Universidad Politécnica de Madrid

Escuela Técnica Superior de

Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Plataforma Web para Intercambio Seguro de Archivos

Autor: Álvaro Rebato Ramírez

Tutor(a): Julio Mariño Carballo

Madrid, junio 2021

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Plataforma Web para Intercambio Seguro de Archivos
Junio, 2021

Autor: Álvaro Rebato Ramírez

Tutor:
Julio Mariño Carballo
Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Es común hoy en día tener una cuenta en Google Drive, ICloud u otra gran empresa que, por una suscripción gratuita o por un módico precio al mes ceden o alquilan un espacio de almacenamiento en sus centros de datos permitiendo a la gente no tener que ocupar sus discos duros con sus datos. Esta práctica, aparte de permitir a los usuarios disponer de mayor espacio en su máquina para otros programas, es de gran utilidad para disponer de esos archivos en cualquier momento en cualquier lugar.

Este proyecto se fija en estas tecnologías para ser una aproximación académica de una plataforma de almacenamiento e intercambio de archivos para usuarios de la UPM que además identifique y muestre al usuario si el archivo que le han compartido pudiera suponer una amenaza en su dispositivo.

La plataforma contiene un sistema de gestión de usuarios gracias al cual, partir de un email de la Universidad Politécnica de Madrid, segrega los usuarios según su extensión de correo, otorgando a los personales docentes la potestad de administrar más opciones que los alumnos.

Los usuarios interactuarán con esta plataforma a través de una interfaz de usuario limpia, intuitiva, compacta y muy sencilla que, entre otras cosas, proporciona *feedback* a los usuarios mediante animaciones y eventos desarrollados teniendo en cuenta las tendencias actuales en el diseño web.

Abstract

It is common nowadays to have an account on Google Drive, ICloud or other large companies that, for a free subscription or for a small monthly fee, give or rent storage space in their data centres allowing people not to have to occupy their hard drives with their data. This practice, apart from allowing users to have more space on their machine for other programmes, is very useful for making those files available at anytime, anywhere.

This project looks at these technologies to be an academic approach to a file storage and sharing platform for UPM users that also identifies and shows the user if the file that has been shared with them could pose a threat to their device.

The platform contains a user management system based on registering with an email from the Universidad Politécnica de Madrid. Users are segregated according to their email extension, giving teaching staff the power to manage more options than students.

Users will interact with this platform through a clean, intuitive, compact and very simple user interface that, among other things, provides feedback to users through animations and events designed taking into account current trends in web design.

Tabla de contenidos

Tabla de figuras

1 Introducción

La velocidad a la que se comparten y descargan archivos hoy en día abre la puerta a una infinidad de ficheros maliciosos que pueden poner en peligro los datos de nuestros ordenadores dejándolos a la merced de ciberdelincuentes que pueden pedir un rescate por ellos, someter a chantaje al propietario e incluso venderlos.
Lo que se propone con este trabajo es una aproximación académica de un servicio de almacenamiento en la nube para usuarios de la UPM en el que los usuarios podrán almacenar y compartir con otros integrantes de la plataforma sus archivos de forma segura. Cada recurso subido a la plataforma será analizado individualmente y, en caso de ser malicioso, los usuarios serán alertados apropiadamente para evitar daños derivados de la descarga y uso de archivos maliciosos en sus dispositivos.

1.1 Objetivos

- Creación de una plataforma de almacenamiento e intercambio seguro portable y fácilmente mantenible que pueda desplegarse en cualquier servidor.
- Los usuarios que usarán esta plataforma están comprendidos por alumnos y profesores de la Universidad Politécnica de Madrid.
- Todos los archivos subidos a esta plataforma deben ser analizados por un analizador de programas maliciosos.
- Los archivos se podrán compartir con cualquier usuario registrado en la plataforma.
- Los usuarios deben conocer el veredicto del análisis para evitar ejecuciones accidentales de programas maliciosos tras su descarga.
- La interfaz de los usuarios debe ser limpia e intuitiva ajustándose a los diseños actuales de interfaces.

Para este proyecto se llevarán a cabo diferentes tareas derivadas del diseño e implementación del sistema descrito previamente como son el diseño de un sistema, despliegue de varias tecnologías que mejor se ajusten al problema, creación de bases de datos, diseño de interfaces web, programación web en *front-end*, programación de servicios en *back-end*.

1.2 Solución propuesta

Tras analizar el proyecto a realizar se ha llegado a la conclusión que la plataforma puede estar formada por diferentes componentes que cumplan una tarea concreta y específica. Estos componentes pueden separarse en una interfaz de usuario, un sistema de almacenamiento, una base de datos y un analizador de archivos. Tras definir estos módulos se ha buscado una tecnología que permita de una forma sencilla encapsular estas funcionalidades, agilizar su despliegue y facilitar la programación y el mantenimiento del funcionamiento de cada componente. Para este objetivo se opta por la tecnología de los contenedores ya que son entornos especialmente diseñados para la creación de aplicaciones independientes y contenidas en cualquier entorno.

Se ha decidido hacer uso de los contenedores de *Docker* para esta solución. *Docker* permite la creación de contenedores en sistemas *Linux* y *Windows* que funcionan como máquinas virtuales modulares las cuales son fácilmente manipulables. El usuario de esta tecnología puede crear cada contenedor a medida, moverlo de maquina y desplegarlo allí donde necesite.

De este modo se ha optado por la siguiente estructura:

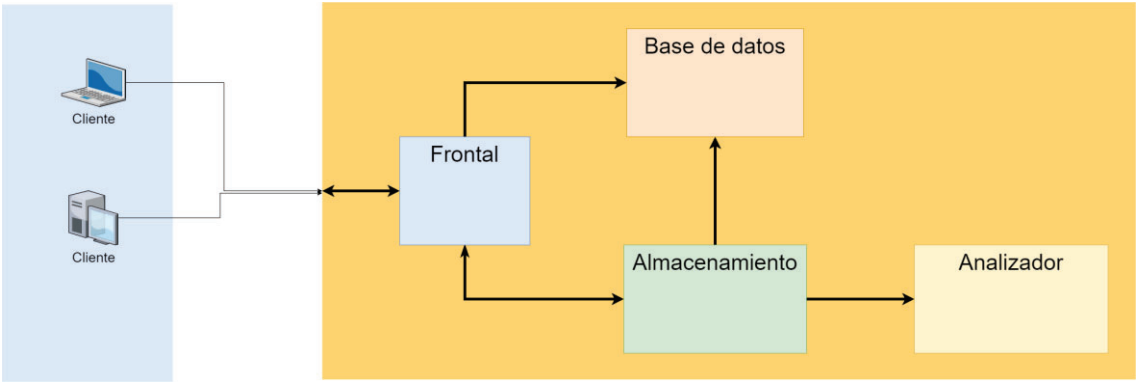

Ilustración 1: esquema de arquitectura

Para el desarrollo de este proyecto se ha decidido implementar los siguientes contenedores para conseguir los objetivos propuestos:

- **Frontal**: este contenedor actuará como interfaz entre el usuario y la plataforma. Basado en un *framework* que facilita el desarrollo web con un patrón MVC (Modelo Vista Controlador) que permita un desarrollo para la comunicación cliente-servidor ágil. Es el que va a servir a los usuarios hacer todas las interacciones con la plataforma.
- **Base de datos**: este contenedor servirá de base de datos relacional para consultas desde otros contenedores y persistencia de datos.
- **Almacenamiento**: en este contenedor se almacenarán los archivos además de ordenar el análisis de los archivos.
- **Analizador**: será el contenedor que realice los análisis de los archivos que los usuarios suban a la plataforma. Estos análisis deben ser rápidos y eficaces para garantizar la usabilidad de la plataforma.

2 Trabajos previos

El almacenamiento en la nube está basado en la computación en la nube, la cual cuenta con una larga carrera. La idea de computación en la nube comenzó a desarrollarse en los años 60 como una "red intergaláctica de ordenadores" mediante el Proyecto MAC, desarrollado por el MIT. El objetivo de este proyecto era hacer posible el uso simultáneo de un por al menos dos individuos. Entre finales de los 60 y principios de los 70 se construyó ARPANET (*Advanced Research Projects Agency Network*) con el objetivo de crear una red de ordenadores conectados desde diferentes puntos de Los Estados Unidos que sirviera como medio de comunicación entre instituciones académicas y estatales [1] [2].

2.1 Almacenamiento en la nube

La computación en la nube propició la creación de tecnologías como las Máquinas Virtuales, iniciadas por *IBM* y las Redes Virtuales Privadas (VPN) así como los servicios de almacenamiento en la nube.

Estas tecnologías fueron desarrollándose hasta nuestros días donde tenemos sistemas de almacenamientos distribuidos y para uso personal. Existen muchos proveedores de almacenamiento en la nube, aunque los más importantes son [3]:

- **Dropbox**: creado en 2007 por dos alumnos del *MIT*. Se trata de un servicio para almacenar y compartir archivos en la nube con sincronización en todos los dispositivos del usuario. Con el tiempo han incorporado herramientas que ayudan a la productividad de los clientes como integraciones con *Slack*, *Zoom*, *Adobe*, etc. [4]
- **Google Drive**: se crea en el año 2012 complementando a su predecesor Google Docs que integra todas las herramientas de Google como *Gmail*, *Calendar*, *Google Talk*, entre otras, así como procesadores de textos, procesadores de código e incluso integración con *Google Colab* [5].
- **OneDrive**: Microsoft lanzó esta plataforma en 2014 como servicio de almacenamiento y copia de seguridad para los archivos y carpetas de los usuarios. Permite elegir varias carpetas del ordenador para realizar copia de seguridad periódica [6].
- **ICloud**: este servicio de Apple lleva alojando contenido de sus usuarios desde 2012. Soportando la sincronización automática entre todos los dispositivos de la marca Apple de todo tipo de archivos como documentos, fotos o videos además de fotos y música [7].

2.2 Análisis de malware

Se define como *malware* cualquier programa que ejecuta acciones perjudiciales para un dispositivo, servidor o red.

Este tipo de programas son utilizados por ciberdelincuentes con el fin de obtener rédito ya bien sea económico, información o chantaje entre otros. Tanto empresas como usuarios privados pueden ser objetivo de un ataque el cual puede ser dirigido o aleatorio [8].

Estos programas en función de su actividad se pueden categorizar en varios tipos, como, por ejemplo:

- **Criptomineros**: son utilizados para el minado encubierto de criptodivisas en el ordenador de la víctima.
- **Spyware**: son utilizados para obtener información de los archivos de la víctima además de monitorizar procesos y conexiones para obtener contraseñas o información confidencial.
- **Troyano**: se infiltra en el ordenador de la víctima y permanece oculto para realizar otras funciones maliciosas como el espionaje.
- **Ransomware**: este tipo de software encripta los archivos de la víctima para luego pedir un rescate monetario por la recuperación de los mismos.
- **Gusano**: estos programas buscan replicar, propagar e infectar otros ordenadores de la misma red en la que se encuentran.
- **Botnets**: son clústeres de ordenadores infectados con un *malware* que pueden ser utilizados de forma remota para realizar otros ataques.

Frente a estas amenazas existen principalmente dos tipos de análisis para detección de *malware* [9]:

- **Análisis estático:** es un tipo de análisis basado en técnicas heurísticas, es decir, técnicas de comportamiento que analizan el archivo sin detonar su ejecución. Este tipo de análisis principalmente procede a desensamblar el archivo y analizar su código en busca de alguna información o patrón que identifique la amenaza.
- **Análisis dinámico:** el análisis dinámico realiza una monitorización completa del comportamiento de un programa detonando este fichero en un entorno controlado como máquinas virtuales. Estos tipos de análisis son más efectivos que los estáticos ya que incluso sirven para detectar nuevos malwares aun no detectados. Por contraparte, el tiempo y coste de análisis es muy superior.

3 Requisitos

3.1 Requisitos funcionales

RF01: un usuario podrá registrarse en la plataforma por medio de su correo de la UPM.

RF02: los usuarios con correo docente tendrán el rol de administrador.

RF03: un usuario registrado podrá iniciar sesión en la plataforma.

RF04: el usuario podrá visualizar todos los archivos que han sido subidos por él, así como los que han sido compartidos con él.

RF05: el usuario podrá subir archivos, clasificarlos y compartirlos con otros usuarios de la plataforma.

RF06: los usuarios tendrán un límite de cuota.

RF07: los administradores podrán modificar la cuota de los usuarios.

RF08: los administradores podrán eliminar usuarios.

RF09: el sistema actualizará la cuota del usuario cuando realice una subida de un archivo.

RF10: el sistema notificará a los usuarios con los que se comparta un archivo.

RF11: los usuarios podrán descargar los archivos de su propiedad o compartido con ellos.

RF12: los usuarios podrán eliminar los archivos de su propiedad.

RF13: los administradores podrán eliminar los archivos que hayan sido compartidos con ellos.

RF14: el sistema almacenará los archivos de los usuarios.

RF15: el sistema realizará un análisis estático de malware a los archivos subidos.

RF16: los usuarios podrán realizar diferentes búsquedas sobre sus documentos.

RF17: los usuarios podrán visualizar el veredicto de sus archivos y los compartidos con ellos.

RF18: el sistema deberá ofrecer al usuario información de cuota consumida y cuota total.

3.2 Requisitos no funcionales

RNF01: solo los administradores podrán gestionar a los usuarios.

RNF02: la interfaz deberá tener un diseño amable con el usuario.

RNF03: la plataforma deberá estar operativa el 99% del año.

RNF04: se ha de asegurar la confidencialidad de los datos.

RNF05: se guardará la sesión del usuario durante 1h.

4 Tecnologías utilizadas

Para la construcción y programación de esta plataforma se ha hecho uso de diferentes tecnologías las cuales se han aplicado para funciones específicas del sistema. Estas tecnologías han sido seleccionadas en función de los requisitos de la plataforma en cada una de sus partes.

4.1 Ubuntu

Ubuntu es un sistema operativo basado en *Linux* de código libre. Se ha hecho uso de esta distribución porque se trata de un sistema ligero con un consumo de recursos bajo y con un amplio catálogo de software hecho especialmente para este sistema. Además, al tratarse de una distribución *open source* puede obtenerse de forma gratuita.

La versión utilizada para este proyecto se trata de la *18.04 Bionic Beaver* en su versión con soporte a largo plazo (LTS) [10].

Ilustración 2: logo de Ubuntu

4.2 Docker

Docker es una tecnología de despliegue de contenedores de software estandarizados que permite a los desarrolladores e ingenieros de software la construcción rápida, eficiente y simple de aplicaciones capaces de desplegarse en cualquier sistema operativo en el que se pueda instalar *Docker* [11].

Esta tecnología ha sido la elegida para contener los diferentes módulos de la plataforma por ser una de las más famosas y extendidas para la creación de contenedores, es fácilmente instalable en un sistema *Linux* y permite su manejo mediante línea de comandos. Asimismo, cuenta con una documentación muy detallada que facilita mucho su uso y con una comunidad muy amplia para consulta de dudas.

Ilustración 3: logo de Docker

4.3 Python

Python es un lenguaje de programación orientado a objetos con semántica dinámica creado a finales de los años 80 [12].

Se elige este lenguaje porque es uno de los lenguajes más utilizados por los programadores por su versatilidad y cuenta con una de las comunidades más extensas del mundo tecnológico. Python cuenta además con multitud de documentación y guías que hacen del desarrollo de aplicaciones una tarea más sencilla.

Ilustración 4: logo de Python

4.4 Django

Se ha utilizado Django para el contenedor frontal porque es un *framework* de código libre para el desarrollo de sitios web seguros y mantenibles. Ofrece una gran cantidad de herramientas para facilitar el desarrollo como bases de datos embebidas, motores de renderizado de plantillas, protocolos de seguridad, etc. Su arquitectura permite la escalabilidad de los proyectos web en diferentes aplicaciones independientes las unas de las otras.

Al ser un *framework* enfocado en *Python* hace que sea completamente portable y pueda ejecutarse tanto en sistemas *Linux* como en sistemas *Windows* [13].

Ilustración 5: logo de Django

4.5 Flask

Se ha utilizado *Flask* en el contenedor de almacenamiento porque es un *microframework* de código libre muy ligero capaz de desplegar aplicaciones web o aplicaciones *REST* [14] que facilita desarrollos muy ágiles. Fue creado basado en *Python* y con el motor de renderizado de plantillas *Jinja2*. Al no depender de librerías externas deja en manos del programador todas las herramientas a utilizar e implementar. Esta carencia permite la compatibilidad con las últimas tecnologías, así como una base de código más ligera haciendo que su ejecución sea más rápida [15].

Ilustración 6: logo de Flask

4.6 MariaDB

MariaDB es una base de datos relacional basada en *MySQL* de código libre. Se ha escogido esta base de datos porque se caracteriza por su alta velocidad de consulta e indexado de datos. Permite la manipulación de grandes conjuntos de datos con bastante optimización, es capaz de almacenar llamadas a procedimientos SQL [16].

Ilustración 7: logo de MariaDB

4.7 PHPMyadmin

Phpmyadmin es una herramienta de software para la administración y manejo de bases de datos relacionales *MySQL* y derivados, como en este caso, *MariaDB*.

Se elige esta herramienta porque ofrece una interfaz web para realizar operaciones sobre una base de datos de forma intuitiva y automática. Permite crear, eliminar, manipular y relacionar unas tablas con otras desde su interfaz gráfica, así como una línea de comandos para realizar consultas. Adicionalmente ofrece la creación de procedimientos SQL con parámetros, característica con la que crear consultas complejas o secuencias de consultas, almacenarlas y ejecutarlas [17].

Ilustración 8: logo de phpMyAdmin

4.8 HTML

HTML es el estándar para Lenguaje de Marcas de Hipertexto, este lenguaje es en el que se va a programar la interfaz de usuario ya que se trata del lenguaje más común en el desarrollo de páginas web, más concretamente para la creación y diseño gráfico del contenido de la página. Define y ordena el contenido que el navegador tiene que renderizar en función de una estructura definida por etiquetas. Este lenguaje permite definir estilos de las etiquetas, contenidos y funcionalidades [18].

Ilustración 9: logo de HTML

4.9 Bootstrap

Bootstrap es un *framework* de *front-end* basado en *CSS* (Hojas de Estilo en Cascada) para la creación y personalización de interfaces de usuario que añade diferentes tipografías, plantillas, formularios, botones, etc. Para simplificar el desarrollo de páginas web y dotar de diferentes estilos y personalización a la plantilla de *html* [19].

Ilustración 10: logo de Bootstrap

4.10 JavaScript

JavaScript es un lenguaje de *scripting* enfocado al desarrollo web que permite la interacción, manipulación y validación de datos en plantillas *html*. Se ejige este lenguaje porque puede ser interpretado por todos los navegadores del mercado y se ejecuta en la parte del cliente, permitiendo realizar acciones de forma dinámica sin necesidad de volver a renderizar una página web. Además, al ser un fragmento que se ejecuta en el lado del cliente esto hace que todo el computo ocurra en el ordenador del usuario, liberando de carga al servidor [20].

Ilustración 11: logo de JavaScript

4.11 JQuery

JQuery es una librería del lenguaje de programación *JavaScript* que ofrece herramientas para manipular elementos de *DOM* (Modelo de Objetos del Documento), la renderización del documento *html* por parte del navegador. Se utiliza JQery porque puede acceder directamente al *DOM* y modificar dinámicamente cualquier elemento que conforme la interfaz web. Puede ser utilizado en scripts de *JavaScript*. Adicionalmente, contiene *AJAX* que permite hacer peticiones asíncronas al servidor y modificar partes del contenido web sin necesidad de refrescar toda la página [21].

Ilustración 12: logo de jQuery

4.12 Assemblyline

Assemblyline es una herramienta de detección y análisis de *malware* desarrollado por el Centro Canadiense para la ciberseguridad (*Canadian Centre for Cyber Security*) [22].

Este componente se ha elegido porque realiza un análisis sobre cualquier archivo mediante un proceso rápido aplicando una serie de técnicas de detección de malware mediante análisis estático para detectar secciones que puedan llegar a suponer una amenaza para un usuario en caso de ejecución.

Su integración es mediante *Docker*, por lo que es fácilmente desplegable en esta plataforma junto a los demás contenedores.

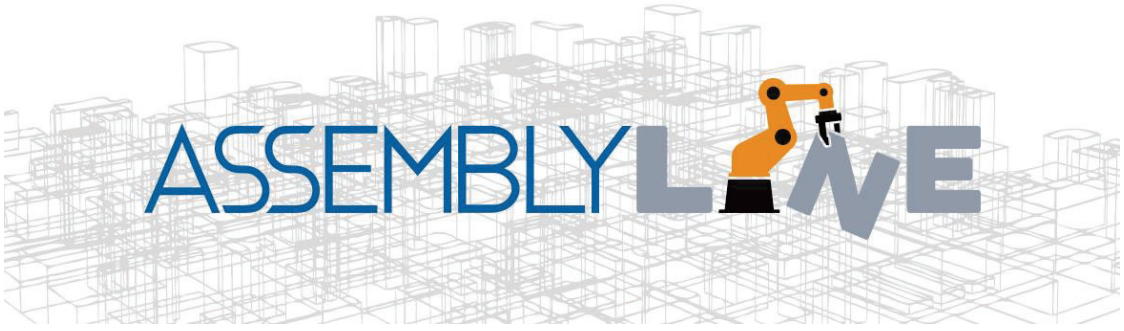

Ilustración 13: logo de Assemblyline

5 Desarrollo

En este capítulo se va a describir el proceso que se ha llevado en el desarrollo de este proyecto hasta su versión final. Este apartado se va a separar por cada bloque que compone este sistema y se va a profundizar individualmente en cada componente. Como se muestra en el siguiente diagrama, consta de el contenedor frontal, el contenedor de base de datos, el contenedor de almacenamiento y el contenedor de análisis.

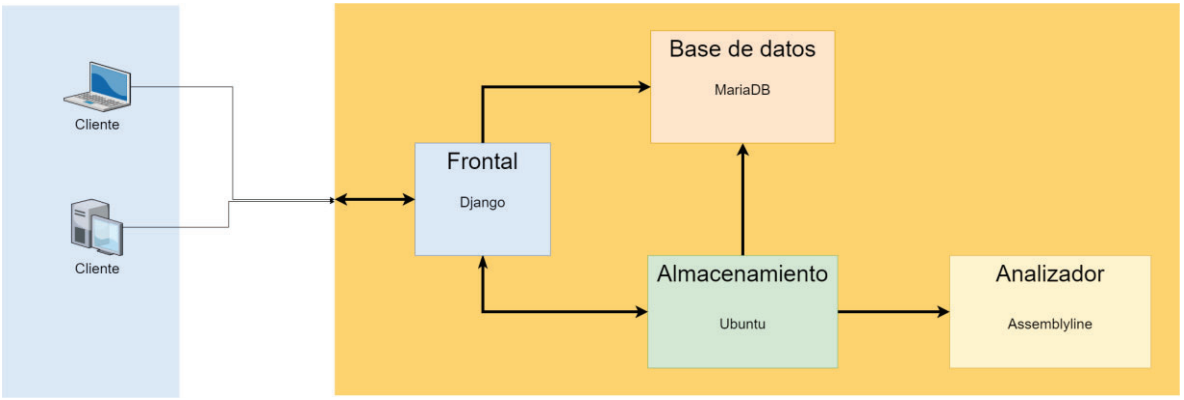

Ilustración 14: diagrama de arquitectura

Para el despliegue de todos los contenedores son necesarios al menos dos archivos que deben llamarse obligatoriamente *docker-compose.yml* y *Dockerfile*.

- **Docker-compose**: en este fichero de tipo *YAML* se definen los componentes y servicios que van a componer el contenedor y con el comando *docker-compose up* crea e inicia todos los servicios de la configuración. Esta herramienta permite la creación de aplicaciones multicontenedores independientes personalizados.
- **Dockerfile**: es el fichero en el que se encuentran los comandos a ejecutar en la construcción de un contenedor. Este fichero se puede compatibilizar con el de *docker-compose* añadiendo el *flag --build* al comando anterior.

5.1 Diagrama de red

Una de las características de los contenedores *Docker* es el control de red de los subsistemas de los contenedores. Un contenedor puede conectarse a otro contenedor, servicio corriendo localmente, otro ordenador de la red e incluso a contenedores de otro tipo. Por defecto, en un sistema multicontenedor, con las opciones y conexiones de red de serie, los contenedores se mantienen aislados e incomunicados los unos de los otros, haciendo imposible las conexiones y comunicaciones entre ellos. Para habilitar las conexiones, *Docker* incluye controladores de red para sus contenedores para dotar de las funciones básicas de red [23].

- **Controlador host:** este controlador elimina el aislamiento de red entre el contenedor y el host. Con esta configuración el contenedor es accesible desde cualquier servicio en ejecución dentro de la máquina. Los contenedores que se han incluido a esta interfaz son la base de datos, el analizador y el frontal. Este último exponiendo su puerto 8000 al exterior. Esta configuración se ha pensado para los contenedores que deben tener más de una conexión, como iba a ser el analizador en una primera fase de diseño, y es el caso de la base de datos.

- **Controlador bridge:** es un tipo de interfaz que conecta los contenedores incluidos en un mismo controlador entre ellos como si fuera un *switch*. Esta interfaz dota al contenedor de una dirección *ip* dentro del rango 172.17.X.X/16. Los contenedores que se han añadido a esta interfaz son los que tienen una única conexión, es decir, solo establece comunicación con un contenedor. En este grupo está la conexión del contenedor frontal con el contenedor de almacenamiento. Se añade el almacenamiento al *bridge* y no al host aprovechando la configuración de red de los demás contenedores. Al ya encontrarse al host son totalmente accesibles desde el almacenamiento y únicamente había que crear la conexión con el frontal, protegiendo así accesos desde el exterior.

Por lo que con esta configuración el diagrama de conexiones quedaría del siguiente modo donde se muestra al tipo de interfaz que se conecta cada contenedor.

En color azul se encuentran las conexiones por medio de la interfaz de host y en color rojo las conexiones que utilizan la interfaz del controlador bridge.

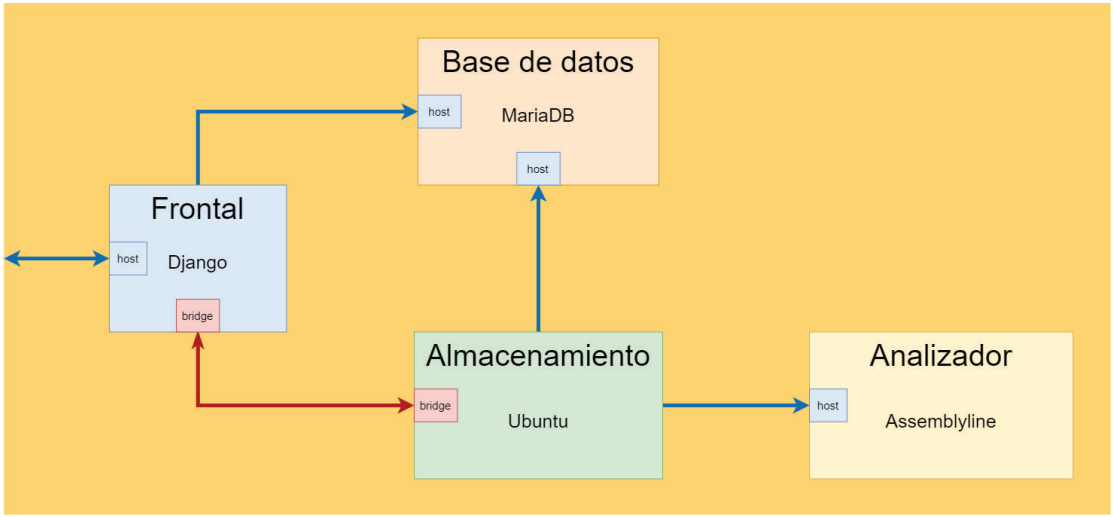

Ilustración 15: diagrama de red

5.2 Contenedor frontal

Este contenedor es el encargado de comunicarse e interactuar con el navegador del usuario. Para ello, se ha optado por un contenedor *Docker* de Django que sigue el patrón modelo vista controlador (MVC). Este patrón de diseño se basa en la separación de la lógica de negocio de la vista y del controlador. La lógica de negocio se encarga de la manipulación de datos, la vista es el conjunto de los elementos que se muestran en el cliente y, el controlador, es la interfaz de comunicación entre el modelo y la vista.

5.2.1 Ficheros de despliegue

Para el despliegue de este contenedor se han creado los archivos *Dockerfile* y *docker-compose* [24] los cuales son los encargados de crear, configurar y desplegar el contenedor levantando un servicio web que expone el puerto pertinente hacia el exterior de la máquina.

5.2.1.1 Fichero Docker-compose

El contenido del fichero *docker-compose* es el siguiente:

```
web:
  build: .
  environment:
    MYENV: EXAMPLE
  volumes:
    - .:/code
web_migrate:
  extends:
    service: web
  command: python manage.py migrate
web_run:
  extends:
    service: web
  command: python manage.py runserver 0.0.0.0:8000
  ports:
    - "8000:8000"
```

Ilustración 16: fragmento docker-compose frontal

Este archivo se encarga de migrar los modelos de la aplicación por si se ha producido una modificación en algún esquema interno con el comando:

```
python manage.py migrate
```

Y a continuación iniciar el servicio web en la dirección IP de origen de la máquina en el puerto 8000 mediante el comando:

```
python manage.py runserver 0.0.0.0:8000.
```

5.2.1.2 Fichero Dockerfile

El contenido del fichero *Dockerfile* es el siguiente:

```
FROM python:3.8
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
WORKDIR /code
COPY requirements.txt .
RUN pip3 install -r requirements.txt
COPY . /code/
```

Ilustración 17: fragmento Dockerfile frontal

Estas líneas tienen la función principal de definir la versión de *Python* que se va a utilizar en este contenedor y la instalación de los paquetes necesarios para el correcto funcionamiento del sistema. Para ello, mediante el comando *pip*, se descargan e instalan los paquetes definidos en el archivo *requirements.txt.*

5.2.2 Estructura del contenedor frontal

Este contenedor es el encargado de comunicarse con la parte del cliente y comunicarse con la base de datos y el almacenamiento para gestión, modificación, envío y recepción de datos. Se estructura en dos bloques a su vez:

- **Front-end**: En la parte de front-end se encuentran los diferentes archivos que conforman los archivos html que conforman la estructura de la interfaz de usuario, el fichero css que da los estilos y posicionar visualmente los componentes html y las funciones JavaScript para la interacción cliente-servidor.
- **Back-end:** aquí se desarrolla la lógica de la aplicación que permite su funcionamiento.

5.2.3 Diseño interfaz de usuario

La interfaz de usuario tiene el papel fundamental de atraer usuarios y mantenerlos en el tiempo. Para ello, la interfaz debe ser auto explicativa, es decir, el usuario debe entender su uso de un primer vistazo. Si la interfaz fuera muy compleja o confusa podría ahuyentar a los usuarios.

Para ello se han seguido algunos consejos de *Adobe* [25] para el diseño web como los siguientes:

- **Mantener la interfaz consistente:** la apariencia y estructura de todas las pantallas deben ser consistentes entre sí. Se ha mantenido la misma fuente de letra y colores, así como la estructura general en todas las pantallas.

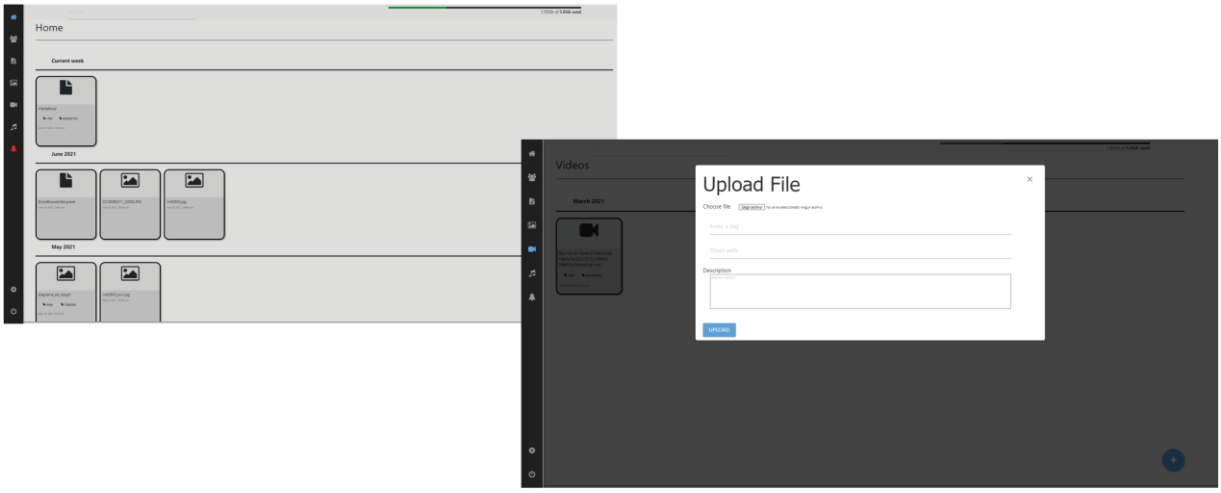

Ilustración 18: capturas interfaz

- **Navegación sencilla de usar:** se ha optado por un menú lateral para navegar por las diferentes secciones. Este menú muestra, de un vistazo, con qué secciones cuenta la interfaz de la aplicación, en qué sección se encuentra el usuario y si este tiene notificaciones.

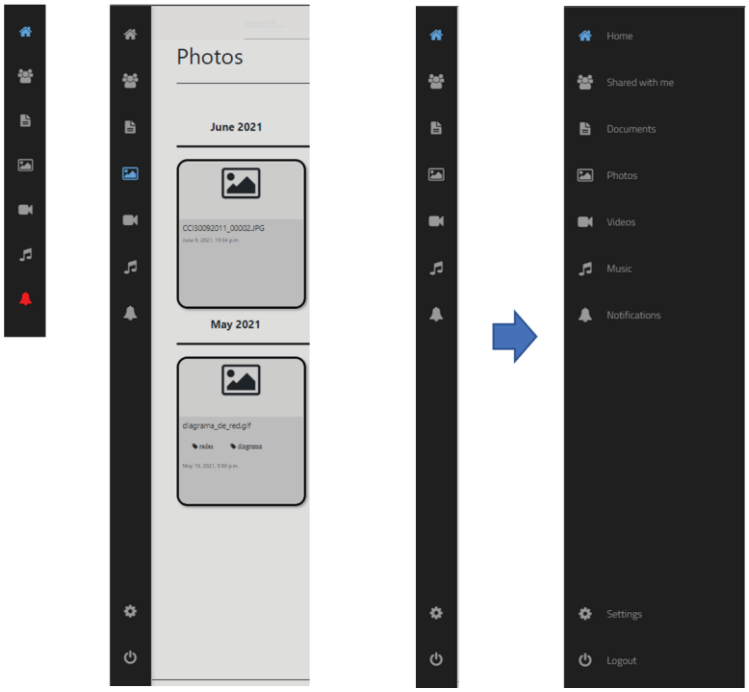

Ilustración 19: capturas menú lateral

El menú a su vez se despliega cuando el usuario sitúa el ratón por encima de él, desplegando el nombre de las secciones.

Adicionalmente, en la parte superior se incorpora una barra de búsqueda para que el usuario pueda identificar archivos con mayor rapidez según su nombre o tags.

Adicionalmente, en la parte superior se incorpora una barra de búsqueda para que el usuario pueda identificar archivos con mayor rapidez según su nombre o tags.

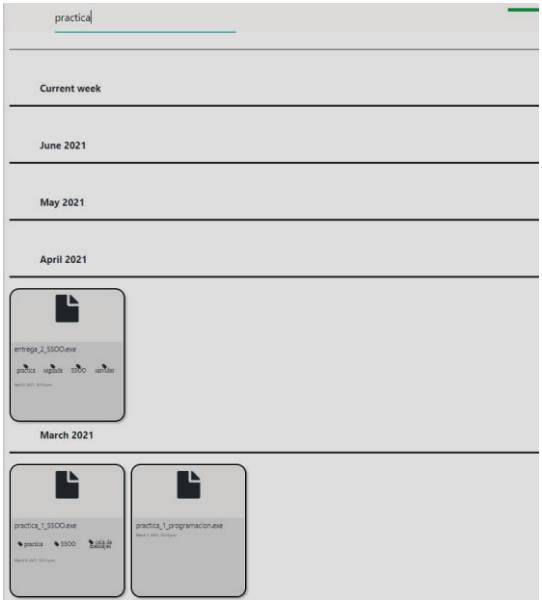

Ilustración 20: captura barra de búsqueda

- **Hacer que las cosas parezcan que funcionan:** las acciones interactivas son elementos con gran importancia para que el usuario obtenga *feedback* durante su navegación y sirven de verificación de que sus acciones tienen consecuencias.

  El buscador hace una búsqueda en tiempo real ofreciendo los resultados que se ajustan al texto introducido.

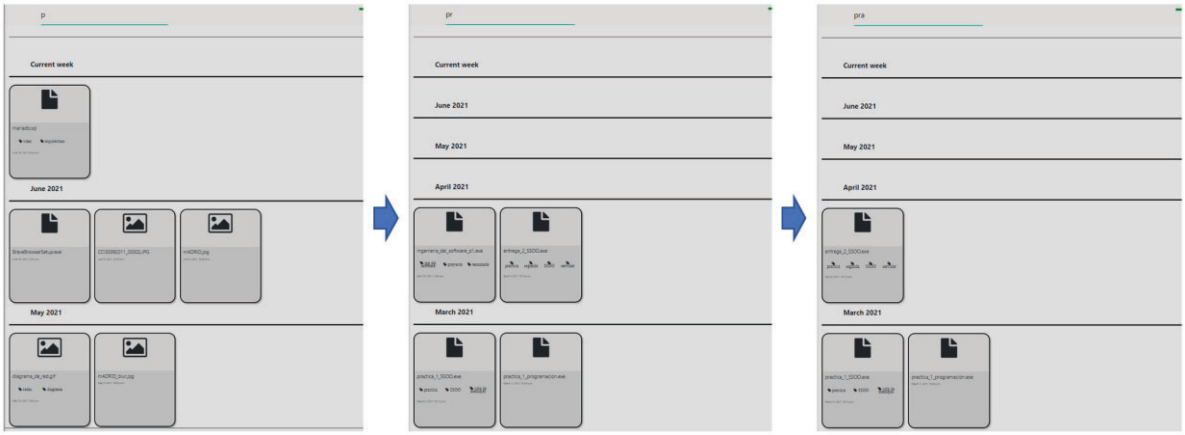

Ilustración 21: secuencia de búsqueda

El formulario de búsqueda aparece sobre el fondo con la finalidad que el usuario mantenga la sensación de seguir en la misma sección.

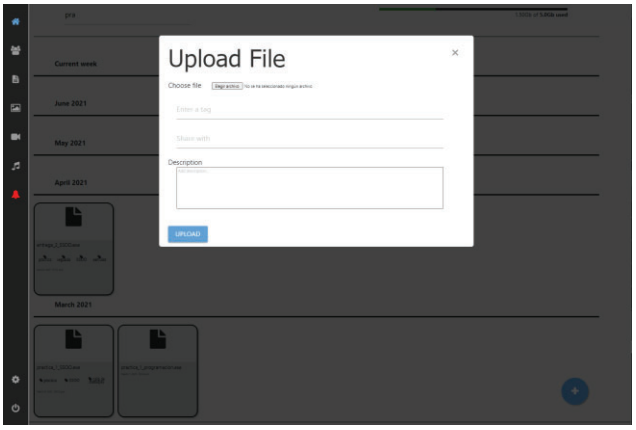

Ilustración 22: captura formulario de subida de archivo

En el formulario de búsqueda, al compartir se ofrece autocompletado al usuario para evitar errores.

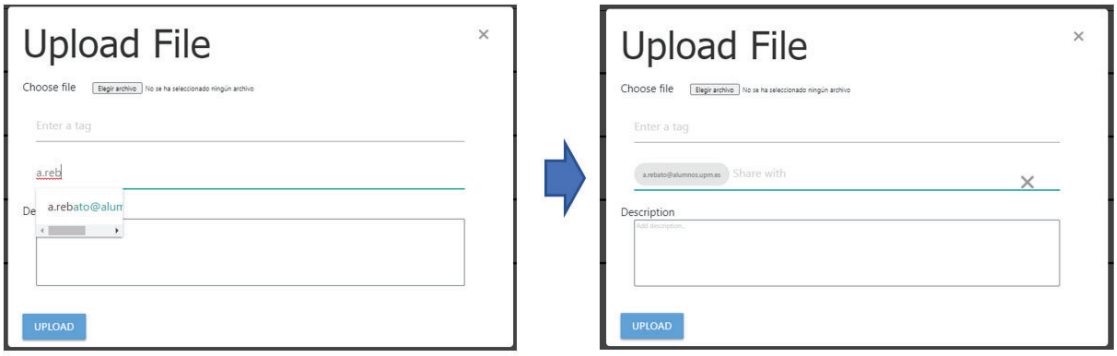

Ilustración 23: autocompletado de correo electrónico

Las tarjetas de los archivos reaccionan al pasar el ratón por encima haciendo un efecto de iluminación.

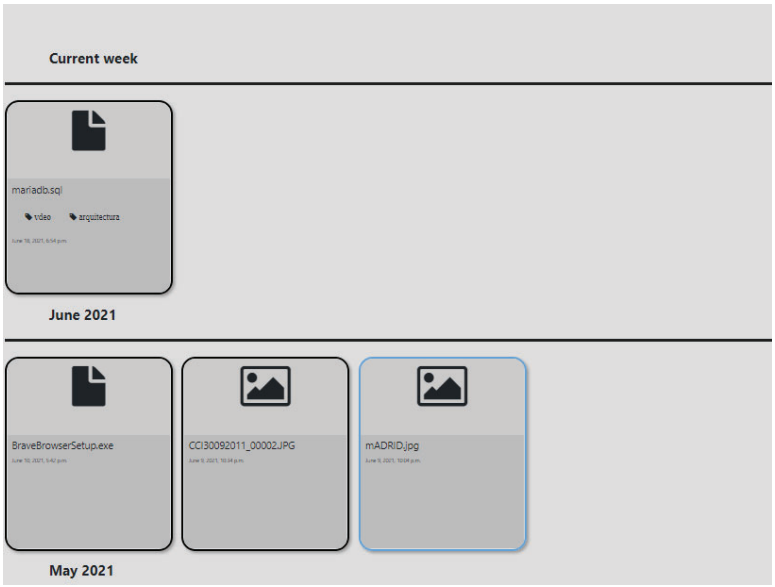

Ilustración 24: captura interacción con archivos

Al seleccionar un archivo se acciona un desplegable con los datos que se hayan provisto al haberlo subido a la plataforma donde se encuentran los botones para descargar o eliminar, otros metadatos como descripción, usuarios con los que se ha compartido, tags, entre otros y en la parte superior el veredicto del análisis sobre ese archivo con un código con los colores de un semáforo indicando el riesgo de ese archivo en la ejecución en un ordenador.

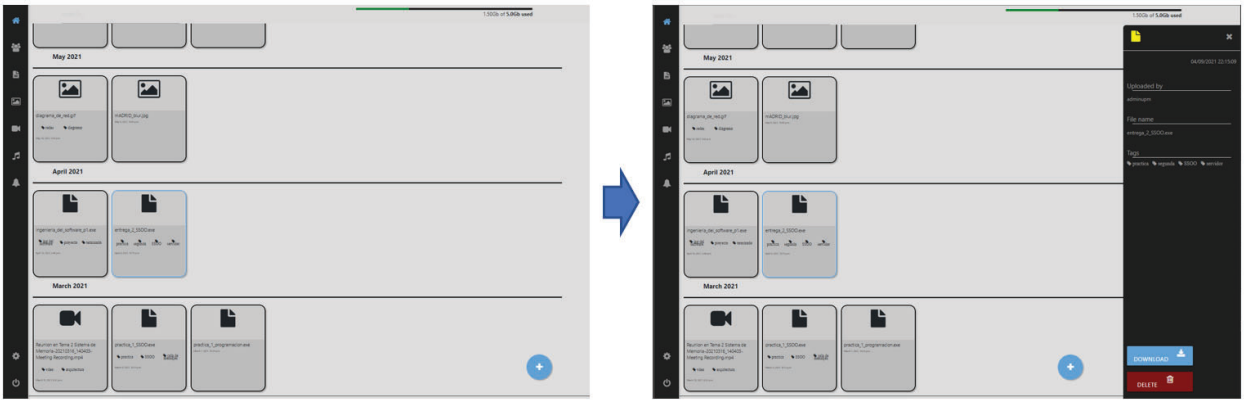

Ilustración 25: menú de archivo

5.2.4 Comunicación con el servidor

Para que la plataforma sea interactiva es necesario establecer unos sistemas de comunicación con el servidor para realizar operaciones sobre los recursos.

Estas comunicaciones se realizan mediante el envío de formularios y peticiones dinámicas utilizando la librería *Axios* [26]. Estas peticiones dinámicas tienen como primer objetivo aliviar la cantidad de información que renderizar y como segundo objetivo dar una navegación interactiva sin necesidad de refrescar la página cada vez que un usuario actualice una información.

Estos efectos pueden verse por toda la plataforma en estos ejemplos:

- Eliminación dinámica de notificaciones.

Ilustración 26: eliminación de notificaciones

- Actualización dinámica de información de archivos.

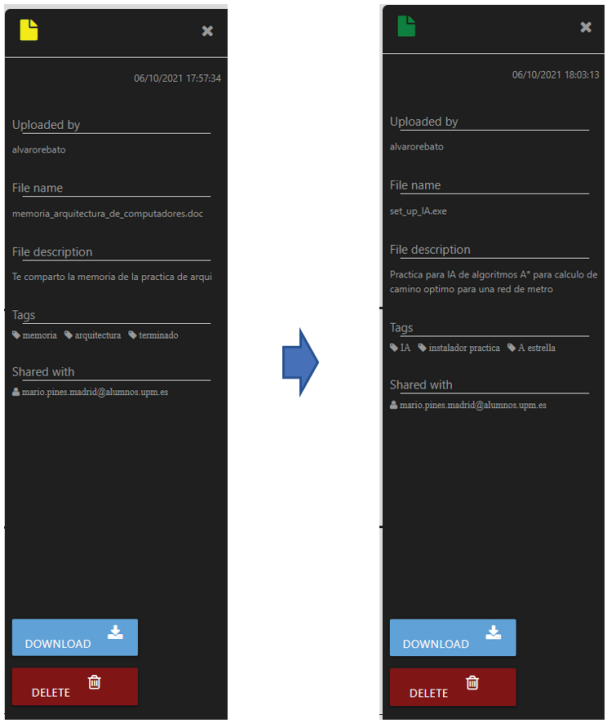

Ilustración 27: actualización dinámica de descripciones

- Actualización de cuota.

Ilustración 28: cambio en cuota de datos

- Descarga de archivos.

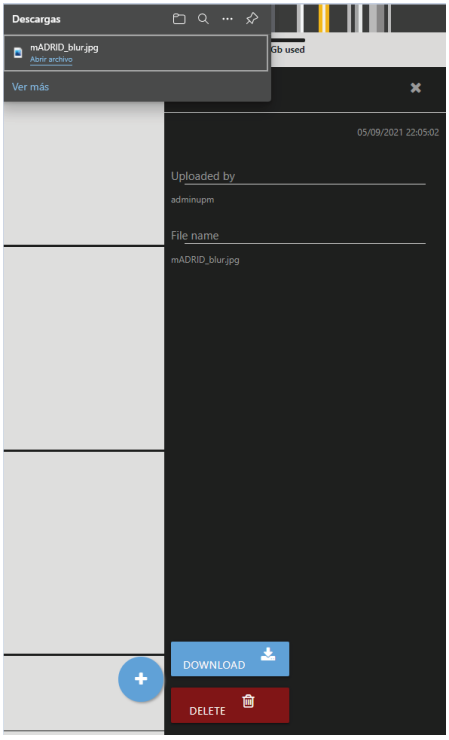

Ilustración 29: descarga de archivos

- Eliminación de archivos.

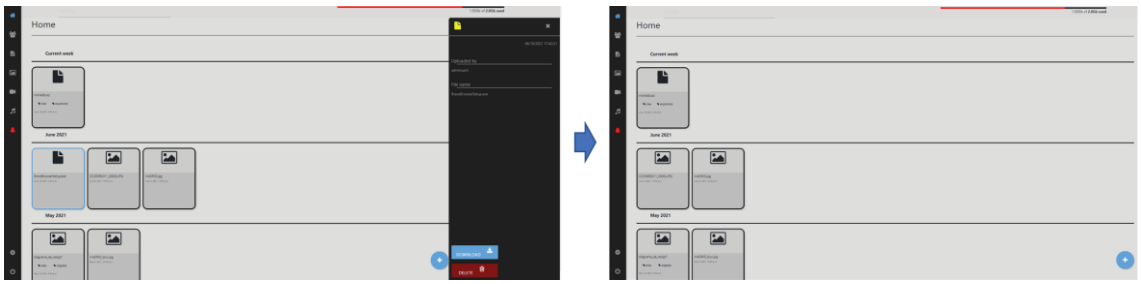

Ilustración 30: eliminación de archivos

5.2.5 Back-end del contenedor frontal

En la parte de *back-end* no solo se encuentran las funciones principales de comunicación con la interfaz si no también la lógica de negocio y seguridad de la aplicación. Se encarga, entre otras cosas, de verificar el permiso de acceso a datos, autenticar usuarios, enviar y recibir datos.

El servidor *Django* se ha configurado para exponer el puerto 8000 hacia el exterior, por lo que se van a aceptar todas las conexiones en ese puerto y para los siguientes *endpoints*:

```
urlpatterns = [
    path('', views.home, name='home'),
    path('home/', views.home, name='home'),
    path('signup/', views.signup, name='signup'),
    path('login/', views.login, name = 'login'),
    path('logout/', views.logout, name = 'logout'),
    path('download/<int:file_id>', views.download, name='download'),
    path('delete/', views.delete, name='delete'),
    path('clear_notification/', views.clear_notification, name='clear_notification'),
    path('delete_user/', views.delete_user, name='delete_user'),
    path('update_quota_user/', views.update_quota_user, name='update_quota_user'),
    path('file/<int:file_id>', views.file, name='delete'),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Ilustración 31: endpoints de la plataforma

/signup/: acepta los métodos http *POST* y *GET*.

- **GET**: crea el formulario para registrarse mediante nombre de usuario, email, contraseña y confirmación de contraseña. Se utiliza el siguiente formulario de *Django* el cual se renderiza junto la plantilla *signup.html*.

```
class SignUpForm(UserCreationForm):
    email = forms.EmailField(max_length=254, help_text='Required. Inform a valid email address.')

    class Meta:
        model = User
        fields = ('username', 'email', 'password1', 'password2', )
```

Ilustración 32: formulario registro de usuario

- **POST**: recibe el formulario de registro completado y comprueba que este sea válido. Una vez validado se realizan comprobaciones de seguridad que, aunque se utiliza un token *csrf,* se deben realizar las comprobaciones en la parte servidor por si se pudiera dar el caso que se haya modificado la petición entre el dispositivo cliente y el servidor.

  En el formulario se solicita al usuario que introduzca su contraseña dos veces a la hora de registrarse y en el *back-end* se comprueba que sean iguales como se puede observar en el siguiente fragmento de código.

```
if form_dup.cleaned_data.get('password1') != form_dup.cleaned_data.get('password2'):
    return render(request, 'error.html', {'error': '400: BAD REQUEST', 'description': 'passwords does not match'})
```

Ilustración 33: verificación de contraseñas

Se comprueba que el email del usuario no exista dentro de los usuarios registrados.

```
conn = mysql.query(
    'CALL check_user_exist(%(email)s)', {'email': email})
users = conn.fetchall()
if users:
    return render(request, 'error.html', {'error': '400: BAD REQUEST', 'description': 'email already in use'})
```

Ilustración 34: comprobación de existencia de usuario

Se comprueba que el email de registro pertenezca al dominio UPM *(@alumnos.upm.es y @upm.es)*. Si el email de registro contiene el dominio @upm.es el usuario será designado como administrador ya que ese tipo de email pertenece al equipo docente.

```
try:
    email_split = email.split('@')
    if email_split[1] == 'alumnos.upm.es':
        user_type = 0
    elif email_split[1] == 'upm.es':
        user_type = 1
    else:
        return render(request, 'error.html', {'error': '400: BAD REQUEST', 'description': 'this email is not valid'})
except:
    return render(request, 'error.html', {'error': '400: BAD REQUEST', 'description': 'something went wrong with the email'})
```

Ilustración 35: asignación de rol al nuevo usuario

Una vez realizadas estas comprobaciones se procede a insertar al usuario en la tabla de usuarios y crear su entrada en la tabla de cuota. Se inicia la sesión de usuario y se crea una cookie de sesión en el que se insertan los datos de usuario para poder redirigir a la pantalla principal y que comience a utilizar la plataforma.

```
mysql.query('CALL insert_new_user(%(user)s, %(mail)s, %(tipo)s)', {
             'user': username, 'mail': email, 'tipo': user_type})

form.save()
user = authenticate(username=username, password=raw_password)
auth_login(request, user)

conn = mysql.query(
    'CALL check_user_exist(%(email)s)', {'email': email})
users = conn.fetchone()
request.session['uname'] = users['user_name']
request.session['umail'] = users['user_email']
request.session['utype'] = users['user_type']
request.session['uid'] = users['user_id']
request.session['quota_used'] = users['quota_used']
request.session['quota_limit'] = users['quota_limit']

return redirect('home')
```

Ilustración 36: inserción de usuario

/login/: acepta métodos *GET* y *POST*.

- **GET**: renderiza la plantilla *login.html* y la manda al cliente.
- **POST**: recibe un formulario con el email y contraseña del usuario y se procede a la verificación de su identidad.

  En el caso que el usuario exista y haya introducido mal la contraseña la redirección será a la pantalla de *login* de nuevo.

  En el caso que el email del usuario no se encuentre, la redirección será a la pantalla de registro.

  En caso de que haya sido exitosa la autenticación se crea una cookie de sesión con los datos del usuario y se hace la redirección a la pantalla principal.

```
email = request.POST['email']
user = User.objects.get(email__iexact=email)
password = request.POST['password']
if user.check_password(password):
    conn = mysql.query(
        'CALL check_user_exist(%(email)s)', {'email': email})
    users = conn.fetchone()
    request.session['uname'] = users['user_name']
    request.session['umail'] = users['user_email']
    request.session['utype'] = users['user_type']
    request.session['uid'] = users['user_id']
    request.session['quota_used'] = users['quota_used']
    request.session['quota_limit'] = users['quota_limit']

    auth_login(request, user)
    return redirect('home')
```

Ilustración 37: creación de token de sesión para usuario

/logout/: acepta el método *GET* y se limpian los datos de sesión del usuario invalidando su cookie de sesión y obligando a hacer *login* en su siguiente visita.

/home/: acepta métodos *GET* y *POST*.

- **GET**: se encarga de preparar y enviar los datos al navegador cliente para su posterior visualización.

  Realiza varias peticiones para obtener los datos de visualización básicos de los archivos del usuario según sus tipos (documentos, música, archivos compartidos con ese usuario…) y se filtran por fecha (por la última semana y por meses).

```
sql_query = 'CALL get_all_user_files(%(i_user_id)s, %(i_file_type)s)'
sql_params = {'i_user_id': int(request.session['uid']), 'i_file_type': None}
with mysql.query(sql_query, sql_params) as conn:
    for user in conn.fetchall():
        file_to_add = user
        file_to_add['file_tags'] = user['file_tags'].split('|')
        if file_to_add['file_date'] >= d:
            if 'Current week' in user_files:
                user_files['Current week'].append(file_to_add)
            else:
                user_files.update({'Current week': []})
                user_files['Current week'].append(file_to_add)
        else:
            key_parse = str(months[file_to_add['file_date'].month] + " " + str(file_to_add['file_date'].year))
            if  key_parse in user_files:
                user_files[key_parse].append(file_to_add)
            else:
                user_files.update({key_parse: []})
                user_files[key_parse].append(file_to_add)
```

Ilustración 38: clasificacion temporal de archivos

Seguidamente se comprueba si el usuario tiene notificaciones pendientes para avisar al usuario en el caso de que existan.

```
sql_query = 'CALL get_user_notifications(%(i_user_id)s)'
sql_params = {'i_user_id': int(request.session['uid'])}
with mysql.query(sql_query, sql_params) as conn:
    user_notif = conn.fetchall()
```

Ilustración 39: comprobación de notificaciones

Se obtiene la cuota consumida y total con la que se va a visualizar el espacio de almacenamiento disponible para ese usuario, así como todos los emails de los usuarios de la plataforma para poder realizar el autocompletado de emails en el formulario de subida de archivos.

En el caso que el usuario de la sesión fuese un administrador se añade a los datos de envío el nombre de todos los usuarios, así como de sus espacios disponibles y ocupados de almacenamiento para la pantalla de administración.

```
if int(request.session['utype']) == 1:
    sql_query = 'CALL get_all_users_config'
    users = []
    with mysql.query(sql_query) as conn:
        for i in  conn.fetchall():
            i['quota_used'] = float(i['quota_used'])/10**3
            i['quota_limit'] = float(i['quota_limit'])
            users.append(i)
    to_front['users'] = users
```

Ilustración 40: datos para gestión de usuario

- **POST**: recibe un formulario con un archivo proveniente de un usuario y prepara los datos para enviar dicho elemento al almacenamiento para su posterior guardado y procesado.

Se calcula el tamaño del binario y clasifica la extensión para adjudicar el tipo de archivo. Estas extensiones se encuentran en el script *files_extensions.py*.

```
file_to_storage = io.BytesIO(file_to_storage)
file_to_storage.seek(0)
file_to_storage.seek(0, 2)
file_size = file_to_storage.tell()
file_to_storage.seek(0)

                ...

file_type = str(filename).split('.')
if len(file_type) > 1:
    extension = file_type[-1].lower()

    if extension in files_extensions.DOCUMENTS:
        extension = 'DOCUMENT'
    elif extension in files_extensions.IMAGE:
        extension = 'PHOTO'
    elif extension in files_extensions.VIDEO:
        extension = 'VIDEO'
    elif extension in files_extensions.AUDIO:
        extension = 'MUSIC'
    elif extension == 'zip':
        extension = 'ZIP'
    else:
        extension = 'FILE'

else:
    file_type = 'FILE'
```

Ilustración 41: cálculo de tipo de archivo

Una vez se tienen todos los datos necesarios para hacer el envío al almacenamiento, se completa el formulario de la petición.

```
respose = requests.post(url=STORAGE_SERVER_IP + '/store_file', data={
    'i_file_name': str(filename),
    'i_file_user': int(request.session['uid']),
    'i_file_size': float(file_size),
    'i_file_type': str(extension),
    'i_file_date': datetime.now(),
    'i_file_tags': str(tags),
    'i_file_shared': str(mails),
    'i_file_description': str(description),
    'user_email': str(request.session['umail'])
}, files={'file': (str(filename), file_to_storage, "application/octet-stream")})
```

Ilustración 42: envío de archivo al almacenamiento

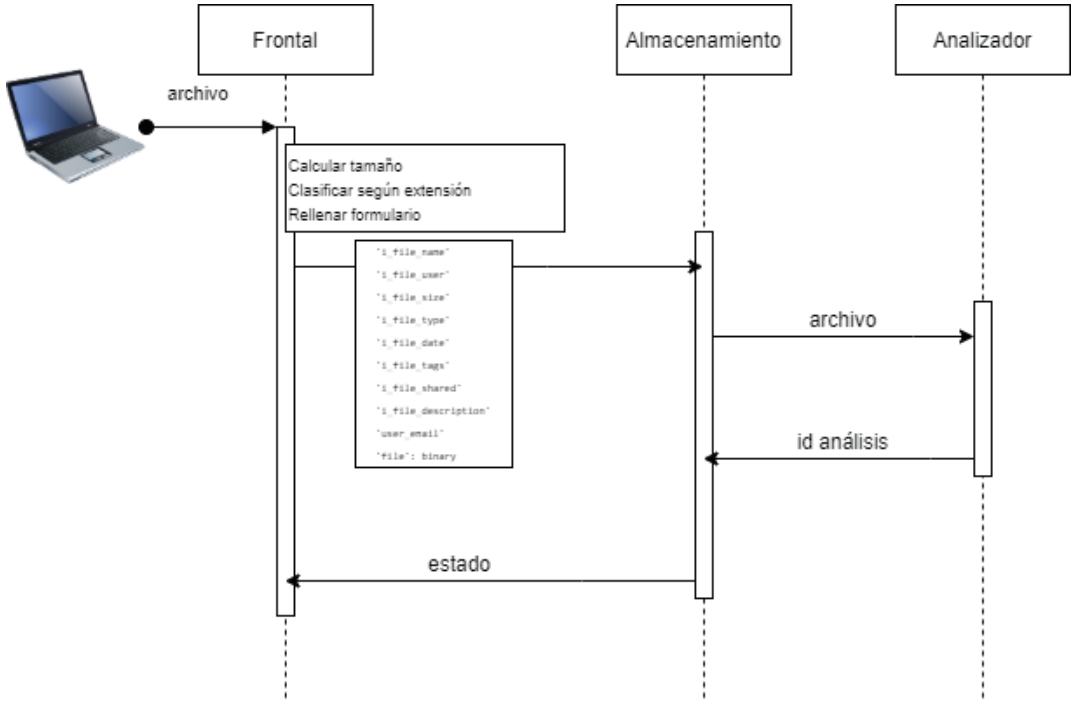

Ilustración 43: diagrama de comunicación en subida de archivos

/download/{file_id}/: admite peticiones *GET* y se encarga de iniciar la descarga del archivo pedido por medio de su identificador único.

En primer lugar, se comprueba que el usuario de la sesión tenga permisos para descargar el archivo mediante una consulta en la base de datos que comprueba:

- El usuario es el propietario del fichero.
- Otro usuario ha compartido con este usuario el fichero.

Si el usuario cumple cualquiera de estas dos condiciones se hace la petición al almacenamiento para recepción de archivo y envío al navegador cliente como adjunto.

```
response_file = requests.get(url=STORAGE_SERVER_IP + '/get_file/' + str(file_id), stream=True, headers={'Accept': '*/*'})

                                ...

response = HttpResponse(response_file.content, content_type="application/octet-stream")
response['Content-Disposition'] = f'attachment; filename="{filename}"'
return response
```

Ilustración 44: petición de descarga al almacenamiento

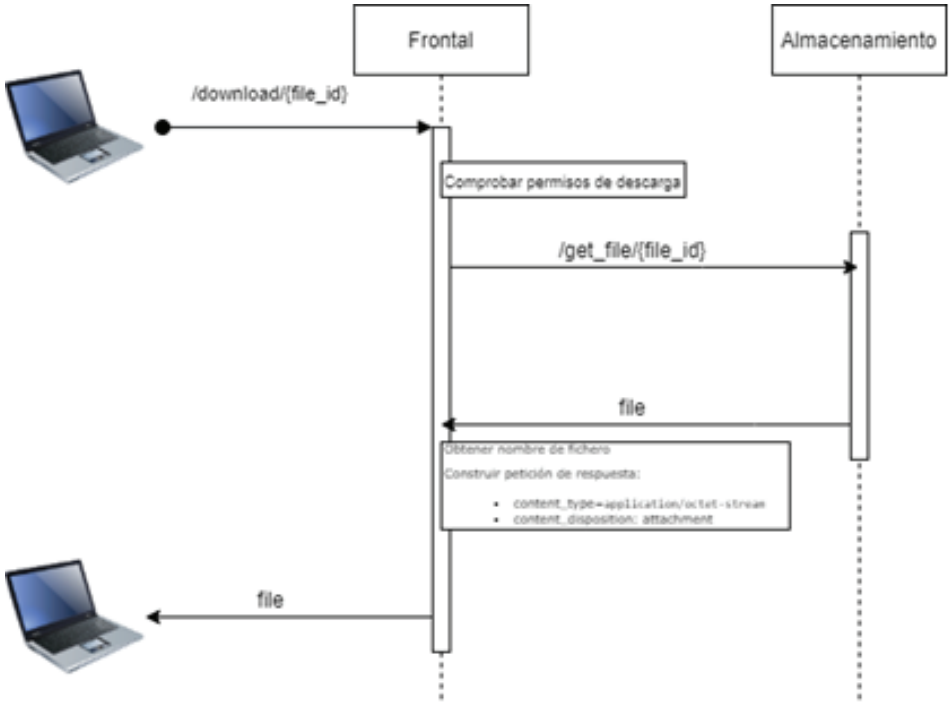

Ilustración 45: diagrama de comunicación con el almacenamiento

/delete/{file_id}/: admite peticiones *POST* recibiendo en el cuerpo de la petición el identificador de fichero a eliminar. Se comprueba que usuario que envía la petición tiene los permisos necesarios para realizar la operación:

- El usuario es el propietario del fichero.
- Otro usuario ha compartido con este usuario el fichero y este usuario es de tipo administrador.

Si se cumplen los requisitos se manda la petición de borrado al almacenamiento y se actualiza el espacio consumido por el usuario.

```
sql_query = 'CALL get_file_with_id(%(i_file)s)'
sql_params = {
    'i_file': upload_form['file_id']
}
with mysql.query(sql_query, sql_params) as conn:
    file_info = conn.fetchone()

sql_query = 'CALL get_quota(%(i_quota_user)s)'
sql_params = {
    'i_quota_user': file_info['user_id']
}
with mysql.query(sql_query, sql_params) as conn:
    quota_data = conn.fetchone()

new_quota = quota_data['quota_used'] - file_info['file_size']

sql_query = 'CALL update_quota_used(%(i_quota_user)s, %(i_quota_new)s)'
sql_params = {
    'i_quota_user': file_info['user_id'], 'i_quota_new': new_quota
}
mysql.query(sql_query, sql_params)
```

Ilustración 46: actualización del espacio ocupado por el usuario

Finalmente se elimina la entrada de ese archivo de la tabla ficheros. Al encontrarse sus referencias de modo *ON DELETE CASCADE*, el resto de las entradas que contengan el id de fichero se eliminarán.

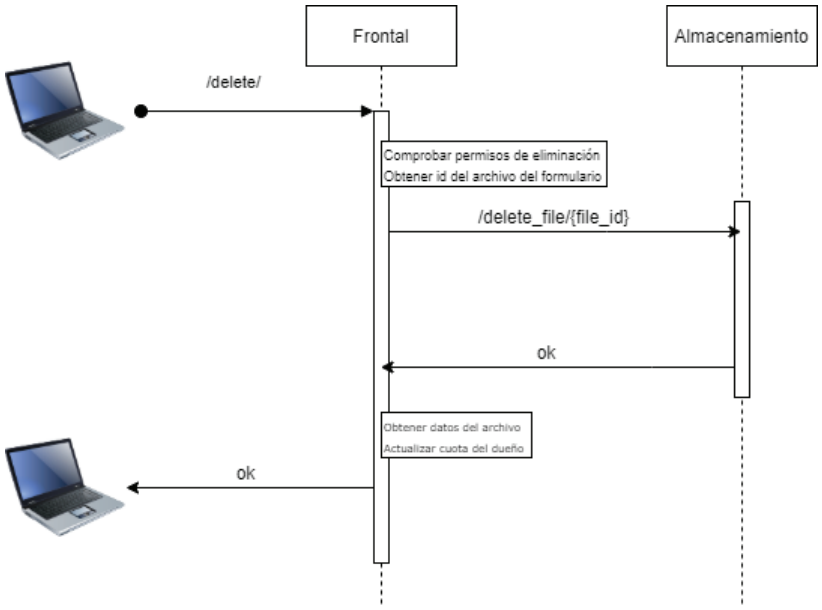

Ilustración 47: diagrama de comunicación en borrado de archivo con el almacenamiento

/clear_notification/: recibe una petición *POST* para marcar una notificación como vista por parte del usuario.

/delete_user/: recibe una petición *POST* con el id del usuario a eliminar. Esta operación solo se le ofrece a los administradores, aunque, un requisito para completarse es que el usuario de la sesión sea de tipo administrador.

```
if request.session['utype'] == int(1):
    sql_query = 'CALL delete_user(%(user_name)s, %(user_mail)s)'
    sql_params = {'user_name': upload_form['uname'], 'user_mail': upload_form['umail']}
    mysql.query(sql_query, sql_params)
    return HttpResponse(str('ok'))
```

Ilustración 48: petición de borrado de un usuario

/update_quota_user/: admite peticiones *POST* que actualizan el límite de almacenamiento de un usuario. Esta operación solo puede ser realizada por un administrador y se puede efectuar sobre cualquier usuario.

5.3 Contenedor base de datos

Este contenedor se encarga de desplegar un servicio de base de datos donde se almacenan los datos de los usuarios, archivos y otros datos relevantes necesarios de mantener a largo plazo. Esta base de datos es de tipo *MariaDB* la cual se gestiona a través de *phpMyadmin*. Se ha optado por este tipo de base de datos por tratarse de una solución ligera, rápida y capaz de almacenar procedimientos que van a utilizarse para realizar operaciones sobre la base de datos.

La utilización de las llamadas a procedimientos almacenados es una función que tiene muchos beneficios en el momento de desarrollo ya que estos procedimientos actúan como funciones que son capaces de recibir parámetros y ejecutar varias sentencias SQL en una llamada. De este modo, se pueden hacer operaciones secuenciales con una consulta desde otro servidor, decrementando el tráfico de red y el tiempo de respuesta. Adicionalmente, al encontrarse almacenados y compilados en la base de datos, su llamada y ejecución es mucho más rápida y eficiente que enviando la consulta declarada de forma literal. Así mismo, se pueden definir procedimientos que se utilicen en varias partes del código evitando código redundante.

Para el diseño del almacenamiento de los datos se ha realizado mediante un modelo relacional que lo constituye cinco entidades las cuales se relacionan entre ellas generando tablas entre sí. Se separan estas entidades en usuarios, archivos, cuota, archivos compartidos y estado de los archivos. La relación entre estas entidades puede verse en el siguiente diagrama:

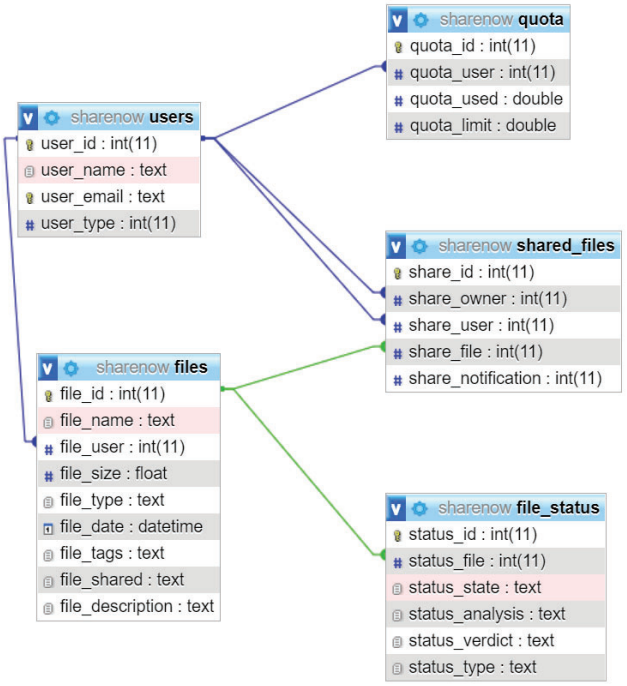

Ilustración 49: diagrama de tablas de base de datos

Estas tablas tienen la siguiente función:

- **Users**: donde se almacenan el nombre, email y rol de un usuario. Esta tabla se utiliza para crear relaciones con otras entidades y ampliar los datos de un usuario.

- **Files**: almacena los datos relevantes sobre los archivos como el nombre, el id del usuario propietario, tamaño (en bytes), tipo (documento, archivo de música, archivo de video, archivo de imagen), fecha de subida, los tags definidos por el propietario, los usuarios con los que se comparte y la descripción. Como la descripción puede tratarse de un texto extenso se va a almacenar en base64.

- **File status**: guarda los datos del estado de análisis de cada archivo. Los datos referentes al estado de un archivo son el id del fichero, el estado del fichero (escaneando o escaneado), el identificador de análisis del servicio de análisis, el resultado del análisis.

- **Shared files**: almacena la relación entre el dueño de un fichero con otro usuario. La relación se define con el id del dueño, el id del fichero y el id del usuario con el que se comparte. Se añade también un *flag* que controla la notificación para el usuario con el que se ha compartido el archivo.

- **Quota**: relaciona un usuario con el uso que ha hecho del almacenamiento de la plataforma donde se registra su espacio ocupado y la cuota disponible.

Para la comunicación de los módulos de Python con la base de datos se ha creado una clase que maneja la conexión con el contenedor SQL con el siguiente fragmento de código:

```
def __connect__(self):
    self.conn = pymysql.connect(host=self.host,
                                user=self.user,
                                password=self.password,
                                db=self.db,
                                charset='utf8mb4',
                                cursorclass=pymysql.cursors.DictCursor,
                                autocommit=True,
                                init_command="SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;",
                                port=3306)
    self.cursor = self.conn.cursor()
```

Ilustración 50: conexón desde módulo Python a base de datos

Para la conexión a la base de datos son necesarios los parámetros de la autenticación del usuario, la base de datos objetivo y el puerto 3306 que es el puerto por defecto para el protocolo MySQL, el cual utiliza *Mariadb*. Otros parámetros utilizados son:

- **init_command**: se ha configurado como `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE` con el que el motor *SQL* bloqueará los registros utilizados y evitará condiciones de carrera para conseguir consistencia en los datos y que las transacciones sean aisladas.
- **cursorclass**: establece el tipo de estructura de datos que va a devolver el cursor de *pymysql*, en este caso un diccionario.
- **autocommit**: por defecto se encuentra desactivado. Al configurarlo como *True* indica la modificación como permanente y de este modo, visible para el resto de los usuarios.

Y también se encarga de realizar las consultas mediante llamadas a procedimientos almacenados a los que se les pueden añadir parámetros en caso de necesitarse con el siguiente fragmento:

```
def query(self, sql, params=None):
    self.__connect__()
    if params:
        self.cursor.execute(sql, params)
    else:
        self.cursor.execute(sql)
    return self.cursor
```

Ilustración 51: envío de consulta a base de datos

5.4 Contenedor almacenamiento

En este contenedor se ubican principalmente los archivos que los usuarios hayan subido a la plataforma además de:

- Servicio de comunicación con el contenedor frontal.
- Servicio de comunicación con el contenedor de análisis.
- Servicio de sondeo constante de estado de análisis de ficheros.

Para su despliegue se han definido los ficheros *Docker* pertinentes que despliegan un sistema operativo *Ubuntu* en su versión *18.04* y que se encuentrará en funcionamiento permanente. Para el fichero *Dockerfile* se ha hecho uso de su imagen oficial en la plataforma *DockerHub* [27]. Para el fichero *docker-compose* se ha utilizado la siguiente configuración con la que se indica su reinicio en caso de error con el comando `on-failure` y el comando `["sleep","infinity"]` para que se mantenga siempre en funcionamiento.

```yaml
version: "3"
services:
  ubuntu:
    container_name: storage-server
    image: ubuntu:18.04
    restart: on-failure
    command: ["sleep","infinity"]
```

Ilustración 52: fragmento docker-compose almacenamiento

Este contenedor se encuentra conectado al resto de contenedores ya que es el único que interactúa con todos ellos.

Los archivos de los usuarios se ubican en la carpeta */srv/* del sistema de ficheros de *Ubuntu* en donde se crearán las carpetas de los usuarios según se vayan registrando y cuyo identificador será el email del propietario. De esta forma se puede acceder a los archivos de forma directa para las operaciones de descarga, eliminación y creación.

Por otro lado, como se ha comentado previamente existen dos servicios los cuales van a servir para establecer comunicación con el contenedor frontal y con el contenedor de análisis.

5.4.1 Servicio Flask

Se ha aprovechado el diseño minimalista y ligero de *Flask* para desarrollar esta aplicación *REST* que conecta el contenedor frontal con el de almacenamiento y recibe operaciones sobre los archivos guardados en las carpetas de los usuarios. Así mismo, se encarga de mandar a analizar los ficheros que se suben a la plataforma. Las operaciones que realiza este servicio son las siguientes:

- **/store_file/:** esta operación de tipo *POST* recibe los datos relacionados con un nuevo archivo. Estos datos relacionan este archivo con su propietario, usuarios con los que se comparte y otros metadatos en el formulario de la petición. El binario del fichero en cuestión se encuentra en el campo de datos. Las operaciones que realiza se pueden ver el siguiente diagrama:

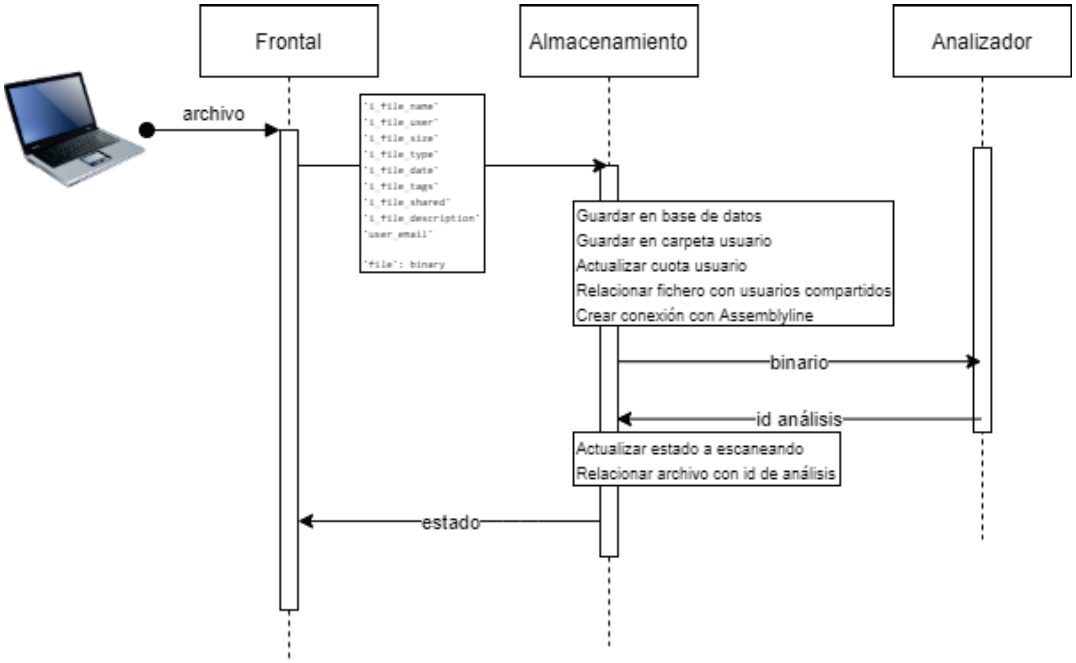

Ilustración 53: diagrama de comunicación almacenamiento de archivo

- **/get_file/<file_id>/:** esta operación de tipo *GET* recibe el id de un archivo con el que consulta nombre y propietario para mandar dicho elemento como `application/octet-stream` para su posterior descarga.

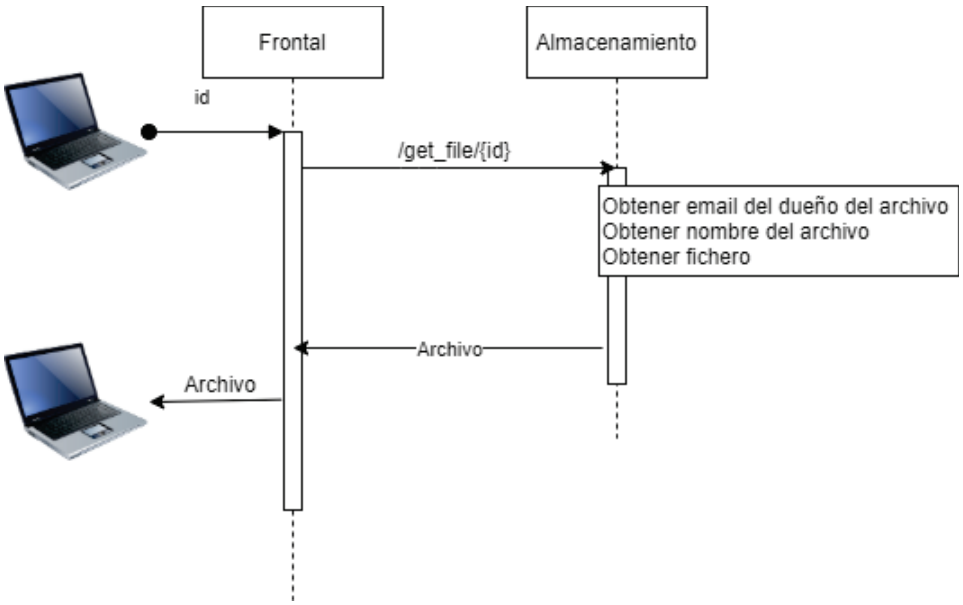

Ilustración 54: diagrama de comunicación en petición de descarga

- **/delete_file/<file_id>**: esta operación de tipo *DELETE* recibe un id de un fichero con el que se consultan los datos necesarios para acceder a él y eliminarlo.

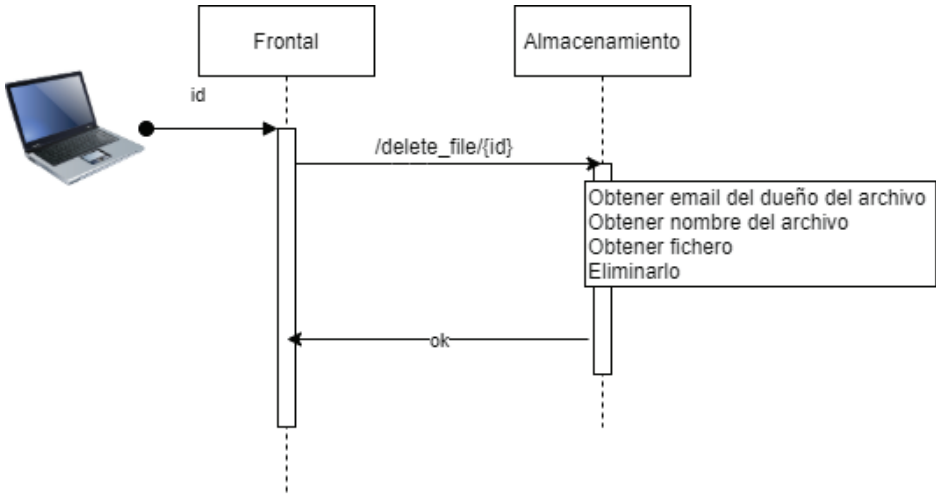

Ilustración 55: diagrama de comunicación eliminación de archivo

- **/new_user:** esta operación de tipo *POST* crea la carpeta de un usuario en el momento que se completa su alta en la aplicación, para ello recibe en el formulario el email con el que se ha registrado esa persona.

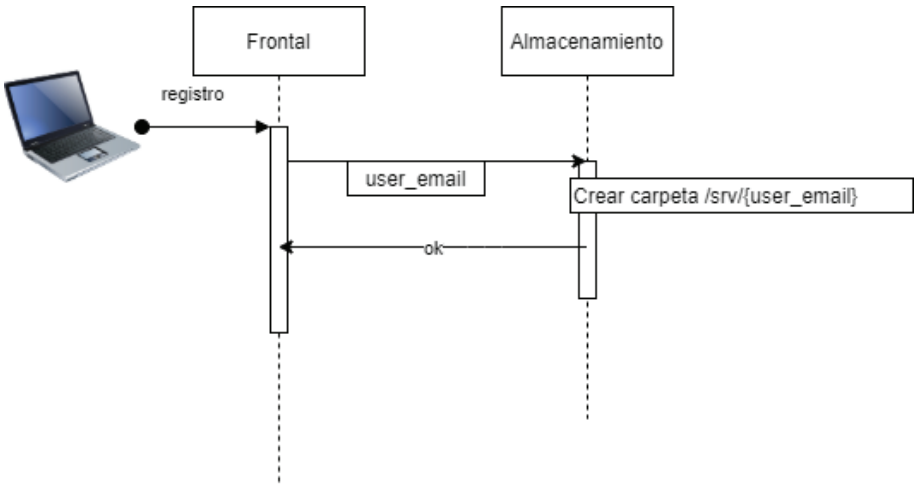

Ilustración 56: diagrama de comunicación registro de usuario

5.4.2 Servicio de sondeo de análisis

Este servicio realiza un sondeo o "polling", una operación constante en bucle encargada de recoger todos los ficheros que están siendo escaneados y preguntar al contenedor de análisis si ha concluido su análisis. En el caso de que el escaneo de un archivo haya concluido, se encargará de recoger la puntuación, clasificarla y actualizar los campos pertinentes en su entrada de la base de datos y pasar a preguntar por el siguiente archivo. En cuanto haya preguntado por el último fichero en análisis esperará un minuto hasta volver a realizar el sondeo.

Estos datos de examen se obtienen por un reporte que genera el propio *Assemblyline* con los resultados del análisis en el que entre otros campos encontramos la puntuación con la que podemos realizar un veredicto y si se ha detectado el tipo o familia de malware (virus, gusano, spyware, troyano…) en caso de que fuera un archivo malicioso. El reporte contiene información adicional que para este proyecto no es relevante.

Una vez ha terminado el escaneo de un fichero, este servicio procede a eliminarlo del contenedor de análisis para minimizar el espacio ocupado en la plataforma y evitar duplicidad de archivos en los contenedores.

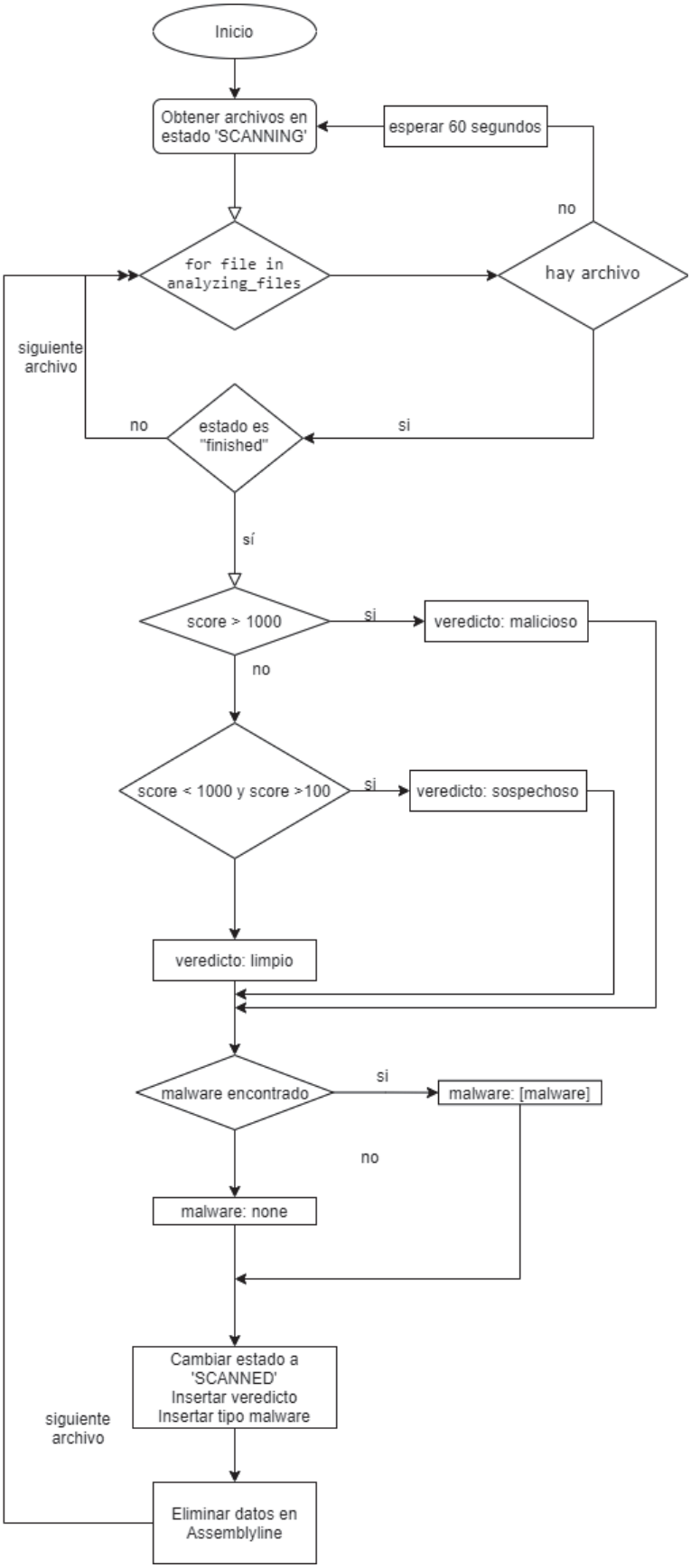

Ilustración 57: diagrama de flujo de servicio de sondeo

5.5 Contenedor de análisis

En este contenedor se despliega la herramienta de análisis Assemblyline. Este software desarrollado por el Centro Canadiense para la Ciber Seguridad tiene el fin de automatizar el análisis de archivos. A esta herramienta podemos añadirle otros paquetes para la mejora de análisis como nuevas reglas de escaneo, antivirus u otras tecnologías comerciales.

Este software puede implementarse como un contenedor de Docker, por el que se ha decantado en este proyecto, o para una aplicación "multi-host" de *Kubernetes* [28]. Así mismo, también se incluyen dos tipos de integraciones: una mínima y una completa. La principal diferencia es la cantidad de servicios que levanta, esto no afecta al análisis.

Para el funcionamiento de la versión mínima de *Assemblyline* el archivo de configuración incorpora y ejecuta los siguientes servicios:

- **Elastic Stack**: que incorpora *Elasticsearch* [29] y que se utiliza como base de datos no relacional en la que se guardan los resultados y que posteriormente se pueden consultar al terminar un análisis [30].
- **Redis**: que actúa como un bróker de mensajes y caché y que se utiliza para el encolado y gestor de archivos para análisis [31].
- **Nginx**: es un software para servicios web que sirve como balanceador de carga, conexiones HTTP, proxy inverso, entre otras funciones y sirve para poder conectarse a *Assemblyline* [32].

Para la conexión y comunicación con esta herramienta se ha hecho uso de la *API* que integra *Assemblyline* y se ha desarrollado un módulo Python para la conexión y comunicación con el contenedor analizador. Los *endpoints* utilizados de esta API son los siguientes:

- **/api/v4/auth/login/:** el primer paso para autenticarse y poder utilizar los demás métodos que ofrece la *API* es esta operación *POST* en la que deben ir las credenciales de usuario y contraseña. Si la respuesta es correcta se recibirá el token con el que realizar las demás consultas. Este token verifica al usuario y tiene una duración, se devuelve en las cookies en el campo XSRF-TOKEN y se debe incluir en las cabeceras de cada petición a la *API*.

```
login = self.session.post('https://' + self.host  + '/api/v4/auth/login/',
                          data={
                              'user': self.credentials['user'],
                              'password': self.credentials['password'],
                          },
                          verify=False)
if login.status_code == 200:
    login_response = json.loads(login.text)
    if "XSRF-TOKEN" in login.cookies:
        self.session.headers.update({"X-XSRF-TOKEN": login.cookies['XSRF-TOKEN']})
```

Ilustración 58: autenticación vía API al analizador

- **/api/v4/auth/logout/:** esta petición *GET* pone fin a la conexión invalidando el token.

```
logout = self.session.get('https://' + self.host  + '/api/v4/auth/logout/',
                          verify=False)
logout_response = json.loads(logout.text)
if logout_response['api_error_message'] == '' and logout_response['api_response']['success'] == True:
    self.authenticated = False
```

Ilustración 59: deautenticación del analizador

- **/api/v4/submit/:** este método *POST* envía un archivo para iniciar su análisis. Admite modificar unos parámetros para su escaneo. Todos los archivos se analizarán de la misma forma y con escaneo profundo para minimizar los falsos negativos.

```
params = {
    "deep_scan": True,
    "description": "Inspection of file: " + filename
}
```

Ilustración 60: parámetros de análisis de archivo

El binario del archivo se incluye en el campo de ficheros de la petición:

```
submit = self.session.post('https://' + self.host  + '/api/v4/submit/',
                           data=data,
                           files={'bin': file},
                           verify=False)
submit_response = json.loads(submit.text)
```

Ilustración 61: envío de archivo a análisis

Como respuesta tenemos el id de análisis que se le asigna a ese archivo:

```
submit_response = json.loads(submit.text)

if submit_response['api_error_message'] == '':
    return submit_response['api_response']
else:
    return None
```

Ilustración 62: respuesta del analizador al mandar un archivo

- **/api/v4/submission/full/{sid}/:** con este método *GET* obtenemos el reporte del escaneo de un archivo. La respuesta es un archivo de tipo Json con los datos y resultados del análisis.

```
report = self.session.get('https://' + self.host  + f'/api/v4/submission/full/{sid}/',
                          verify=False)
report_response = json.loads(report.text)
if report_response['api_error_message'] == '':
    return report_response['api_response']
else:
    return None
```

Ilustración 63: obtención de resultados del analizador

- **/api/v4/submission/is_completed/{sid}/:** por medio de este método *GET* se consulta si el examen de un archivo ha concluido o en su defecto sigue en proceso.

```
status = self.session.get('https://' + self.host  + f'/api/v4/submission/is_completed/{sid}/',
                          verify=False)
status_response = json.loads(status.text)
```

Ilustración 64: consulta de estado de análisis

En caso de que haya terminado, a la respuesta se añade el campo *api_response*. En caso contrario el análisis sigue en curso.

```
if status_response['api_response']:
    sample_status = 'finished'
else:

    report = self.get_report(sid)
    if report['times']['completed'] is not None:
        sample_status = 'error'
    else:
        sample_status = 'running'
```

Ilustración 65: respuesta de estado

- **/api/v4/submission/{sid}/:** este método *DELETE* elimina el binario y los resultados del análisis.

```
purge = self.session.delete('https://' + self.host  + f'/api/v4/submission/{sid}/')
purge_response = json.loads(purge.text)

if purge_response['api_error_message'] == '':
    if purge_response['api_response']['success'] == True:

        removed = True
    else:
```

Ilustración 66: eliminación de datos de un archivo del analizador

6 Resultados y conclusiones

A lo largo del desarrollo de este proyecto se ha realizado un estudio y análisis exhaustivo de un gran número tecnologías que se pudieran ajustar a la solución. A la hora de elegir las tecnologías adecuadas se ha tenido en cuenta principalmente que estas se pudieran conectar y trabajar de forma conjunta. Cabe no olvidar otros procesos en el desarrollo del software como son la definición de requisitos y diseño de la arquitectura los cuales han sido de gran vitalidad para agilizar el desarrollo.

Gracias a la complejidad y el reto que supone un sistema de estas características se ha podido indagar en el funcionamiento y características de los contenedores, de sus funciones y posibilidades hasta formar un entorno donde se encuentren varios contenedores independientes ejecutándose y comunicándose entre sí. Tratándose, además, de una tecnología en constante crecimiento para el desarrollo de servicios y microservicios y ganando popularidad en las empresas.

En el desarrollo de la interfaz de usuario, parte vital el todo desarrollo web y que supone en muchas ocasiones la diferencia entre el éxito de un negocio o la desaparición de este, ha supuesto un reto en su desarrollo. Se trata de un apartado en constante evolución, con continuo cambio de tendencias y se ha intentado ofrecer una solución a la altura de las tendencias actuales persiguiendo una apariencia sencilla, funcional e interactiva.

Una de las partes más complicadas para la cumplimentación de este sistema ha sido la búsqueda del elemento que pudiera encajar en la función del analizador, para el que se ha estudiado los tipos de análisis y herramientas para este propósito. Al final se ha dado con la herramienta que mejor se ajustaba a las necesidades de rapidez, consumo de recursos, sencillez de instalación y documentación clara y concisa para desarrolladores.

Durante el desarrollo han ido ocurriendo contratiempos que han retrasado la fecha de finalización pero que han servido para indagar y entender cada tecnología a fondo, cuáles son sus funcionalidades y características más básicas que en muchas ocasiones se pasan por alto, como por ejemplo la gestión de redes de *Docker*, o la posibilidad de utilizar procedimientos almacenados de *MariaDB*. Sin olvidarnos del funcionamiento de *Assemblyline*, entre otros.

Termino muy satisfecho con este proyecto que me ha servido para explorar muchas tecnologías que se utilizan hoy en día en varias empresas, como el desarrollo de *front-end*, *back-end*, integración de tecnologías mediante *API*, creación y gestión de una base de datos, conexiones de red, programación de servicios y microservicios además del análisis, estudio y criba de tecnologías. Todos estos conocimientos me serán de gran utilidad para mi trayectoria profesional.

7 Análisis de Impacto

El almacenamiento en la nube supone una buena solución en diferentes ámbitos del día a día para mucha gente y empresas.

Impacto personal

En el espacio doméstico y personal, este tipo de servicios se utilizan principalmente para cumplir la función de copia de seguridad de datos personales ya que ofrece la persistencia de los datos ante problemas comunes como la eliminación accidental, perdida de dispositivos USB, fallos en los dispositivos de almacenamiento domésticos entre otros posibles sucesos. Conjuntamente, la localización en la nube de estos archivos hace posible su uso y disfrute en cualquier momento, desde cualquier sitio, con cualquier dispositivo ya que en muchos proveedores no es necesaria la instalación de software adicional. Y uno de los mayores puntos fuertes de este sector es su amplia oferta, el número de empresas que ofrecen espacios de almacenamiento es suficientemente amplia como para que cada persona pueda encontrar una tarifa que se ajuste a sus necesidades.

Por el contrario, este tipo de tecnologías pueden provocar un cierto escepticismo basado en el desconocimiento, ideas sesgadas en muchas ocasiones formadas por sucesos puntuales como noticias de robo de información o suplantación de identidad e incluso la incertidumbre sobre qué podrían hacer estas empresas sobre los datos personales de las personas.

Impacto empresarial

El almacenamiento en la nube es, para muchas empresas, un servicio fundamental para el correcto funcionamiento de varios de sus departamentos. Evita en gran medida la necesidad de adquisición de grandes equipos de almacenamiento ahorrando así mucho dinero a pequeñas y grandes empresas además muchas de las empresas que ofertan estos almacenamientos permiten la ampliación de espacio ajustándose a las necesidades de la empresa y acomodando el espacio necesitado al espacio requerido. El intercambio, actualización y acceso de archivos es hoy una necesidad para el desarrollo ágil de proyectos y que sucede en muchas ocasiones gracias a el almacenamiento distribuido.

Impacto social

Cada día millones de archivos se comparten a través de las plataformas de almacenamiento. Para acceder a estos recursos es necesario el acceso a internet por lo que esta infraestructura es de vital importancia. Es necesario, por tanto, de una conexión de red que, es imposible de alcanzar en muchos puntos del mundo. Hay muchos proyectos en desarrollo para llevar conexiones

de calidad a zonas sin acceso a internet, pero hasta que eso ocurra la cantidad de usuarios que pueden hacer uso del almacenamiento en la nube es limitado.

Impacto económico

El almacenamiento en la nube tiene una gran importancia en el desarrollo e innovación de un modelo de negocio que necesita de mano de obra cualificada para su crecimiento. Esto unido a una creciente demanda de almacenamiento se verá reflejado en la ampliación de los centros de datos y la apertura en otras localizaciones que darán empleo a una gran cantidad de personas para soporte, desarrollo y mantenimiento reflejándose así en un ligero aumento en la riqueza social de los lugares donde se localicen estas infraestructuras de almacenamiento.

Impacto medioambiental

Las altas demandas de energía para garantizar el constante funcionamiento de estos servicios es una de las mayores amenazas hacia el medio ambiente. Estos centros de datos necesitan de un gran volumen de electricidad que, por el momento, no puede ser satisfecho en su mayoría por energías renovables o alternativas si no que ha de ser electricidad generada mediante carbón. Este tipo de producción, tratándose de una de las fuentes más contaminantes, genera efectos muy negativos alterando ecosistemas, biomas y poniendo en riesgo a especies de animales y dañando la capa de ozono.

No obstante, estas empresas junto a los gobiernos están transicionando sus fuentes de energía hacia otras más sostenibles y mejorando la eficiencia energética de sus centros de datos con la finalidad de reducir la huella de carbono que han generado durante todos estos años [33].

Impacto cultural

Cada día se infectan miles de ordenadores por archivos que se descargan sin conocer su contenido. Esto supone un riesgo para los dispositivos por lo que la calificación de que un archivo puede ser potencialmente peligroso supondría un cambio en el comportamiento social incitando a desconfiar de archivos de fuentes desconocidas. Por el momento, ninguna de las tecnologías de almacenamiento del mercado ofrece esa información a los usuarios.

8 Bibliografía

[1] «Computer History Museum,» 2021. [En línea]. Available: https://www.computerhistory.org/internethistory/.

[2] A. Mohamed, «A history of cloud computing,» 9 April 2018. [En línea]. Available: https://www.computerweekly.com/feature/A-history-of-cloud-computing.

[3] W. Chai, «cloud computing,» December 2020. [En línea]. Available: https://searchcloudcomputing.techtarget.com/definition/cloud-computing.

[4] «Dropbox,» [En línea]. Available: https://www.dropbox.com/es_ES/.

[5] «Google Drive,» [En línea]. Available: https://www.google.com/intl/es_es/drive/.

[6] «Microsoft onedrive,» [En línea]. Available: https://www.microsoft.com/es-es/microsoft-365/onedrive/online-cloud-storage.

[7] «Apple iCloud,» [En línea]. Available: https://www.apple.com/es/icloud/.

[8] «Defining Malware: FAQ,» 04 January 2009. [En línea]. Available: https://web.archive.org/web/20180920112045/https://docs.microsoft.com/en-us/previous-versions/tn-archive/dd632948(v=technet.10).

[9] R. Sihwail, O. Khairuddin y K. A. Zainol Ariffin, «A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis,» *International Journal on Advanced Science Engineering and Information Technology,* 2018.

[10] «Ubuntu,» [En línea]. Available: https://ubuntu.com/download/desktop.

[11] «Docker,» [En línea]. Available: https://www.docker.com/.

[12] «Python,» [En línea]. Available: https://www.python.org/.

[13] «Django website,» [En línea]. Available: https://www.djangoproject.com/.

[14] «¿Qué es una API de REST?,» Red-Hat, [En línea]. Available: https://www.redhat.com/es/topics/api/what-is-a-rest-api.

[15] Flask, "flask.palletsprojects," 2010. [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/.

[16] «MaríaDB,» [En línea]. Available: https://mariadb.org/.

[17] «phpMyadmin,» [En línea]. Available: https://www.phpmyadmin.net/.

[18] «HTML: Lenguaje de etiquetas de hipertexto,» Mozilla, [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTML.

[19] «Bootstrap,» [En línea]. Available: https://getbootstrap.com/.

[20] «JavaScript,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript.

[21] «jQuery,» [En línea]. Available: https://jquery.com/.

[22] «Assemblyline,» 2021. [En línea]. Available: https://cyber.gc.ca/en/assemblyline.

[23] «Networking overview,» [En línea]. Available: https://docs.docker.com/network/.

[24] «Overview of Docker Compose,» [En línea]. Available: https://docs.docker.com/compose/.

[25] N. Babich, «The 12 Do's and Don'ts of Web Design,» 5 Feb 2018. [En línea]. Available: https://xd.adobe.com/ideas/principles/web-design/12-dos-donts-web-design-2/.

[26] «axios,» [En línea]. Available: https://axios-http.com/.

[27] «Docker Hub Ubuntu,» Docker, [En línea]. Available: https://hub.docker.com/_/ubuntu.

[28] «¿Qué es Kubernetes?,» Kubernetes, [En línea]. Available: https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/.

[29] «¿Qué es Elasticsearch?,» [En línea]. Available: https://www.elastic.co/es/what-is/elasticsearch.

[30] «Elastic Stack,» [En línea]. Available: https://www.elastic.co/es/elastic-stack.

[31] «¿Qué es Redis?,» [En línea]. Available: https://aws.amazon.com/es/elasticache/what-is-redis/.

[32] «¿Qué Es Nginx y Cómo Funciona?,» [En línea]. Available: https://kinsta.com/es/base-de-conocimiento/que-es-nginx/.

[33] A. Beardmore, «Uncovering the Environmental Impact of Cloud Computing,» 12 october 2020. [En línea]. Available: https://earth.org/environmental-impact-of-cloud-computing/.

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Thu Jul 01 19:12:55 CEST 2021
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)