramienta de Docker encargada de crear aplicaciones multi-contenedor.
- Imagen Docker: instancia de la configuración/estado de un contenedor.
- Contenedor Docker: paquete de código y dependencias de una aplicación que permite una ejecución rápida y fiable.
- Sistema de librerías de componentes (SLC): programa encargado de archivar componentes electrónicos mediante una serie de características propias.
- HTTP: protocolo de comunicación mediante archivos en la red.
- GranaSAT: grupo de electrónica aeroespacial de la UGR.
- Base de datos: conjunto de datos almacenados bajo un o varios contextos.
- Servidor: computador/as disponible/s para atender a las peticiones de un cliente.
- Código abierto: código fuente de software de dominio público el cual se puede utilizar y redistribuir de forma gratuita.
- Symfony2: framework de PHP diseñado para crear aplicaciones basado en el Modelo Vista Controlador.
- PHP: lenguaje de programación dedicado mayormente a desarrollo web desde el lado del servidor.
- phpMyAdmin: aplicación web escrita en PHP que se encarga de la administración de bases de datos MySQL.
- Github: plataforma que usa Git para compartir código fuente de proyectos software.
- Altium: software de diseño de PCBs.
- MySQL: sistema de gestión de base de datos en código abierto. Es el más utilizado del mundo.
- MariaDB: sistema de gestión de base de datos basado en MySQL.

- Javascript: lenguaje de programación interpretado del lado del cliente, más concretamente del navegador web.
- Microservicios: arquitectura que divide una aplicación en pequeños procesos independientes.

1. INTRODUCCIÓN

En este documento se va a documentar el proyecto de fin de carrera que me fue asignado a través de mi tutor, Andrés Roldán Aranda, del grupo GranaSAT, el grupo de investigación de electrónica aeroespacial de la UGR.

El proyecto consiste en la implementación y adaptación de un sistema de librerías de componentes para los alumnos de la UGR que precisen de él a la hora de diseñar placas de circuito impreso. El entorno está desarrollado dentro de contenedores Docker (docker-compose, más concretamente), los cuales usan imágenes de software libre del repositorio de imágenes de Docker Hub como componentes del docker-compose que serán desplegados cada uno en su respectivo contenedor.

La intención es que el software disponga de las suficientes funcionalidades y aspectos para poder solventar el problema del control del stock de componentes y que los componentes introducidos en la base de datos mediante el software se vean claramente representados por este, con sus campos y características adecuados para ello. También, se busca que el programa esté adaptado para el programa Altium.

También debe proporcionar las funcionalidades para que los alumnos puedan registrarse, identificarse, introducir nuevos componentes o modificarlos y la infraestructura back-end debe de poder permitir todas estas acciones.

2. OBJETIVOS

Los objetivos de este proyecto son varios, algunos ya han sido explicados pero es importante volver a mencionarlos para una mejor comprensión:

- Proporcionar a los estudiantes una aplicación adaptada a las funcionalidades que precisen un control adecuado del stock de componentes
- Modificar el software mediante nuevas funcionalidades para así adaptarlo adecuadamente a los estudiantes y a los usuarios en general.
- Permitir el uso de la aplicación únicamente a usuarios autorizados con el fin de salvaguardar la integridad de los datos introducidos
- Utilizar software de código abierto con el fin de ahorrar el coste económico que supondría usar software de código cerrado y de disponer de el amplio abanico de soluciones que proporciona la comunidad de usuarios/desarrolladores
- Estudio/repaso y aplicación práctica de los conocimientos adquiridos durante la carrera, más concretamente los implicados en desarrollo web (front-end y back-end), bases de datos y servidores web
- Conocer nuevas tecnologías, métodos y soluciones para poder abordar los retos que suponen la realización de este proyecto

3. ANÁLISIS

En primer lugar, es necesario determinar el entorno en el cual se va a lanzar el software a utilizar para el fin de este proyecto para supeditar la búsqueda del programa al sistema operativo, características y demás a este entorno.

Quedó claro desde el principio que la aplicación iba a ser lanzada en el servidor de **GranaSAT**, ya que los alumnos que van a utilizar el programa están directamente relacionados con este grupo. Entonces, al conectarme al servidor mediante las credenciales dadas para ello, introduje el comando **lsb_release -a**, el cual da un pequeño resumen sobre la distribución en la cual está desplegado el servidor.

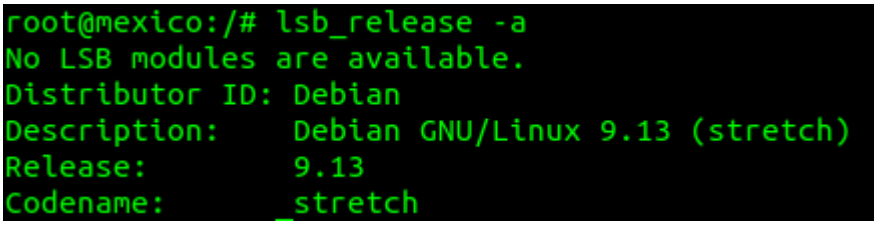


Figura 3.1: Captura de lsb_release -a hecha en el servidor de GranaSAT

Al confirmar que el SO del servidor es **Linux**, más concretamente Debian, decidí, entonces, enfocar mi búsqueda hacia los sistemas de librerías de stock de componentes basados en Linux. Teniendo en cuenta también que el software debía de ser en código abierto, eso reafirmó aún más la necesidad de que la búsqueda fuese de una aplicación en Linux, ya que es bien sabido que Linux es de código abierto.

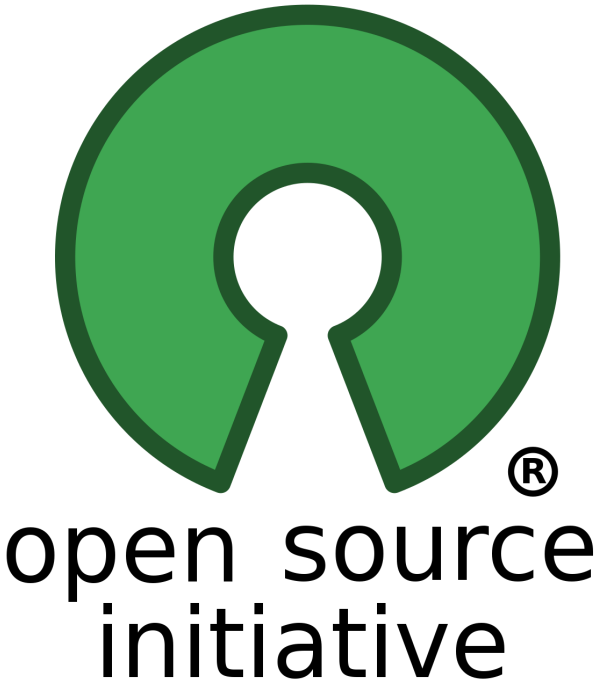


Figura 3.2: Logotipo de Open Source (código abierto), tomada del artículo de Wikipedia sobre código abierto **[20]**

3.1. Búsqueda de sistemas de librerías de componentes

En ésta búsqueda de candidatos he tenido en cuenta los siguientes puntos:

- Código abierto
- Gratuito (o, al menos, para la mayoría de sus funcionalidades)
- Basado en Linux

Realizando la búsqueda me topé con esta página web llamada **GoodFirms**, una página que se dedica a hacer reseñas sobre servicios y productos software. Ésta dispone de una página en la que tiene un artículo que trata sobre los 8 mejores SLC gratis y de código abierto bajo su criterio.

En la propia página muestra una tabla con los distintos softwares que entran dentro de este top 8 pero no todos son gratis del todo, puesto que hay algunos que son gratis sólo los primeros 14 días o sólo tienen una versión de prueba. Además, hay algunos programas que el número de usuarios que permiten es limitado a tan sólo uno y eso era un factor determinante para no escoger dicho software con dicha característica. Al final, quedaron cuatro candidatos, los cuales resumo en en la siguiente tabla:

Nombre	**Descripción**	**Nº Usuarios**	**Nº Productos**
Odoo	Ideal para pequeñas y grandes organizaciones	Ilimitado	Ilimitado
PartKeepr	Ideal para componentes electrónicos	Ilimitado	Ilimitado
ABC Inventory	Ideal para PYMEs	Ilimitado	Ilimitado
Stockpile	Ideal para pequeñas empresas y usuarios individuales	Ilimitado	Ilimitado

3.2. Análisis de los candidatos

Pese a que en la página GoodFirms se indicaba que los softwares **Odoo** y **ABC** Inventory eran gratis del todo, resulta que no era así. Ambos son gratuitos en la mayoría de sus funcionalidades, pero no gratuitos del todo. No sería problema si no fuese porque Odoo tiene como gratis los primeros 15 días funcionalidades esenciales para este proyecto (inventario) que luego pasan a ser de pago y ABC Inventory ofrece un diseño muy rudimentario y anticuado a mi parecer, sin mencionar que no es gratuito del todo. Es por esto que fueron descartados antes de realizar un análisis más profundo de ellos.

Al quedar tan sólo los candidatos **Stockpile** y **PartKeepr** caí en la cuenta de que en la memoria del TFG del alumno Jesús Rodríguez Pérez también se realiza un análisis de ambos softwares y da la casualidad que los dos SLCs que

resultan analizados son estos dos últimos.

Tomando como referencia los resultados de su análisis decidí, entonces, descartar Stockpile, puesto que es un SLC online que no permite modificaciones y ajustes al gusto y eso no es lo que se busca. Además, ya se ha indicado en la tabla anterior que PartKeepr es un SLC especializado y diseñado para componentes electrónicos en el que se ofrece total libertad de uso y modificación, por lo que vi oportuno realizar un examen exhaustivo de la aplicación y del soporte que ésta ofrecía.

A continuación, voy a proceder a realizar una descripción de PartKeepr.

3.2.1. PartKeepr

PartKeepr es un sistema de librerías de componentes en código abierto especializado en componentes electrónicos escrito en **PHP** y que usa **Symfony2**, un framework de PHP diseñado para el desarrollo de aplicaciones web basado en el patrón **MVC** (modelo-vista-controlador).

Entre los requisitos ya mencionados (Linux), se requiere tener instalado PHP **7.0 o 7.1** como mínimo (o superior) y una base de datos **MySQL** (o basada en él) o **PostgreSQL**. La última versión sacada es la **1.4.0**.

La captura de abajo corresponde a una de las dos versiones de prueba. La otra versión de prueba disponible corresponde a la versión más actualizada de la rama master del Github de PartKeepr.

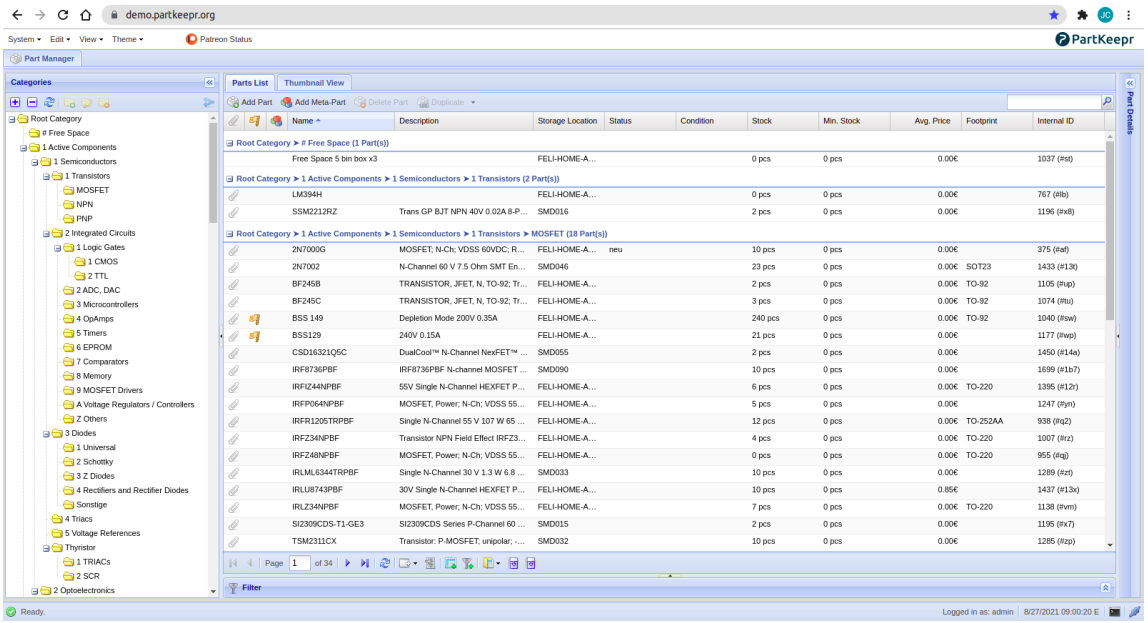
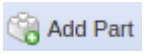


Figura 3.3: Captura de la demo de PartKeepr

3.2.1.1. Profundizando

Siguiendo en la página de la demo y tomando como ejemplo el componente **IRF8736PBF**, podemos ver el gran abanico de campos y funcionalidades que tenemos a la hora de añadir/editar un componente. Para añadirlo basta ir a la pestaña 

que tenemos arriba del menú de componentes ya registrados en la base de datos y para editarlos clicar sobre ellos para que se muestre la ventana de abajo, en la cual se encuentran los campos que definen cada componente.

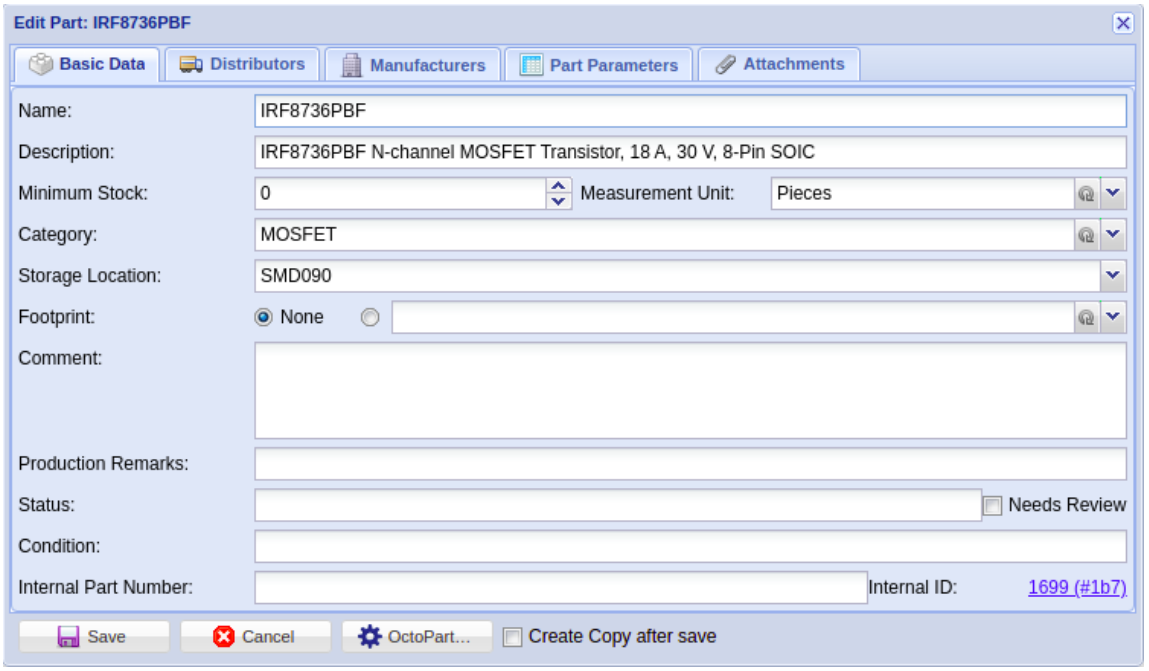


Figura 3.4: Panel Edit Part

Los campos de la pestaña, por ejemplo, **Basic Data**, corresponden a la tabla **Part** de la base de datos de PartKeepr.

Figura 3.5: Filas y columnas de la tabla Part en phpMyAdmin

Al clicar en un componente se despliega también una ventana lateral en la que aparecen otros campos de la fila de la tabla Part que corresponde a este componente. En la misma ventana y más abajo (captura de al lado), aparecen los parámetros físicos del componente y los archivos añadidos sobre el componente, tales como las imágenes y/o PDFs. La primera de las imágenes introducidas será la que se visualice a modo de vista previa del componente.

Cabe añadir que arriba del todo de la imagen podemos gestionar la cantidad de stock de dicho componente, tanto añadir como quitar y que podemos editar

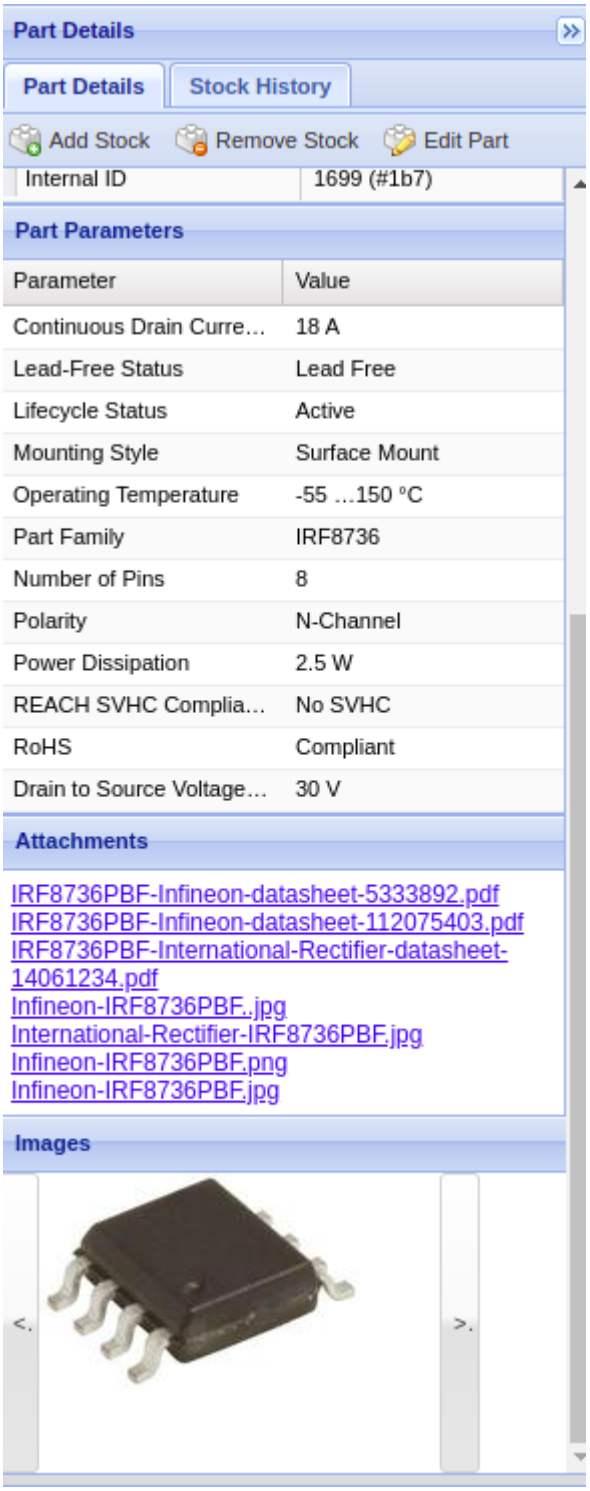


Figura 3.6: Panel lateral Part Details

también el componente.

Volviendo un poco atrás, es de mencionar que podemos tener una visualización de los componentes distinta a la que se nos ofrece de primeras, pudiendo optar por que se nos muestren todos los componentes a partir de su imagen de vista previa ya mencionada. Entonces, si volvemos a la página principal y clicamos en la pestaña **Thumbnail View** podemos obtener dicha visualización, como podemos apreciar en la captura de abajo.

Figura 3.8: Panel de vistas previas de componentes Thumbnail View

Si nos fijamos en la pestaña de arriba del todo llamada View, podremos acceder al panel de información del sistema en el cual está lanzado el PartKeepr (**System Information**).

Part Manager | System Information

Name	Value
Libraries	
Doctrine DBAL	2.5.2
Doctrine ORM	2.5.4
PHP	
allow_url_fopen	1
max_execution_time	30
memory_limit	128M
Metadata Cache Implementation	Doctrine\Common\Cache\ApcCache
post_max_size	8M
Query Cache Implementation	Doctrine\Common\Cache\ApcCache
upload_max_filesize	2M
PartKeepr	
Data Directory	/home/demo.partkeepr.org/PartKeepr/data/files
Disk Space (Free)	17.4 GB
Disk Space (Total)	58.9 GB
Disk Space (Used)	41.6 GB
PartKeepr Version	GIT development version Commit 8a6380e46efc81fd52c650995ad4ab62b7cbcedf Short Commit 8a6380e4
System	
Operating System Release	Debian GNU/Linux 9.13 (stretch)
Operating System Type	Linux
PHP Version	7.1.33-39+0~20210701.57+debian9~1.gbp1cdf2c

Figura 3.9: Panel de información del sistema System Information

Como podemos observar en la captura, tenemos disponibles unos cuantos parámetros sobre el sistema y sobre el propio programa, tales como la versión de éste último o el tipo de sistema operativo en el que se está lanzando y su distribución. Si recordamos lo explicado atrás, veremos que el sistema operativo y su distribución coinciden con los del servidor de GranaSAT, por lo que presupuse que no habría problemas de algún tipo de incompatibilidad o semejante.

Junto a otras funcionalidades no tan notorias pero tenidas en cuenta a la hora de valorar este programa, me decidí por escoger PartKeepr como el SLC que iba a utilizar como objeto de este proyecto.

3.2.1.2. Base de datos

La base de datos está compuesta de **cincuenta tablas**, las cuales se pueden ver (más o menos) en la captura de abajo, realizada mediante la demo de phpMyAdmin tras importar un backup realizado desde dentro del contenedor para una migración posteriormente hecha (procesos explicados detalladamente más adelante):

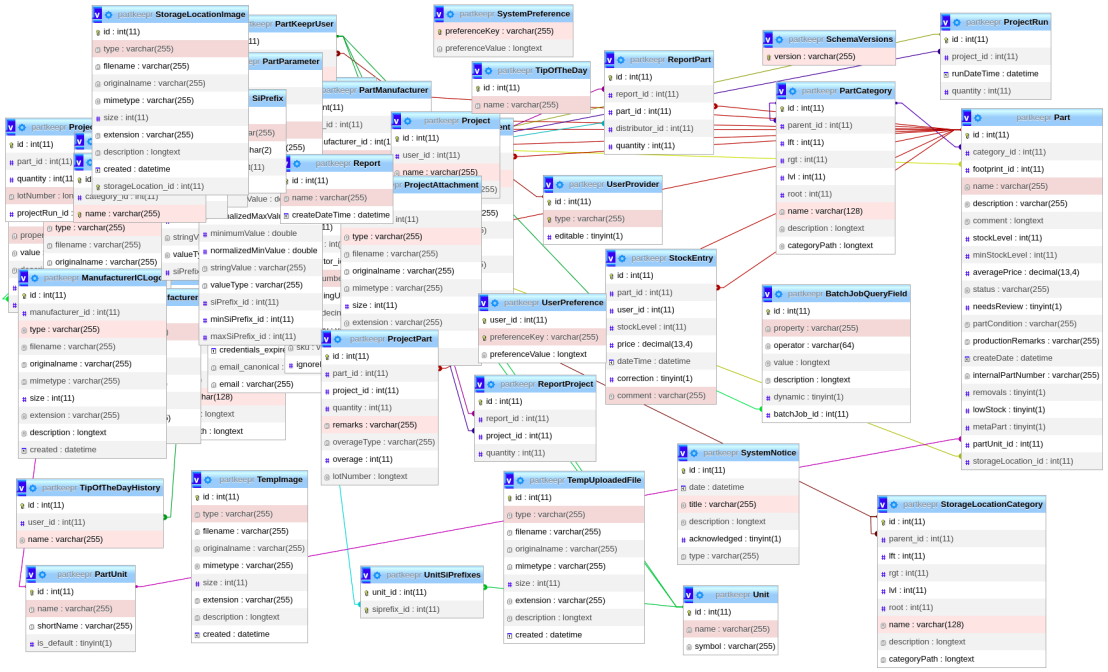


Figura 3.10: Conjunto completo de tablas de PartKeepr en phpMyAdmin

La captura de abajo fue hecha en el contenedor de la base de datos posteriormente a la instalación. En ésta se demuestra que el número de tablas que componen la base de datos es igual a cincuenta.

```
MariaDB [partkeepr]> SELECT COUNT(*) from Information_Schema.Tables where
table_schema = 'partkeepr';
+----------+
| COUNT(*) |
+----------+
|       50 |
+----------+
1 row in set (0.01 sec)
```

Figura 3.11: Captura de SELECT COUNT(*) dentro de la base de datos de PartKeepr

4. DISEÑO Y ARQUITECTURA

Paso a explicar como se ha planeado el diseño del entorno y la arquitectura que tiene ésta. La explicación de la estructura de la base de datos se omite ya que ya se ha explicado y, además, ya venía implementada de serie.

La idea fue desplegar un entorno el cual dispusiese de la **aplicación** y la **base de datos** actuando en conjunto pero por separado, de manera que ambos servicios puedan estar encapsulados y puedan configurarse y modificarse sin afectar a la integridad del otro, lo que corresponde a una arquitectura de **microservicios**. Esto es posible mediante la herramienta **Compose**, que básicamente sirve para implementar **aplicaciones multi-contenedor**. Docker Compose se implementará mediante un archivo llamado **docker-compose.yaml**, cuya implementación se detallará más adelante.

El archivo docker-compose.yaml dispone de una serie de líneas de código por cada servicio/contenedor que se quiera implementar. En estas líneas, estará contenida la imagen de cada contenedor, a la cual se hará un **pull** al repositorio en el caso de no estar la imagen guardada localmente.

La aplicación estará implementada en base a un repositorio local, clon de uno remoto, ya que buscamos una aplicación ya creada y en código abierto. Este repositorio deberá disponer de carpetas de volúmenes asociadas a carpetas de volúmenes dentro de los contenedores. De ésta manera, proporcionamos una réplica de los datos, lo que supone una seguridad de preservación de los datos, y una manera ágil de añadir, mover o borrar datos, ya que al estar conectadas las carpetas podemos modificar los contenidos de las de dentro sin tener que meternos dentro de los contenedores constantemente.

Normalmente, al iniciar cada contenedor manualmente deberíamos indicar en la propia línea de comandos la carpetas que van a enlazarse, no obstante en el archivo de docker-compose.yaml podemos dejarlo indicado, de manera que no haga falta escribirlo cada vez que se quiera levantar el entorno.

Preferentemente, desplegamos dos contenedores, uno para la base de datos y otro para la aplicación, los cuales se usarán de forma separada, lo que incrementa la encapsulación de los contenidos aún más. Los cambios deberán de ser y serán dinámicos tanto en la aplicación como en la base de datos a medida que se vayan realizando o, dicha de otra forma, los cambios serán visibles por pantalla al momento de haberlos realizado (es probable que se necesite refrescar la pantalla o volver a realizar ciertos procesos para poder visualizar los cambios).

Una aproximación al caso real del entorno descrito en los párrafos anteriores podría ser el esquema siguiente:

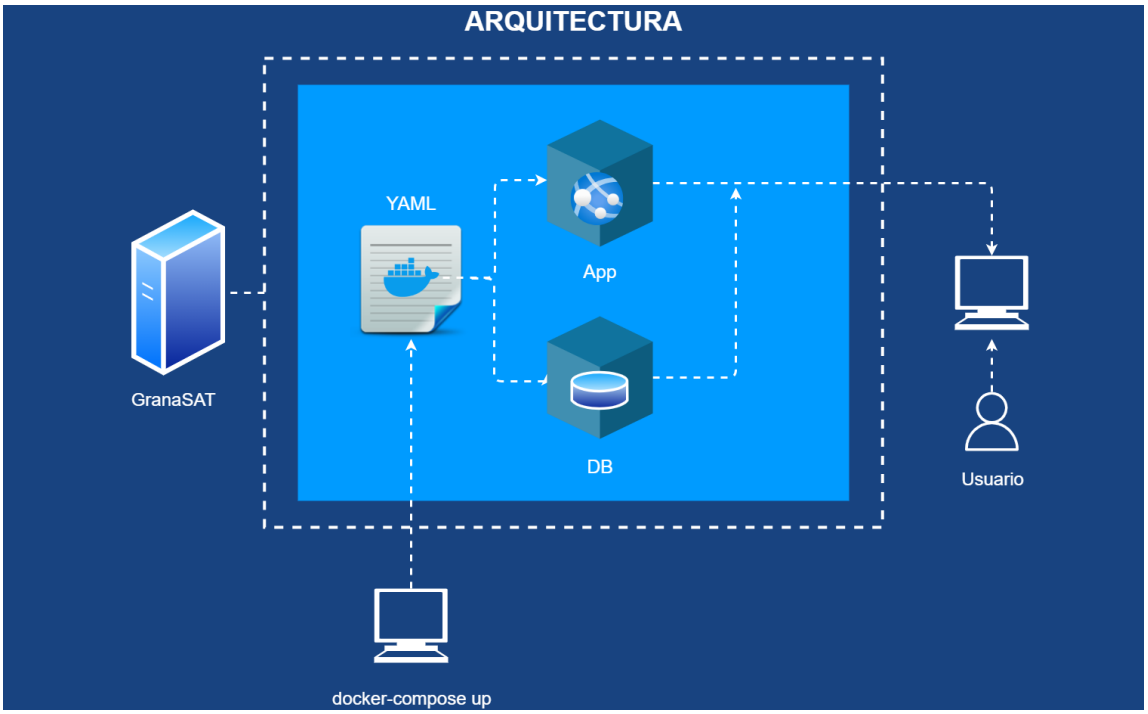


Figura 4.1: Esquema de arquitectura del entorno a implementar

5. IMPLEMENTACIÓN

Ahora, procederé a explicar todo el proceso de implementación que he llevado a cabo a lo largo de todo el proyecto. Las instalaciones y pruebas fueron realizadas previamente en local para trasladar el entorno una vez configurado. Una vez teniendo claro cuál va a ser la arquitectura del entorno, procedemos a la implementación.

5.1. Búsqueda de repositorios

Para empezar, como se decidió diseñar el entorno en base a una arquitectura de **microservicios**, busqué archivos de **docker-compose** que desplegaran el PartKeepr mediante una imagen y una serie de contenedores. Como la instalación mediante docker-compose era más laboriosa a partir de la **imagen original** que desde otra imagen **alternativa**, la cual tenía una instalación más sencilla y rápida, opté en primer lugar por hacer esa.

La instalación correspondía a la del Github de un usuario llamado mhubig. En la página principal, se explica de manera breve y sencilla como realizar la instalación. Cabe añadir que esta imagen utiliza la versión **1.4.0** de PartKeepr, la última. Tomando como base el docker-compose.yml que se encuentra en este Github, se ha creado otro adaptado al servidor de GranaSAT.

Como, finalmente, el proyecto se desarrolló a partir del Github original, he optado por explicar aquí y de forma profunda y extensa la instalación mediante la imagen original. En el **anexo** se encuentran los pasos seguidos para instalación mediante la imagen alternativa (la de **mhubig**). No obstante, es necesario mencionar la función que ha tenido la anterior versión de PartKeepr dentro del proyecto.

Como primeramente se optó por la instalación de la imagen alternativa y la instalación se realizó correctamente, se movió el entorno al servidor de GranaSAT para que ciertos usuarios pudieran utilizar la aplicación en base a sus necesidades. Resultó en una base de datos llena de componentes.

Fue entonces cuando surgió la idea de intentar realizar la instalación mediante la imagen original y después realizar una **migración** de la base de datos y del contenido hacia el nuevo entorno, intentando mantener en la medida de lo posible toda la integridad de los datos del primer entorno.

5.2. Instalación

La instalación procedió siguiendo los pasos de la instalación descrita en el directorio **docker/** del Github original. PartKeepr tiene dos imágenes, pero sólo puede usarse la imagen llamada **partkeepr/development** debido a que, tal y como se indica en la propia página de instalación, la imagen **partkeepr/base-dev** no está lista para ser ejecutada.

Tras hacer una clonación local de la rama master del repositorio (**git clone https://github.com/partkeepr/PartKeepr.git**) se empezó la instalación. La instrucciones están propiamente descritas en el enlace del párrafo anterior, pero como se le han añadido unas cuantas pinceladas personales, veo necesario relatar mi experiencia:

- Nos situamos **docker/development/**
- Debemos de copiar/modificar el archivo **github.env.dist** como un archivo llamado **github.env**.
    - Veremos que dentro del archivo nos encontramos esto

      Personal GitHub token
      COMPOSER_AUTH={"github-oauth":{"github.com":
      "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"}}
    - Se nos indica crear añadir un token de acceso de Github que previamente hayamos creado en el campo relleno por el conjunto de X. Sin embargo, se nos sugiere directamente borrar la línea, ya que Github tiene un límite de accesos y si tenemos que realizar

muchas veces acciones con los comandos de docker-compose podemos tener problemas debido a esto, así que opté por esta segunda opción.

- Hacemos pull de las imágenes a usar (**docker-compose pull**), que serán las de **partkeepr/development:latest** y la de **mariadb:10.1** (base de datos).
- Levantamos de forma individual cada contenedor siguiendo el orden y las indicaciones de las instrucciones.
- Ahora está listo el entorno para ser instalado mediante la instalación situada en **setup/**, al cual podemos acceder introduciendo la dirección **http://127.0.0.1:8082/setup/** en el navegador web.
- En el tercer paso de la instalación, se nos pide una llave de autenticación. Para encontrarla, me basé en la instalación de la imagen alternativa e introduje el siguiente comando en la terminal (situado en el directorio en el que se ha levantado el entorno): **docker-compose exec app cat app/authkey.php**. Obtuve, entonces, la llave y proseguí la instalación, la cual no tuvo ni tiene mucha complicación.
- La aplicación fue, finalmente, instalada.

Finalizada, aparentemente, la instalación, surgieron varios problemas de los cuales hubo que hacerse cargo, estando estos directamente relacionados con la base de datos. No obstante, primero es necesario explicar el proceso de migración.

5.3. Migración

Para empezar, era necesario hacer una migración de la base de datos del primer entorno (a la que llamaremos base de datos antigua) a la base de datos del entorno nuevo (base de datos nueva). Los pasos seguidos para realizar la migración de la base de datos fueron los siguientes:

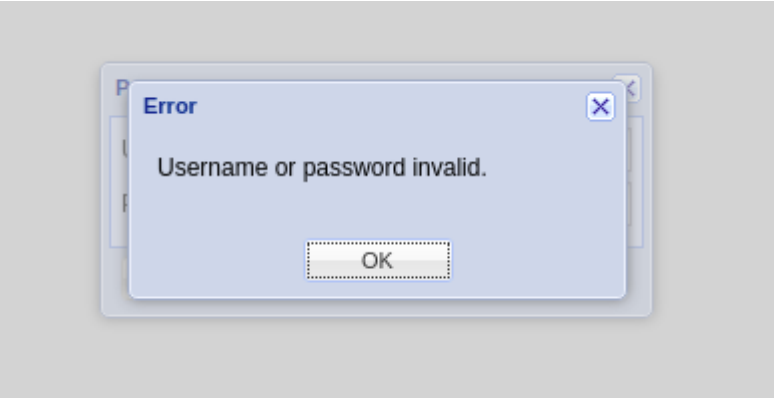
- Entrar en el contenedor de la base de datos antigua: **docker exec -it <nombre_contenedor/ID_contenedor> bash**
- Realizar el volcado de la base de datos a un archivo con extensión **SQL**

mediante la autenticación como usuario root predeterminado en el docker-compose.yml: **mysqldump -u usuario_root -p pass_root nombre_bd > archivo.sql**

- Sacar el archivo SQL del contenedor hacia el host: **docker cp <ID_contenedor/nombre_contenedor>:/ruta_archivo /ruta_host**
- Introducir archivo en el contenedor nuevo: **docker cp /ruta_archivo <ID_contenedor/nombre_contenedor>:/ruta_contenedor**
- Importar el archivo SQL a la base de datos que queramos **mysql -u usuario_root -p pass_root nombre_bd < archivo.sql**

5.4. Problema 1: Auto Login erróneo

En principio, este proceso bastaría. No obstante, surgió uno de los primeros problemas relacionados con la base de datos y es que, al realizar de nuevo el setup e intentar loguearme, salía una ventana indicando que el usuario y la contraseña son incorrectos, sin ni siquiera aparecer el formulario de login, era como si se estuviese autenticando un usuario con un nombre y/o contraseña incorrectos.



5.4.1: Mensaje de error del login

Para solventar esto, probé a hacer una comparación entre las dos bases de datos mediante la aplicación **phpMyAdmin**, creando dos bases de datos dentro de la aplicación a partir de los archivos SQL extraídos de las bases de datos antigua y nueva y abrirlas en una ventana cada una para colocarlas una al lado de la otra y hacer una comparativa de las tablas y los datos.

Aparentemente, la migración se había realizado correctamente, puesto que compartían los mismos datos y las tablas y sus columnas eran las mismas. Pero, entonces, caí en la cuenta de que la tabla **FOSUser** (la usada para los providers de autenticación en Symfony) tenía más columnas en la base de datos antigua (que, recuerdo, es la de la imagen alternativa) que en la nueva. Probé, entonces, a modificar la base de datos antigua borrando las columnas extra que tenía e introduciendo los datos de los usuarios de la nueva en la antigua también. Después, volví a hacer la migración de la base de datos de la misma forma que he descrito más arriba.

Fue así como, tras hacer otra vez el setup, desapareció el problema anterior y pude entrar dentro de la aplicación.

5.5. Problema 2: Auto Login como administrador

Sin embargo, surgió otro error y este era que la autenticación se hacía de forma automática como administrador, tomando el usuario registrado como tal en el setup.

Buscando en las issues del Github de PartKeepr me topé con ésta issue. Básicamente, un usuario describe un problema de login y la desarrolladora de PartKeepr le sugiere comprobar el valor de un parámetro en particular del archivo **parameters.php**. Al acceder a este archivo, me llamaron la atención otros campos relacionados con el autenticación automática, siendo estos los siguientes:

```
$container->setParameter('partkeepr.frontend.auto_login.enabled',
true);
$container->setParameter('partkeepr.frontend.auto_login.password',
'admin');
$container->setParameter('partkeepr.frontend.auto_login.username',
'admin');
```

Como se puede ver, el parámetro **auto_login** colocado como permitido (true) y con un usuario y contraseña **admin**, supuestamente para una autenticación automática como administrador. El caso es que yo no tenía un usuario con tales campos y la intención tampoco era una autenticación automática, entonces puse a **false** el campo **enabled** y deje a valor nulo los otros dos campos, quedando tal que así:

```
$container->setParameter('partkeepr.frontend.auto_login.enabled',
false);
$container->setParameter('partkeepr.frontend.auto_login.password',
'');
$container->setParameter('partkeepr.frontend.auto_login.username',
'');
```

Una vez cambiados estos campos y reiniciando el entorno, ya aparecía por fin el formulario de login para poder loguearme con el usuario de mi elección.

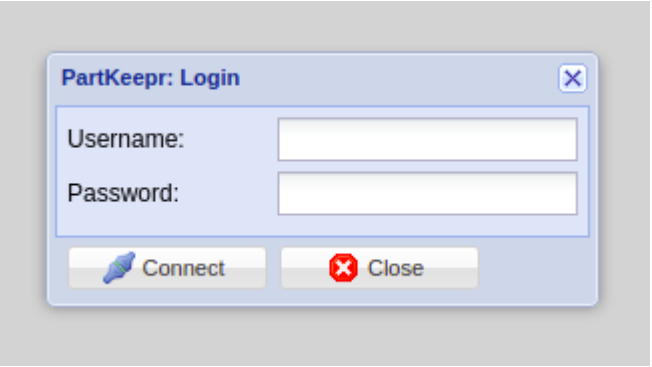


Figura 5.5.1: Formulario de login

Faltaba por realizar la migración de las imágenes y PDFs de cada componente, puesto que se había migrado la base de datos pero no el contenido de los volúmenes. Para esto, simplemente hizo falta copiar el contenido de los volúmenes antiguos a los nuevos.

5.6. Problema 3: Error con la vista previa de componentes

Hecho esto último, cuando pinchaba en un componente para que saliese su ventana de información se mostraban todos sus datos, incluidos los enlaces a todas sus imágenes (ya que hay algunos que tienen más de una) y a sus PDFs. Lamentablemente, hubo otro problema que resolver, ya que en la ventana de información, abajo del todo, se muestra una de las imágenes de este componente en un recuadro como vista previa y en algunos componentes no aparecía la foto. En su lugar, había un recuadro blanco con un mensaje que decía **500 Server Error.**

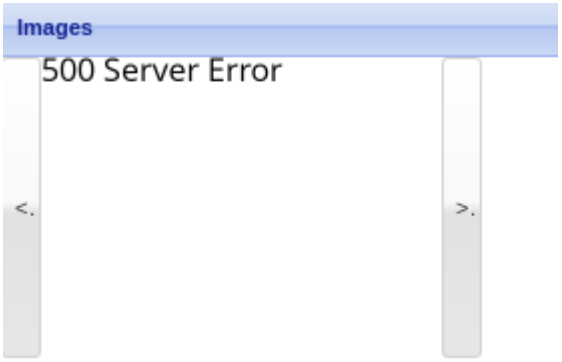


Figura 5.6.1: Mensaje de error en vista previa de componente

El error 500 es un error de respuesta de estado de HTTP que indica que el servidor no ha podido llevar a cabo una acción por una condición indeterminada. Es una respuesta ambigua que da el servidor cuando no encuentra otro código que dé una respuesta más específica pero no significa que error por elemento no encontrado (**404**), por lo cual la imagen existe pero no puede ser mostrada.

Dadas las circunstancias, volví a utilizar la funcionalidad de inspección de página. La pestaña de **Console** imprimía cierta información de error relacionada con una función llamada **getImage** seguida del código de error 500.

Como esto no me dio la suficiente información, me moví a la pestaña **Network** del inspector, puesto que esta pestaña de información sobre los movimientos en la red, más concretamente la del servidor.

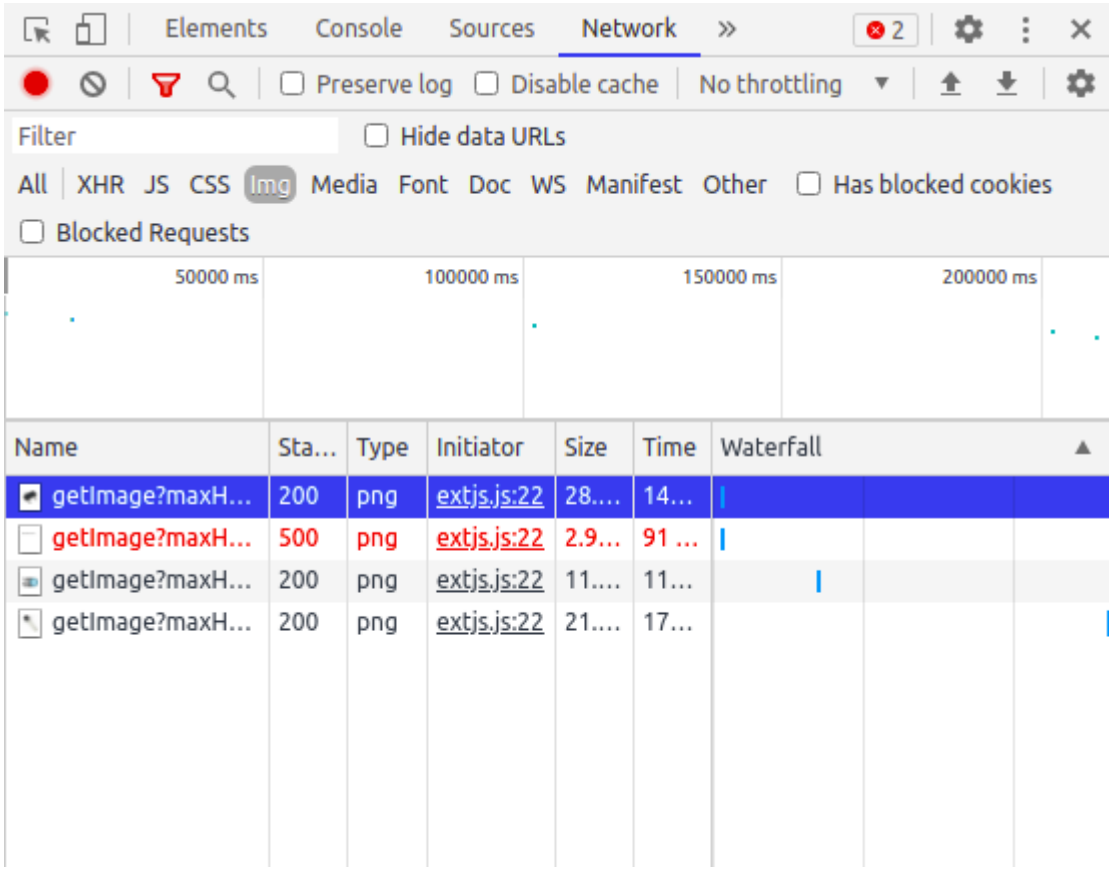


Figura 5.6.2: Pestaña de inspección de redes dentro del inspector de página de Google Chrome

Si nos fijamos, en la imagen de arriba se puede ver como, pinchando en la subpestaña **Img**, podemos ver el tráfico relacionado con las imágenes. Cuando se pincha en un elemento cuya imagen puede verse en el recuadro de vista previa, el código que se muestra es de **200**, código éxito de HTTP. En el caso contrario, el texto se muestra en rojo y el código sigue siendo 500.

Me fijé, entonces, en la extensión de la imagen mostrada. En todas las líneas se indica que la imagen es **PNG** pero sólo en una da error. Fue cuando caí en la cuenta de que, probablemente, dicha imagen estaba en otra extensión y así fue, estaba en **JPG**, junto a muchas otras imágenes. Además, echando un vistazo a la base de datos, me fijé que en la tabla que recoge las imágenes añadidas a los volúmenes (**PartAttachment**) hay una columna llamada

extension en la cual figura la extensión de la imagen, junto a otras columnas y respectivos valores.

En el trozo de código SQL podemos ver la estructura de la tabla (se comentarán ciertos campos más adelante).

```
DROP TABLE IF EXISTS `PartAttachment`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `PartAttachment` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `part_id` int(11) DEFAULT NULL,
 `type` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
 `filename` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
 `originalname` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
 `mimetype` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
 `size` int(11) NOT NULL,
 `extension` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
 `description` longtext COLLATE utf8_unicode_ci,
 `created` datetime NOT NULL,
 `isImage` tinyint(1) DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `IDX_76D73D864CE34BEC` (`part_id`),
 CONSTRAINT `FK_76D73D864CE34BEC` FOREIGN KEY (`part_id`)
REFERENCES `Part` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1023 DEFAULT CHARSET=utf8
COLLATE=utf8_unicode_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
```

La manera en la cual vi una forma adecuada de solucionar este problema fue cambiar las extensiones de las imágenes y, si no estaba automatizado, los valores de las extensiones en la tabla PartAttachment, para lo cual, entonces, realicé los siguientes pasos:

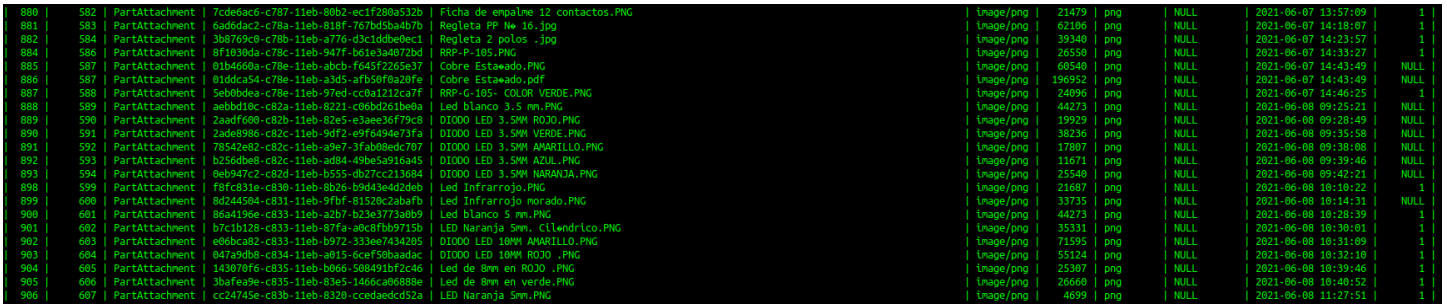
- Primero, se modificaron todos los formatos de las imágenes de .jpeg a .png para eso, hay que instalar **imageclick** con **sudo apt-get install imagemagick** y se usó el comando **mogrify -format png nombre_imagen.jpeg** (para hacerlo con todas las imágenes, usaremos **mogrify -format png *jpeg**)

- Después, se borraron todas las imágenes con formato .jpeg con **rm * .jpeg**

Pese a este cambio, seguían sin verse las imágenes, puesto que no se habían cambiado las extensiones de la tabla de forma automática y la imagen es solicitada mediante la construcción de sus valores en las columnas **filename** y **extension**, por lo cual se seguían llamando a las imágenes por su extensión antigua. Entonces:

- Se accedió al contenedor de la base de datos para introducirse dentro de ella.
- Se modificaron las celdas vacías de la columna extensión y que no tuviesen valor de **pdf** mediante **update PartAttachment SET extension = 'png' where extension != 'pdf'**
- También se modificaron los valores de la columna **mimetype** en los que el valor fuese **image/png** mediante **update PartAttachment SET mimetype = 'image/png' where mimetype = 'image/jpeg'**.

Abajo tenemos un trozo de la base de datos en el que ambos campos fueron cambiados.



880	582	PartAttachment	7cde6ac6-c787-11eb-80b2-ec1f280a532b	Ficha de empalme 12 contactos.PNG	image/png	21479	png	NULL	2021-06-07 13:57:09	1
881	583	PartAttachment	6ad6dac2-c78a-11eb-818f-767bd5ba4b7b	Regleta PP Nº 16.jpg	image/png	62106	png	NULL	2021-06-07 14:18:07	1
882	584	PartAttachment	3b8769c0-c78b-11eb-a776-d3c1ddbe0ec1	Regleta 2 polos .jpg	image/png	39340	png	NULL	2021-06-07 14:23:57	1
884	586	PartAttachment	8f1030da-c78c-11eb-947f-b61e3a4072bd	RRP-P-105.PNG	image/png	26550	png	NULL	2021-06-07 14:33:27	1
885	587	PartAttachment	01b4660a-c78e-11eb-abcb-f645f2265e37	Cobre Estañado.PNG	image/png	60540	png	NULL	2021-06-07 14:43:49	NULL
886	587	PartAttachment	01ddca54-c78e-11eb-a3d5-afb50f0a20fe	Cobre Estañado.pdf	image/png	196952	png	NULL	2021-06-07 14:43:49	NULL
887	588	PartAttachment	5eb0bdea-c78e-11eb-97ed-cc0a1212ca7f	RRP-G-105- COLOR VERDE.PNG	image/png	24096	png	NULL	2021-06-07 14:46:25	1
888	589	PartAttachment	aebbd10c-c82a-11eb-8221-c06bd261be0a	Led blanco 3.5 mm.PNG	image/png	44273	png	NULL	2021-06-08 09:25:21	NULL
889	590	PartAttachment	2aadf600-c82b-11eb-82e5-e3aee36f79c8	DIODO LED 3.5MM ROJO.PNG	image/png	19929	png	NULL	2021-06-08 09:28:49	NULL
890	591	PartAttachment	2ade8986-c82c-11eb-9df2-e9f6494e73fa	DIODO LED 3.5MM VERDE.PNG	image/png	38236	png	NULL	2021-06-08 09:35:58	NULL
891	592	PartAttachment	78542e82-c82c-11eb-a9e7-3fab08edc707	DIODO LED 3.5MM AMARILLO.PNG	image/png	17807	png	NULL	2021-06-08 09:38:08	NULL
892	593	PartAttachment	b256dbe8-c82c-11eb-ad84-49be5a916a45	DIODO LED 3.5MM AZUL.PNG	image/png	11671	png	NULL	2021-06-08 09:39:46	NULL
893	594	PartAttachment	0eb947c2-c82d-11eb-b555-db27cc213684	DIODO LED 3.5MM NARANJA.PNG	image/png	25540	png	NULL	2021-06-08 09:42:21	NULL
898	599	PartAttachment	f8fc831e-c830-11eb-8b26-b9d43e4d2deb	Led Infrarrojo.PNG	image/png	21687	png	NULL	2021-06-08 10:10:22	1
899	600	PartAttachment	8d244504-c831-11eb-9fbf-81520c2abafb	Led Infrarrojo morado.PNG	image/png	33735	png	NULL	2021-06-08 10:14:31	NULL
900	601	PartAttachment	86a4196e-c833-11eb-a2b7-b23e3773a0b9	Led blanco 5 mm.PNG	image/png	44273	png	NULL	2021-06-08 10:28:39	1
901	602	PartAttachment	b7c1b128-c833-11eb-87fa-a0c8fbb9715b	LED Naranja 5mm. Cilíndrico.PNG	image/png	35331	png	NULL	2021-06-08 10:30:01	1
902	603	PartAttachment	e06bca82-c833-11eb-b972-333ee7434205	DIODO LED 10MM AMARILLO.PNG	image/png	71595	png	NULL	2021-06-08 10:31:09	1
903	604	PartAttachment	047a9db8-c834-11eb-a015-6cef50baadac	DIODO LED 10MM ROJO .PNG	image/png	55124	png	NULL	2021-06-08 10:32:10	1
904	605	PartAttachment	143070f6-c835-11eb-b066-508491bf2c46	Led de 8mm en ROJO .PNG	image/png	25307	png	NULL	2021-06-08 10:39:46	1
905	606	PartAttachment	3bafea9e-c835-11eb-83e5-1466ca06888e	Led de 8mm en verde.PNG	image/png	26660	png	NULL	2021-06-08 10:40:52	1
906	607	PartAttachment	cc24745e-c83b-11eb-8320-ccedaedcd52a	LED Naranja 5mm.PNG	image/png	4699	png	NULL	2021-06-08 11:27:51	1

Figura 5.6.3: Captura de imágenes cambiadas de formato en la base de datos

Y fue a partir de aquí cuando la versión local del entorno funcionaba sin problemas.

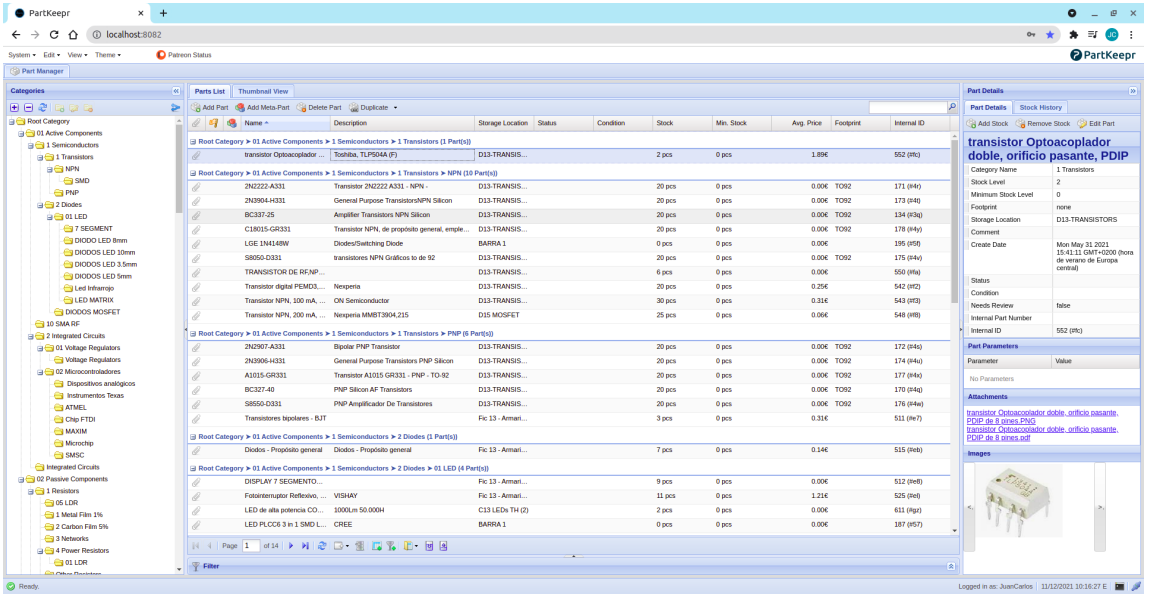


Figura 5.6.4: Captura de PartKeepr instalado localmente

Instalado y configurado el entorno en local, llegó la hora de trasladar el entorno al servidor, como se hizo con el entorno antiguo pero, en este caso, se explicará cómo se hizo.

Para empezar, se creó un nuevo **docker-compose.yml** hecho a partir del que está en el Github. Se le añadieron nombres en particular a los contenedores y una clave para la API de Octopart que está embebida en la aplicación de PartKeepr a la hora de consultar cada componente.

Se definieron dos redes, una red **traefik** ya configurada en el servidor (de ahí la indicación de **external: true**) y una subred llamada **partmanager_altium_network_jc**.

```
---
version: "3.7"

volumes:
    db:

services:
    app:
        image: partkeepr/development:latest
        container_name: partmanager_partkeep_jc
```

```
        # Use this instead if you want to develop the docker
infrastructure
        #build:
            #context: ./app
            #args:
                #SRC_IMAGE: partkeepr/base-dev:latest

        ports:
            - 8082:80
            #- 443:443

       volumes:
            - "../..:/var/www/pk"
        env_file:
            - ./github.env
        environment:
            GITHUB_DEBUG_UID: 1000
            #ADD_PHPINFO_FILE: 1
            #PARTKEEPR_FORCE_UPDATE: "yes"
            PARTKEEPR_OCTOPART_APIKEY: *******
        networks:
            - traefik
            - partmanager_altium_network_jc
        labels:
            - "traefik.docker.network=traefik"
            - "traefik.enable=true"
            -
"traefik.frontend.rule=Host:partmanagerjc.granasat.space"
            - "traefik.port=80"

    db:
        image: mariadb:10.1
        container_name: partmanager_mariadb_jc
        volumes:
            - db:/var/lib/mysql
        environment:
            MYSQL_ROOT_PASSWORD: *******
            MYSQL_USER: *******
            MYSQL_PASSWORD: *******
            MYSQL_DATABASE: *******
        networks:
            - partmanager_altium_network_jc
    initdb:
```

```
        build: initdb
        container_name: partmanager_initdb_jc
        depends_on:
            - db
        environment:
            MYSQL_USER: *******
            MYSQL_ROOT_PASSWORD: *******
            MYSQL_DATABASE: *******
            # Setting this to yes will enable reset upon running
            #RESET_DATABASE: 'yes'
            GITHUB_DEBUG_UID: 1000
        volumes:
            - "../../data:/data"
        networks:
            - partmanager_altium_network_jc


networks:
 traefik:
   external: true
#    name: traefik
 partmanager_altium_network_jc:
   driver: bridge
   ipam:
    config:
     - subnet: 192.168.184.0/24
```

Después, se movió toda la carpeta del proyecto comprimida como un **tar.gz** al servidor de GranaSAT mediante SSH. Finalmente, la aplicación fue iniciada mediante docker-compose up -d y actualmente se encuentra en la dirección https://partmanager.granasat.space/

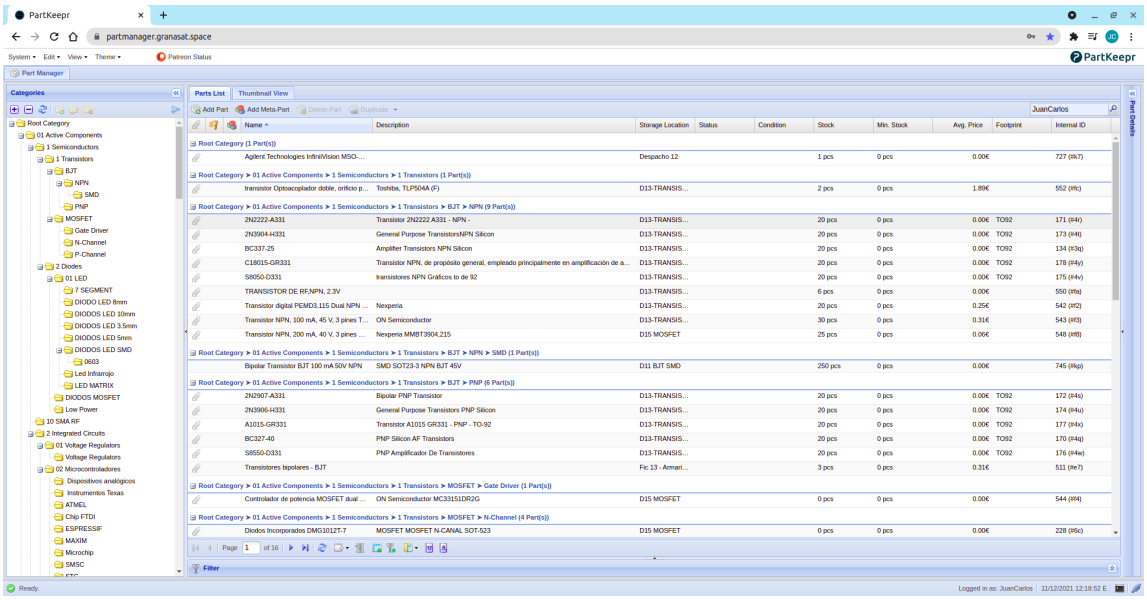


Figura 5.6.5: Captura de PartKeepr instalado en el servidor de GranaSAT

6. PUNTOS A REALIZAR Y RESULTADOS

Una vez implementado el entorno, se definieron una serie de puntos que se pusieron como objetivo para poder configurar y adaptar la aplicación a las necesidades de los usuarios de PartKeepr. A continuación, voy a proceder a listar todos los puntos y los procesos que se han realizado para poder aplicarlos al entorno:

1- Quitar privilegios de edición a según qué usuarios:

De manera predeterminada, PartKeepr no distingue de usuarios administradores y los que no lo son, pese a que tiene una tabla en la base de datos (**PartKeeprUser**) que contiene una columna que indica qué usuario es administrador (1) y cual no (0). Nuestra intención era que PartKeepr tuviese en cuenta qué usuario tenía los privilegios de administrador para que, de esa forma, sólo este tipo de usuario tuviese el poder de añadir, editar o eliminar componentes.

A partir de esto, he realizado los siguientes pasos para poder implementar este cambio:

1. Situarnos en **web/** (**cd web/**)
2. Conectarnos a la base de datos y realizar una consulta en la tabla **PartKeeprUser** que nos devuelva el nombre de usuario y si tiene privilegios de administrador y guardarlo en un archivo: **docker exec -it partkeepr_db_1 bash -c 'mysql -uroot -proot --database partkeepr --batch -e "select username, admin from PartKeeprUser"' > admin_privileges.txt**
3. Realizar **sudo chmod 777 admin_privileges.txt** en el caso de necesitarlo.

Ya tenemos listados los usuarios con sus permisos. Antes de proceder a explicar las modificaciones hechas en el código, veo necesario explicar cómo he descubierto dónde he tenido que realizarlas.

Para impedir que los usuarios no administradores editen los componentes y sólo puedan visualizarlos decidí deshabilitar las acciones que permitían hacer eso, o sea, deshabilitar los botones de edición en el caso de que no fuesen administradores. Estos botones son los siguientes:

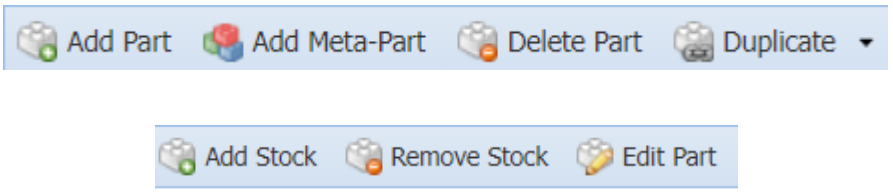


Figura 6.1.1: Barras de funcionalidades primaria (arriba) y del panel lateral (abajo)

Lo que hice entonces fue buscar dentro del directorio local de PartKeepr y desde la terminal cada uno de los textos de cada botón, de manera que eso me llevase al código de creación del botón. También utilicé la funcionalidad de inspeccionar la página de la aplicación y usar su funcionalidad de seleccionar interactivamente los elementos que quería inspeccionar para seguidamente acceder al código fuente mediante la pestaña **Sources** (más abajo se ilustra esto último mediante una captura).

Pero antes de eso, es necesario obtener primero el usuario logueado y encontrar la manera de determinar si ese usuario tiene los privilegios. Para ello, inspeccioné el código de la manera explicada arriba clicando en la barra de estado de abajo de la aplicación en la que se indica el usuario logueado. En la siguiente imagen se resume el proceso:

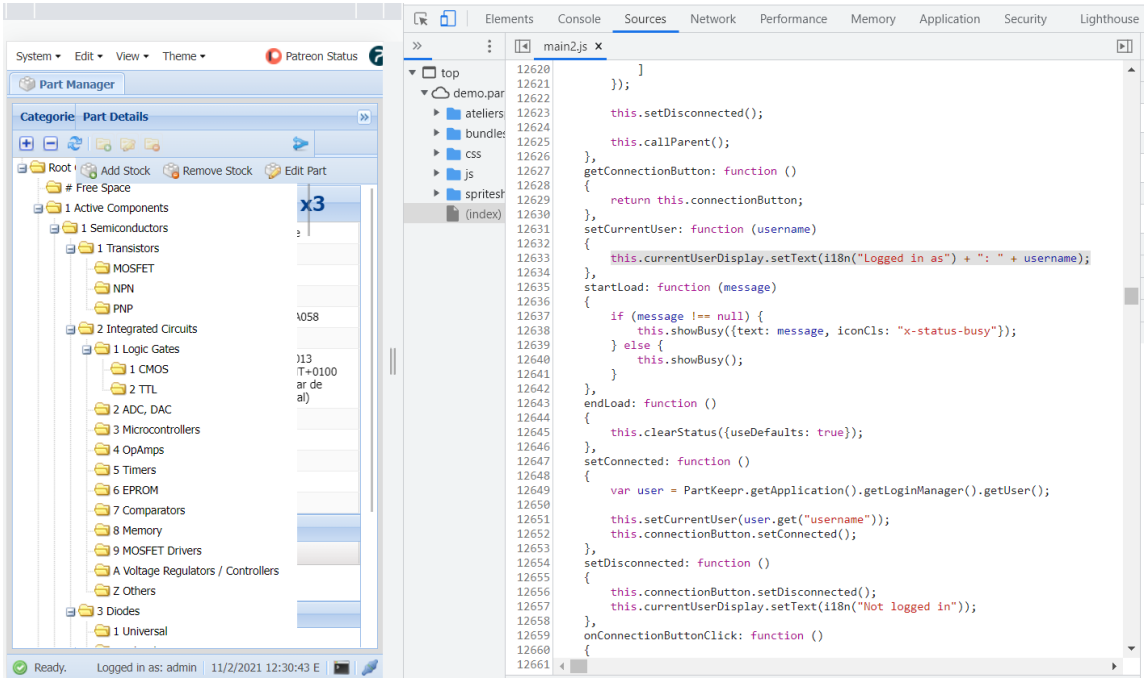


Figura 6.1.2: Línea de código buscada y seleccionada dentro de la funcionalidad de inspección de código fuente dentro del inspector de página de Google Chrome

Este archivo es un archivo compilado que recoge todos los archivos Javascript en uno solo a la hora de realizar el setup. Por lo cual, no es el archivo en el que hacer la modificación. Para saber cual es en el que hay que realizar las modificaciones hay que dirigirse unas líneas más arriba hasta que aparezca el nombre del archivo origen, el cual está indicado como parte de un objeto llamado PartKeepr que es el que engloba todo el front-end, ya que se está usando la librería **ExtJS** que sirve para la orientación a objetos en JS.

En este caso, la parte y consecuente archivo es **StatusBar**, por lo que busqué en el directorio local dicho archivo y lo encontré. Ahí localicé la línea que estaba buscando dentro de una función y realicé la siguiente modificación:

- Colocar las siguientes líneas de código dentro la función setCurrentUser: function (username) (línea 50 del archivo StatusBar.js):

```
setCurrentUser: function (username)
{
    localStorage.setItem("Current", username);

    var xhr = new XMLHttpRequest;
    xhr.open("GET","admin_privileges.txt");
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
            var arr = xhr.responseText;

            if (arr.includes(localStorage.getItem("Current")+"\t1")) {
                localStorage.setItem("Current_priv", username+"\t1");
            }
            else {
                localStorage.setItem("Current_priv", username+"\t0");
            }
        }

    }

    xhr.send();

    this.currentUserDisplay.setText(i18n("Logged in as") + ": " + username);
},
```

Figura 6.1.3: Código de setCurrentUser() modificado

En resumidas cuentas, se trata de introducir la variable **username** ya rellenada con el usuario actual mediante otras funciones a la variable **Storage** mediante la función **localStorage**, con la que se guardará la información que queramos de manera indefinida y mediante un índice (**"Current"**) y de una solicitud asíncrona mediante XMLHttpRequest que desde el cliente recoge algo del servidor, en este caso el archivo con los permisos. En el caso de que se realice correctamente, el contenido del archivo se almacena en un array. El condicional consiste en una comprobación de si dentro del array se encuentra la cadena compuesta por el valor actual de "Current" en localStorage sumado con el símbolo de tabulación y 1 ("\t1"), que simulan cómo está escrito en el archivo que recoge los usuarios y sus permisos o, dicho de otra, forma comprueba si ese usuario es administrador.

En caso afirmativo, se guarda la cadena **username + "\t1"** en un nuevo índice de Storage llamado "Current_priv" el cual será utilizado en un condicional que comparará la **username** + **"\t1"** con el valor de "Current_priv" y en el caso de ser iguales, el usuario podrá acceder a la funcionalidad.

Una vez obtenido esto, realicé la búsqueda de las líneas de código correspondientes a la creación de los botones. Me topé, entonces, con una función llamada **fireEvent**, la cual comprobé que sirve para realizar distintas acciones, entre las cuales y relacionadas con los botones sirve para realizar la acción de pulsar un botón, sea la que sea. En mi caso, lo que quería era deshabilitar las acciones de pulsar los botones mostrados. Entonces, coloqué el condicional en las siguientes líneas de código:

- **PartsGrid.js**:
    a. **this.fireEvent("duplicateItemWithBasicData");** (hay dos, rodear las dos)
    b. **this.fireEvent("duplicateItemWithAllData");** (esta y las dos anteriores corresponden al botón de "**Duplicate**")
    c. **this.fireEvent("addMetaPart");** (esta corresponde al botón de "**Add Meta Part"**)
    d. **this.fireEvent("editPart", record);** (con esto deshabilitamos abrir el panel de edición de componente al hacer **doble click** en el panel de datos de componentes situado justo en el centro de la aplicación, ya que esta línea corresponde a una función que reacciona al doble click)
- **PartDisplay.js**:
    e. **this.fireEvent("editPart", this.record);** (corresponde al botón de "**Edit Part**" del menú desplegable que sale al hacer click en un componente)
- **PartStockWindow.js**:
    f. Rodear todo el código interno de las funciones **addStock** y **removeStock**, que corresponden a los botones homónimos del menú desplegable anteriormente mencionado
- **EditorGrid.js**
    g. **this.fireEvent("itemAdd");** (corresponde al botón "**Add Part**")
    h. **this.fireEvent("itemDelete");** (corresponde al botón "**Delete**")

Una vez hecho esto, para poder observar los cambios debemos realizar el setup de nuevo y borrar la caché, debido a que los archivos compilados tienen que borrarse y volverse a crear para poder visualizar los cambios.

2- Impedir que se pueda entrar en el setup a usuarios no autorizados

Para salvaguardar la integridad del contenido que se pueda ver afectada acceder al setup, lo que he hecho ha sido crear un **.htaccess** en **web/setup/**, que es el directorio correspondiente al setup. Un archivo .htaccess es un archivo que se implementa para determinar el comportamiento de un servidor Apache, en este caso el nuestro.

Colocando las directivas

order deny,allow
deny from all

en el directorio indicado, podemos evitar el acceso a cualquier persona que intente entrar en el setup.

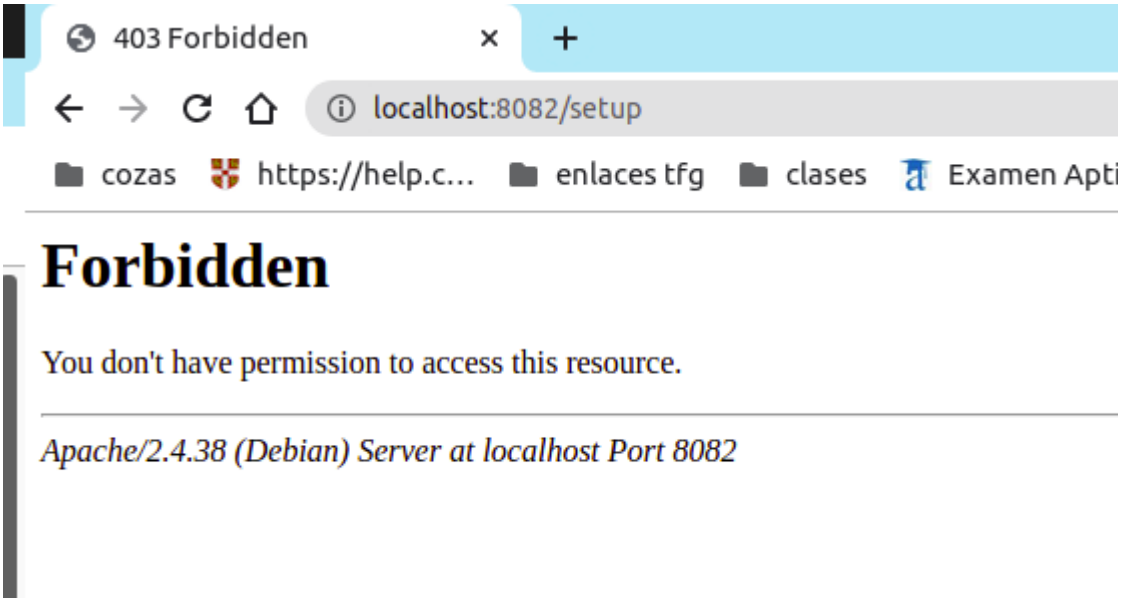


Figura 6.2.1: Captura en la que se muestra el permiso denegado

3- Logo GranaSAT como logo de carga de página y de favicon

Para darle a la estética de la aplicación un toque más personal, decidí cambiar el logo PartKeepr por de GranaSAT como favicon y en el momento de carga justo al iniciar la aplicación.
Con la funcionalidad de inspección de página, encontré varios archivos fuente en la pestaña de **Sources**, entre ellos el archivo CSS, llamado **ec39aa1.css**. Mirando entre las reglas del archivo, me topé con una llamada **#loader-logo**, la cual una de las declaraciones era la siguiente:

```
background-image: url(../bundles/partkeeprfrontend/images/partkeepr_loader.svg);
```

Ésta ruta correspondía al logo de PartKeepr que aparecía como logo de carga, por lo cual coloqué la imagen del logo de GranaSAT en ese directorio y sustituí el nombre de la imagen en la ruta puesta en la declaración anteriormente mostrada. La declaración actual es la siguiente:

```
background-image: url(../bundles/partkeeprfrontend/images/granasat_logo.jpg);
```

Después de esto, supuse que el favicon se llamaría tal cual, entonces busqué por su nombre en el buscador del directorio de PartKeepr y encontré un elemento llamado **favicon.ico**, que correspondía con una imagen a escala muy reducida del logo. Entonces, simplemente sustituí la imagen por la del logo de GranaSAT pero con el mismo nombre y extensión.

Después de estos cambios, realicé el setup de nuevo y borré la caché y los cambios se realizaron correctamente. Más abajo están las imágenes que prueban los cambios en logo de carga (izquierda) y en el favicon (derecha)

Figura 6.3.1: Logo de GranaSAT adjudicado como símbolo de carga (izquierda) y como favicon (derecha)

4- Indicar la distribución de Linux en la que se está ejecutando el PartKeepr

Como se ha explicado anteriormente, siguiendo la pestaña View y clicando en System Information, se puede acceder a un panel de información en el cual podemos acceder a cierta información técnica sobre el sistema que se ha levantado en los contenedores.

Uno de los campos de información es la del sistema operativo del contenedor (**Operating System Release**). El problema es que a la hora de imprimir el sistema operativo, imprimía "unknown", entonces traté de buscar la línea de código que se encargaba de esto ejecutando el comando **grep -ri "Operating System Release" PartKeepr/** en la terminal. Eso me llevó al archivo **SystemService.php**, el cual es el encargado de mostrar la información de la pestaña.

Ahí se puede ver como en la función **getSystemInformation()** se va rellenando un array de objetos **SystemInformationRecord** que cada uno de ellos corresponde a un campo de información, uno de ellos siendo el del sistema operativo, más concretamente la línea **$aData[] = new SystemInformationRecord('Operating System Release', $os->getRelease(),**

'System');. Como podemos ver, la función a la que llama es **getRelease()** y ésta está contenida en el archivo **OperatingSystem.php.**

Entrando a éste último y accediendo al código de esta función, vemos como se hace una llamada al comando **lsb_release -d -s**, el cual debería de dar el nombre del sistema operativo y su distribución.

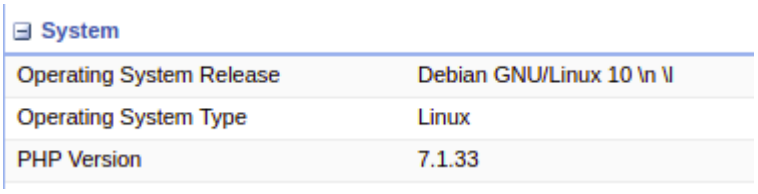
```
public function getLinuxDistribution()
    {
        /* Try executing lsb_release */
        $release = @exec('lsb_release -d -s', $void, $retval);

        if ($retval === 0 && $release !== '') {
            return $release;
        }

        //@todo we need better handling here
        return 'unknown';
    }
```

Al ya saber que este comando no funciona, decidí probarlo en la terminal del contenedor de la aplicación y efectivamente no funcionaba. Entonces, decidí probar otro comando.

Di con el comando **cat /etc/issue**, que imprime la versión de Linux seguido de un par de campos más. También probé a concatenar el comando con **head -n 1**, ya que me di cuenta de que el comando anterior a secas imprime también un salto de línea. El resultado fue el comando **cat /etc/issue | head -n 1** y cree, entonces, una función en **SystemService.php** con ese comando, llamada **getOSVersion()**, la cual, entonces, forma ahora parte del array de campos de System Information.

```
public function getOSVersion()
    {
        $release = @exec('cat /etc/issue | head -n 1');

        return $release;
    }
```

Como se puede ver en la imagen de abajo, los cambios de realizaron correctamente



System	
Operating System Release	Debian GNU/Linux 10 \n \l
Operating System Type	Linux
PHP Version	7.1.33

Figura 6.4.1: Captura de System Information en la que se ve el comando funcionando

5- Saber cuánto ocupan los ficheros de la base de datos e imprimirlo en la System Information

Para añadir algo más de información técnica sobre el entorno, se decidió añadir un campo más a System Information que determinase el tamaño total que ocupan los archivos de la base de datos.

Para ello, volví a **SystemService.php** y cree una nueva función llamada **getDatabaseSize()** que ejecuta el comando **du -sh ../data/** de manera que de esta forma se sabe cual es el tamaño total de del volumen **/data** formado por todos sus subdirectorios y respectivos archivos (imágenes, PDFs…).

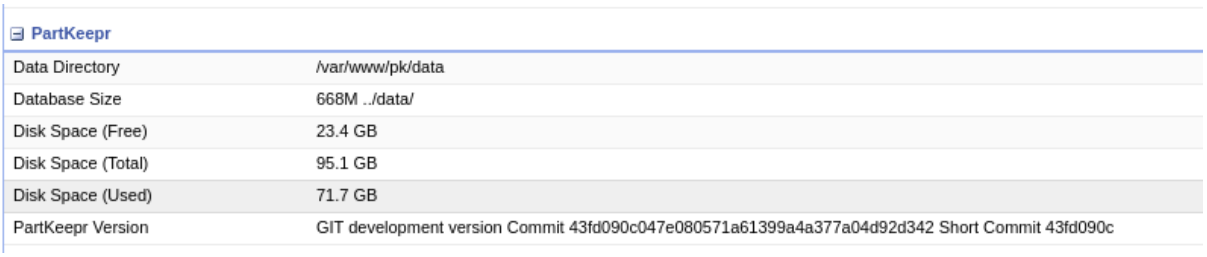
```
public function getDatabaseSize()
    {
        $release = @exec('du -sh ../data/');

        return $release;
    }
```

Después, se llama a esta función desde la función **getSystemInformation()**, para que forme parte del array que se encarga de todos los campos de información.

```
$aData[] = new SystemInformationRecord(
            'Database Size',
            $this->getDatabaseSize(),
            'PartKeepr'
        );
```

Cabe destacar que la primera parte del objeto **SystemInformationRecord** el nombre del campo, el segundo es la llamada a la función y el tercero es la

sección a la que pertenece. Entonces, una vez hecho ésto, tenemos el nuevo campo en la sección PartKeepr



PartKeepr	
Data Directory	/var/www/pk/data
Database Size	668M ../data/
Disk Space (Free)	23.4 GB
Disk Space (Total)	95.1 GB
Disk Space (Used)	71.7 GB
PartKeepr Version	GIT development version Commit 43fd090c047e080571a61399a4a377a04d92d342 Short Commit 43fd090c

Figura 6.5.1: Captura de System Information donde se ve el nuevo campo y comando funcionando

6- Cambio en la distribución de los directorios del proyecto y del docker-compose.yml

Se me sugirió la idea de modificar la distribución y, en parte, el contenido de la carpeta del proyecto debido a que dentro de ésta había ciertos componentes que no los usábamos para nada y el archivo docker-compose.yml estaba relativamente “escondido”, teniendo que movernos varios directorios hacia adentro para levantar el entorno. Es por eso que hubo que hacer varias modificaciones:

1. **Sacar el archivo docker-compose.yml al directorio raíz:** antes de realizar esto, hay que modificar las líneas que indican los volúmenes que usan los servicios de **app** e **initdb**, ya que al estar ambos antes en **docker/development/** los volúmenes estaban referidos con **../../** delante (dos directorios arriba). Lo que hay que hacer es, simplemente, reescribir la línea referida a los volúmenes de app tal que así **./:/var/www/pk**, ya que ésta manera queremos decir que el volúmen es todo el directorio PartKeepr/ y **/data:/data** en la línea de volúmenes de **initdb**.
2. **Colocar el volúmen initdb y el archivo github.env en el directorio raíz**
3. **Eliminar el directorio docker/**

Abajo podemos comparar las dos distribuciones antes...

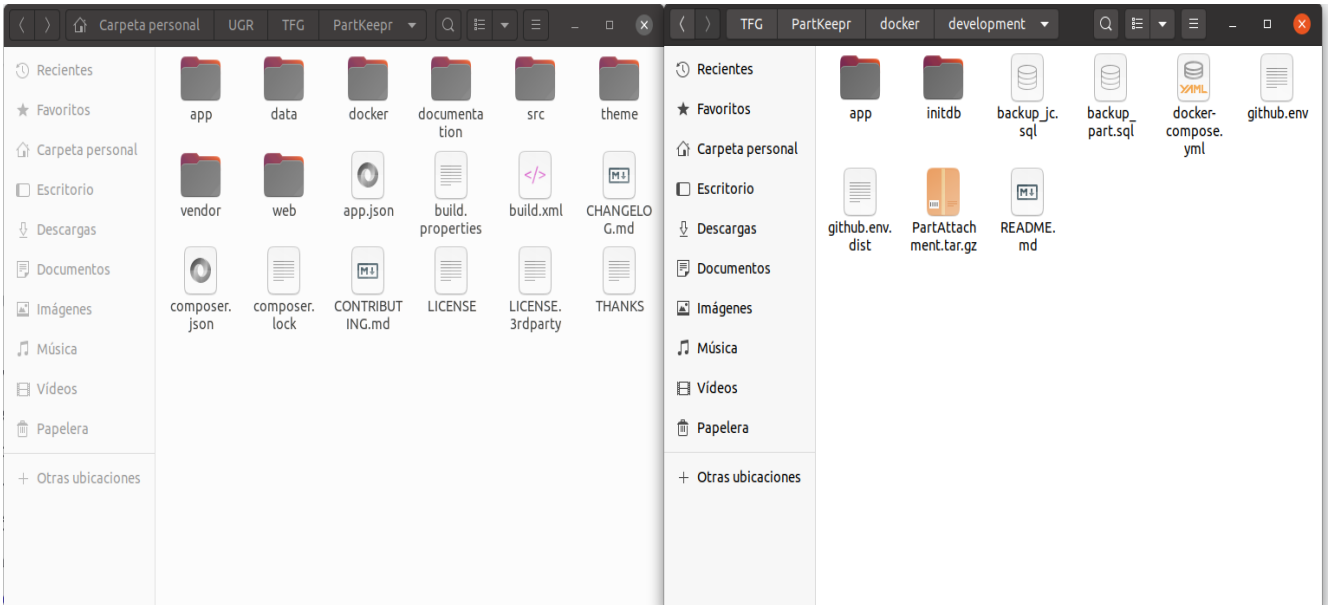


Figura 6.6.1: Distribución de directorios antes de la modificación

Y después...

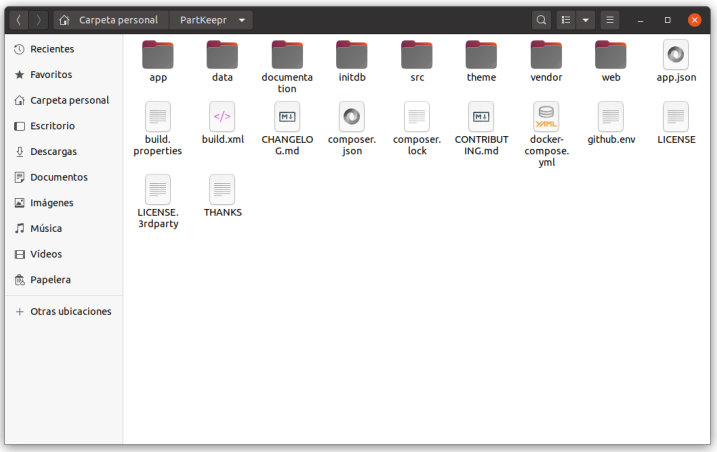
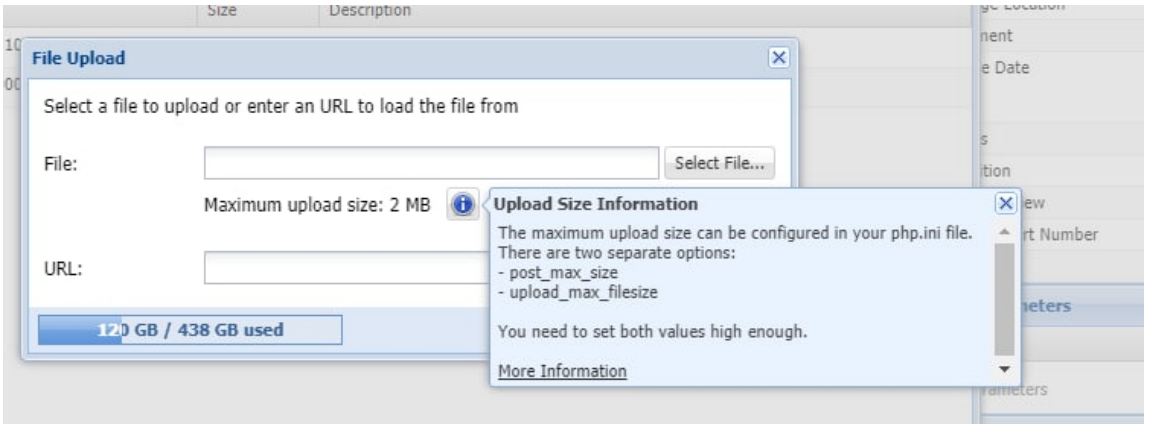


Figura 6.6.2: Distribución de los directorios después de la modificación

Siendo después una muy buena forma de tener todos los directorios localizados y dándole a este conjunto una perspectiva más simple y concisa.

7- Permitir utilizar ficheros más grandes que 2 MB

A la hora de subir un archivo (JPG, PNG, PDF) la aplicación nos indica de que hay un límite de tamaño para los archivos que se quieren subir, 2 MB.



6.7.1: Captura de File Upload antes de la modificación del tamaño máximo de subida

Se comentó, entonces, la idea de aumentar el tamaño permitido para poder subir contenido más pesado en el caso de necesitarlo. Por lo cual, se hicieron las modificaciones que finalmente lo permitieron.

Al ser PartKeepr un programa escrito en PHP para el back-end, el tamaño por defecto de subida son 2 MB. Para cambiar esto, se debe crear en el directorio raíz (o modificar) el archivo **php.ini** que es, básicamente, el fichero de configuración de PHP.

Entonces, una vez dentro del archivo, habrá que escribir (si no están escritas ya) las variables upload_max_filesize (la cantidad que aparecerá en las ventanas de subida de archivos) y post_max_size (el máximo). En este caso, las variables y los valores deben indicarse tal que así.
Tras reiniciar el entorno de nuevo (**docker-compose down** y **docker-compose up**) veremos los resultados

6.7.2: Captura de los campos y sus respectivos valores introducidos en php.ini

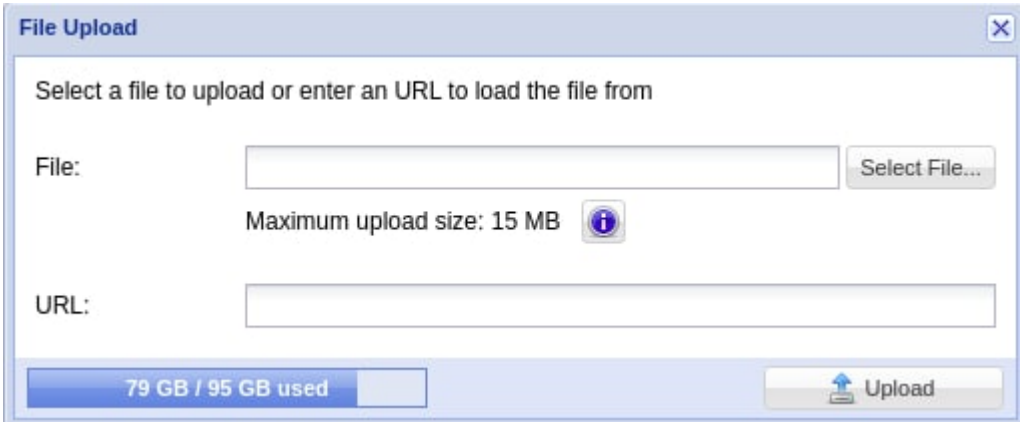


Figura 6.7.3: Captura de File Upload con el cambio introducido

7. CONCLUSIONES

La realización de este proyecto me ha llevado a refrescar muchas competencias y conocimientos adquiridos durante todos estos años de carrera de ingeniería informática y a comprobar que muchas cosas aprendidas han servido para desarrollar un caso práctico y a solventar las situaciones y problemas que han surgido a medida que se ha ido avanzando en él.

También he aprendido el uso de nuevas tecnologías, filosofías y metodologías, en especial la ingeniería inversa, puesto que sin ella nada de esto hubiese sido posible y los microservicios, sin los cuales tampoco podría haberse llevado a cabo el proyecto.

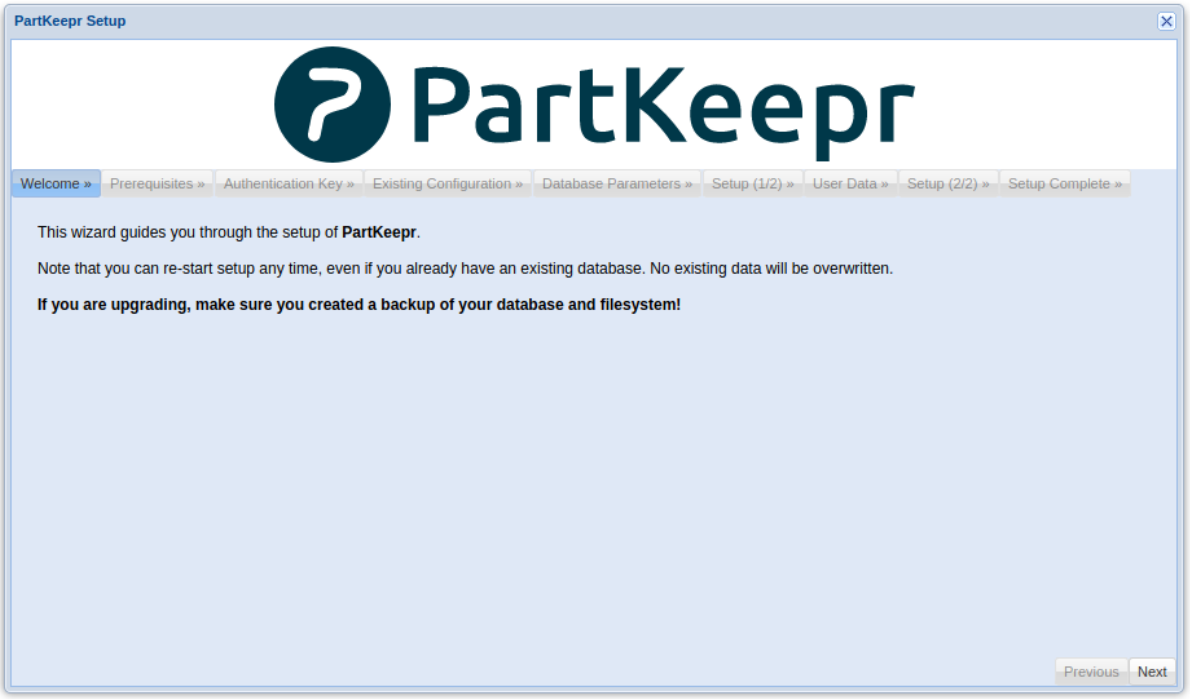
Y, sobre todo, he puesto en práctica valores como la constancia, el arrojo, la voluntad de encontrar soluciones adecuadas y correctas y las ganas de hacer el trabajo bien hecho, valores puestos en práctica durante la carrera pero especialmente en este proyecto.

ANEXO I: Instalación de PartKeepr (imagen alternativa)

- Ejecutar el siguiente script:

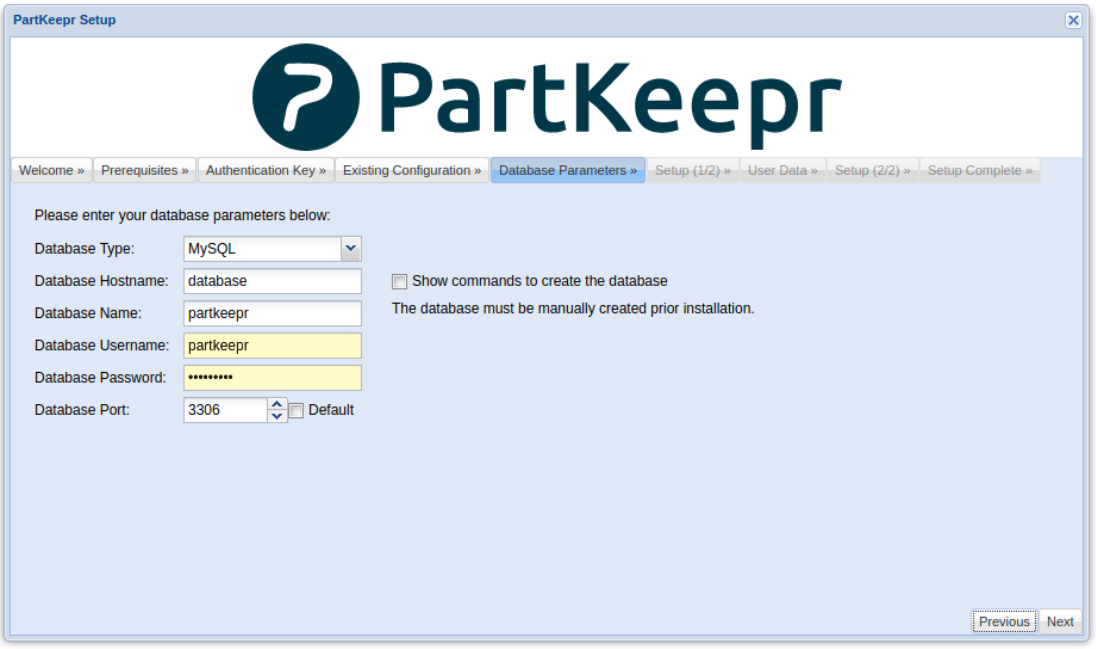
#!/bin/bash
git clone https://gitlab.com/jchermoso/partkeepr.git
chmod -Rf 777 partkeepr/
cd partkeepr
git pull
docker-compose up -d

- Entrar en localhost:8080/setup (puede tardar un poco al principio). Si todo ha ido bien, saldrá esta imagen



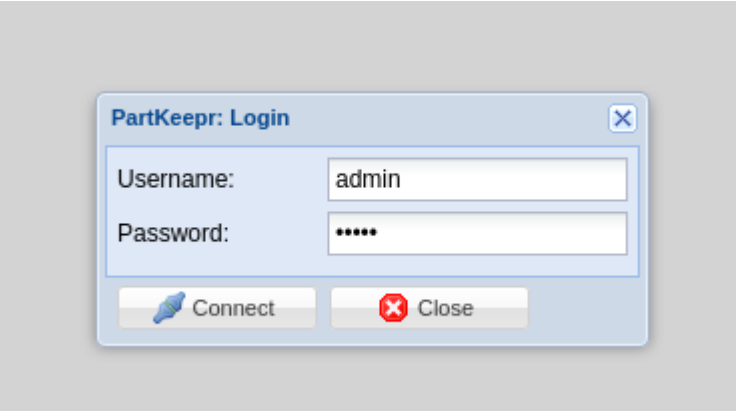
- Seguir adelante hasta llegar a la ventana encabezada por la pestaña Authentication Key. Una vez ahí, pedirá una llave de autenticación, entonces se tendrá que poner el siguiente comando en la terminal (estando dentro del repositorio local de Partkeepr clonado anteriormente): **docker-compose exec partkeepr cat app/authkey.php**

- Una vez hecho esto, copiar la llave de autenticación y seguir avanzando hasta que aparezca la siguiente ventana, que tendrá de los campos rellenados de forma predeterminada por el docker-compose



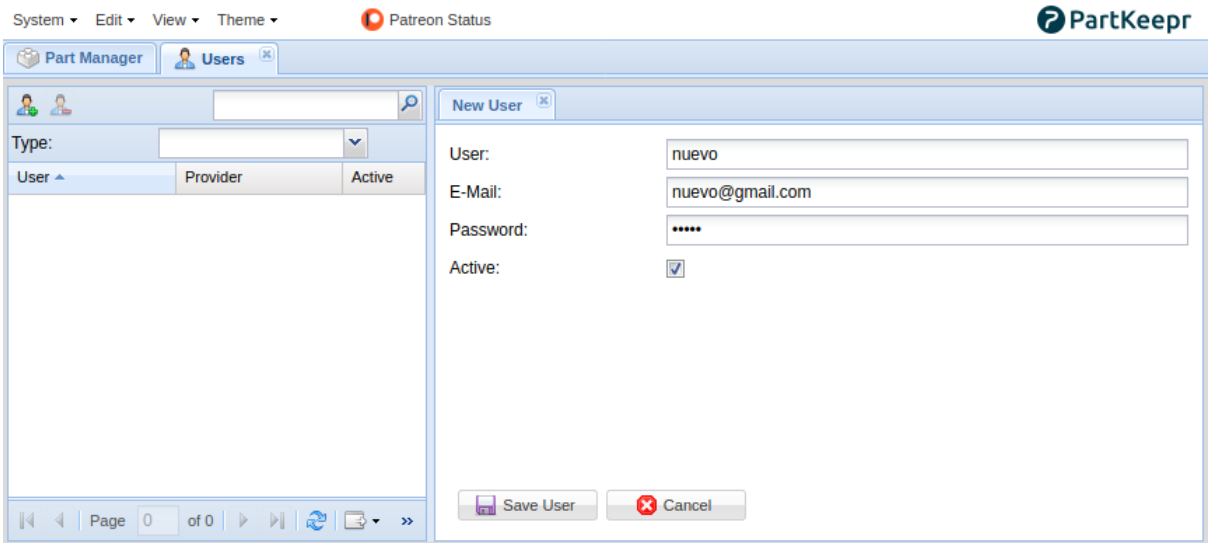
- En la ventana encabezada por User Data pedirá crear un usuario que haga de administrador. Este usuario será el encargado de crear los primeros usuarios, componentes… hasta que se creen nuevos usuarios dentro de PartKeepr.

- Seguir adelante hasta que aparezca la siguiente ventana para loguearse dentro del PartKeepr



Crear nuevos usuarios

- Una vez dentro de PartKeepr, ir a Edit > Users clicar en Add User 

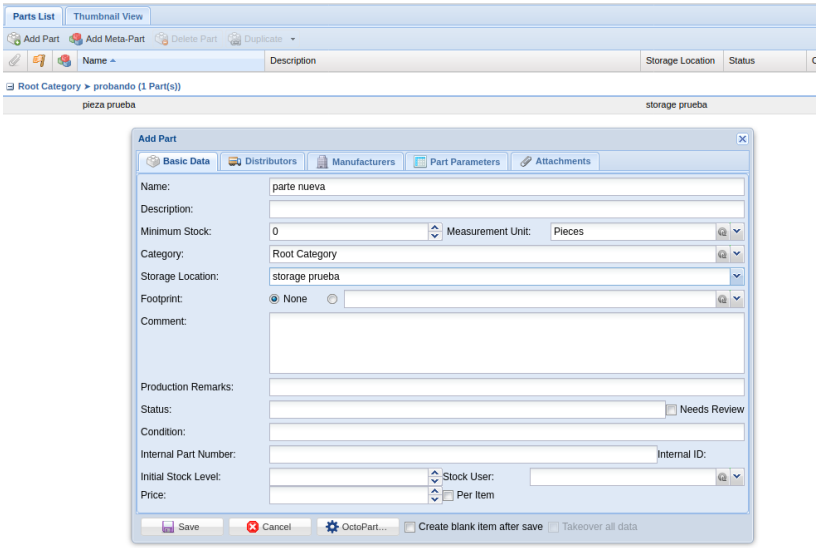


- Ahora se podrá acceder a PartKeepr con este nuevo usuario

Añadir nuevos componentes

- Para añadir un nuevo componente, es necesario que exista previamente un Storage Location. Para crear un SL hay que ir a Edit > Storage Location y clicar en Add Storage Location 

- Una vez hecho esto, volver a la pestaña Part Manager y clicar en Add Part y rellenar los campos que sean pertinentes (el nombre y el Storage Location son campos obligatorios a rellenar)



ANEXO II: Comandos

Docker

- Levantar docker-compose en segundo plano: **docker-compose up -d**
- Parar docker-compose (todos los contenedores): **docker-compose stop**
- Entrar en contenedor:
    - **docker ps** (para saber el ID del contenedor al que se quiere entrar)
    - **docker exec -it ID bash**
- Conseguir ćlave de autenticación para setup de PartKeepr (versión original): **docker-compose exec app cat app/authkey.php**
- Copiar archivo de contenedor a host: **docker cp <containerId>:/file/path/within/container /host/path/target**

Mysql

- Entrar como root (indispensable para crear/modificar bases de datos): **mysql -uroot -p** (la contraseña predeterminada es root)
- Hacer dump para backup: **mysqldump -uroot -p nombre_bd > archivo.sql**
- Importar archivo .sql a base de datos: **mysql -uroot -p nombre_bd < archivo.sql**
- Usar determinada base de datos: **use nombre_bd**
- Consultar valores de filas de cierta tabla: **select * from nombre_tabla;**
- Modificar campo: **update nombre_table SET columna = 'valor' where condicion**

BIBLIOGRAFÍA

[1] PartKeepr: https://www.partkeepr.org/

[2] Github de PartKeepr (original): https://github.com/partkeepr/PartKeepr

[3] Github de PartKeepr (alternativa): https://github.com/mhubig/docker-partkeepr

[4] GranaSAT: https://granasat.ugr.es/

[5] Demo de PartKeepr: https://demo.partkeepr.org/

[6] Memoria del TFG de Jesús Rodríguez Pérez https://digibug.ugr.es/bitstream/handle/10481/63850/Memoria%20Jesus%20Rodriquez.pdf?sequence=1&isAllowed=y

[7] Página de Goodfirms que usó como base para la búsqueda de candidatos para el sistema de librerías de componentes a usar https://www.goodfirms.co/blog/best-free-open-source-inventory-management-software-systems

[7] Odoo https://www.odoo.com/es_ES/app/inventory

[8] Stockpile http://www.thecanvus.com/

[9] Página de Wikipedia de Symfony https://es.wikipedia.org/wiki/Symfony

[10] Demo de phpMyAdmin https://demo.phpmyadmin.net/master-config/index.php

[11] Wiki de PartKeepr https://wiki.partkeepr.org/wiki/Main_Page

[12] Documentación de Docker Compose https://docs.docker.com/compose/

[13] XMLHttpRequest para leer archivo de texto https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest

[14] Directivas para archivo .htaccess https://stackoverflow.com/questions/5046100/prevent-access-to-files-in-a-certain-folder

[15] Comandos para comprobar la versión de Linux en la terminal https://lamiradadelreplicante.com/2012/11/03/comprueba-la-version-de-linux-desde-la-terminal/

[16] php.ini http://tecmint.com/increase-file-upload-size-in-php/

[17] Providers de Symfony

https://diego.com.es/providers-de-seguridad-en-symfony
[18] Descripción de tablas de usuarios de PartKeepr
https://wiki.partkeepr.org/wiki/Developers/User_System
[19] Issue sobre autenticación automática
https://github.com/partkeepr/PartKeepr/issues/491#issuecomment-464885868
[20] Artículo de Wikipedia sobre código abierto
https://es.wikipedia.org/wiki/C%C3%B3digo_abierto
[21] Web de Altium https://www.altium.com/