Ingeniería informática

DISEÑO E IMPLEMENTACIÓN DE UN AGREGADOR DE ALMACENAMIENTO- DESIGN AND IMPLEMENTATION OF A STORAGE AGGREGATOR

Realizado por
Jose Alonso Ortega Elías

Tutorizado por
José Antonio Onieva González

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, Septiembre 2021

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADA/O EN INGENIERIA INFROMATICA

DISEÑO E IMPLEMENTACIÓN DE UN AGREGADOR DE ALMACENAMIENTO

DESIGN AND IMPLEMENTATION OF A STORAGE AGGREGATOR

Realizado por
Jose Alonso Ortega Elías

Tutorizado por
José Antonio Onieva González

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2021

Fecha defensa: octubre de 2021

Agradecimientos

Quería agradecer a mi tutor por su apoyo y guía en todo el desarrollo de este proyecto.

A mi familia por su apoyo incondicional en esta etapa académica. A mi hermana por su ánimo y apoyo a lo largo de la realización de este proyecto y en especial a mi madre por no perder nunca la esperanza en mí.

Resumen

Recientemente han surgido multitud de servicios en la nube que ofrecen cierta cantidad de almacenamiento. Por lo cual una idea interesante seria agrupar estos servicios, además de permitir agregar otros dispositivos. Por lo tanto, el objetivo de este proyecto es desarrollar una aplicación la cual sea capaz de, a través de una serie de reglas (políticas de almacenamiento), escoger el medio de almacenaje al cual enviar el archivo elegido. Estas políticas estarán definidas por el propio usuario y serán ellas las encargadas, de forma totalmente transparente, del destino de los archivos que suba el usuario a la aplicación. Esos "medios" consistirán en las nubes más importantes (OneDrive y drive) y aquellos dispositivos, en la red local, con capacidad de almacenamiento. Para añadir estos últimos se realizará una búsqueda en la red local, devolviendo aquellos que contengan la capacidad de almacenamiento. Esta búsqueda se realizará a través del protocolo SSDP. También tendrá la capacidad de mostrar en una misma página todos los archivos pertenecientes a las conexiones establecidas, pudiendo borrarlos o acceder al contenido de las carpetas.

Ya que este proyecto se encuentra enmarcado en el paradigma del Edge Computing se usarán contenedores para simular este comportamiento. Los contenedores permiten el despliegue de aplicaciones "cerca" del usuario cumpliendo así con el objetivo de Edge Computing, que la distancia recorrida por los datos sea la menor posible. Con esto se conseguirá desplegar la aplicación web en cualquier entorno compatible con contenedores.

Palabras clave:

Docker, Aplicación web, Almacenamiento, Red local, Edge Computing, Rclone.

Abstract

Recently there are plenty of cloud services that offer different amounts of storage. For this reason, an Interesting Idea will be to join these services and allow them to connect other devices too. Therefore, this project's goal is to develop an application that, can choose a storage mode to send the chosen files by using determinate rules (storage´s policies). These policies will be defined by the own user and they will handle, by Itself, the final location of the files that the user uploads to the application before. The locations will consist of the most Important clouds (OneDrive and drive) and the devices in the local network with storage ability. To add the devices we will make a search In the local network, returning the ones with storage ability. This search will be made through the SSPD protocol. The application will have the ability to show in a same page all the files related to a connection, delete them or access to the folder´s content.

As this project belongs to the edge computing paradigm we will use containers to simulate this behavior. The containers allow the deploy of the application "near" the user, this accomplish with the goal of edge computing, that the Information runs the less distance possible. With this we will get to deploy the web application in any compatible environment with containers.

Keywords:

Docker, web application, storage, local network, Edge Computing, Rclone.

Abreviaturas

BD Base de datos

SSDP Simple Service Discovery Protocol

DLNA Digital Living Network Alliance

UPnP Universal Plug and Play

HTML HyperText Markup Language

CSS Cascading Style Sheets

HTTP Hypertext Transfer Protocol

API Application programming interface

UDP User Datagram Protocol

XML Extensible Markup Language

SFTP Secure File Transfer Protocol

SSH Secure SHell

IP **I**nternet Protocol

SQL Structured Query Language

YAML YAML Ain't Markup Language

Índice

1

Introducción

1.1 Motivación

En los últimos años han surgido una gran cantidad de servicios en la nube que ofrecen de forma gratuita una cierta cantidad de espacio. Aunque por separado parezcan que ofrecen poco almacenamiento en conjunto puede ser una cifra que considerar. Esta gran cantidad de recursos en la nube se vuelven difíciles de organizar y pueden provocar que el usuario "pierda" alguna información previamente almacenada.

Aunque organizar las diferentes nubes es un problema que ya se está resolviendo, con soluciones como biupBOX [18], nosotros añadiremos algo más. Añadiremos la capacidad de agregar otras formas de almacenamiento. Esto permitirá al usuario aumentar la capacidad de almacenamiento a su disposición con diferentes dispositivos que permitan el almacenamiento de información sin que este tenga que indicar el lugar de almacenaje, gracias a una serie de políticas que definiremos. Se

trataría, por tanto, de un agregador de espacio personal y modular para todos los dispositivos con los que cuente el usuario.

Otro problema que considerar es la gran cantidad de datos que se produce en una empresa. Y no sólo con ficheros de datos, sino también con todo el stream de datos que hoy en día (y aún más en el futuro) generan los dispositivos IoT (Internet of Things). Por ello el paradigma del Edge Computing es muy útil aquí, ya que cuanto menos sea la distancia que recorran esos datos menor será su latencia. Esto llega a ser muy significativo cuando se maneja una gran cantidad de datos. Por lo tanto, un agregador de almacenamiento que permitiera, a través de distintas extensiones/módulos, agregar de forma transparente todo el espacio disponible y no utilizado de los distintos dispositivos con los que ya cuenta la empresa sería de gran utilidad a la misma.

1.2 Objetivos

El TFG consistirá en el desarrollo y diseño de un agregador de almacenamiento, que contará con la capacidad de comunicarse con las nubes más importantes (OneDrive, Google drive…) y cualquier otro dispositivo, con capacidad de almacenamiento, que se encuentre disponible en la red local.

Implementaremos una serie de políticas que utilizaremos para distribuir la información que llegue al agregador. Esto será un procedimiento transparente al usuario para que este no se tenga que preocupar del lugar donde guardar la información (e.g. balanceo de espacio libre en los distintos servicios), aunque el mismo usuario podrá definir sus propias políticas en el agregador (e.g. material multimedia al espacio ofrecido por el smartphone). El agregador proporcionará a su vez la información necesaria para que el usuario, de ser preciso, pueda encontrar donde se ha almacenado la información, así como estadísticas de uso y distribución.
La aplicación permitirá otra serie de características como: la descarga de archivos consulta de estadísticas, consulta de los archivos de cada almacenamiento, borrar archivos etc.

1.3 Estructura de la memoria

Esta memoria está estructurada en 7 capítulos. A continuación, entraremos en más detalle en cada una de ellas:

- **Capítulo 1 - Introducción**: en este capítulo trataremos de dar una visión preliminar del tema que aquí se va a tratar a través de la motivación y el objetivo a alcanzar, además de la estructura de la memoria.
- **Capítulo 2 - Tecnologías usadas**: en este capítulo enumeraremos las distintas tecnologías con las cuales hemos desarrollado este trabajo. Daremos una pequeña introducción sobre ellas y explicaremos en que parte del desarrollo van enmarcadas.
- **Capítulo 3 - Metodología y fases de trabajo**: en este capítulo detallaremos la metodología que hemos usado y el por qué. Además, explicaremos las diferentes fases del trabajo.
- **Capítulo 4 - Análisis de requisitos**: en este capítulo hablaremos sobre los distintos requisitos tanto funcionales como no funcionales que encontraremos en nuestro sistema. Añadiremos también los distintos casos de uso que encontremos en nuestro sistema y diagramas de actividad de los procesos más importantes.
- **Capítulo 5 - Diseño del proyecto:** en este capítulo discutiremos sobre la estructura de nuestro proyecto, hablaremos sobre los distintos componentes que la forman y describiremos tanto su comportamiento como su relación entre sí.
- **Capítulo 6 - Implementación y pruebas**: en este capítulo abordaremos la parte práctica de este trabajo. Con las tecnologías mencionadas anteriormente elaboraremos el desarrollo del sistema paso por paso. Además, elaboraremos una serie de pruebas para corroborar el correcto funcionamiento del sistema.
- **Capítulo 7 - Conclusión y líneas futuras**: en este capítulo comentaremos las conclusiones que hemos alcanzado al finalizar el proyecto y las futuras mejoras que podrían surgir para el sistema.

Además, al final de la memoria están incluidas las referencias, (aquellas que hemos consultado para la realización de este trabajo) y un manual de instalación, (que incluirá los requisitos para su despliegue e instrucciones para poner en funcionamiento el sistema).

2

Tecnologías usadas

Introducción

En este capítulo se hablará sobre las distintas tecnologías sobre las que se basa este proyecto.

En cada una de ellas se dará una breve introducción, se explicará en detalle su uso y se desarrollará el porqué de su elección.

2.1 Symfony

Symfony es un framework PHP para desarrollar aplicaciones web basado en el patrón Modelo-Vista-Controlador. Un framework agiliza el desarrollo de aplicaciones al automatizar muchos de los patrones empleados. Un framework también agrega estructura al código, lo que facilita al desarrollador que escriba un código mejor, más legible y fácil de mantener. En última instancia, un framework facilita la programación, ya que empaqueta operaciones complejas en declaraciones simples.

A continuación, daremos más detalles sobre cada componente del patrón:

- El modelo está formado por la información que maneja el framework (en este caso la base de datos) y todo lo relacionado con la gestión de esta [2]. Se usará MySQL como motor de base de datos.

- La vista es la encargada de representar los datos provenientes del controlador de una forma visible para el usuario [2]. Symfony usa Twig, un motor de plantillas para la generación de vistas. En él podemos usar tanto HTML como CSS y JavaScript. En Twig podemos usar {{}} o {%%} para insertar sentencias propias de un lenguaje de programación, tales como condiciones, almacenar variables e iteraciones entre muchas otras cosas [11].

- El controlador es aquel componente que hace de intermediario entre la vista y el modelo. Se encarga de responder a eventos solicitando información al modelo que luego distribuirá hacia la vista [2].

Symfony ofrece una funcionalidad llamada componentes de symfony. Estos son una serie de librerías PHP que permiten implementar ciertas características ya desarrolladas. Por ejemplo, todo el manejo de la BD, uso de cuentas de correo, manejo de peticiones HTTP y sus respuestas, etc. En este proyecto trabajaremos con algunas de ellas como HttpFoundation, Routing, etc.

Además, ofrece una gran variedad de tutoriales y documentación para que cualquier usuario pueda aprender los conceptos de la manera más fácil posible apoyado sobre una gran comunidad.
Para ejecutar symfony en nuestro ordenador necesitaremos instalar PHP 7.2.5 o superior, composer para instalar paquetes PHP y el cliente de symfony con el cual correremos un servidor local para desplegar nuestra aplicación [3].

Con symfony desplegaremos una aplicación web que usaremos para interactuar y mostrar los datos que nuestra aplicación genere.

Hemos elegido symfony para nuestro proyecto por la existencia de experiencia previa. Otro factor ha sido que al estar basado en PHP este es un lenguaje muy parecido a Java y al haber trabajado con él durante toda la carrera la transición a PHP ha sido muy sencilla.

2.2 Rclone

Rclone es un programa sobre línea de comandos que nos permite gestionar los ficheros pertenecientes a sistemas de almacenamiento en nube, además de soportar una serie de protocolos como SFTP, HTTP, WebDAV, FTP y DLNA.

Rclone nos proporciona una seria de comandos con el mismo funcionamiento que algunos de los comandos UNIX más utilizados como `cp`, `mv`, `ls`, `tree`, `rm`, etc. Como ya hemos comentado antes, Rclone está diseñado para ser usado en línea de comandos, pero también permite el uso de scripts o a través de su API. Rclone es un software de código abierto escrito en Go. Posee una gran comunidad con un foro que atenderá todas tus dudas al instante [4].

Usaremos la API para así poder desacoplarla de la aplicación web. Se realizarán una serie de peticiones HTTP sobre esta API, cambiando los parámetros de las llamadas, para así controlar y modificar todo lo relacionado con las conexiones establecidas. Entre ellas se incluirá la creación de una nueva conexión, el listado de los archivos, el espacio disponible en un dispositivo, etc.

Se ha optado por Rclone por su gran versatilidad. Gracias a Rclone todo el trabajo de las conexiones (crearlas, manejar los ficheros…) se realiza de forma transparente. Solo debemos de proveerle de los datos que necesite y él se encargará de todo. Además, si en el futuro se decide implementar más tipos de conexiones solo tendremos que comprobar que ya están configuradas en Rclone y seguir los pasos para su uso. Rclone posee muchas más funcionalidades de las que hemos mencionado o que vayamos a usar, pero, serán interesantes si en un futuro decidiéramos aumentar las características de nuestra aplicación.

2.3 OAuth 2.0

OAuth 2 es un framework de autenticación que permite a una aplicación obtener un acceso (limitado) a un servicio HTTP como Facebook, GitHub o Twitter [6]. OAuth nos permitirá realizar operaciones (previamente autorizadas) sobre el servicio sin necesidad de estar intercambiando continuamente unas credenciales.

En el encontramos distintos roles que intervendrán en su desarrollo.

Propietario del recurso

Es el usuario que da permisos para que la aplicación pueda acceder y realizar acciones sobre el recurso autorizado.

Servidor de recursos

Se considera aquel que dispone de los recursos a los que deseamos acceder. Podremos obtener esos recursos a través de un token de acceso.

Cliente

Aquella aplicación que necesita acceder al recurso protegido. Antes deberá ser autorizada por el usuario y esa autorización será validada por la API.

Servidor de autorización

Tras la validación de las credenciales de usuario este generara un token de acceso para que el cliente haga uso de los datos y acciones a los que tenga permiso.

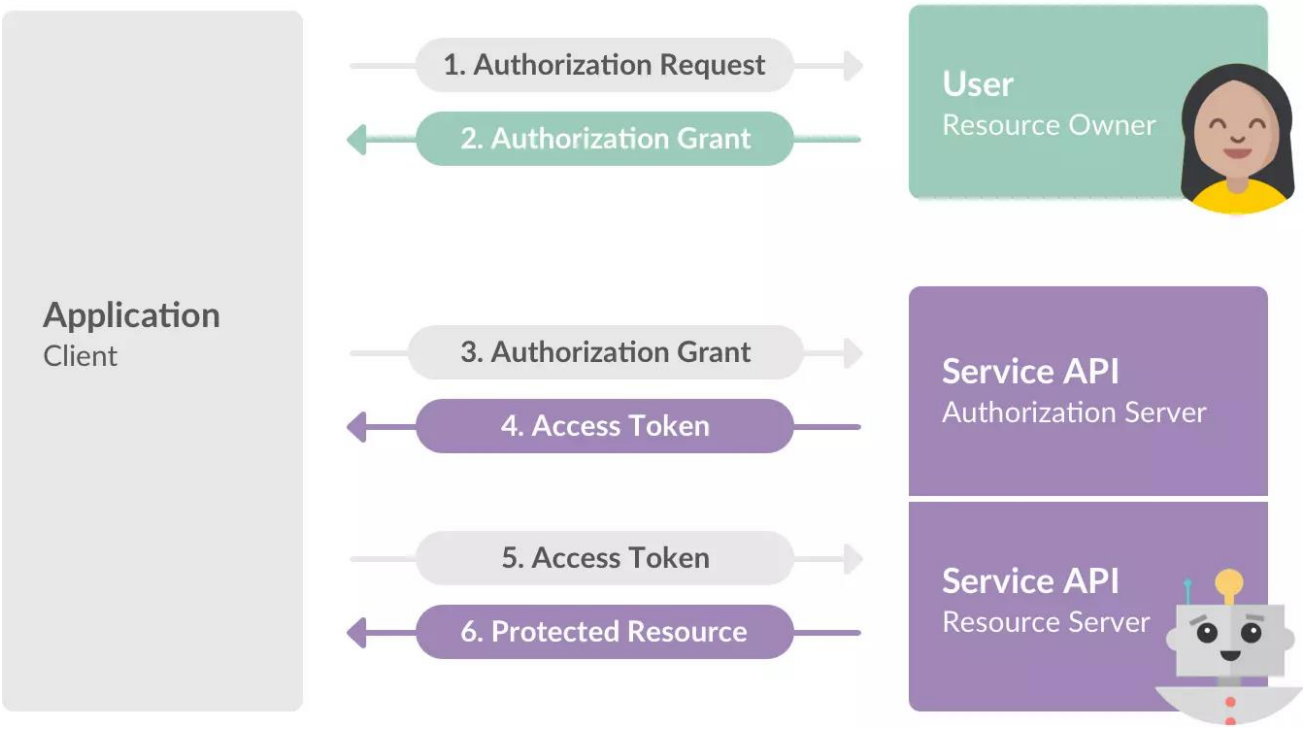

Figura 2.1 Diagrama de flujo para oauth 2.0 [19]

El token de acceso permite mantener el acceso a la API durante un tiempo prolongado y permite al usuario mantenerlo tras un tiempo de inactividad.

La información intercambiada entre cada parte implicada se detallará en la parte de diseño.

Se ha elegido OAuth 2.0 ya que es el protocolo usado por OneDrive y Drive para expedir tokens de acceso y estos tokens son usados por Rclone para identificarnos en esas conexiones.

2.4 SSDP

Es un protocolo que sirve para la búsqueda de servicios en una red. SSDP es la base del protocolo UPnP. UPnP es un conjunto de protocolos de comunicación que permiten que los dispositivos conectados a la red puedan detectar a otros dispositivos, comunicarse y compartir datos. A su vez usaremos DLNA para anunciar los servicios de nuestros dispositivos. DLNA es un superconjunto de UPnP.

Dentro de una red DLNA podemos encontrar diferentes roles [15]:

- **Digital Media Servers**: Dispositivos que proporcionan almacenamiento.
- **Digital Media Controllers**: Aquellos dispositivos que controlan el flujo de datos. Establecen el origen y el destino de estos.
- **Digital Media Renderers**: Dispositivos capaces de manejar el flujo de datos entrante. Como televisiones, móviles, etc.

El dispositivo que desee buscar un servicio en la red enviara un mensaje hacia la dirección multicast conocida 239.255.255.250 sobre el puerto 1900 a través del protocolo UDP (funciona como un anuncio que enviamos a todos los dispositivos de la red). Si algún dispositivo cumple con los requisitos responderá con la IP, el puerto y un puntero de descripción al fichero de descripción (formato XML). Este mensaje se manda en UDP unicast [5].

Se ha escogido este protocolo ya que a medida que se iba implementando en el proyecto cubría todos los requisitos requeridos.

Usaremos SSPD para encontrar aquellos dispositivos que tengan el servicio que nosotros queremos. En concreto será Digital Media Server que es aquel que nos indica que un dispositivo integra capacidad de almacenamiento.

Al usar DLNA se anunciará nuestro dispositivo como *Digital Media Servers* ya que este es el servicio que buscamos.

2.5 SFTP

SFTP está basado en el protocolo SSH que permite el acceso seguro a dispositivos remotos para el intercambio de ficheros. Con SFTP todos los datos intercambiados están cifrados. Es posible añadir un grado más de seguridad con el uso de claves publica y privadas. SFTP opera en el puerto 22 como SSH [8]. También posee otras características como: visualizar directorios, cambiar el nombre o borrar archivos.

Se ha elegido este protocolo ya que se encuentra implementado en Rclone, con ello todas nuestras conexiones requerirán el mismo manejo.

En el dispositivo de destino estableceremos un servidor SSH con su nombre, contraseña y puerto. Desde Rclone nos conectaremos con esos parámetros más la IP, que obtendremos al haber usado SSDP para la búsqueda de dispositivos.

2.6 HTTP

HTTP es el protocolo detrás de la World Wide Web. Cualquier transacción realizada en la web utiliza HTTP. HTTP proporciona una forma estandarizada para que los ordenadores pueden comunicarse entre sí.

HTTP mantiene una estructura cliente-servidor, es decir, el cliente inicia la comunicación y el servidor responde al mensaje. HTTP proporciona una serie de métodos para las peticiones realizadas por el cliente. Definiremos dos de ellas ya que serán las que usemos.

- Petición POST: Se usará para proporcionar cierta información al servidor, usualmente esta provocará un cambio en el estado del servidor.
- Petición GET: Para solicitar un recurso del servidor web.

Tras el envío de la petición el cliente esperara una respuesta. La respuesta estará formada por el HEAD, información acerca del servidor y la respuesta, y el contenido de la respuesta en sí [10]. En el HEAD se encuentra un dato importante, que es el código de respuesta. Este indicara si todo ha salido bien o si ha surgido algún problema [20].

2.7 Docker

Docker es un software de código abierto que permite de una manera sencilla crear, desplegar y correr aplicaciones usando contenedores. Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de forma rápida y confiable de un entorno informático a otro.

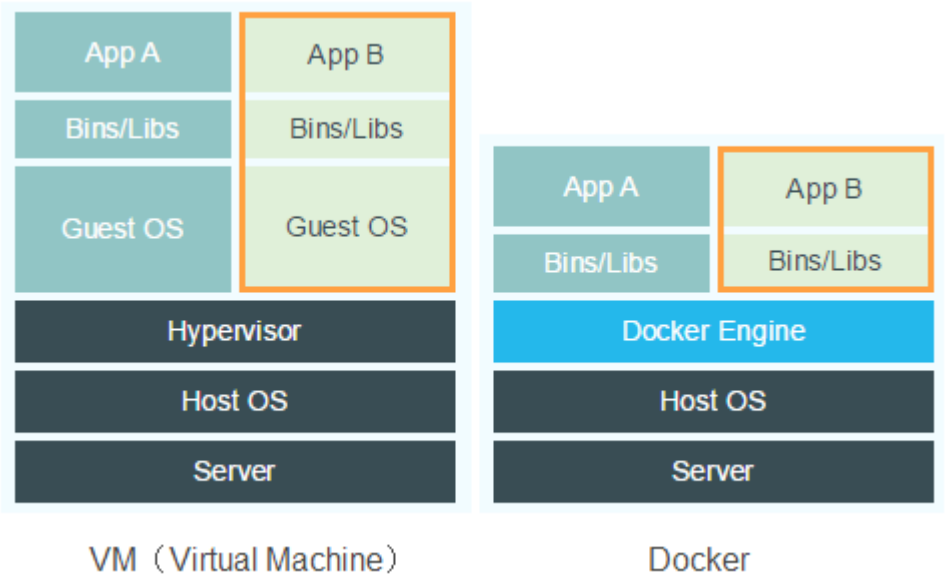

Figura 2.2 Docker vs Máquina virtual [21]

Los contenedores comparten los recursos con el anfitrión en lugar de virtualizar todo un sistema operativo. Esto permite que los contenedores compartan el mismo kernel del sistema operativo pudiendo ejecutarse cada uno como procesos aislados. En el caso de las máquinas virtuales, al virtualizar el hardware físico, cada una incluye una copia completa del sistema operativo, haciéndolas más pesadas y lentas en el arranque [12].

Algunas ventajas de usar Docker son:

- Portabilidad: Ya que todo funciona bajo el mismo software podemos ejecutar nuestra aplicación en cualquier ordenador siempre que cumpla con los detalles que le hemos especificado (si existieran).
- Menos uso de la memoria: Ya que compartimos el kernel con el host no hay que realizar operaciones de arranque porque ya están realizadas. Esto permite ahorrarnos memoria y reducir tiempo en el arranque del contenedor.
- Seguridad: Docker nos permite configurar el lanzamiento de los contenedores para que queden asilados entre sí. Esto permite que si uno se ve comprometido no afecte a los demás.
- Facilita el testing: Podemos configurar nuestro contenedor para probar partes específicas de nuestra aplicación. Al poder tener distintas configuraciones nos ahorramos tener que desplegarlas en distintos servidores.

Hemos elegido docker por sus ventajas a la hora de aplicarlo a un entorno Fog/Edge computing. Edge computing permite que los datos producidos por los dispositivos de la internet de las cosas se procesen más cerca de donde se crearon. Gracias a su fácil despliegue se tendría la capacidad de

habilitar un dispositivo Edge en casi cualquier entorno. Además, el caso contario también es posible, eliminar el despliegue y ubicarlo en otro punto es muy sencillo.

Componentes básicos

Imagen:

Es una plantilla con instrucciones para crear un contenedor Docker. Para crear tu propia imagen se precisa de un *dockerfile* que es un fichero con los pasos necesarios para construir tu imagen.

Contenedor:

Es una instancia ejecutable de una imagen.

Volúmenes:

Son usados para guardar información de los contenedores de forma persistente, ya que cuando estos se borran se pierde todo su contenido.

Red: Es un canal que permite la comunicación entre todos los contenedores aislados [14].

Dentro del sistema de redes de Docker encontramos 5 tipos de controlador.

- **Bridge/NAT**: Es el controlador por defecto. Se utiliza cuando tienes un único host y quieres que los contenedores se comuniquen entre sí. Los contenedores también podrán comunicarse con el host y el exterior. Los docker recién creados se unirán a la red por defecto (docker0). Se podrá crear tu propia red bridge que tendrá el mismo esquema que la predeterminada, pero ofrecerá un conjunto distinto de direcciones IP. Al crear tu propia red se generará un DNS interno con los nombres e IPs de las máquinas en su red, esto no ocurre si se conectan a la red por defecto.

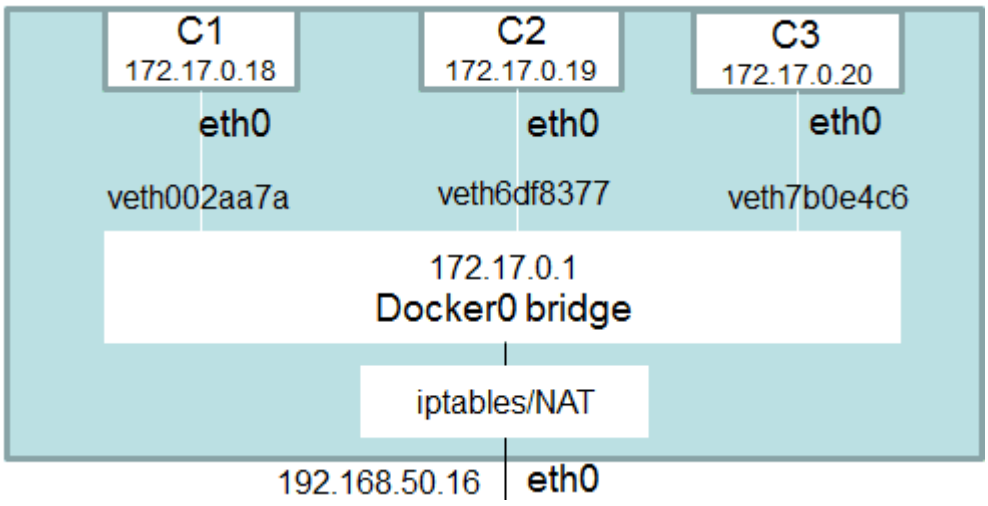

Figura 2.3 Red docker en modo bridge [22]

- **Host**: Lo que hace esta red es eliminar el aislamiento entre el host y los contenedores, por lo que un contenedor no recibe su propia IP, sino que utiliza la del host. Para poder acceder a nuestros contenedores tendremos que usar la IP del host más el puerto con el que lo hayamos mapeado. Los contenedores tendrían acceso a las mismas direcciones que el host.
- **Overlay**: Se utiliza cuando los contenedores se ejecutan en un clúster y necesito que hablen entre ellos independientemente de donde se encuentren dentro del clúster. Funcionaria igual que el modo bridge.
- **None**: Con esto decimos a Docker que no queremos que nuestro contenedor este asociado a ninguna red.
- **Macvlan**: Este controlador asigna direcciones Mac a los contenedores para que parezcan dispositivos físicos, es decir, otorga una IP de nuestra red local a cada contenedor. Para que este tipo de redes funcione necesitamos que la tarjeta de red del host esté en modo promiscuo, lo cual significa que necesita estar a la escucha de todos los paquetes que viajan por dicha red. Así conseguimos que los contenedores puedan ser alcanzados desde fuera de la red.

Arquitectura Docker

Docker utiliza una arquitectura cliente-servidor. El cliente y el demonio de Docker se comunican mediante una API REST, a través de sockets UNIX o una interfaz de red [13].

Se compone de:

- Docker Daemon: Escucha las solicitudes de la API de Docker para el manejo de contenedores e imágenes. También puede comunicarse con otros demonios para administrar los servicios de Docker.
- Docker Client: Es la forma principal con la que los usuarios interactúan con la API de Docker. El cliente envía un comando al demonio y este se encarga de ejecutarlo.
- Docker Registry: En él se almacenan las imágenes que generamos, permitiéndonos poder usarlas desde cualquier parte. Docker Hub es un registro público que cualquiera puede usar, en el encontraremos las imágenes oficiales de la mayoría de software que queremos usar.

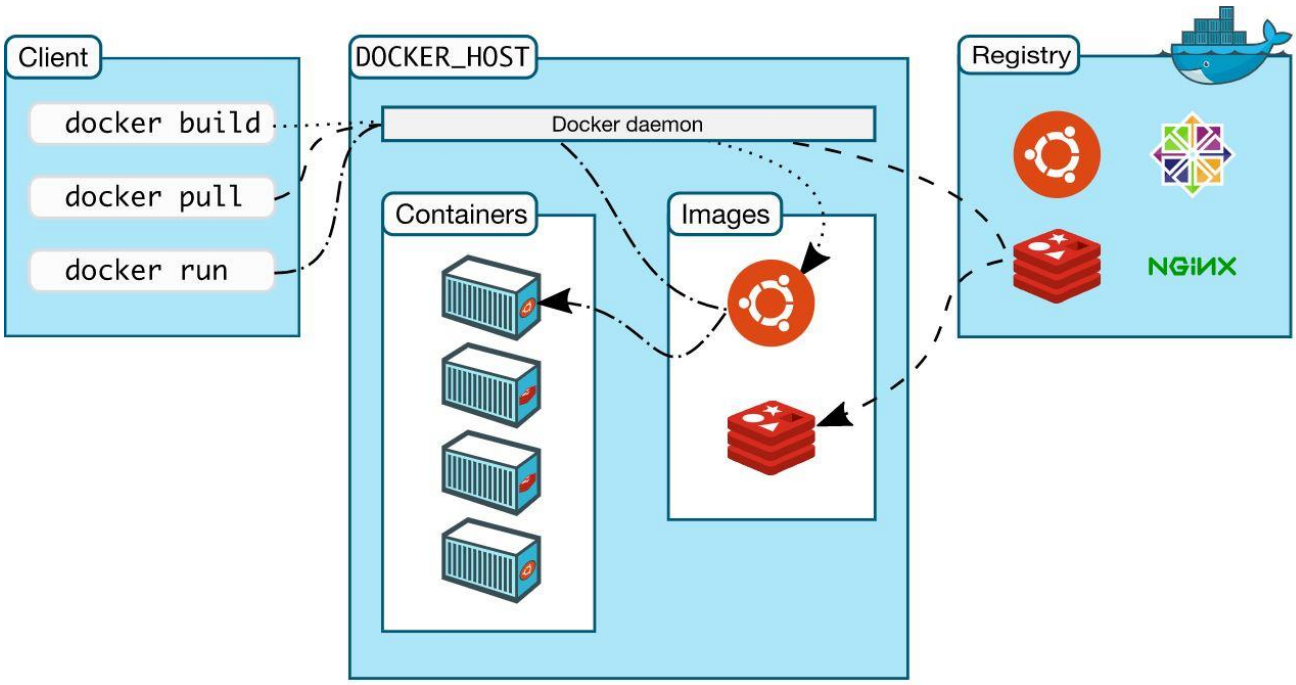

Figura 2.4 Arquitectura de docker [13]

2.8 MySQL

MySQL es un sistema de gestión de base de datos relacionales de código abierto. Una base de datos es una colección estructurada de registros o datos almacenados en un sistema informático y organizados de tal manera que se pueden buscar y recuperar rápidamente la información que en ella reside. Las siglas SQL corresponden al lenguaje en el que se encuentra implementado.

Una base de datos MySQL contiene uno o más tablas, cada una de las cuales contiene registros o filas. Dentro de estas filas son varias columnas o campos que contienen los datos en sí.

Hay tres formas principales de interactuar con MySQL: usando una línea de comando, a través de una interfaz web como phpMyAdmin y a través de un lenguaje de programación como PHP [24]. En este proyecto se usará la tercera opción a través del componente de symfony Doctrine [23].
Se ha elegido MySQL por tener experiencia previa con él, se ha visto en la carrea, y por su gran documentación al ser de los más utilizados.

2.9 GitHub

GitHub es un sistema de control de versiones cuya principal característica es el almacenamiento de código. GitHub crea un entorno que le permite almacenar su código en un servidor remoto, le brinda la capacidad de compartir su código con otras personas y facilita que más de una persona agregue,

modifique o elimine código en el mismo archivo y proyecto. GitHub está basado en Git, que es un sistema de control de versiones, gratuito y de código abierto.

Los sistemas de control de versiones son software que realizan un seguimiento de cada versión de cada archivo en un proyecto, asignando una marca de tiempo de cuándo se creó esa versión y el autor de esos cambios [25].

Se ha elegido GitHub por haber sido ya utilizado a lo largo de los estudios cursados.

3

Metodología y fases de trabajo

Introducción

En esta sección se discutirá la metodología de trabajo usada y las distintas fases realizadas para el desarrollo de este.

3.1 Desarrollo incremental

Con el desarrollo incremental empezaremos con la construcción del sistema con sus requisitos más básicos y a medida que el cliente (en este caso el tutor) nos dé el visto bueno iremos añadiendo más

especificaciones hasta alcanzar el desarrollo final del sistema. Todas estas fases están relacionadas y además encontramos una gran sinergia entre ellas.

- **Requisitos**: Se presentarán los requisitos tanto funcionales como no funcionales que debe de cumplir el sistema. Incluiremos además el diagrama de casos de uso del sistema.
- **Investigación**: Se realizará una búsqueda exhaustiva de las diferentes tecnologías que nos permitirán desarrollar el sistema. Daremos una breve descripción de cada uno de ellas y especificaremos donde y como se van a utilizar.
- **Diseño e implementación del agregador de almacenamiento**: Con los requisitos y las tecnologías ya definidos entraremos de lleno en la parte troncal del trabajo. Desarrollaremos el agregador de almacenamiento en su forma más básica, con ello nos referimos al almacenamiento en la nube, para luego ir añadiendo algunas formas de almacenamiento más.
- **Diseño de las políticas de almacenamiento**: Investigaremos y desarrollaremos algunas políticas de almacenamiento para nuestro agregador. Con cada una de ellas decidiremos, de forma automática, en qué lugar se ubicará la información que llega al agregador.
- **Diseño e implementación del servicio web**: Desarrollaremos un servicio web para la configuración del agregador por parte del usuario. Mostraremos también algunas estadísticas relevantes para este.
- **Validación y pruebas**: Realizaremos una serie de pruebas en cada parte para asegurarnos que, tanto individualmente como en conjunto se cumplen con todos los requisitos estipulados.
- **Documentación**: A medida que se vaya desarrollando el proyecto y de una forma paralela, se ira cumplimentando la documentación requerida para así tener un mayor control sobre esta y no dejar nada sin documentar.

Hemos elegido este método ya que no todos los requisitos están cerrados [1]. A medida que se avance en el proyecto se podrán introducir nuevas funcionalidades o reestructurar algunas ya definidas, debido a su complejidad. El desarrollo incremental nos permite retroceder si encontramos algo nuevo que añadir o eliminar y al tener continuas "entregas" al tutor podemos saber si se avanza adecuadamente.

3.2 Análisis de las Iteraciones

Número	Fecha	Nombre	Comentarios
1	11/03/2021	Requisitos	Analizados los mismos se ha decidió extender los requisitos no funcionales y eliminar algunos funcionales que dificultaban en exceso el desarrollo del trabajo.
2	25/03/2021	Investigación	Se ha dado el visto bueno a las tecnologías a usar, pero se decide realizar un mayor esfuerzo en el análisis de Rclone.
3	29/04/2021	Diseño e implementación del agregador de almacenamiento	La estructura inicial ya no es válida y se plantea el uso de la API de Rclone.
4	27/05/2021	Diseño de las políticas de almacenamiento	Se deciden cual va a ser la estructura para definir las políticas y aquellas que serán implementadas.
5	17/06/2021	Diseño e implementación del servicio web	Es necesario rediseñar la parte de la creación de conexiones.
6	22/07/2021	Validación y pruebas	Tras comprobar que se cumplen todos los requisitos, se diseñan las pruebas de validación.
7	2/9/2021	Documentación	La documentación necesitas de algunos ajustes gramaticales y la reestructuración del capítulo de diseño.

4

Análisis de requisitos

Introducción

Antes de empezar con el diseño del proyecto, deberemos establecer los requisitos (tanto funcionales como no funcionales) que debe de cumplir nuestro sistema y los casos de uso que desarrollaremos.

4.1 Requisitos funcionales

Los requisitos funcionales hacen referencia al comportamiento que debe cumplir el sistema.

ID	Nombre	Descripción
RF1	Almacenar archivo	El sistema permitirá la subida de fichero, que se almacenará en alguna conexión disponible, a través del sistema de políticas establecido.
RF2	Descargar archivo	El usuario podrá descargar un archivo de las conexiones establecidas, que almacenará su navegador.

RF3	Eliminar archivo	Se podrán eliminar archivos de las conexiones establecidas.
RF4	Añadir dispositivos	Se podrá añadir distintos espacios de almacenamiento. Tales como: OneDrive, Drive y cualquier dispositivo que permita DLNA y SFTP.
RF5	Estadísticas	Se mostrarán información sobre la conexión como: historial de subidas y descargas, espacio usado etc.
RF6	Políticas	A través de un fichero, debidamente estructurado, el usuario podrá establecer una serie de políticas, ya configuradas, con sus propios argumentos.
RF7	Autenticación	Un usuario podrá crear una cuenta a través de un nombre y una contraseña. A esta cuenta estarán vinculados las conexiones que cree.
RF8	Visualizar espacio	Al usuario se le mostrará el contenido de sus conexiones en una misma página.

4.2 Requisitos no funcionales

Son aquellas cualidades que el sistema debe de tener.

ID	Nombre	Descripción
RNF1	Acceso local	La aplicación podrá ser consultada a través de cualquier dispositivo que esté conectado a la misma red en la que se encuentre desplegada.
RNF2	Aplicación fácil de usar	La aplicación estará desarrollada de tal forma que la curva de aprendizaje del usuario sea mínima.
RNF3	Escalabilidad	Si en un futuro de decide de aumentar sus capacidades no debe de haber problemas para añadirlas.
RNF4	Responsive	Será capaz de adaptarse a cualquier tipo de pantallas.

4.3 Diagrama de casos de uso

Usando como base los requisitos definidos anteriormente podemos diseñar el diagrama de casos de uso.

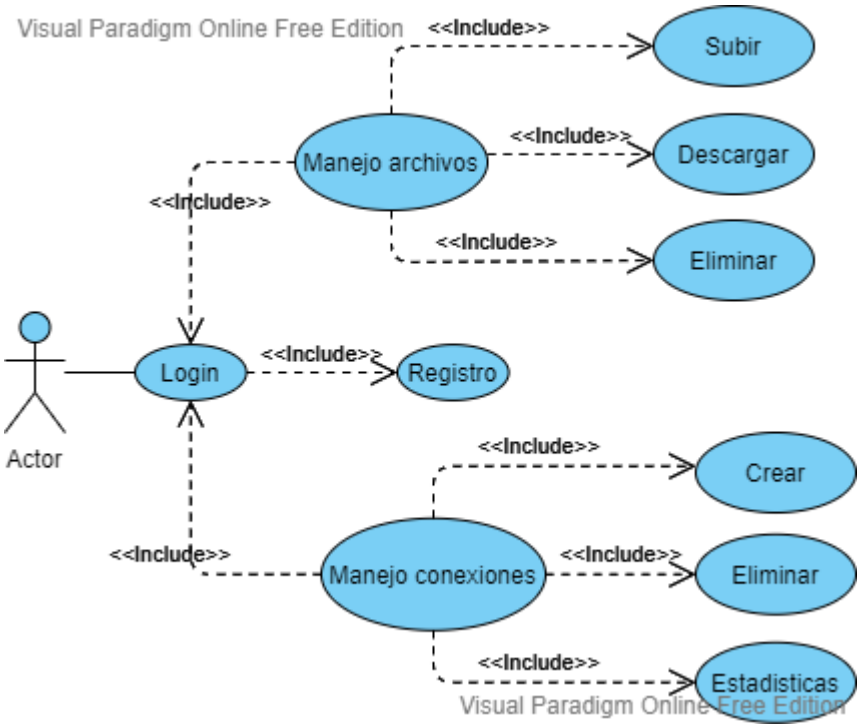

Figura 4.1 Diagrama de casos de uso

Nos encontramos con un solo actor (el usuario) que puede realizar dos grandes tareas. Manejar archivos y manejar conexiones. Estos a su vez incluyen tareas básicas para su uso. Dentro del manejo de archivos nos encontramos subir (mandar un archivo a un dispositivo de almacenamiento), descargar (bajar el archivo seleccionado) y eliminar (elimina el archivo seleccionado). Dentro del manejo conexiones encontramos: crear (estableceríamos una conexión de un tipo determinado), eliminar (dentro de las conexiones ya creadas borraríamos la que hemos seleccionado) y estadísticas (información acerca de esa conexión). Todo esto posible gracias a un sistema previo de login (con un usuario y contraseña) y asociado a un registro anterior.

4.4 Diagrama de actividad

A continuación, usaremos diagramas de actividad para modelar el flujo de las actividades más complejas.

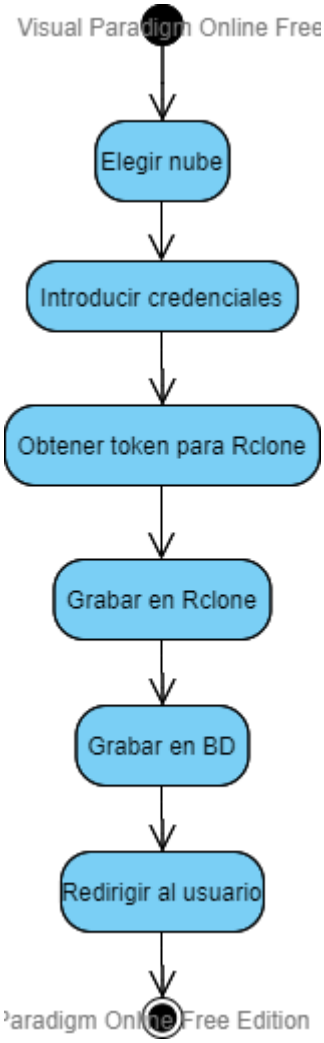

Figura 4.2 Diagrama de actividad para la creación de una conexión nube

Pasaremos a detallar el flujo de acción en la figura 5.2. Tras elegir la nube se deberá introducir las credenciales para identificarse. Tras la validación de estas se devolverá un código con el cual se generará el token de acceso. Con este token crearemos la conexión en Rclone y grabaremos en la BD una fila para asignar esa conexión al usuario activo. Tras lo cual se redirigirá al usuario a una página determinada.

Figura 4.3 Diagrama de actividad para la subida de ficheros

Pasaremos a detallar el flujo de acción en la figura 5.3. Tras seleccionar el archivo y la política a aplicar este se subirá a nuestro proyecto. Rclone tomara el archivo y lo enviara al destino elegido (a través de la política). Insertaremos en la BD una fila que nos servirá como un historial y se volverá a redirigir al usuario.

Se ha omitido el control de errores entre cada acción para aportar legibilidad al diagrama. Toda acción conlleva un control de errores.

5

Diseño de la aplicación

Introducción

En esta sección daremos detalles del diseño del proyecto. Describiremos la función de cada componente y una breve explicación de como lo realizan. Se proporcionarán más detalles en la parte de implementación.

5.1 Diseño inicial

Nuestra aplicación consta de 4 módulos, que a su vez estarán implementados cada uno sobre un contenedor. La elección de contenedores se debe a que esta tecnología permite la implementación de un entorno Edge Computing, el cual es la línea en la que se basa este proyecto.

A continuación, se explicará la función de cada módulo:

- **La aplicación web**: Esta esta implementada en el framework symfony y actuará como una capa visual para los datos que maneje nuestra aplicación. Se encargará de la comunicación con los otros módulos. Es el centro de todo el proyecto.
- **La API Rclone**: Esta estará implementada con el software Rclone y responderá a las peticiones que ejecutemos desde nuestra aplicación web. Nos permitirá conectarnos a los distintos espacios de almacenamiento y realizar distintas tareas sobre estos.
- **La base de datos**: En este caso hemos elegido MySQL y en ella guardaremos los datos necesarios para el correcto funcionamiento de nuestra aplicación.
- **API SSDP:** Consiste en otro proyecto symfony (con la mínima cantidad de dependencias) que actúa como una API y nos devolverá los dispositivos disponibles en la red local.

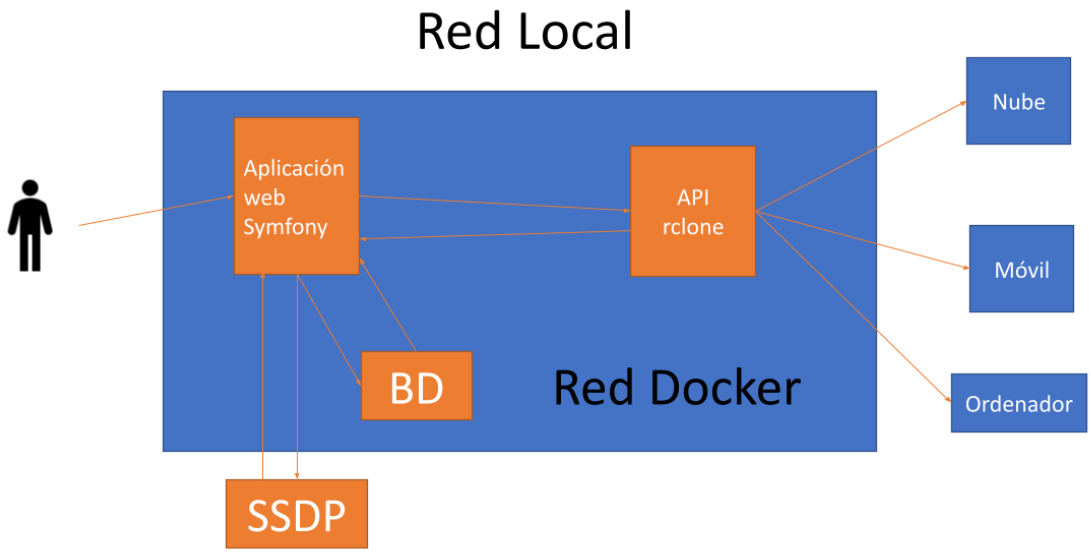

Figura 5.1 Diseño de la aplicación

Este diseño a su vez será implementado en docker (la figura 5.1 muestra su diseño en esta tecnología). En siguientes secciones se detallarán como se llevará a cabo.

5.2 Diseño de la aplicación web

Como ya explicamos anteriormente symfony se basa en el patrón de diseño modelo-vista-controlador. En este apartado veremos cómo están relacionados esos conceptos. En este apartado comentaremos los componentes más básicos para la creación de una página en symfony. Mas adelante se explicarán las creadas para este proyecto.

5.2.1 Controladores

Cuando nuestra aplicación reciba una solicitud (en forma de URL) esta llamará al controlador encargado de manejar esa solicitud que devolverá una respuesta. Esta puede ser generar una página (vista) o llamar a otro controlador. Los controladores también pueden devolver datos, pero en nuestro caso generarán una página o nos redirigirán a otro controlador, que generará otra página.

```
<?php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
class PaginaInicioController extends AbstractController
{
   /**
    * @Route("/", name="pagina_inicio")
    */
   public function index(): Response
   {       return $this->render('pagina_inicio/index.html.twig', [
         'controller_name' => 'PaginaInicioController',
      ]); } }
```

El acceso a los controladores se realiza a través del componente Routing. Este puede definirse tanto en YAML, XML, PHP o usando anotaciones. En este proyecto se usarán anotaciones. Routing permite asociar a una determinada URL un controlador, esto permite que al introducir una URL esta invoque al controlador al que se encuentra ligado. Estas anotaciones se sitúan encima de aquel controlador al que van a estar asociadas. Se proporcionará dos argumentos: la URL a la que responderá el controlador y un nombre para poder referirse al controlador de una manera sencilla en otras partes del proyecto [16].

Tras la llamada al controlador este tendrá que devolver una respuesta. En el código anterior se dirige al usuario a la página implementada en el fichero index.html.twig. Es posible enviar información hacia esa página en la forma de pares clave=>valor. Siendo la clave el identificador para acceder y el valor la información en sí.

5.2.2 Vistas

Sea cual sea la respuesta del controlador al final se generará una página (vista).

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>{% block title %}Welcome!{% endblock %}</title
      {% block stylesheets %}
      {% endblock %}
      {% block javascripts %}
         {#{{ encore_entry_script_tags('app') }}#}
      {% endblock %}
   </head>
   <body>
      {%block navbar %}
         {% if is_granted('ROLE_USER') %}
            <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
               <a class="navbar-brand" href="{{path('lista_archivos')}}">Listado</a>
               <a class="navbar-brand" href="{{path('upload')}}">Subir</a>
               <a class="navbar-brand" href="{{path('lista_conexion')}}">Conexiones</a>
               <a class="navbar-brand" href="{{path('app_logout')}}">Log Out</a>
            </nav>
         {% endif %}
      {%endblock%}
   {% block body %}{% endblock %}
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>
</body></html>
```

Para la creación de las vistas usaremos Twig, que es un motor de plantillas desarrollados para PHP. El código de las páginas estará compuesto por HTML, Bootstrap y las sentencias propias de Twig.

Se utiliza HTML para dividir el código en distintos bloques para tener cierta organización.

Bootstrap se encarga de todo el aspecto visual, tales como, botones, tablas, barra de navegación etc.

Twig es aquel que modifica y muestra los datos recibidos desde el controlador. En Twig se destacan dos sintaxis especiales:

- **{{…}}:** Para mostrar los datos por pantalla.
- **{%…%}:** Contiene la sintaxis propia de Twig. Podrá albergar declaraciones de variables, bucles, llamadas a funciones etc.

Twig posee un sistema de herencia el cual permite generar una estructura base que seguirán todas aquellas paginas que lo deseen [26]. El código anterior muestra la plantilla base a partir de la cual

heredaran las demás en el proyecto, por lo cual es la plantilla base. A través de la definición de bloques, en la plantilla base, las plantillas hijas usarán estos mismos bloques para redefinir el contenido.

5.2.3 Modelo

En esta sección trataremos como symfony se comunica con la base de datos. Este comportamiento es gracias al componente Doctrine. Crearemos en nuestro proyecto una clase entidad en la cual definiremos todos las variables que deseamos que tenga nuestra tabla. Doctrine asociara esas variables con atributos y creara la tabla correspondiente. Esto permite manejar las clases como si de un objeto se tratara (en un lenguaje orientado a objetos).

```
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-
dbal/en/latest/reference/configuration.html#connecting-using-a-url
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
#
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://jose:jose@db:3306/TFG?serverVersion=5.7"
#DATABASE_URL="postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=13&char
set=utf8"
###< doctrine/doctrine-bundle ###
```

En el archivo env, en la raíz de nuestro proyecto, nos encontramos la configuración de la base de datos. Deberemos de elegir cual es nuestro motor de base de datos e introducir los datos que nos pidan.

Para MySQL quedaría así mysql://db_user:db_password@IP:puerto/db_username.

- db_user es el nombre del usuario con acceso a la base de datos.
- db_password es la contraseña asociada al usuario.
- IP es la dirección donde se encuentra nuestra base de datos.
- puerto donde se despliegue la base de datos.
- db_username es el nombre que hemos dado a nuestra base de datos.

A continuación, será en el controlador donde realizaremos las acciones correspondientes para guardar o borrar entradas en la base de datos.

```
$entityManager = $this->getDoctrine()->getManager();
```

```
$user = new User();
$user->setUsername(‘’);
$user->setPassword(‘’);
$entityManager->persist($user);
$entityManager->flush();
```

Al tratarse de un controlador podremos obtener la clase Doctrine, y el gestor de entidades de este. Así a través de la variable entityManager obtenemos la forma para manejar la base de datos.

Para introducir datos en la BD se crea un objeto asociado a la entidad que contendrá los datos a grabar. A este introducimos los atributos que luego pasaran a ser campos en las tablas de la BD. Después se llama al método persist para indicar que se va a realizar un cambio, pero esto no ejecuta ninguna sentencia. Para que se produzca el registro se deberá también usar la función flush.

Para buscar un campo en la BD deberemos acudir a la clase Repository. Cuando se crean las entidades, además se crea una clase asociada llamada repository. Cada entidad tiene una asociada con el nombre de la entidad más la palabra repository. En ella se encuentran los métodos para buscar datos en las entidades asociadas. Todas las clases repository tienen implementados cuatro métodos de búsqueda, aunque se pueden definir métodos propios.

- **find($id):** Devuelve la fila de la tabla que contenga ese ID.
- **findOneBy($criterio):** Devuelve la primera fila que cumpla ese criterio.
- **findAll():** Devuelve todas las filas de la tabla.
- **findBy($criteria):** Devuelve todas las filas que cumplan el criterio.

Al terminar no es necesario llamar a la función flush ya que no se altera la BD.

Para borrar una entrada en la BD primero llamaríamos a un método de búsqueda para después borrar el objeto con remove.

```
$conexione = $ConexionRepository->findBy($criteria);
$entityManager->remove($conexione);
$entityManager->flush();
```

Tras lo cual llamamos a flush para ejecutar la sentencia.

5.3 Diseño de la API Rclone

Rclone será aquel encargado de todo lo relacionado con la comunicación con los distintos dispositivos, por lo tanto, tendremos que comunicarnos con él para realizar todas las tareas pertinentes.

Para ello se usará la API que este trae implementada. Se iniciará Rclone como un servidor HTTP que se encontrará a la escucha de peticiones. A través de estas peticiones se obtendrá información acerca de un dispositivo o se realizará una tarea sobre este que modificará su estado. Las peticiones se realizarán usando POST con el formato JSON más la información necesaria para su ejecución. JSON es un formato de texto sencillo para el intercambio de datos. Básicamente es una secuencia de pares clave=>valor.

La petición consta de dos parámetros:

- La IP donde este alojado el servidor más una dirección que indica el comando a usar.
- Las opciones que requiera ese comando. Esta será la parte en formato JSON. La cantidad de información para enviar dependerá de cada comando.

Por ejemplo, si nuestro servidor se encuentra en la dirección 192.168.0.105 y queremos borrar una conexión en concreto la petición quedaría:

1º 192.168.0.105/config/delete seria nuestra dirección.

2º {"name":"nombre de la conexión que se desea borrar"} seria al JSON.

La lista concreta de comandos y sus opciones puede encontrarse aquí [17].

5.3.1 Conexiones a los espacios de almacenamiento

En nuestra aplicación hemos implementado 4 formas de almacenamiento, dos nubes (OneDrive y Google drive), móvil y ordenador.

Para las nubes se usará OAuth 2.0, ya que este protocolo permite obtener un token con el cual autenticarse para realizar tareas sobre esa nube. Este token será usado por Rclone para poder interactuar con la nube. Antes de empezar con el flujo de OAuth se debe de crear un proyecto para cada nube que usemos. Este proyecto contendrá un *client ID*, un *client secret*, los permisos que concede el token expedido y una URL de respuesta que albergará el código con el cual se obtendrá el token.

En la página de Rclone viene explicado cómo se realiza este registro para OneDrive [30] y para drive [31].

En este punto el proyecto creado en el registro de aplicación de Google se encuentra en modo pruebas. Esto impone una serie de limitaciones como que hay que establecer con anterioridad los correos que tendrán permiso para obtener token en nuestra aplicación. En OneDrive no se han encontrado esas limitaciones.

El *client ID y client secret* se generan al crear el proyecto. La URL de respuesta estará compuesta de la dirección en la que se encuentre nuestro proyecto más la ruta que apunte al controlador que gestiona la obtención del token y su posterior modificación al formato que Rclone utiliza.

Para OneDrive seria: http://localhost:8080/inicio/lista_conexion/crear_onedrive.

Para drive seria: http://localhost:8080/inicio/lista_conexion/crear_drive.

Siendo localhost la dirección de *loopback* en nuestra máquina.

Los permisos serán aquellos que nos permitan control total de todos los ficheros además de aquel que nos permite un acceso prolongado a la nube, aunque este tendrá una caducidad. Para permitir ese acceso prolongado se nos otorgara dos tokens. El primero será aquel que sirva como autenticador, pero tendrá una caducidad. El segundo será aquel que usemos para regenerar el primer token para así poder tener acceso a la nube durante un periodo de tiempo mayor. Tras la caducidad del segundo token deberemos de volver a pedir un token de acceso.

Vamos a explicar el intercambio de información que se produce para obtener el token.

El primer paso será obtener el código con el cual obtendremos el token. La URL a la que se envía esta petición es distinta para cada servicio. Para ello se deberá de realizar una petición GET con los siguientes parámetros:

- ***client_id*:** El ID creado por nuestro proyecto.
- ***scope*:** Todos aquellos permisos necesarios para manipular la nube.
- ***redirect_uri*:** La URL de respuesta que contendrá el código tras autenticarnos.
- ***response_type*:** El tipo de respuesta que deseamos. En nuestro caso será *code.*

Tras el envío de la petición se nos pedirá que nos autentiquemos en la nube. Tras ello se nos responderá en la URL que especificamos anteriormente más el código.

Tras la obtención del código realizaremos una segunda petición (ahora con POST) para obtener el token. Los parámetros que requiere son:

- ***client_id*:** El ID creado por nuestro proyecto.
- ***client_secret*:** El cliente secreto creado por nuestro proyecto.
- ***redirect_uri*:** La URL de respuesta que contendrá la respuesta a la petición. Deberá ser la misma que en la primera petición.
- ***code*:** El código de autenticación obtenido en la primera petición.

Si la petición es correcta la respuesta contendrá en formato JSON los siguientes parámetros:

- ***token_type:*** Identifica el tipo de token devuelto.

- ***expires_in:*** La vida útil restante del token de acceso en segundos.
- ***access_token:*** El token que nos permite manipular la nube.
- ***scope:*** La lista de permisos dados al token.
- ***refresh_token:*** Si se ha marcado el permiso de *offline* este token permitirá un acceso prolongado regenerando el token de acceso.

Para solicitar un nuevo token de acceso podemos volver a realizar todo el proceso o usar el token de refresco. Para lo segundo seria el mismo proceso cambiando el código de autenticación por el token de refresco y la respuesta seria la misma ya con el token de acceso regenerado [32].

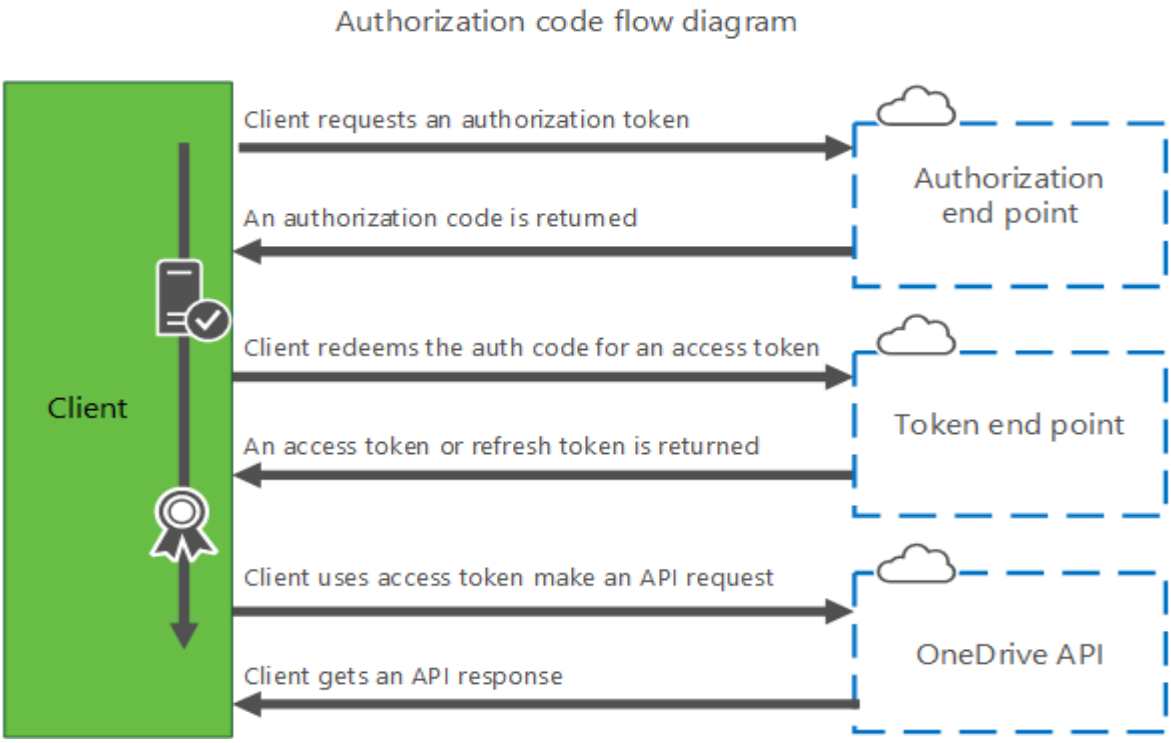

Figura 5.2 Flujo de OAuth en OneDrive [32]

Para los otros dos dispositivos se ha concebido un escenario un poco más complejo. Se necesitará un servidor SSH a través del cual Rclone accederá al dispositivo. Uno de los parámetros que se necesitan es la dirección IP, está la obtendremos a través de un servidor DLNA. Este anunciara sus servicios en la red más cierta información adicional (en la que se encuentra la IP). A través de SSDP obtendremos la IP y así se podrá establecer una conexión por SFTP. Para móvil se han usado dos apps para implementar DLNA [27] y SFTP [28].

En la app de DLNA no hay que seleccionar ninguna carpeta en específico ya que solo queremos anunciar el servicio.

En la app de SFTP debemos de establecer un nombre y una contraseña. En las opciones, en directorio raíz, debemos seleccionar DCIM para que apunte a nuestro almacenamiento principal.

Para DLNA en Linux se ha usado DLNA ReadyMedia [29]. Tras su instalación no es necesario configuración. Aunque se puede cambiar el nombre con el cual se anuncia.

Para ssh en linux se ha usado el que viene por defecto con la distribución.

Se ha usado el puerto 2222 para las conexiones SFTP.

5.4 Base de datos

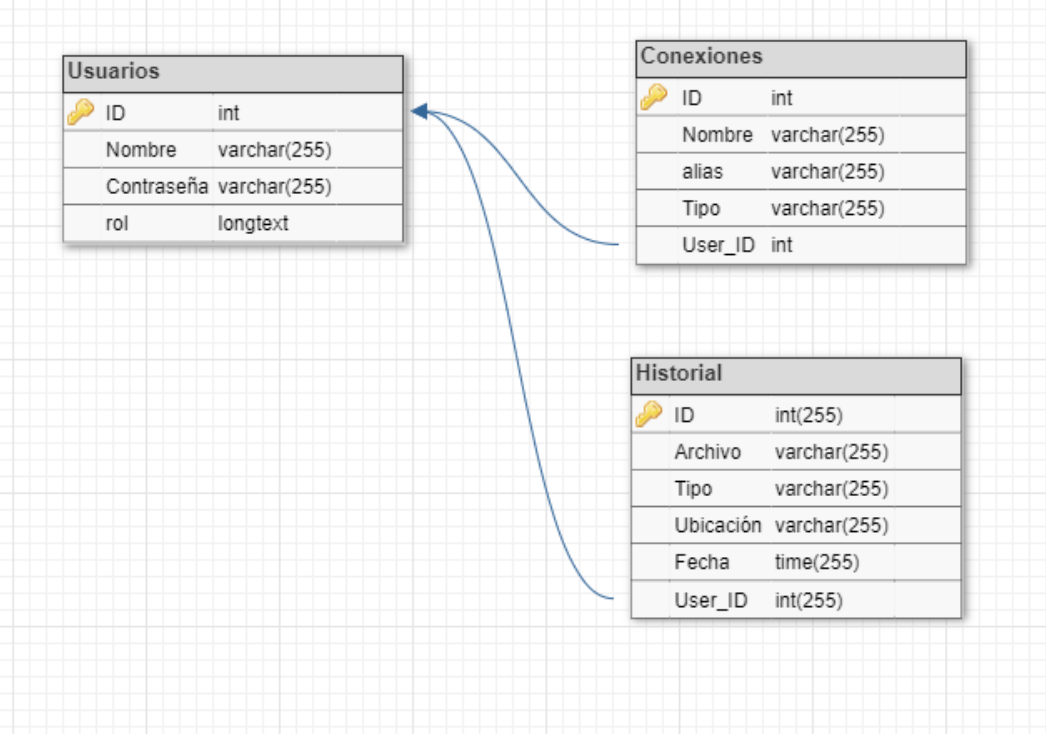

Figura 5.3 Diagrama de clase para la BD

La BD esta implementada de tal manera que nos facilite la asociación de ciertos campos para el buen funcionamiento de la aplicación y para mostrar información útil al usuario. En la figura 5.3 podemos ver la relación que guardan entre si las entidades. A continuación, describiremos cada una.

Usuarios: Aquella tabla en la que se registra las cuentas que tienen permisos para entrar a la aplicación.

- **ID:** El identificador único para cada fila de la tabla.
- **Nombre:** Es uno de los campos usados para el *login* en la aplicación. Es un identificador para el usuario que se registra.

- **Contraseña:** El segundo campo usado para el *login.* Junto con el nombre forma la pareja para entrar en la aplicación.
- **Rol:** Aquel que el usuario tiene en la aplicación. Todos los usuarios registrados tendrán el mismo.

Conexiones: Aquella tabla que guardas las conexiones activas para un usuario concreto.

- **ID:** El identificador único para cada fila de la tabla.
- **Nombre:** Identificador de la conexión en Rclone.
- **Alias:** Seudónimo asociado al nombre, para no mostrar este. El usuario podrá cambiarlo para que sea más representativo.
- **Tipo:** El tipo de la conexión. Serán drive, OneDrive, sftp_movil y sftp_ordenador. Estos serán nombres internos con los cual referimos a las dos nubes y a los dos tipos de dispositivos.
- **User_ID:** El ID del usuario al que está ligado esta conexión.

Historial: Aquella tabla que guarda un listado de las subidas y descargas de ficheros. Para que el usuario sepa la ubicación de estos para posteriores consultas.

- **ID:** El identificador único para cada fila de la tabla.
- **Nombre:** Nombre del archivo.
- **Tipo:** Si la operación ha sido subida o descarga de archivo.
- **Ubicación:** Procedencia del archivo en su descarga o lugar al que se ha subido.
- **Fecha:** Cuando se produjo la operación.
- **User_ID:** El ID del usuario al que está ligado este historial.

5.5 API SSDP

Tratará de otro proyecto en symfony (con la cantidad mínima de dependencias, es decir con el menor número de componentes) cuya función será una API que nos devuelve el escaneo de la red en la que se encuentra. De la respuesta que nos dé nos interesa la IP y nombre de los dispositivos encontrados. La respuesta devuelta se encuentra en formato JSON. Este escaneo se realiza a través del protocolo SSDP, su comportamiento se discutirá en la sección de implementación.

5.6 Contenedores

Todo el despliegue del proyecto se encuentra en un archivo docker-compose.yml. En el encontramos todos los parámetros necesarios para el correcto funcionamiento de nuestros contenedores. Cada

componente mostrado en la figura 5.1 tiene su contendor asociado, salvo la aplicación principal. Esta contará con dos, uno para su ejecución y el otro que hará de servidor web. Salvo el contenedor SSDP los demás se encuentran desplegados en una red personalizada llamada symfony. Esta red contiene un sistema DNS con los nombre e IPS de los contenedores asociados. Con ello no tenemos que preocuparnos si estos cambiasen su IP. El contenedor SSDP se encuentra fuera para poder escanear la red local, ya que si se desplegara en la red docker solo tendría acceso a los contenedores desplegados en ella y no a los dispositivos conectados a la red local.

La descripción del archivo docker-compose.yml se realizará en la sección de implementación.

6

Implementación y pruebas

Introducción

En el capítulo anterior hemos hablado sobre las diferentes componentes que tiene nuestra aplicación, su diseño y ciertos comportamientos que se dan en ellos. En esta sección discutiremos con más detalles como están implementados esos componentes. Intentaremos ser lo más minuciosos posible, pero sin llegar a sobrecargar al lector.

6.1 Implantación en contenedores

Como ya se ha comentado en la parte de diseño en la subsección dedicada a los contenedores todo el despliegue de nuestro proyecto estará contenido en un archivo docker-compose.yml. Este archivo contiene toda la configuración necesaria para el correcto despliegue de nuestro proyecto.

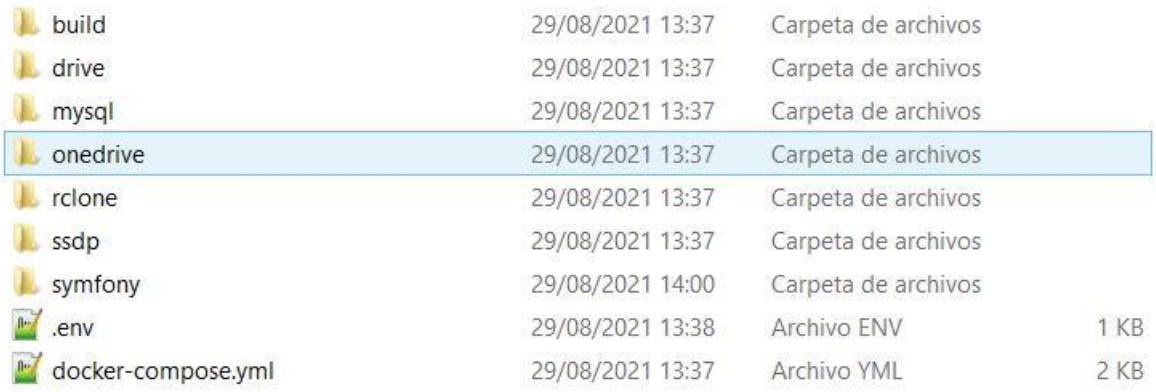

Figura 6.1 Estructura de nuestro proyecto

En la figura 6.1 se muestra aquellas carpetas que componen nuestro proyecto. Todas ellas tienen su relación con el fichero docker-compose.yml.

- **Build:** Contiene la configuración de los dos contenedores encargados de la aplicación web.
- **Drive:** Es un proyecto PHP que nos ayudará a obtener el token de acceso para drive. Este proyecto esta sacado de GitHub [33].
- **Mysql:** Volumen asociado al contenedor de MySQL para no tener que generar manualmente la BD y las tablas cada vez que se despliegue. Toda esa información se encuentra aquí.
- **OneDrive:** Es un proyecto PHP que nos ayudara a obtener el token de acceso para OneDrive. Este proyecto esta sacado de GitHub [34].
- **Rclone:** Esta carpeta contiene el archivo de configuración que Rclone usa para las conexiones. Además, la usaremos como espacio común para que la aplicación web y Rclone puedan acceder a los archivos subidos y bajados.
- **SSDP:** Contiene el proyecto symfony que implementa el protocolo SSDP. Se usará para buscar dispositivos, con capacidad de almacenamiento, en la red en la cual se despliegue.
- **Symfony:** Contiene la aplicación principal del proyecto.
- **env:** Archivo que contiene variables que se usaran en el docker-compose.
- **Docker-compose.yml:** Archivo el cual pone en marcha todo el proyecto.

A continuación, se describe la configuración de cada contenedor en docker-compose:

```
version: '3'
```

Este parámetro indica con que versión de docker se está trabajando. Se empezó trabajando con la versión 3 y al no dar problemas se ha terminado con ella. Relación de la versión del docker-compose y docker se puede encontrar aquí [35].

Services

En esta sección se declaran los contenedores.

Cada contenedor tiene una seria de sentencias que completan su configuración. Se dará una definición general para después explicar los más representativos de cada uno.

- **Db:** En el ejemplo para MySQL indica el nombre para el servicio, aunque luego se combine con el de la carpeta en el que se encuentra docker-compose. Esta sería el nombre por defecto.
- **Image:** Es la imagen elegida para iniciar el contenedor.
- **Build:** Comprende una serie de instrucciones para la creación de un *dockerfile*. **Context**, indica la ruta en la que se encuentra el fichero dockerfile. **Dockerfile**, indica el nombre del fichero *dockerfile*.
- **Command:** Es el comando que se ejecuta al iniciar el contenedor.
- **Container_name:** Nombre que sustituye al por defecto.
- **Volumes:** Mapeamos una dirección interna del contenedor con una externa en nuestro proyecto. Cualquier cambio en el contenido de una de las direcciones provocara el mismo cambio en la otra.
- **Environmment:** Declaración de variables para la configuración del contenedor.
- **Ports:** Asociación de puertos del contenedor con puertos de la máquina en la que se despliega.
- **Network:** La red en la que se encuentra desplegado el contenedor.

Contenedor para MySQL

```
db:
    image: mysql:8.0.20
    command: --default-authentication-plugin=mysql_native_password
    container_name: mysql
    volumes:
      - "db_app:/var/lib/mysql"
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
    ports:
      - 3306:3306
    networks:
      - symfony
```

En el comando indicamos el plugin de autenticación que se empleará al conectarnos con MySQL.

En el volumen indicamos la dirección en la que se encuentra la información de MySQL. Gracias a esto no habrá que generar las tablas en cada despliegue, se importan.

En la parte de las variables se declaran la contraseña de MySQL, el nombre de la BD y un usuario con su contraseña que tendrá todos los privilegios sobre la BD que se ha creado. Estas se encuentran en el fichero env.

Se mapea el puerto por defecto de MySQL con su igual. Deberá de encontrase sin usar en la máquina anfitriona.

Se conecta este contenedor a la red personalizada symfony.

Contenedor php

```
php:
    container_name: php
    build:
      context: .
      dockerfile: build/php/Dockerfile
      args:
        TIMEZONE: ${TIMEZONE}
    volumes:
      - ./symfony/:/var/www/symfony/
      - ./drive/:/var/www/drive/
      - ./onedrive/:/var/www/onedrive/
      - ./rclone/archivos/:/var/www/symfony/public/uploads/
    networks:
      - symfony
```

En *context* indicamos donde buscar el archivo *dockerfile* y en *dockerfile* file indicamos su ubicación.

En *args* nos encontramos otra variable para PHP, también se encuentra en el fichero env.

En los volúmenes nos encontramos: el proyecto de symfony y los proyectos para la obtención de tokens que se están copiando al directorio en el que se almacena nuestra aplicación web. El ultimo relaciona la ubicación donde se suben los archivos en la web con una carpeta en nuestro proyecto. Esto nos permite comunicar los dos contenedores (php y Rclone).

Por último, la red a la que se conecta es symfony.

Contenedor nginx

```
nginx:
    container_name: nginx
    build:
      context: .
      dockerfile: build/nginx/Dockerfile
    volumes:
      - ./symfony/:/var/www/symfony/
    ports:
```

```
      - 8080:80
    networks:
      - symfony
```

En la parte del volumen se copia toda la aplicación web al directorio de ejecución.

Mapeamos el puerto que usa nginx con un puerto libre en nuestra máquina.

Por último, se conecta a la red symfony.

Contenedor Rclone

```
rclone:
    container_name: rclone
    image: rclone/rclone
    command: rcd --rc-serve --rc-addr :5572 --rc-user jose --rc-pass jose
    ports:
      - 5572:5572
    volumes:
      - ./rclone:/config/rclone
      - ./rclone:/logs
      - ./rclone/archivos:/home
    environment:
      - PHP_TZ=Europe/London
      - PUID=1000
      - PGID=1000
    networks:
      - symfony
```

En el apartado de comando se especifica el cual dará inicio al servidor HTTP.

En la parte de los volúmenes indicamos la dirección del archivo de configuración y el de los logs. Además, usamos el directorio home como directorio de trabajo. En el aparecerán los ficheros subidos a la web al estar relacionado con el mismo directorio al que está relacionado la web.

También se encuentra ligado a la red symfony.

Contenedor SSDP

```
symfony_ssdp:
    container_name: ssdp
    image: 'bitnami/symfony:1'
    ports:
      - '8000:8000'
    volumes:
      - './ssdp:/app'
    network_mode: "host"
```

Se usará una imagen de bitnami que ya cuenta con todo lo necesario para funcionar.

Se mapea el puerto 8000 que es el por defecto.

En los volúmenes copiamos nuestro proyecto SSDP al directorio de trabajo.

Se conectará a la red en la que se encuentre nuestra máquina anfitriona.

```
volumes:
  db_app:
```

```
networks:
    symfony:
```

Definimos el volumen usado en el fichero. Sino se encuentran en esta etiqueta serán carpetas creadas por nosotros.

Definición para la red personalizada. Sino se indica lo contrario esta red estará definida en modo *bridge*. Hemos creado esta red para mejorar la comunicación de los contenedores.

6.2 Aplicación symfony

Nuestra aplicación web tiene la siguiente estructura de directorios:

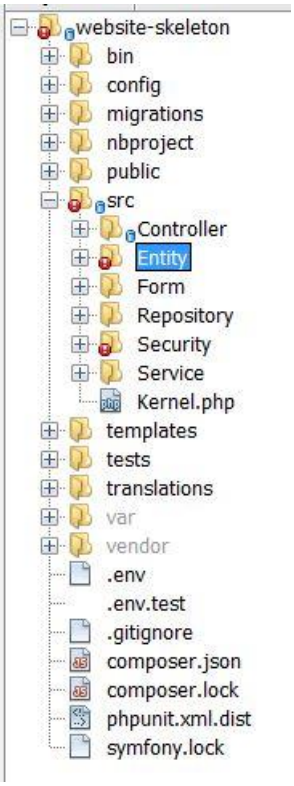

Figura 6.2 Estructura de la aplicación web

- **bin**: Contiene el punto de entrada principal a los comandos que se introducen desde la línea de comandos, con ellos podremos realizar una serie de operaciones que nos facilitarán algunas tareas, como la de crear entidades.
- **config**: Está formado por un conjunto de archivos de configuración que nos permitirán editar el comportamiento de varios componentes.
- **public**: Es el directorio raíz de la web. En el almacenaremos temporalmente los archivos para su subida o descarga. También se almacenarán las políticas de cada usuario.

- **src**: Contiene todas las clases PHP. En el encontramos: los controladores, las entidades, formularios, las clases *repository*, la carpeta Security encargada de varios aspectos del *login* y los servicios.

  Los servicios son clases de apoyo implementadas para aliviar la carga de los controladores. La carpeta *form* contiene formularios construidos en php. Estos pueden ser insertados en las vistas para su visualización.
- **templates**: Aquí guardamos todas las plantillas Twig.
- **test:** Contendrá las clases php que usaremos para evaluar la aplicación.
- **var**: Contiene cachés, logs y archivos generados en tiempo de ejecución por la aplicación.
- **vendor**: Contiene todos los paquetes instalados por Composer, incluyendo el propio Symfony.

El archivo env contiene las variables de entorno usadas por nuestra aplicación.

El archivo composer.json contiene un listado de todos los componentes instalados.

A continuación, se detallará el flujo de vistas que el usuario seguirá en la aplicación. Se explicará que archivos intervienen en su ejecución y la función de cada página en la web.

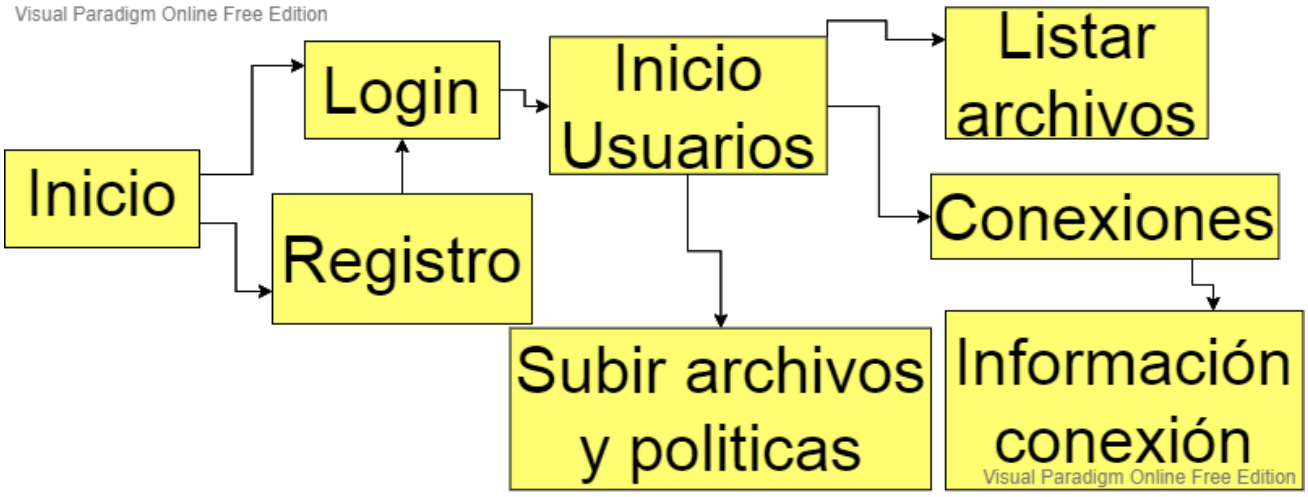

Figura 6.3 Flujo de las vistas en la aplicación web

Inicio

En ella interviene la vista almacenada en el fichero pagina_inicio/index.html.twig y el controlador PaginaInicioController.php. Es la encargada de dar la bienvenida a la aplicación y responde a la URL dirección/, donde dirección es la IP donde se encuentra desplegada la web más el puerto especificado en el contenedor de nginx.

Login

En ella interviene la vista almacenada en el fichero security/login.html.twig y el controlador SecurityController.php. Es la encargada de la comprobación de credenciales y responde a la URL dirección/login, donde dirección es la IP donde se encuentra desplegada la web más el puerto especificado en el contenedor de nginx.

Registro

En ella interviene la vista almacenada en el fichero registration/register.html.twig y el controlador RegistrationController.php. Es la encargada del registro de los usuarios y responde a la URL dirección/register, donde dirección es la IP donde se encuentra desplegada la web más el puerto especificado en el contenedor de nginx.

Inicio Usuarios

En ella interviene la vista almacenada en el fichero inicio.html.twig y el controlador InicioController.php. Es la página que se visita tras el correcto inicio de sesión y muestra las distintas opciones para el usuario. Responde a la URL dirección/inicio, donde dirección es la IP donde se encuentra desplegada la web más el puerto especificado en el contenedor de nginx.

Listar Archivos

En ella interviene la vista almacenada en el fichero lista/inicio.html.twig y el controlador ListaController.php. En ella se muestra el listado de todos los archivos correspondientes a las conexiones establecidas. En el apartado de carpetas nos permite borrarlas o acceder a su contenido y en el apartado de archivos borrarlo o descargarlo. Responde a la URL dirección/ /inicio/lista/{conexion}/{ruta}, donde dirección es la IP donde se encuentra desplegada la web más el puerto especificado en el contenedor de nginx. Conexión y ruta son dos variables que usaremos para saber en qué conexión nos encontramos y la ruta de directorios que hemos seguido.

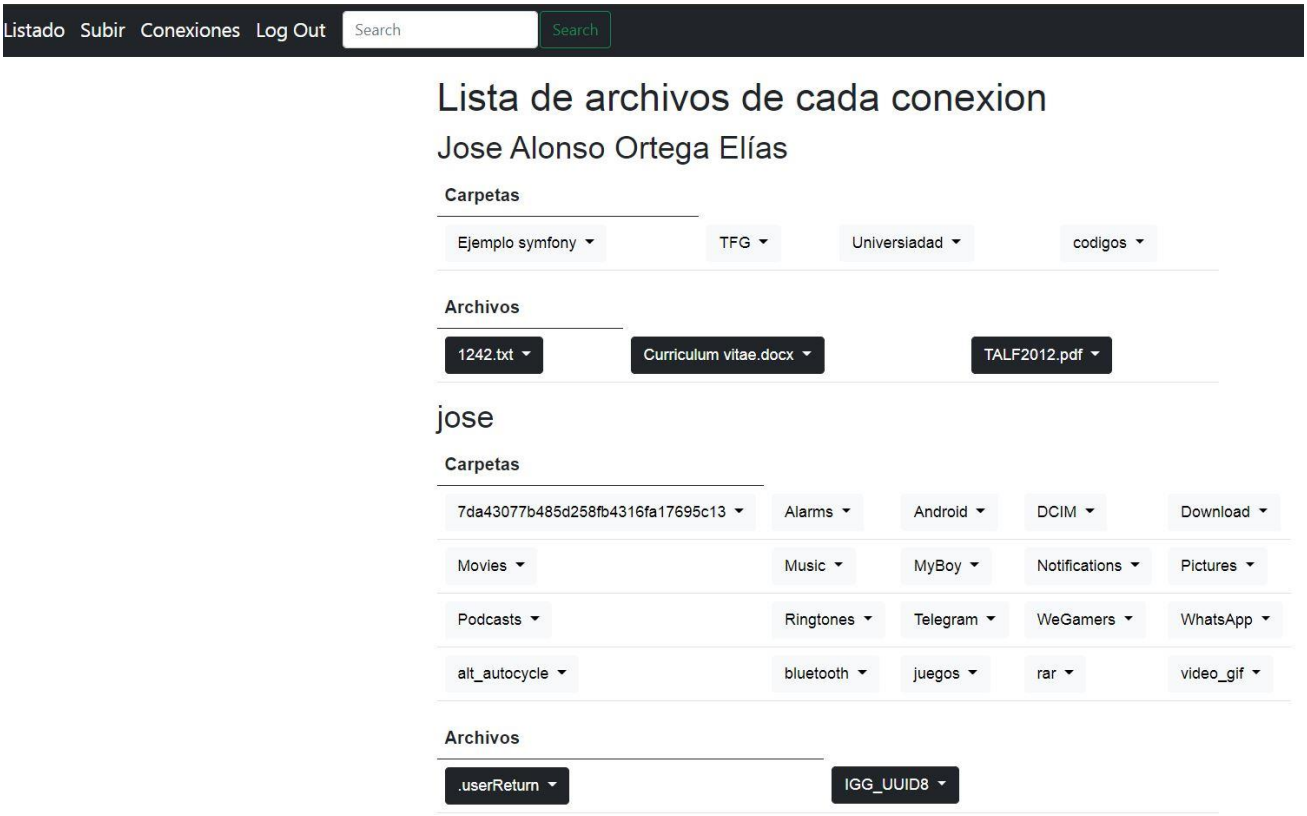

Figura 6.4 Listado de archivos en la aplicación

Conexiones

En ella interviene la vista almacenada en el fichero crear_conexión/index.html.twig y el controlador CrearConexionController.php. Es la encargada de crear las conexiones y visualizar los dispositivos disponibles por medio del escaneo de la red. Responde a la URL dirección/inicio/lista_conexion, donde dirección es la IP donde se encuentra desplegada la web más el puerto especificado en el contenedor de nginx.

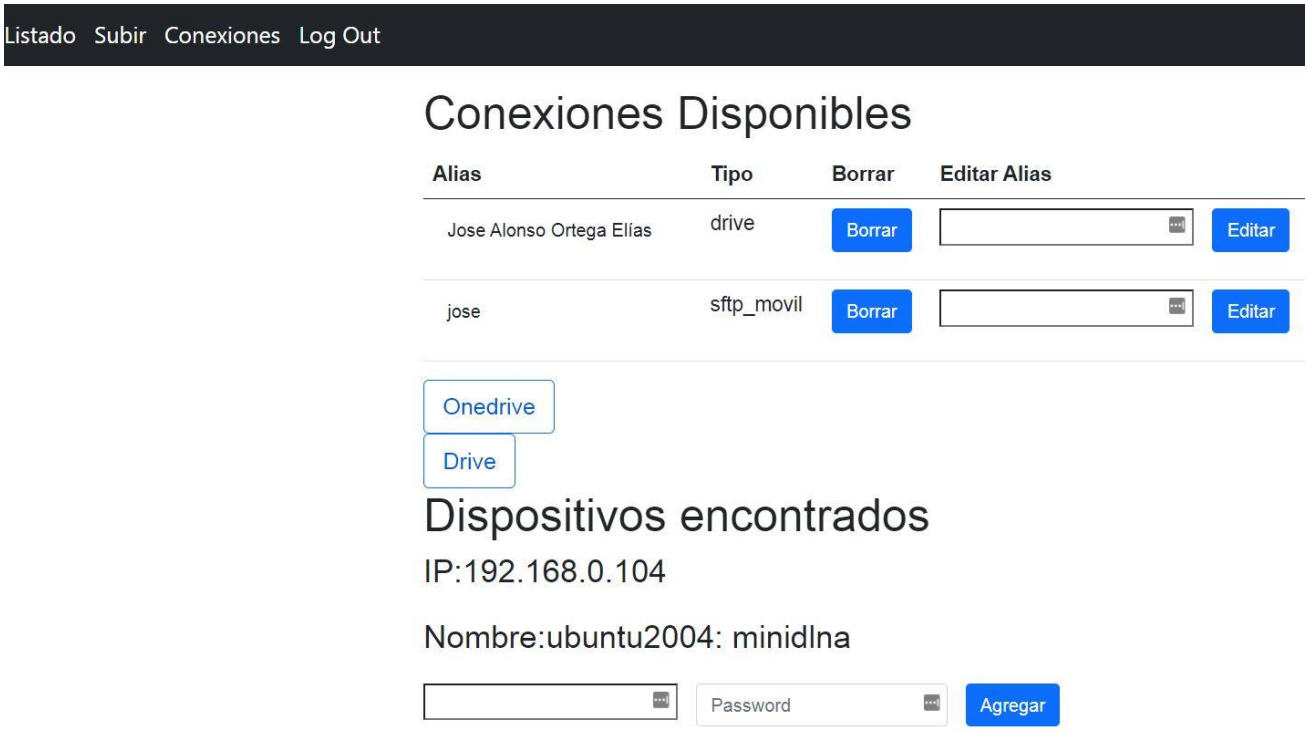

Figura 6.5 Listado de conexiones establecidas y dispositivos disponibles

Información Conexión

En ella interviene la vista almacenada en el fichero estadisticas/index.html.twig y el controlador EstadisticasController.php. Es la encargada de mostrar la división del espacio en la conexión y el historial del usuario. Responde a la URL dirección/inicio/lista_conexion/estadisticas/{conexion}, donde dirección es la IP donde se encuentra desplegada la web más el puerto especificado en el contenedor de nginx. La variable conexión nos indica a quien está asociada estas estadísticas.

Figura 6.6 Uso de memoria e historial de la conexión

Subir archivos y políticas

En ella interviene la vista almacenada en el fichero upload/index.html.twig y el controlador UploadController.php. Es la encargada de subir los archivos y políticas a la web. A continuación, el archivo será enviado con Rclone a la conexión designada y este será borrado de la aplicación. Responde a la URL dirección/inicio/upload, donde dirección es la IP donde se encuentra desplegada la web más el puerto especificado en el contenedor de nginx.

El archivo de políticas esta codificado en formato JSON y está formado por la siguiente estructura.

```
{
  "politicas": {
    "id": {
      "1":
```

Todos los archivos de políticas que vayamos a usar deberán de empezar con la palabra políticas. A continuación, con la palabra id que cumple la función de identificador. Se deberán de colocar los ids de 1 en adelante (el cero está reservado para la política por defecto).

Tras ello se encuentran la definición de cada política. En la aplicación están implementadas 4 tipos. Se comentará el significado de cada parámetro para después profundizar en cada tipo.

- **Nombre:** Pequeña descripción que se mostrara en la web.
- **Tipo:** De las 4 implementadas cual es la elegida.
- **Args:** Argumentos necesarios para el correcto funcionamiento de la política.
- **Destino:** Es el alias de la conexión a la que mandaremos los archivos que cumplan la condición.

La política por defecto es aquella que busca la conexión con más espacio libre disponible.

Tipo Extensión

```
{
        "Nombre":"Las extensiones van...",
        "Tipo":"Extension",
        "Args":"jpg",
        "Destino":"onedrive"
    }
```

Si un archivo contiene la extensión especificada se envía al destino. Como argumento toma aquella extensión que debe de tener el archivo. La extensión se define sin el punto. Solo se puede indicar una.

Tipo Tamaño

```
{
        "Nombre":"Arch mayor que...",
        "Tipo":"Tamaño",
        "Args":"< 50000",
```

```
            "Destino":"jose_drive"
        }
```

Esta política evalúa si el tamaño del archivo cumple con lo especificado. Si la comparación es exitosa el archivo es enviado a destino. Aquí nos encontramos dos argumentos. El primero es la comparación, están definidos <, >, =. El segundo sería el tamaño en bytes que se comparará con el tamaño del archivo.

Tipo Personal

```
{
            "Nombre":"Lo mando a..",
            "Tipo":"Personal",
            "Args":"",
            "Destino":"jose_movil"
        }
```

Esta política es usada para enviar el archivo a un destino sin ningún tipo de condición.

Tipo Patrón

```
{
            "Nombre":"Si el nombre contiene",
            "Tipo":"Patron",
            "Args":"/.*a.*/",
            "Destino":"jose_linux"
        }
```

Esta política comprueba el nombre del fichero respecto a una expresión regular especificada. Como argumento toma una expresión regular entres dos /. Una guía para crear expresiones regulares en php puede encontrase aquí [36].

Cuando una de las políticas no se cumple se recurre a la por defecto.

Un archivo bien construido de políticas se ve así:

```
{
  "politicas": {
    "id": {
      "1": {
        "Nombre":"PDF adrive",
        "Tipo":"Extension",
        "Args":"pdf",
        "Destino":"mi_drive"
      },
      "2": {
        "Nombre":"Mayores que 10000 al ordenador.",
        "Tipo":"Tamaño",
        "Args":"> 10000",
        "Destino":"Linux_oficina"
      },
      "3": {
        "Nombre":"OneDrive personal",
        "Tipo":"Personal",
        "Args":"",
```

```
            "Destino":"jose_onedrive"
        },
        "4": {
            "Nombre":"Termina en _movil",
            "Tipo":"Patron",
            "Args":"/.*movil/",
            "Destino":"móvil_jose"
        }
    }
  }
}
```


Ahora se detallarán las funcionalidades de las clases ubicadas en la carpeta *Service*. Como ya se ha comentado *Service* contiene aquellas clases que ayudan a los controladores a reducir su carga.

La clase que se encarga del manejo de la BD tanto de crear filas como borrarlas es BD.php.
La clase encargada de la obtención del token para drive se llama driveToken.php.
La clase que se encarga de la subida de ficheros a la web, la subida del archivo de políticas y la implementación de las políticas es FileUploader.php.
La case encargada de todas las peticiones al servidor de Rclone y al proyecto SSDP es httpClient.php.
La clase encargada de la obtención del token para OneDrive es onedriveToken.php.
La clase que se encarga del manejo de las políticas es politicas.php.
Los métodos que no encajan en ninguna de las otras clases se implementan en Varios.php.

6.3 API Rclone

Toda la implementación de Rclone se encuentra en docker-compose. Para que Rclone se convierta en un servidor HTTP se debe ejecutar el comando rcd [37]. Este modificará el comportamiento de Rclone para que este a la escucha de las peticiones diseñadas en su API. Por si solo esto no vale y se deberá añadir algunas opciones más. Con la opción --rc-serve se inicia el servidor, este se encuentra ubicado por defecto en la dirección 127.0.0.1:5572. Para utilizar algunas de las operaciones de la API se necesita una autorización. Esto puede anularse con --rc-no-auth, pero nosotros vamos a añadir un usuario y contraseña para agregar seguridad. Con --rc-user y --rc-pass definimos un usuario y contraseña que debemos añadir a todas las peticiones que hagamos al servidor. El último paso será definir la dirección y puerto al cual responde el servidor. Con --rc-addr :5572 indicamos que su dirección es la misma que la máquina donde se ejecuta y asignamos el puerto por defecto que usa Rclone.

6.4 API SSDP

Su función es implementar una API que utiliza el protocolo SSDP para escanear la red en busca de dispositivos disponibles. Se ha desarrollado como un proyecto a parte del principal ya que es necesario su despliegue en una red distinta. La red personalizada *symfony* solo contiene los contenedores que se le han asignado en docker-compose. Por ello se ha desplegado este proyecto en la misma red en la que se encuentra la máquina anfitriona para así poder acceder a todos los dispositivos (que contengan el servicio de almacenamiento) que se encuentren en esa red.

Este proyecto solo contendrá un controlador que llamará a la clase que implementa SSDP y devolverá el resultado de esa llamada (en formato JSON). Solo devolverá aquellos dispositivos que se estén anunciando y contengan el servicio de almacenamiento.

El escaneo por SSDP esta implementado gracias al código de GitHub [38]. Esta clase proporciona una serie de métodos relacionados con SSDP. Se ha elegido aquel que permite filtrar por el tipo de servicio. Tras especificar que busque solo dispositivos con el servicio de almacenamiento, nos devolverá una colección de datos que usaremos para mostrar y configurar conexiones.

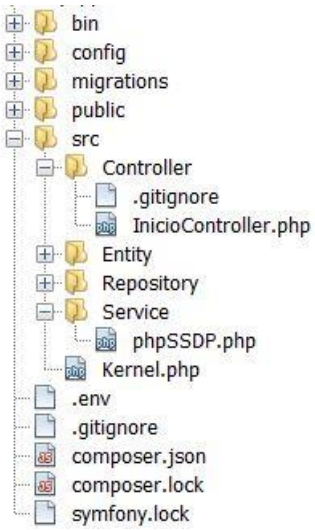

Figura 6.4 Árbol de directorios del proyecto SSDP

6.5 Pruebas

El framework symfony incorpora una librería independiente llamada PHPUnit para realizar test. Cada test se encuentra implementado en una clase PHP ubicada en el directorio /test. Nos encontramos con 3 tipos:

- **Pruebas Unitaria:** Se encargan de comprobar el correcto funcionamiento de los elementos más básicos como clase y métodos.
- **Pruebas de integración:** Se encargar de comprobar si la interacción entre las distintas clases es la correcta.
- **Pruebas de aplicación:** Se encargan de comprobar la aplicación en sí. Realizan peticiones HTTP y validan que la respuesta sea correcta.

Nos centramos en las pruebas de aplicación ya que nos permiten probar una página completa. Se realizarán una serie de pruebas para probar partes importantes de la aplicación. La complejidad y variedad de pruebas que se pueden realizar es inmensa, pero intentaremos ser lo más simples posibles para no recargar al lector [39]. Las pruebas están diseñadas para su fallo ya que si son correctas no se muestra nada en la salida. Al fallar se incluirá información útil sobre el fallo.

Las pruebas son:

InicioUsuarioTest

Con esta prueba se comprueba si se puede acceder a una página que requiere un inicio de sesión previo.

```
1) App\Tests\Controller\IncioUsuarioTest::testSomething
Failed asserting that the Response is successful.
HTTP/1.1 302 Found
Cache-Control: max-age=0, must-revalidate, private
Content-Type:  text/html; charset=UTF-8
Date:          Fri, 24 Sep 2021 10:52:45 GMT
Expires:       Fri, 24 Sep 2021 10:52:45 GMT
Location:      /login
Set-Cookie:    MOCKSESSID=db6a024bbdba7ade8ec9c6f7b987a2f4ec0281f77b580f51e0ba00
3c9fc50291; path=/; secure; httponly; samesite=lax
X-Robots-Tag:  noindex
```

Figura 6.5 Salida para IncioUsuarioTest

En la figura 6.5 se aprecia como al intentar acceder sin haber iniciado sesión se nos redirige a la página de login. La prueba consiste en comprobar si es devuelta la página seleccionada o se nos redirige para obtener permiso para verla.

InicioLogginTest

En esta prueba se comprobará lo contrario a la anterior. Se iniciará sesión con un usuario de prueba y se verificara si se mantiene o cambia de página.

Figura 6.6 Salida de InicioLogginTest

En la figura 6.6 vemos que no nos está redirigiendo para iniciar sesión. Observamos que el código de respuesta de HTTP es 200 indicando que todo ha sido correcto.

NumConexionesTest

Se comprobará que las conexiones que el usuario tiene en la BD son las mismas mostradas en la página asignada para ello. Por lo tanto se consultará el número de conexiones asociadas y se comparará con el número de apariciones de la etiqueta HTML que se usa para su visualización.

Figura 6.6 Salida de NumConexionesTest

En la figura 6.6 se muestra que el usaría tendría 2 conexiones. No coinciden ya que en nuestra página hay otro elemento con la misma etiqueta. Por lo cual al tamaño esperado habría que sumarle uno.

RegistroTest

Aquí comprobaremos la validez de los datos introducidos al registro. Nuestro registro comprueba tres errores: nombres de usuario ya escogidos, contraseña en blanco y contraseña menor de 6 caracteres.

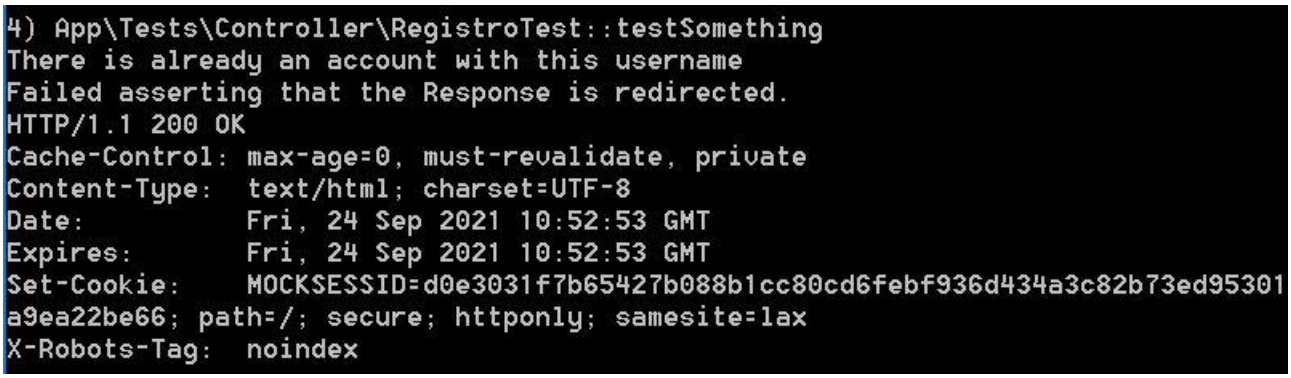

Figura 6.7 Salida RegistroTest

En la figura 6.7 intentamos añadir un usuario con un nombre ya usado. En la primera línea devolvemos el error. Al haber fallado el registro no se nos redirige a la parte de inicio de sesión para ponerlo en marcha, sino que se nos mantiene en la misma página y se nos devuelve un error.

7

Conclusiones y líneas futuras

7.1 Conclusiones

Estoy muy satisfecho con el trabajo final que se ha desarrollado. Aunque en este momento los diferentes tipos de conexiones son limitados, en actualizaciones posteriores se podría llegar a un número considerable. Es ahí donde el sistema de políticas alcanzaría el mayor rendimiento posible.

Con esto en mente y con la evolución actual de la tecnología (cualquier dispositivo ya contiene la capacidad de almacenar información) un proyecto de este estilo resulta muy interesante.

El objetivo principal era, utilizando la tecnología de contenedores, crear un servicio con la facultad de detección de dispositivos con capacidad de almacenamiento y poder hacer uso de estos. Además

de un sistema de políticas que de forma transparente distribuyera los archivos entre las conexiones disponible. Esto se ha cumplido, más el añadido de otras funcionalidades. Sin embargo por falta de tiempo, se han decidido eliminar funcionalidades que, aunque no son parte critica, hubieran añadido más características a esta. Nos referimos al sistema de notificaciones, que se encargaría de emitir alertas como, falta de espacio, desconexiones, nuevos dispositivos en la red etc.

Por la parte personal he sentido un gran avance en mi faceta de ingeniero, ya que nunca había trabajado en un proyecto de estas dimensiones. Aunque symfony era un *framework* que ya conocía, este proyecto me ha permitido profundizar mucho más en él y darme cuenta de su complejidad. En el estudio de la demás tecnología, si bien conocía algunas, el proyecto me ha permitido alcanzar un mayor grado de entendimiento sobre ellas y en aquellas que no conocía asentar una buena base para seguir su estudio.

7.2 Líneas futuras

Aunque este proyecto cumple con casi todos los requisitos iniciales, podría mejorar con las siguientes características:

- **Seguridad:** Durante el desarrollo del proyecto la seguridad no ha sido un objetivo prioritario, por lo cual apenas se ha realizado un esfuerzo en ella. Por lo tanto, añadir esta capacidad al proyecto sería muy útil. Por ejemplo, evitar las inyecciones SQL o reducir la información que aparece en la URL.
- **Nombre de dominio:** El acceso a la aplicación web se realiza a través de la IP de la máquina en la que se ejecuta. Una mejora estaría encaminada a asignar al proyecto un nombre de dominio para que no tuviese que depender de esa IP.
- **Sistema de notificaciones:** No ha sido posible implementar un sistema de notificaciones. Tales como alertas de espacio disponible, dispositivos que aparecen o desaparecen de la red etc. Esto sería una característica muy interesante para añadir.
- **Mejoras visuales:** El aspecto visual responde a las necesidades mínimas del proyecto. Se podría hacer un mayor esfuerzo en la parte visual. Por ejemplo, añadiendo plantillas que mejores la visualización de los datos.

Referencias

[1] I. Sommerville, Ingeniería de software, Pearson Educación, 2011.

[2]Fabien Potencier. François Zaninotto., Definitive guide to symfony, The. Apress, 2007.

[3]"Installing & Setting up the Symfony Framework (Symfony Docs)", Symfony.com, 2021.[Online]. Available: https://symfony.com/doc/current/setup.html#technical-requirements. [Accessed: 01- Jul- 2021].

[4]N. Craig-Wood, "Rclone", Rclone.org. [Online]. Available: https://rclone.org/#what. [Accessed: 01- Jul- 2021].

[5] Kayas, G., Hossain, M., Payton, J. and Islam, S., 2021. SUPnP: Secure Access and Service Registration for UPnP-Enabled Internet of Things. IEEE Internet of Things Journal, 8(14), pp.11561-11580.

[6] Datatracker.ietf.org. 2012. rfc6749. [online] Available at: <https://datatracker.ietf.org/doc/html/rfc6749> [Accessed 24 August 2021].

[7]L. López Magaña, "Qué es OAuth 2", OpenWebinars.net, 2020. [Online]. Available: https://openwebinars.net/blog/que-es-oauth2. [Accessed: 02- Jul- 2021].

[8] https://www.cs.umd.edu/sites/default/files/scholarly_papers/XuWang.pdf

[9]K. JANGLA, ACCELERATING DEVELOPMENT VELOCITY USING DOCKER. [Place of publication not identified]: APRESS, 2019.

[10] Wong, C., 2000. HTTP pocket reference. 1st ed. Sebastopol, CA: O'Reilly.

[11]"¿Qué es Twig? | Documentación para Diseñadores", Docs.tiendanube.com. [Online]. Available: https://docs.tiendanube.com/help/qu-es-twig. [Accessed: 08- Jul- 2021].

[12] J. Alarcón, "¿Qué diferencia hay entre Docker (Contenedores) y Máquinas virtuales (VMWare, VirtualBox...)? - campusMVP.es", campusMVP.es. [Online]. Available: https://www.campusmvp.es/recursos/post/que-diferencia-hay-entre-docker-contenedores-y-maquinas-virtuales.aspx. [Accessed: 26- Aug- 2021]

[13]"Docker overview", Docker Documentation. [Online]. Available: https://docs.docker.com/get-started/overview. [Accessed: 12- Jul- 2021].

[14]G. Torres, "Cómo funcionan las redes en Docker | return(GiS);", return(GiS);, 2020. [Online]. Available: https://www.returngis.net/2020/12/como-funcionan-las-redes-en-docker. [Accessed: 13- Jul- 2021].

[15]Chin-Feng Lai, Yueh-Min Huang and Han-Chieh Chao, 2010. DLNA-Based Multimedia Sharing System for OSGI Framework With Extension to P2P Network. IEEE Systems Journal, 4(2), pp.262-270.

[16]"Routing (Symfony Docs)", Symfony.com. [Online]. Available: https://symfony.com/doc/current/routing.html. [Accessed: 16- Jul- 2021].

[17]N. Craig-Wood, "Remote Control / API", Rclone.org. [Online]. Available: https://rclone.org/rc. [Accessed: 16- Jul- 2021].

[18]Javier Martín, biupBOX: el agregador de nubes, https://loogic.com/biupbox-el-agregador-de-nubes, 2015

[19]Espinosa, O., 2021. diagrama de flujo para oauth. [image] Available at: <https://www.redeszone.net/tutoriales/seguridad/que-es-oauth> [Accessed 25 August 2021].

[20] Developer.mozilla.org. n.d. Códigos de estado de respuesta HTTP - HTTP | MDN. [online] Available at: <https://developer.mozilla.org/es/docs/Web/HTTP/Status> [Accessed 26 August 2021].

[21]programador clic, Docker vs Máquina virtual. [image]. Available: https://programmerclick.com/article/71931412641. [Accessed: 26- Aug- 2021]

[22]R. Unzue Pulido, Red docker en modo bridge. 2017 [Online]. Available: https://www.maquinasvirtuales.eu/configuracion-red-networking-dockers. [Accessed: 26- Aug- 2021]

[23]"Databases and the Doctrine ORM (Symfony Docs)", Symfony.com. [Online]. Available: https://symfony.com/doc/current/doctrine.html. [Accessed: 27- Aug- 2021].

[24]R. NIXON, LEARNING PHP, MYSQL & JAVASCRIPT. [S.l.]: O'REILLY MEDIA, 2021.

[25]S. Guthals and P. Haack, Github, 1st ed. 2019.

[26]"Twig for Template Designers - Documentation - Twig - The flexible, fast, and secure PHP template engine", Twig.symfony.com. [Online]. Available: https://twig.symfony.com/doc/3.x/templates.html. [Accessed: 07- Sep- 2021].

[27]"DLNAServer", Play.google.com. [Online]. Available: https://play.google.com/store/apps/details?id=com.al.dlnaserver&hl=es&gl=US. [Accessed: 07- Sep- 2021].

[28]"Servidor Ssh", Play.google.com. [Online]. Available: https://play.google.com/store/apps/details?id=com.theolivetree.sshserver. [Accessed: 07- Sep- 2021].

[29]"ReadyMedia, un servidor DLNA ligero y muy sencillo de configurar", RedesZone, 2017. [Online]. Available: https://www.redeszone.net/2017/01/07/readymedia-servidor-dlna-ligero-sencillo-configurar. [Accessed: 07- Sep- 2021].

[30]N. Craig-Wood, "Microsoft OneDrive", Rclone.org. [Online]. Available: https://rclone.org/onedrive/#getting-your-own-client-id-and-key. [Accessed: 08- Sep- 2021].

[31]N. Craig-Wood, "Google drive", Rclone.org. [Online]. Available: https://rclone.org/drive/#making-your-own-client-id. [Accessed: 08- Sep- 2021].

[32]"Authorization for OneDrive API for Microsoft Accounts - OneDrive dev center", Docs.microsoft.com. [Online]. Available: https://docs.microsoft.com/en-us/onedrive/developer/rest-api/getting-started/msa-oauth?view=odsp-graph-online#code-flow. [Accessed: 08- Sep- 2021].

[33]"GitHub - thephpleague/oauth2-google: Google Provider for the OAuth 2.0 Client", GitHub. [Online]. Available: https://github.com/thephpleague/oauth2-google. [Accessed: 09- Sep- 2021].

[34]"GitHub - TheNetworg/oauth2-azure: Azure AD provider for the OAuth 2.0 Client.", GitHub, 2021. [Online]. Available: https://github.com/TheNetworg/oauth2-azure. [Accessed: 09- Sep- 2021].

[35]"Compose file", Docker Documentation. [Online]. Available: https://docs.docker.com/compose/compose-file. [Accessed: 09- Sep- 2021].

[36]D. Lázaro, "Expresiones regulares en PHP", Diego.com.es. [Online]. Available: https://diego.com.es/expresiones-regulares-en-php. [Accessed: 10- Sep- 2021].

[37]N. Craig-Wood, "rclone rcd", Rclone.org. [Online]. Available: https://rclone.org/commands/rclone_rcd. [Accessed: 10- Sep- 2021].

[38]"GitHub - liqueurdetoile/phpSSDP: Utility PHP class to search for devices on a local network through UPnP SSDP Discovery.", GitHub. [Online]. Available: https://github.com/liqueurdetoile/phpSSDP. [Accessed: 10- Sep- 2021].

[39]"Testing (Symfony Docs)", Symfony.com. [Online]. Available: https://symfony.com/doc/current/testing.html. [Accessed: 22- Sep- 2021].

Apéndice A

Manual de Instalación

Requerimientos

El proyecto esta diseñado para ejecutarse en una distribucion de linux. Tendremos que tener instalado docker en nuestra máquina. Los puertos 8080, 3306, 5572 y 8000 deberán de estar disponibles.

Instalación

Tras la descarga del proyecto deberemos de situarnos en la raiz de este (como en la figura 6.1). Ahora ejecutaremos el comando que levanta nuestros contenedores a partir del docker-compose.yml.

docker-compose up -d

Y con esto ya estaria nuestro proyecto desplegado, que se encuentra en la IP:8080 de la máquina en la que se esta ejecutando.

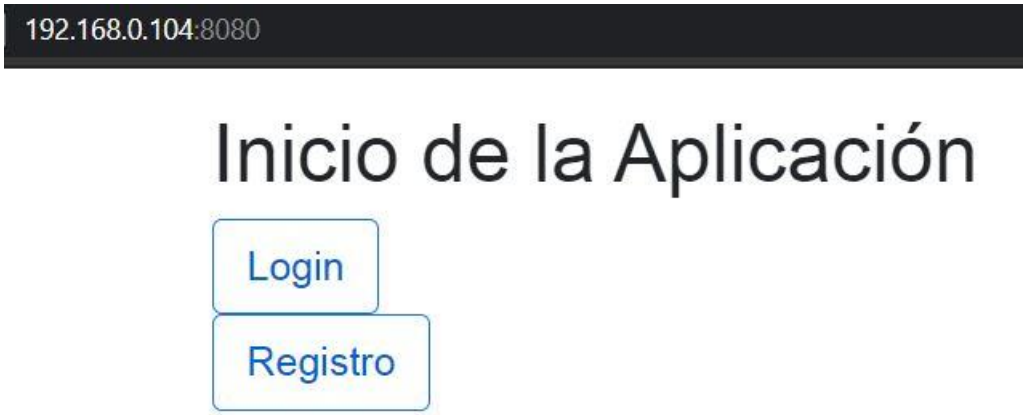

Figura 7.1 Despliegue de la aplicación en la ip de la máquina anfitriona

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga