



**APB2TAL**

# Índice

<b>ÍNDICE</b>	<b>2</b>
<b>1. INTRODUCCIÓN</b>	<b>5</b>
<b>1.1 PRESENTACIÓN</b>	<b>5</b>
1.1.1 GRUPO	5
1.1.2 MEMORIA	5
<b>1.2 FASES</b>	<b>6</b>
1.2.1 PLANTEAMIENTO FASE 0	6
1.2.2 PLANTEAMIENTO FASE 1	8
1.2.3 PLANTEAMIENTO FASE 2	10
<b>2. PROYECTO APB2TAL</b>	<b>11</b>
<b>2.1 ¿QUE USAMOS Y POR QUÉ?</b>	<b>11</b>
2.1.1 DJANGO	11
2.1.1.1 ¿Qué es un framework?	11
2.1.1.2 ¿Qué es Django?	12
2.1.1.3 ¿Como funciona?	12
2.1.1.4 Arquitectura Modelo-Vista-Template (MTV)	12
2.1.1.5 Comparación Modelo-Vista Controlador (MVC)	13
2.1.1.6 ¿Porque lo usamos?	14
2.1.1.7 ¿Como Aplicamos Django?	15
2.1.2 BASES DE DATOS	16
2.1.2.1 Relacional SQLite	16
2.1.2.2 No Relacional Mongo	17
2.1.3 TECNOLOGÍAS	18
2.1.3.1 Docker	18
2.1.3.2 JavaScript & AJAX	19
<b>2.2 PLAN GENERAL DE LA ESTRUCTURA DEL PROYECTO</b>	<b>20</b>
2.2.1 DIAGRAMAS	20

<b>3. FUNCIONAMIENTO PROYECTO</b>	<b>21</b>
<b>3.1 USUARIOS</b>	<b>21</b>
3.1.1 REGISTRO	21
3.1.2 INICIO DE SESIÓN	21
3.1.3 PERFIL	22
<b>3.2 GRUPOS</b>	<b>22</b>
<b>3.3 ADMINISTRADOR</b>	<b>23</b>
3.3.1 ADMINISTRAR USUARIOS	23
3.3.1.1 Aceptar usuario	23
3.3.1.2 Modificar usuario	23
3.3.1.3 Banear usuario	23
3.3.2 ADMINISTRAR GRUPOS	23
3.3.2.1 Crear Grupo	23
3.3.2.2 Asignar Grupo	24
3.3.2.3 Eliminar Grupo	24
3.3.3 DJANGO ADMIN	24
<b>3.4 ARCHIVOS</b>	<b>24</b>
3.4.1 SUBIR ARCHIVO	25
3.4.1.1 Hashear	25
3.4.1.2 Preguntar en la BD	25
3.4.2 ANALIZAR ARCHIVO	25
3.4.2.1 API Virus Total	25
3.4.2.2 Guardar en BD	26
3.4.2.3 Encriptar	27
3.4.3 RESULTADO ARCHIVO	27
3.4.3.1 Resultado	27
3.4.3.2 Descargar	27
3.4.3.3 Eliminar	27
<b>3.5 COMPARTIR</b>	<b>28</b>
3.5.1 COMPARTIR A USUARIO	28
3.5.2 COMPARTIR A GRUPO	28
<b>3.6 LOGS</b>	<b>29</b>
3.6.1 LOGS USUARIO	29
3.6.2 LOGS ADMINISTRATIVOS	29
<b>3.7 CONTACTA</b>	<b>29</b>
3.7.1 ENVIAR MENSAJE A USUARIO	30
3.7.2 ENVIAR MENSAJE ADMINISTRADOR	30
3.7.3 BUZÓN	30
3.7.4 LLAMAR	30

<b>4. ESTADO ACTUAL – REQUISITOS</b>	<b>31</b>
<b>4.1 REQUISITOS DEL PROYECTO</b>	<b>31</b>
4.1.1 CUMPLIDO	31
<b>4.2 APLICACIONES PROPIAS</b>	<b>33</b>
4.2.1 SERVICIO DE CONTACTO	33
4.2.2 ALTA SEGURIDAD	33
4.2.3 CUSTOMIZACIÓN	34
4.2.4 PRODUCCIÓN	34
<b>4.3 APLICACIONES FUTURAS</b>	<b>35</b>
4.3.1 ¿DEUDA TÉCNICA?	35
4.3.2 CONTROL DE USUARIOS	35
<b>5. SEGURIDAD</b>	<b>37</b>
<b>5.1 AUDITORIA WEB</b>	<b>37</b>
5.1.1 PRIMERA AUDITORIA	37
5.1.2 SEGUNDA AUDITORIA	37
5.1.3 COMPARACIÓN - CONCLUSIONES	38
<b>5.2 APLICACIONES DE SEGURIDAD</b>	<b>38</b>
5.2.1 SEGURIDAD APLICADA	38
5.2.1.1 Validación y Sanitización de Datos	38
5.2.1.2 Configuración de Seguridad en Django	38
5.2.2 PFSense	39
<b>6. FINAL</b>	<b>40</b>
<b>6.1 CONCLUSIONES</b>	<b>40</b>
<b>6.2 WEBGRAFÍA</b>	<b>41</b>

# 1. Introducción

## 1.1 Presentación

Proyecto del curso segundo y último año de administración de sistemas y redes con perfil de ciberseguridad. Montar un servicio web, con la capacidad de tener una gestión de usuarios, grupos, análisis de archivos con una Api externa, como núcleo principal, y más

Se empezó a mitades de octubre del 2023 y concluye el 30 de mayo de 2024. Aproximadamente 9 meses para hacer el proyecto, 3 integrantes por equipo.

### 1.1.1 Grupo

El grupo ha sido formado por:

- Albert Xicola Sánchez

Organizador y programador del proyecto.

- Pau Cañadillas Isern

Responsable de ciberseguridad e infraestructura.

- Max Thomas Garín

Recopilación de información y gestor de bases de datos.

Mas información de los integrantes en [Información Memoria](#)

### 1.1.2 Memoria

La memoria del proyecto se ha dividido. Por una parte, tenemos, la memoria en PDF, esta memoria explica como se ve en el índice, el trabajo en las fases, el proyecto en si mismo y toda la parte técnica, junto a análisis de ciberseguridad, y lo que se tiene entre manos hablando sobre el estado actual del proyecto.

Por otra parte, tenemos una página web, generada con GitHub-Pages, que explicamos al detalle, tanto la seguridad del *pfsense*, como su infraestructura, la puesta de producción del proyecto , también se copia el apartado de seguridad y la webgrafía.

Lo que se ha querido tratar con el hecho de publicar la memoria de manera web, es hacer llegar a mas gente nuestro trabajo, y también ver de una manera como se tratan las memorias de los proyectos a gran escala o en entornos reales de trabajo.

[Memoria clic aquí](#)

## 1.2 Fases

Como se ha dicho antes, el Proyecto empezó a mitades de octubre después del mini proyecto, lo cual se han trabajado aproximadamente 9 meses desde su inicio, con un total de 3 participantes.

El proyecto ha constado de 3 etapas, Fase 0, Fase 1 y Fase 2, nosotros de alguna manera también subdividimos las fases o más bien le asignamos un nombre, debido a los grandes cambios que se han realizado.

*\*Aquí se cuenta de una manera mas amigable como se ha trabajado el proyecto.*

### 1.2.1 Planteamiento Fase 0

La primera fase, Constaba del planteamiento del proyecto web con algunas especificaciones:

- Crear un servidor web. (libre elección)
- Crear una base de datos relacional. (libre elección)
- Crear una base de datos no relacional. (libre elección)
- Crear mockups de la página web con las siguientes

Se empezó desde un principio entre los 3 integrantes de organizar el trabajo, y buscar información acerca de cómo hacerlo y el planteamiento.

Pau Cañadillas -> información acerca de las APIS

Max Thomas -> información acerca de desarrollo web en PHP

Albert Xicola -> En busca de gestión de proyectos web

En primera instancia cada uno iba un poco por libre, cada uno investigando con sus correspondientes partes del trabajo

Se buscó información acerca de la API que íbamos a trabajar y el funcionamiento y trato de la misma (Api Virus Total)

En su momento se hicieron pruebas con Python ( apoyado con las pruebas del pre-proyecto) para la API de virus total, que fue enviar y recibir peticiones de archivos, analizados.

Por otra parte, conseguir implementar dichas peticiones y respuestas en un entorno PHP para la incorporación en una web.

Y por último la búsqueda de tecnologías web para el desarrollo del servidor.

Estas fueron las primeras semanas tras un no muy buen planteamiento.

Buscando tecnologías en el mercado nos topamos con REACT, la cual fue nuestra primera base de implementación de servidor web.

REACT - PHP

*React* te ayuda a crear interfaces de usuario interactivas de forma sencilla. Diseña vistas simples para cada estado en tu aplicación, y *React* se encargará de

actualizar y renderizar de manera eficiente los componentes correctos cuando los datos cambien.

La implementación fue bastante tardía, duradera y finalmente fallida. Pero esto se comentará más adelante.

El integrante con el objetivo de búsqueda de tecnologías trató de aprender el lenguaje de REACT durante unas semanas, no obstante, tras un corto periodo de vida, tuvimos que dar un giro de los acontecimientos.

El PHP no nos gustaba, el problema del PHP es que es “antiguo”. Lo que entiendo por antiguo en este contexto es que actualmente hay muchas más opciones de herramientas para desarrollo web en el mercado que les dan mil vueltas a PHP.

Tras 2 semanas más a mitades de noviembre, nos topamos con *Django*. La herramienta prodigio de nuestro proyecto. Una tecnología que le da mil vueltas a PHP en el mundo de los servidores web.

Estamos en la fase 1, nuestra tercera etapa.

Volver a empezar...

Nuestro reto fue crear un proyecto con la unión de muy buenas tecnologías del mundo de las webs, REACT + DJANGO.

Así que tras 1 mes más seguimos tratando de crear nuestro proyecto.

*React* por una parte como *frontend*, *Django* trabajando como *backend*, *mongodb* y *mariadb* como bases de datos NoSQL y SQL respectivamente.

Tras largo estudio, y pruebas con la tecnología, nos topamos con otro problema.

REACT, dado el tiempo dado para la creación del proyecto y la no tanta capacidad para asumir esta herramienta relativamente compleja, tuvimos que desistir y afrontar el proyecto de otra manera.

En la primera fase del proyecto, tuvimos que afrontar los siguientes puntos:

- Crear un servidor web -> Django
- Crear una base de datos relacional -> SQLite (proporcionada por Django)
- Crear una base de datos no relacional -> MongoDB
- 

Estos fueron los requisitos de la fase 0:

- Autenticación de usuarios.
- Creación de usuarios.
- Subir archivos y carpetas.
- Verificar los archivos con VirusTotal.
- Mostrar los resultados.

Aparentemente teníamos todos los puntos muy logrados, no obstante, venían con muchos fallos, seguridad, visualización de datos, bugs y demás, a nuestro parecer no obtuvimos mucha deuda técnica.

Simplemente había que pulir los requisitos, como por ejemplo mostrar debidamente las respuestas JSON.

### 1.2.2 Planteamiento Fase 1

Esta fase, fue a nuestro parecer la más compleja, la incorporación de requisitos más allá de nuestros conocimientos dado los requisitos:

- Compartir los archivos con usuarios o departamentos.
- Registro de archivos subidos por el usuario y eliminados por contener software malicioso.
- Descarga de archivos.
- Panel administrativo donde validar un usuario y asignarlo a un departamento.
- Registro de todos los archivos subidos y eliminados por los usuarios.

A partir de este punto, vimos que esto iba y podría ser algo grande, un poco tarde, pero empezamos a utilizar github.

#### GitHub

Es una plataforma online de desarrollo de software que se usa para almacenar, supervisar y trabajar con proyectos de software. Facilita el intercambio de archivos de código y trabajar en proyectos colaborativos de código abierto.

Nosotros empezamos a usar con mucha intensidad esta herramienta, no solo para el transporte y guardado del archivo, sino porque podíamos gestionar los *deadlines*, tiempos, tareas, documentar errores, que se había hecho y mucho más, gracias al apartado *projects* de *github*.

A partir de este momento el trabajo era más versátil, cada uno tenía pequeñas tareas a realizar y datos que aportar a la documentación.

No solo planificar tareas, sino como he dicho antes, los integrantes tuvimos que, a aprender a usar esta herramienta de manera para guardar los archivos, saber usar *commits*, crear ramas y en general poder guardar de manera rápida ordenada y retráctil, datos o caminos donde se embarcaba el proyecto o partes de este.

Terminado el uso e implementación de la herramienta, seguimos con el proceso del proyecto.

Hasta este punto ya en febrero, seguimos en la búsqueda y “tecleo” de código del proyecto, nos repartimos el trabajo con lo que mejor se le daba a cada uno.

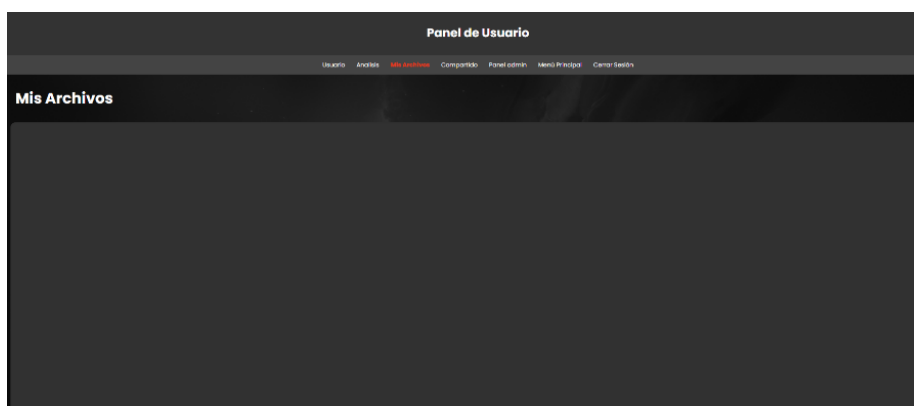
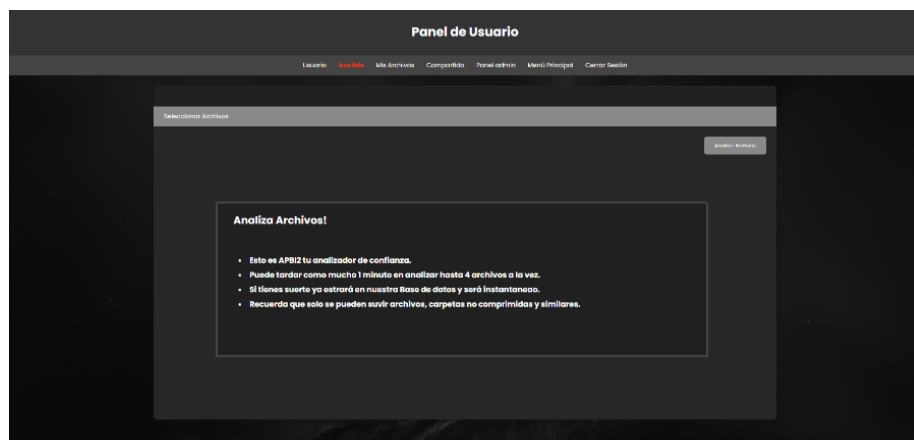
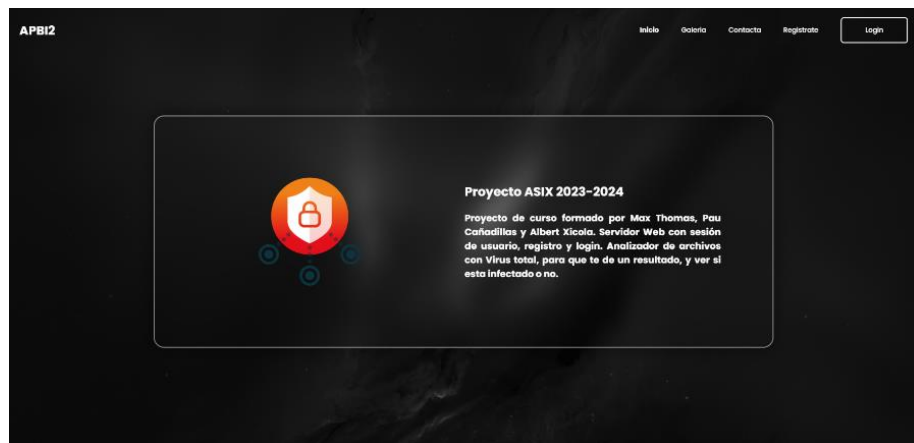
Pudimos realizar los puntos pedidos, excepto algunos, Registros de los archivos eliminados (logs en término corto), un panel administrativo, aparte de el de *Django* predefinido y a pesar de cumplir con el resto, aun así, el proyecto no nos estaba gustando.



¿Funcionaba? **Si**. ¿A la perfección sin ningún tipo de error? **No**.

Tan poco tiempo sin dedicación a *Django*, sin conocimiento previos, sin una buena estructura de proyecto anterior, el proyecto se podía ir a pique, y quedaban pocos días para la entrega. Así que, tras unos días, se entregó ( con una buena presentación, todo se tiene que decir) un buen pre-proyecto sinceramente, presentación, demo y demás muy limpio, aunque todo bajo la supervisión del equipo, sabiendo donde había fallos, fugas, mejoras y sobre todo con deuda técnica.

Aquí se puede apreciar algunas páginas de el proyecto en la Fase .



### 1.2.3 Planteamiento Fase 2

Última fase.

¿Que nos queda?

- Aplicar un cifrado de archivos mientras el archivo ha sido verificado, pero no descargado.
- Aplicar seguridad a la red, configurando un cortafuegos como *pfSense* delante del servidor.
- Auditar la web en busca de vulnerabilidades.
- Pensar y aplicar un par de mejoras al producto.
- + Deuda técnica de la fase 1

Llegados a este punto, y haciendo un pensamiento positivo, y un buen *feedback* se nos ocurre la grandiosa idea de volver a empezar de 0.

Solo nos llevamos una cosa, el conocimiento adquirido.

¿Por qué?

Teníamos 2 grandes motivos.

1. Como se dice al final de la fase 1, se comenta que el proyecto estaba cogido con pinzas, y así era, las tablas no estaban correctamente planteadas, como el uso del propio *django* y sus funcionalidades que nos ofrecía. Sabíamos perfectamente que no estaba correcto, y que las bases estaban mal, eran andamios mal colocados para el avance del proyecto, vivíamos una mentira que todos conocíamos, pero nadie quería decir en voz alta.
2. El efecto interacción-usuario, pero mal (hemos patentado el efecto). ¿Qué queremos decir con esto? Un usuario al interaccionar con una página web, no solo debe ser visualmente agradable, también debe tener una serie de herramientas para el uso objetivo de la página y la manipulación de esta. Esto es un punto importante, porque lo cambia todo, uno no puede empezar a creer que le puede venir bien al usuario, hay unas reglas no escritas, unos tips de usuario y más factores que nos hicieron decidir que cambiar la interfaz era algo casi necesario.

Además de los motivos por los cuales tomamos la decisión, sabíamos o simplemente creíamos, que, llegados a este punto, tendríamos los conocimientos necesarios para abordar la situación y la recreación del nuevo proyecto con mejores bases.

En esta fase final el proyecto lo hemos abordado con gran rapidez, y con un acabado muy bueno, con una gran mejora en todos los aspectos.

- Control de errores
- Velocidad y mejora de rendimiento
- Gestión de usuarios, grupos y archivos
- Manipulación de datos
- Buena interacción
- Bases estables, y menos fallos

Terminamos con todos los requisitos agregando incluso funcionalidades extra.

En esta parte, también decidimos la implementación de JavaScript y AJAX.

JavaScript, es el lenguaje de programación que debes usar para añadir características interactivas al sitio web.

AJAX, permite a las aplicaciones web validar información específica en formularios antes de que los usuarios los envíen. Por ejemplo, cuando un nuevo usuario crea una cuenta, la página web puede verificar automáticamente si hay un nombre de usuario disponible antes de que el usuario pase a la siguiente sección. Gracias a esto también hemos enfatizado en la configuración de *Django* y el uso de buenas prácticas del lenguaje.

Aquí también se han aplicado medidas de seguridad extras y reforzar las que se controlaban poco. Implementación del *pfsense*, puesta a producción y análisis de ciberseguridad.

## 2. Proyecto APB2TAL

### 2.1 ¿Que usamos y por qué?

#### 2.1.1 Django

En el proyecto sea usado principalmente como se ha visto en las fases el framework de Django, que es lo que consideramos como lo principal del proyecto.

Explicación .

##### 2.1.1.1 ¿Qué es un framework?

Imagina que quieres construir una casa. No empiezas de cero, cortando los árboles para hacer madera y fabricando cada ladrillo. En lugar de eso, utilizas herramientas y materiales ya preparados, como ladrillos, cemento, y planos de construcción. Un framework es algo similar, pero para desarrollar software.

Un framework es un conjunto de herramientas y componentes predefinidos que te ayudan a crear aplicaciones de una manera más fácil y rápida. En lugar de escribir todo el código desde cero, puedes usar estos componentes ya hechos y enfocarte en las partes específicas de tu aplicación. Es como si el framework te diera las bases y las estructuras necesarias para construir tu proyecto, ahorrándote tiempo y esfuerzo. Y nosotros en el proyecto usamos el *framework* de *Django*

#### 2.1.1.2 *¿Qué es Django?*

Django es un framework específico para desarrollar aplicaciones web, escrito en el lenguaje de programación Python. Nos ha dado muchas herramientas muy completas y eficientes que nos ha ayudado a construir el sitio web y aplicaciones web de manera rápida y segura.

#### 2.1.1.3 *¿Como funciona?*

Django utiliza una arquitectura conocida como Modelo-Vista-Template (MTV), que está inspirada en el patrón Modelo-Vista-Controlador (MVC). A continuación, te explico en detalle qué es la arquitectura MTV y cómo se compara con MVC.

Mas adelante se explica las aplicaciones de django.

Las aplicaciones de Django funcionan como módulos independientes dentro de un proyecto de Django, permitiendo una estructura organizada y modular del código.

#### 2.1.1.4 *Arquitectura Modelo-Vista-Template (MTV)*

En Django, la arquitectura MTV se descompone en tres componentes principales:

- Modelo (Model - Class):

El modelo es responsable de la lógica de datos y la definición de la estructura de la base de datos. Representa las tablas de la base de datos y proporciona una interfaz para interactuar con los datos.

En Django, los modelos se definen como clases de Python que heredan de `django.db.models.Model`.

- Vista (View):

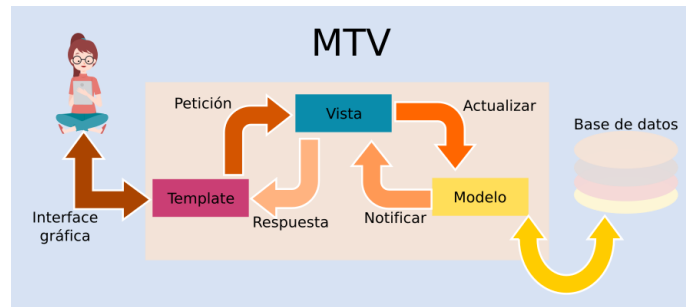
La vista contiene la lógica que maneja las solicitudes del usuario y devuelve respuestas apropiadas. No se encarga directamente de la presentación de los datos, sino de procesar las solicitudes y coordinar la lógica de la aplicación.

En Django, las vistas se definen como funciones o clases en el archivo `views.py`.

- Plantilla (Template):

La plantilla se encarga de la presentación de los datos. Define cómo se mostrarán los datos al usuario final, utilizando un lenguaje de plantillas que permite insertar variables y lógica simple en HTML.

En Django, las plantillas se escriben en archivos HTML que pueden incluir etiquetas y filtros del lenguaje de plantillas de Django.



#### 2.1.1.5 Comparación Modelo-Vista Controlador (MVC)

El patrón MVC es un paradigma de diseño que separa la aplicación en tres componentes principales: Modelo, Vista y Controlador. Aquí está la comparación con MTV en Django:

- Modelo (Model):

Es similar en ambas arquitecturas. En MVC y MTV, el modelo maneja la lógica de datos y la interacción con la base de datos.

- Vista (View):

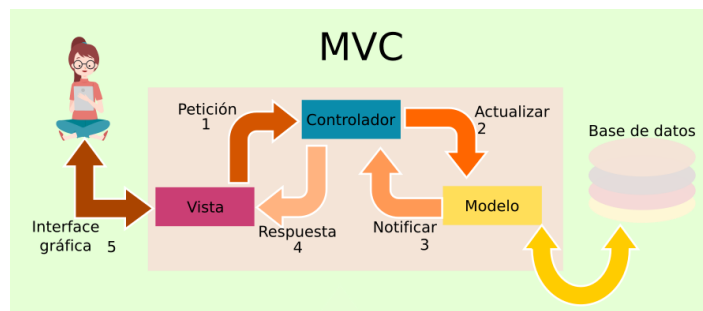
En MVC, la vista se encarga de la presentación de los datos, es decir, del HTML que se envía al cliente.

En MTV, la vista es responsable de la lógica que maneja las solicitudes y coordina las respuestas, pero no genera directamente el HTML. Esta tarea recae en las plantillas.

- Controlador (Controller):

En MVC, el controlador maneja la lógica de la aplicación, procesa las entradas del usuario y selecciona las vistas a mostrar.

En MTV, la función del controlador está dividida entre la vista (que maneja la lógica de la aplicación y las solicitudes) y las plantillas (que se encargan de la presentación).



En resumen, la vista en el patrón MTV de Django combina aspectos del controlador y la vista del patrón MVC, mientras que las plantillas en MTV se encargan exclusivamente de la presentación.

### Diferencia Clave

La diferencia principal entre MVC y MTV está en la terminología y en el enfoque:

- Controlador (MVC) vs. Vista (MTV):

En MVC, el controlador gestiona la lógica de las solicitudes y determina qué vista se debe mostrar.

En MTV, la "vista" (en Django) cumple el rol del controlador de MVC.

Maneja las solicitudes, realiza la lógica de negocio, y selecciona la plantilla (vista en MVC) para renderizar la respuesta.

- Vista (MVC) vs. Plantilla (MTV):

En MVC, la vista es responsable del renderizado de la UI.

En MTV, esta responsabilidad recae en las plantillas.

#### 2.1.1.6 ¿Porque lo usamos?

Popularidad:

En el momento de plantear el proyecto, supimos en primera instancia, que decidir una base en la que montar el proyecto sería una decisión crucial, lo que teníamos claro era la de no usar el viejo y obsoleto PHP, no solo por la mala calidad de manejo de errores, sino, por el hecho dicho antes de que es algo ya que no se suele usar.

Por ello tras investigar varias opciones las cuales uno de sus requisitos era, que fuera popular, decidimos usar django, top 5 en frameworks más usados en desarrollo web.

Django es un potente framework de desarrollo web que facilita la creación de aplicaciones web robustas, seguras y escalables. Proporciona todas las herramientas necesarias para que puedas enfocarte en lo que hace única a tu aplicación, dejando las tareas complejas y repetitivas en manos del framework.

Algunas de sus Características:

- Protecciones Integradas:

Django incorpora medidas de seguridad avanzadas que protegen las aplicaciones contra amenazas comunes, como inyecciones de SQL, cross-site scripting (XSS), y cross-site request forgery (CSRF).

- Desarrollo Rápido:

Django proporciona una estructura bien definida y herramientas preconstruidas que permiten a los desarrolladores avanzar rápidamente desde la idea hasta el producto terminado. Esto incluye un sistema de plantillas para HTML, un ORM para la gestión de bases de datos, y muchas más otras funcionalidades listas para usar cuando se necesiten.

- Manejo de Alto Tráfico:

Django es utilizado por grandes empresas que necesitan manejar un alto volumen de tráfico web, como Instagram, Pinterest, Disqus etc.... Su arquitectura modular permite que las aplicaciones crezcan y se adapten a medida que aumentan las demandas.

#### 2.1.1.7 ¿Como Aplicamos Django?

##### Aplicación de django en nuestra web

En nuestra web django es lo principal, dada la información anterior, se deduce que django lo usamos tanto de frontend como de backend.

Django funciona mediante aplicaciones, componentes.

Las aplicaciones en Django son componentes modulares que te permiten organizar tu proyecto en partes funcionales. Cada aplicación es una entidad autónoma que puede incluir modelos, vistas, plantillas, archivos estáticos y otros elementos necesarios para cumplir con una funcionalidad específica. Esto facilita el desarrollo, mantenimiento y escalabilidad de proyectos web. Aquí te explico cómo funcionan las aplicaciones en Django y cómo puedes trabajar con ellas.

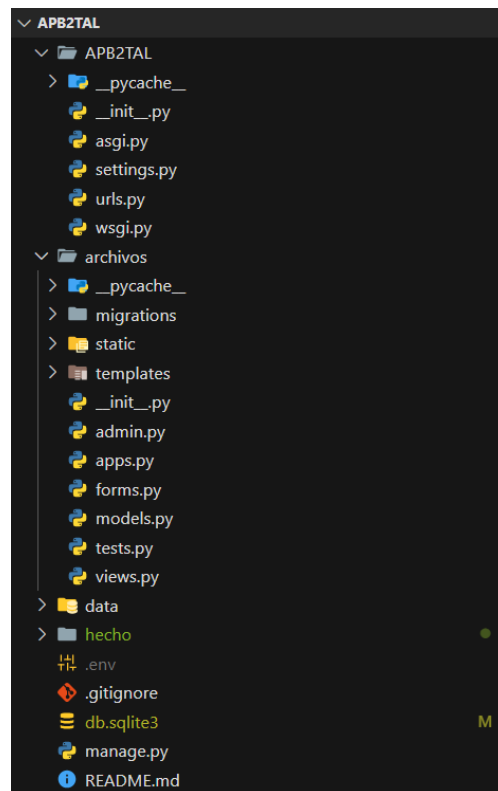
##### Estructura Básica de un Proyecto Django

Un proyecto Django puede contener una o varias aplicaciones. La estructura básica de un proyecto Django con una aplicación se ve así en el caso de nuestro proyecto.

```

APB2TAL/      <- Proyecto
  manage.py   <- Arrenque y núcleo de la aplicación
  APB2TAL/    <- Carpeta de datos del proyecto
    __init__.py <- inicializador python
    settings.py <- Configuración general de proyecto
    urls.py    <- Gestion de rutas
    wsgi.py    <- Punto de entrada para los servidores web compatibles
  Archivos/   <- Aplicacion (solo hay una, archivos que están los datos web)
    __init__.py <- inicializador Python de la app
    admin.py   <- información de la ruta de administración
    apps.py    <- Contiene la configuración de la aplicación
    models.py  <- Modelos de la base de datos que cumple django
    tests.py   <- se utiliza para escribir pruebas (tests) automatizadas
    views.py   <- vistas que manejan las solicitudes del user y devuelven reptas
    migrations/ <- gestionar los cambios en la estructura de la base de datos
    __init__.py <- identificar los directorios que contienen código
    templates/ <- donde se almacenan las plantillas HTML utilizadas
  
```

Esta sería la estructura básica del proyecto, luego vamos a entrar en detalles, y todas las demás funciones y rutas que hay más



Aquí muestro cual sería la estructura final del proyecto, se muestra como a parte de la estructura básica existen otros archivos y datos que hacen el funcionamiento para el objetivo de la web, más adelante enfatizaremos en el código.

## 2.1.2 Bases de datos

### 2.1.2.1 *Relacional SQLite*

#### SQLite como Base de Datos Predeterminada en Django

SQLite es una biblioteca de base de datos relacional escrita en C que implementa un motor de base de datos SQL ligero, autónomo, de servidor cero. Django utiliza SQLite como base de datos predeterminada para nuevos proyectos. Esta elección se debe a varias razones:

- Facilidad de Uso

SQLite es simple de configurar y no requiere una configuración de servidor de base de datos separado. La base de datos se almacena en un único archivo, lo que facilita su distribución y uso en entornos de desarrollo y pruebas.

- Bajo Requisito de Recursos

SQLite es ligero y eficiente en cuanto a recursos, lo que lo hace adecuado para proyectos pequeños o medianos, o para aplicaciones que no requieren una gran cantidad de usuarios concurrentes o características avanzadas de base de datos.



- Portabilidad

Almacenar la base de datos en un solo archivo facilita la portabilidad de la aplicación, ya que la base de datos y la aplicación pueden moverse juntas sin necesidad de configuraciones adicionales.

En nuestro caso usamos dicha BD, para gestionar múltiples datos.

Tenemos un total de 16 tablas de las cuales 8 son las que se usan constantemente.

Django por defecto ya tiene un sistema de usuarios y grupos, que es una ventaja y una facilidad que hemos usado para la web.

<b>archivos_adquisicion</b>	Tabla para asociar archivo con usuario
<b>archivos_archivo</b>	Información de los archivos registrados
<b>archivos_compartido</b>	Asociar los archivos de un usuario a usuario compartido
<b>archivos_mensajes</b>	Datos de mensaje de usuario a usuario
<b>archivos_customuser</b>	Usuarios del proyecto
<b>archivos_customuser_group</b>	Usuarios asociados a grupos
<b>archivos_groupdescription</b>	Tabal únicamente para la descripción del grupo
<b>auth_group</b>	Grupos de el proyecto

Estas son las tablas que se utilizan de normal las 7 primeras con el nombre de *archivo\_* por delante son tablas creadas por nosotros. Las demás como antes se ha mencionado son tablas generadas por django (*auth\_*) . Las 8 tablas restantes, también son generadas por django y en algún caso de alguna librería instaladas, que no se usan de normal, pero van bien tenerlas para un futuro y/o para el correcto funcionamiento.

Usar una BD relacional para estos datos es una decisión muy acertada ya que los datos tratados mayoritariamente se necesitan una relación o al menos tratan con una relación para el funcionamiento, por eso tiene sentido usar una bd relacional , están diseñadas para almacenar datos de manera estructurada y organizada. Ofrece una serie de ventajas clave en términos de estructura de datos, integridad, flexibilidad, confiabilidad, escalabilidad y cumplimiento normativo, lo que la convierte en una opción preferida para una amplia gama de aplicaciones web y empresariales.

Más adelante explicaremos el funcionamiento de cada tabla en producción.

#### 2.1.2.2 *No Relacional Mongo*

MongoDB, qué es, ¿cómo funciona y por qué la usamos?

MongoDB es un sistema de gestión de bases de datos NoSQL (Not Only SQL) que se ha vuelto muy popular en los últimos años debido a su flexibilidad y escalabilidad. A diferencia de las bases de datos relacionales tradicionales, que almacenan datos en tablas con esquemas predefinidos, MongoDB utiliza un

enfoque de almacenamiento de datos basado en documentos JSON (JavaScript Object Notation) que se almacenan en colecciones.

MongoDB utiliza un modelo cliente-servidor en el que los clientes se comunican con un servidor de MongoDB para acceder y manipular datos.

En nuestro caso esta BD la usamos para el registro de logs.

MongoDB permite almacenar datos con un esquema dinámico, lo que significa que cada registro de log no necesita cumplir con un formato predefinido. Esto facilita la captura de diferentes tipos de registros, incluso si varían en estructura o contenido.

Mientras que las bases de datos relacionales pueden ser adecuadas para ciertos tipos de datos y aplicaciones, las bases de datos NoSQL, como MongoDB, ofrecen ventajas significativas en términos de flexibilidad, escalabilidad, velocidad de escritura y operaciones de consulta para almacenar y analizar logs en entornos modernos de desarrollo de aplicaciones.

La aplicación en el proyecto es simple, en concretos movimientos de datos, Django captura dicho movimiento, y lo envía a la BD de mongo para que genere un registro de la información y posteriormente presentarlo al usuario como registro,

Generamos lo que se llama una colección de datos, sin relación ninguna entre sí, en formato *JSON*, y se guardan todos los datos en la misma colección para luego poder obtenerlos y presentarlo al usuario:

Ejemplo de cómo un usuario analiza un archivo, en la colección Logs, se ve un registro de el usuario admin de cómo ha analizado un archivo, con los datos ya de una base de datos relacional (usuario y archivo). El texto que se ve es un texto plano ya tratado en la obtención de la información que se guarda.



## 2.1.3 Tecnologías

### 2.1.3.1 Docker

¿Qué es? ¿Cómo lo usamos y por qué?

Docker es una plataforma de software que simplifica el desarrollo, implementación y ejecución de aplicaciones mediante la utilización de contenedores. Los contenedores son entornos ligeros y portátiles que incluyen todo lo necesario para ejecutar una aplicación, como el código, las bibliotecas, las dependencias y las variables de entorno. Docker proporciona una forma estandarizada de

empaquetar, distribuir y ejecutar aplicaciones, lo que facilita la creación de entornos de desarrollo y despliegue consistentes y reproducibles en diferentes sistemas operativos y plataformas de infraestructura.

Docker lo utilizamos principalmente por varias razones importantes que mejoran significativamente el rendimiento del servidor web y el despliegue de aplicaciones. Razones clave por las que Docker es utilizado:

#### 1. Consistencia del Entorno de Desarrollo:

Docker proporciona un entorno de desarrollo consistente y reproducible para todos los que quieran dicho servicio. Los desarrolladores pueden ejecutar la misma configuración de software y dependencias en sus máquinas locales que en los entornos de producción, lo que reduce los problemas de compatibilidad y facilita la colaboración.

#### 2. Portabilidad de Aplicaciones:

Docker permite empaquetar una aplicación y todas sus dependencias en un contenedor ligero y portátil. Estos contenedores pueden ejecutarse en cualquier sistema operativo que tenga Docker instalado, lo que garantiza la portabilidad de las aplicaciones entre diferentes entornos de desarrollo y producción.

El uso en concreto es para el despliegue de MongoDB, y en un futuro tal vez el servidor entero, para el manejo de datos, y el despliegue del servidor de mongodb.

Se nos es útil para desplegar el servidor, y no tratar con una instalación y migración de datos. Docker nos ha proporcionado facilidad, en el hecho de poder trabajar con más velocidad en todo momento.

##### 2.1.3.2 *JavaScript & AJAX*

AJAX (Asynchronous JavaScript and XML) es una técnica de desarrollo web que permite a las aplicaciones web enviar y recibir datos de un servidor de manera asíncrona (en segundo plano) sin recargar la página web completa. Esto mejora la interactividad y la experiencia del usuario, ya que partes de la página pueden actualizarse de manera dinámica.

La implementación de esta técnica, de desarrollo web, ha sido un acierto, es importante que los proyectos estén al día de las tecnologías, para no quedarse atrás. El Ajax se ha aplicado como se dijo en las fases anteriores para comunicar la base de datos con la interacción del usuario, se ha aplicado solamente en medidas de seguridad.

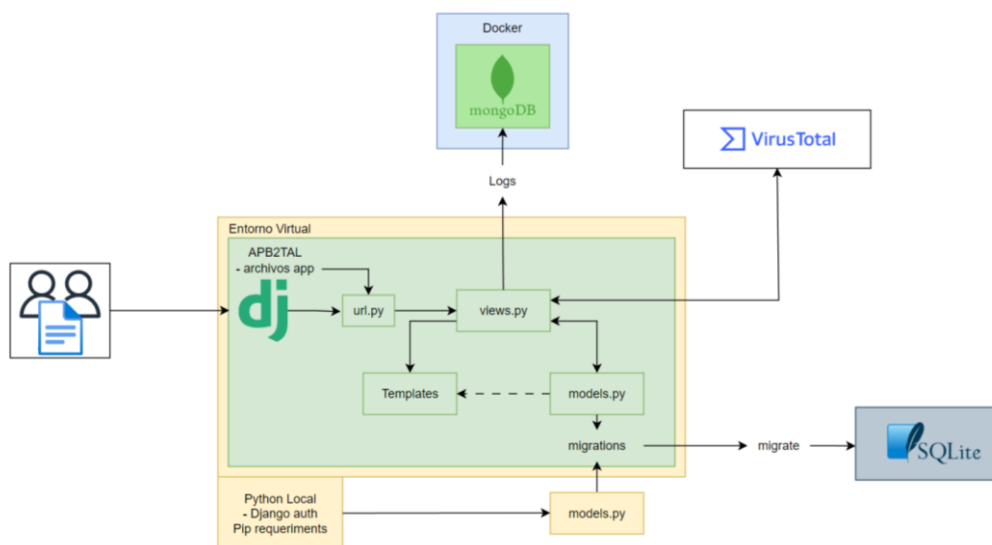
JavaScript es un lenguaje de programación interpretado y de alto nivel, que se ejecuta en el lado del cliente en los navegadores web. Es uno de los componentes

centrales de la tecnología web, junto con HTML y CSS. Y por ello hemos querido aplicarlo, creemos que la interacción con el usuario es la primera impresión que se da, y que por ello tras aprender e investigar se usa constantemente el JavaScript lo usamos en casi todas las vistas.

## 2.2 Plan general de la estructura del proyecto

### 2.2.1 Diagramas

Aquí se puede ver como estaría estructurado el proyecto a nivel tecnológías



Explicación:

Aquí voy a entrar un poco en terreno del funcionamiento, ha faltado explicar, si no se ha comentado ya, que de normal se ha estado trabajando sobre un entorno virtual de Python, con sus librerías para evitar problemas de incompatibilidad con dispositivos

Se crea el entorno e instala todo, tenemos el Django como núcleo que conecta con ambas BD.

Mongo como bien se ha explicado proviene de un contenedor de Docker que desde las vistas genera los logs y se comunica. Y, por otra parte, tenemos SQLite.

Se puede ver que las migraciones para ponerlas en la BD vienen de dos lados por una parte las clases creadas manualmente, (tablas archivos\_) y por otra parte cuando creamos las migraciones, también vienen de las clases de la librería de Django del entorno virtual y de allí nacen las tablas que se ven en la base de datos.

Por último, tenemos la conexión con la API de virus total, que se crea cuando se necesita enviar algo.

## 3. Funcionamiento Proyecto

### 3.1 Usuarios

#### 3.1.1 Registro

El usuario ingresa las credenciales del formulario default de Django username, nombre, apellido, correo y contraseña uno y dos para confirmar (las cuales se hashean).

Los datos que se ingresan se registran en la tabla `archivos_customuser`, que es una tabla generada por la migración de la clase `auth_user` (Default de django) junto a la clase `CustomUser`, esta combinación es para agregar campos a mas de los que ofrece django. Agregar campos o modificar clases lo contempla django.

La función llama al formulario de la clase y crea el usuario haciendo comprobaciones necesarias de seguridad.

Función: [clic aquí](#)

Clase: [clic aquí](#)

Formulario: [clic aquí](#)

#### 3.1.2 Inicio de sesión

El usuario ingresa las credenciales existentes, la función comprueba , si existen entra en su perfil, inicia sesión con username y contraseña.

Función: [clic aquí](#)

Formulario: [clic aquí](#)

Tras registrarte e iniciar sesión, el usuario no puede hacer nada, tan solo contactar con administración y visualizar su perfil. En el perfil se ve un mensaje de “esperando a que el administrador te acepte en el sistema. Y eso también es un campo del usuario el `is_accepted`, que mientras este en false, el usuario no podrá hacer gran cosa. Esta función es especial porque la misma función se puede aplicar como un tag a las demás funciones que les bloquee el acceso de los usuarios que no estén aceptados.















Función: [clic aquí](#)

### 3.1.3 Perfil

Estando en el perfil, el usuario puede editar sus credenciales, username, nombre, apellido, correo, contraseña y edad, cada vez que quiere editar los datos, con el id de usuario actual hace un post a sus datos nuevos para actualizar.

Función: [clic aquí](#)

La tabla de usuario:

	Name	Data type	Clave Primaria	Foreign Key	Único	Check	No es NULL	Collate	Generated
1	id	integer							NULL
2	password	varchar (128)							NULL
3	last_login	datetime							NULL
4	is_superuser	bool							NULL
5	username	varchar (150)							NULL
6	first_name	varchar (150)							NULL
7	last_name	varchar (150)							NULL
8	email	varchar (254)							NULL
9	is_staff	bool							NULL
10	is_active	bool							NULL
11	date_joined	datetime							NULL
12	is_accepted	bool							NULL
13	edad	integer							NULL

Id	Identificador usuario
password	Contraseña usuario
Last_login	Ultimo inicio de sesión de usuario
is_superuser	Si el usuario puede tener permisos especiales o no
username	Nombre de usuario
first_name	Nombre
last_name	Apellido
email	Correo electrónico
is_staff	Administrador o no
is_Active	Activo en el sistema (ban)
date_joined	Fecha de registro
is_ascepted	Si esta aceptado o no
edad	Edad del usuario

## 3.2 Grupos

La tabla del grupo es *auth\_groups* que tiene id y nombre, más una clase de la app de django que agrega descripción.

Cada usuario puede ver en que grupo esta y que usuarios están también en el grupo.

Función#1: [clic aquí](#)

Función#2: [clic aquí](#)

## 3.3 Administrador

El administrador es aquel usuario con poder sobre los usuarios, grupos, logs y mensajes, al menos cuando hablo del atributo *is\_staff*. Gracias a este atributo, ha resultado más fácil manejar el acceso o no las rutas administrativas y funciones que solo debería poder hacer el administrador.

Función: [clic aquí](#)

### 3.3.1 Administrar usuarios

Función: [clic aquí](#)

#### 3.3.1.1 *Aceptar usuario*

El administrador tiene la tarea de aceptar y gestionar el comportamiento de los usuarios, por ello también puede y debe aceptar a los usuarios registrados según el criterio,

#### 3.3.1.2 *Modificar usuario*

El administrador como se ha comentado antes tiene la capacidad de administrar usuarios, incluso pudiendo cambiar valores de este, como nombre, apellido, username y correo electrónico.

#### 3.3.1.3 *Banear usuario*

Puede banear (*is\_active*) a los usuarios, y puede otorgar permisos especiales incluso hacer administrador.

### 3.3.2 Administrar Grupos

#### 3.3.2.1 *Crear Grupo*

El administrador también tiene la capacidad de crear y gestionar grupos. De hecho, es el único que puede crear el grupo, agrega nombre y descripción y listo.

Por una parte, el nombre e id de grupo se guardan en la tabla default de Django (*auth\_group*) mientras que la descripción y la clave que pasa a ser forane de *auth\_griup* al ser campo agregado se va a la clase de archivos *GroupDescription*.

Función: [clic aquí](#)

Clase: [clic aquí](#)

### 3.3.2.2 Asignar Grupo

El administrador es el que decide que usuario pueden ir a que grupo. Generando un registro en la tabla `archivo_customuser_grupo`. Los puede tanto agregar como eliminar.

Función: [clic aquí](#)

### 3.3.2.3 Eliminar Grupo

Por último, el administrador podrá también eliminar el grupo, eliminando las coexiones del registro `archivo_customuser_grupo`.

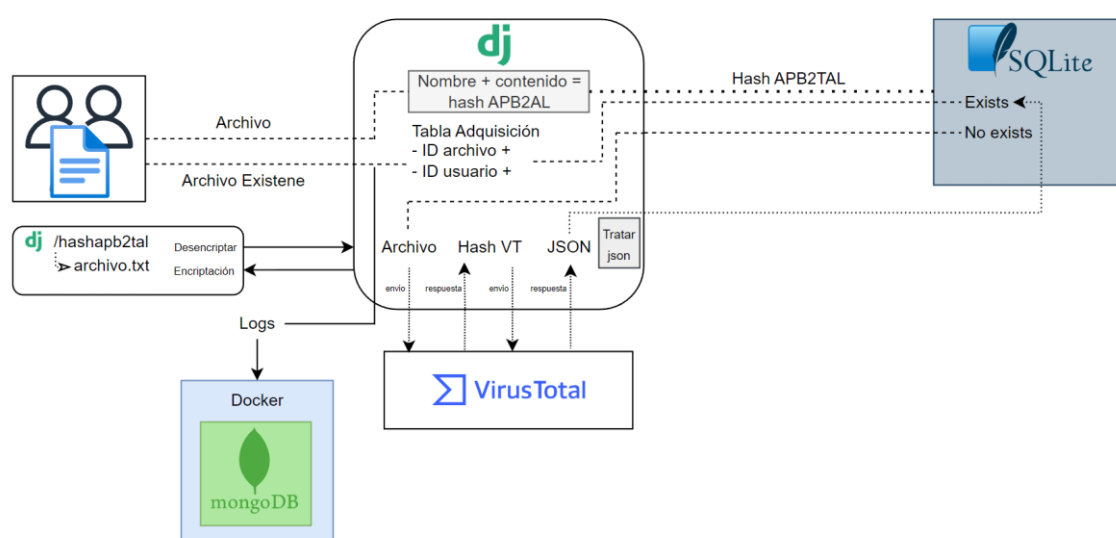
Función: [clic aquí](#)

## 3.3.3 Django Admin

Django permite un panel de administración de datos predefinidos en la ruta `/admin`. Todo aquel con el tributo true de `is_staff`, podrá acceder, lo que en el proyecto no usamos dicho panel, hemos creado el nuestro, pero está bien saberlo.

## 3.4 Archivos

En este apartado vamos a explicar como gestionamos los archivos, la subida el análisis y la respuesta. Aquí se muestra un diagrama genérico de cómo se trata el archivo, pero se va a concretar en los siguientes apartados.



Función completa de [analizar archivos](#)



### 3.4.1 Subir archivo

#### 3.4.1.1 *Hashear*

En primera instancia el usuario tras subir el archivo, lo que hace el programa es generar un hash respecto al nombre y contenido del usuario, es una manera de que en nuestro sistema no existan archivos repetidos al menos de momento en la base de datos.

#### 3.4.1.2 *Preguntar en la BD*

Una vez tenemos el hash, preguntamos a nuestra base de datos si existe en la tabla *archivos\_archivo*, si existe algún registro con ese hash, en caso positivo, nos devolverá el archivo, ¿cómo? Agregando un registro en la tabla *archivo\_adquisicion*, donde se agrega la id de usuario y la id de archivo, así asociamos usuario con archivo.

En caso de que no exista continua con el recorrido, empezando por analizar el archivo.

### 3.4.2 Analizar archivo

#### 3.4.2.1 *API Virus Total*

¿Qué es virus total?

VirusTotal es un servicio en línea que permite analizar archivos y URLs sospechosos para detectar virus, malware y otras amenazas. Para facilitar la integración de sus funcionalidades en aplicaciones y servicios de terceros, VirusTotal ofrece una API (Interfaz de Programación de Aplicaciones). La API de VirusTotal nos permite acceder programáticamente a las capacidades de análisis y reporte de VirusTotal.

Para utilizar la API de VirusTotal, necesitamos una clave de API, que se obtiene registrándose en el sitio web de VirusTotal. La API utiliza peticiones HTTP para interactuar con los servicios de VirusTotal.

Limitaciones

VirusTotal ofrece varios planes de acceso a su API, que incluyen desde una versión gratuita con limitaciones en el número de peticiones diarias, hasta versiones de pago con mayores capacidades y funcionalidades avanzadas.

Como nosotros usamos el plan gratuito , lo que hay que tener en cuenta es lo siguiente:

4 archivos por minuto. No más de 120 o 240MB por archivo dependiendo como uses la API. Solo 500 envíos al día.

Teniendo en cuenta esto, seguimos con la explicación.

Primeramente, se envía el archivo a la API, y nos va a devolver un hash, el cual guardamos en la base de datos en la tabla *archivos\_archivo*, posteriormente este hash lo volvemos a enviar para que nos de una respuesta JSON, la cual tratamos.

Nos quedamos con los positivos (detecciones malignas) y el hash del api de virus total

### 3.4.2.2 Guardar en BD

Una vez tenemos los valores necesarios guardamos los datos. Como se ha dicho antes en la tabla *archivos\_archivo*:

	Name	Data type	Clave Primaria	Foreign Key	Único	Check	No es NULL	Collate	Generated
1	id	integer							NULL
2	id_APB2TAL	varchar (640)							NULL
3	archivo_hash	varchar (640)							NULL
4	nombre_archivo	varchar (255)							NULL
5	positivos	integer							NULL
6	current_time	datetime							NULL
7	tamaño	varchar (255)							NULL

Id	Identificador archivos
Id_APB2TAL	Nuestro hash
archivo_hash	Hash del api de VT
nombre_archivo	El nombre del archivo subido
positivos	Detecciones malignas proporcionado por la respuesta
current_time	Hora y fecha del análisis
tamaño	Tamaño del archivo

La función que hace el análisis genera y envía todos los datos de la tabla para crear el registro.

Posteriormente tras crear el registro el archivo se mueve de la carpeta inicial del servidor donde sube el usuario, a la carpeta de “hecho”, donde a partir de una idea simple que tuvimos en la hora de guardar el archivo, lo que hacemos es generar una carpeta con el nombre del hash que hacemos y guardar el archivo, de una manera simple evitamos repetidos, a pesar de que sabemos que existen métodos de hashéo más seguros para evitar problemas que puede haber.

### 3.4.2.3 *Encriptar*

Este es un requisito del proyecto el de encriptar los archivos.

Cada vez que se guarda el archivo lo que hace el servidor es recoger una clave que se almacena en el archivo .env (clave la cual se genera si no existe el archivo, porque es el cual se ignora tras subir en el repositorio de github .gitignore) y encripta el archivo.

Se genera con la librería Fernet. Es un método de cifrado simétrico del módulo de criptografía "cryptography" de Python, que proporciona cifrado autenticado para garantizar la confidencialidad y la integridad de los datos. Esta biblioteca es fácil de usar y muy segura, ya que combina cifrado AES (Advanced Encryption Standard) en modo CBC (Cipher Block Chaining) con HMAC (Hash-based Message Authentication Code) para garantizar que los datos no se han alterado.

Función [clic aquí](#) generar clave

Función [clic aquí](#) encriptar

## 3.4.3 Resultado Archivo

### 3.4.3.1 *Resultado*

El resultado del archivo se muestra en el apartado de archivos, se muestran los dos hashes, los positivos, el nombre, el tamaño y la fecha de análisis. Si el archivo tiene mas de x positivos se marca como peligroso y no puedes acceder ni interaccionar, de hecho, se elimina de los archivos guardados en carpeta.

Función [clic aquí](#)

### 3.4.3.2 *Descargar*

El archivo puede ser o no descargado por el usuario, la descarga se puede realizar dependiendo si el archivo es malicioso o no. Al no ser malicioso el usuario se lo va a descargar, el proceso es buscar el archivo en el servidor y antes de dárselo al usuario aplica la función de *cargar\_clave*, para desencriptar el usuario con la clave.

Función [clic aquí](#)

### 3.4.3.3 *Eliminar*

El botón de eliminar archivo, elimina el registro de la tabla adquisición que junta archivo y usuario, para así seguir teniendo los datos de el archivo, en caso de volver a analizarlo.


Función [clic aquí](#)

## 3.5 Compartir

Por cada archivo que se analiza (que no sea malicioso), el usuario puede compartirlo. Se trabaja con la tabla de compartidos, selecciona el id de usuario que comparte y se destina el id de archivo al id de usuario o grupo destino.

Clase [clic aquí](#)

Función [clic aquí](#)

	Name	Data type	Clave Primaria	Foreign Key	Único	Check	No es NULL	Collate	Generated	
1	id	integer								NULL
2	archivo_id	bigint								NULL
3	grupo_destinatario_id	integer								NULL
4	grupo_propietario_id	bigint								NULL
5	usuario_destinatario_id	bigint								NULL
6	usuario_propietario_id	bigint								NULL

Actualmente no hay función para que grupo pueda compartir archivo, es algo que se agrego para un futuro que se puede tener en cuenta.

### 3.5.1 Compartir a usuario

El archivo va de usuario a usuario, en la función de compartir archivo se hacen todas las particiones y registros. El usuario al que es compartido podrá visualizar el archivo, eliminar el registro de compartido (no el archivo en sí mismo) y podrá descargarlo.

Y el usuario que comparte también en el panel de compartidos verá los archivos que ha compartido y eliminar el registro de compartido. Si se elimina el archivo en sí, se eliminarán los registros de compartidos en cascada

Función [clic aquí](#)

### 3.5.2 Compartir a grupo

Compartir grupo viene de la misma función solo que agregando una id de grupo, todo aquel que este en ese grupo podrá solamente descargar el archivo y ver información, no podrá eliminar el compartido de otros usuarios, solo los suyos en el grupo.

Función [clic aquí](#)

## 3.6 Logs

Los logs en nuestro servidor son una manera de reflejar lo ocurrido en tiempos determinados por cada usuario. Se registran las horas del archivo analizado, archivo eliminado, compartido de quien y que archivo a quien o quienes y eliminar el registro que se ha compartido.

Todos los logs son cadenas de texto con las variables correspondientes. Dicha cadena sea la función que sea se registra en la base de datos de mongo.

Usuario comparte el archivo – se genera el mensaje de compartición de archivo – se sube a MongoDB.

Funcionalidad [clic aquí](#)

### 3.6.1 Logs usuario

Cada usuario tiene un apartado donde puede ver un seguimiento de sus acciones individualmente.

Función [clic aquí](#)

### 3.6.2 Logs Administrativos

El administrador tiene un apartado donde puede ver los movimientos del demás usuario, de tal manera que pueda tener un mejor control de los usuarios.

Función [clic aquí](#)

## 3.7 Contacta

El apartado de contacta es un añadido personal del proyecto, dados los requisitos de la web, creemos que este apartado es importante. Que un administrador se pueda comunicar con el usuario y viceversa, es una interacción importante, en nuestro caso el tipo de contacto tiene diversas opciones, ya sea petición ( como para aceptar el usuario), queja (importante dar respuesta a los usuarios) y sugerencia (para poder mejorar, dar feedback y demás).

Por ello el apartado de comunicación.

### 3.7.1 Enviar Mensaje a Usuario

Dependiendo de que usuario eres puedes mandarle de un modo u otro los mensajes, en caso de ser un usuario común, vas a poder enviar un mensaje, que no deja de ser una tabla con atributos, título, cuerpo, y tipo de mensaje. El receptor es automático, el administrador.

Función [clic aquí](#)

Clase [clic aquí](#)

### 3.7.2 Enviar Mensaje Administrador

El administrador al contrario que el usuario común, puede escoger a que usuario puede enviar el mensaje, solo que no tiene tipo de mensaje, simplemente es consulta hacia el usuario,

Función [clic aquí](#)

### 3.7.3 Buzón

El buzón trabaja con Ajax i JavaScript, la tabla de mensajes tiene un atributo llamado leído, que se actualiza a true cuando un usuario sea admin o no lee el mensaje, es decir va al buzón, si está en false, saldrá una notificación.

Por una parte, el usuario común tiene el listado de mensajes recibidos. El administrador tiene secciones que dividen los tipos de mensajes. Todo el mundo puede borrar los mensajes (el registro de la tabla)

### 3.7.4 Llamar

Por último, en el apartado de contacto tenemos, un botón que envía un URL de llamada para cada dispositivo a un número en concreto.

## 4. Estado Actual – Requisitos

### 4.1 Requisitos del Proyecto

El proyecto debía cumplir con una serie de parámetros y requisitos obligatorios. Aquí se presenta una lista detallada de los mismos junto con una explicación de cómo se han abordado y completado al 100%.

#### 4.1.1 Cumplido

Tras el trabajo desempeñado, hemos alcanzado el cumplimiento del 100% de los requisitos establecidos. A continuación, se describe cada requisito y su implementación:

- Autenticación de usuarios

Se ha implementado un sistema de autenticación robusto que permite a los usuarios registrarse y acceder al sistema de manera segura mediante el uso de credenciales (nombre de usuario y contraseña). La autenticación se refuerza mediante la utilización de hashing para proteger las contraseñas almacenadas.

- Creación de usuarios

El sistema permite la creación de nuevos usuarios mediante un formulario de registro. Los datos ingresados se validan y almacenan en la base de datos, garantizando que solo usuarios autorizados puedan acceder a la plataforma.

- Subir archivos y carpetas

Se ha desarrollado una funcionalidad que permite a los usuarios subir archivos a la plataforma. Los archivos se almacenan en el servidor y se gestionan a través de una estructura organizada de directorios.

- Verificar archivos con Virus Total

Integración con el servicio de Virus Total para escanear los archivos subidos en busca de posibles amenazas de malware. Los archivos son enviados a Virus Total y se reciben los resultados de la verificación, asegurando que los archivos almacenados sean seguros.

- Mostrar los resultados

Los resultados de las verificaciones realizadas por Virus Total se muestran de manera clara y detallada a los usuarios. Esta funcionalidad permite a los usuarios conocer el estado de seguridad de sus archivos.

- Compartir archivos con usuarios o departamentos

Implementación de una funcionalidad que permite compartir archivos con otros usuarios o departamentos específicos dentro de la organización. Se han establecido permisos y controles de acceso para asegurar la confidencialidad y la integridad de los archivos compartidos.

- Registro de archivos subidos por el usuario y eliminados por contener software malicioso.

Se ha creado un sistema de registro (log) que documenta cada archivo subido y cualquier archivo eliminado debido a la detección de software malicioso. Este registro es accesible a los administradores para auditorías y monitoreo.

- Descargar archivos

Los usuarios tienen la capacidad de descargar archivos desde la plataforma de manera segura. Esta funcionalidad asegura que solo los usuarios autorizados puedan acceder y descargar los archivos almacenados.

- Panel administrativo para validar un usuario y asignarlo a un departamento

Desarrollo de un panel administrativo que permite a los administradores validar nuevos usuarios y asignarlos a los departamentos correspondientes. Esto facilita la gestión y organización de los usuarios dentro del sistema.

- Registro de todos los archivos subidos y eliminados por los usuarios

Un registro exhaustivo de todas las actividades relacionadas con la subida y eliminación de archivos por parte de los usuarios. Este registro ayuda a mantener un control detallado y una auditoría completa de todas las operaciones realizadas.

- Aplicar un cifrado a los archivos mientras el archivo ha sido verificado, pero no descargado

Implementación de técnicas de cifrado para proteger los archivos mientras están almacenados en el servidor, especialmente en el periodo entre la verificación y la descarga. Esto asegura que los archivos no sean accesibles en texto plano en caso de acceso no autorizado.



- Aplicar seguridad a la red, configurando un firewall como pfSense delante del servidor

Configuración de un firewall utilizando pfSense para proteger el servidor contra ataques y accesos no autorizados. Esta medida asegura que solo el tráfico legítimo llegue al servidor, proporcionando una capa adicional de seguridad.

- Auditar la web en busca de vulnerabilidades

Realización de auditorías de seguridad para identificar y mitigar vulnerabilidades en la aplicación web. Esto incluye pruebas de penetración y análisis de seguridad para asegurar que la plataforma esté protegida contra amenazas comunes.

- Pensar y aplicar un par de mejoras al producto

Evaluación y aplicación de mejoras adicionales al producto basadas en retroalimentación de los usuarios y análisis interno. Estas mejoras pueden incluir optimizaciones de rendimiento, nuevas funcionalidades o mejoras en la interfaz de usuario.

Con el cumplimiento de estos requisitos, el proyecto se encuentra en un estado completo y funcional, listo para ser utilizado de manera segura y eficiente.

## 4.2 Aplicaciones Propias

Durante el proyecto hemos desarrollado, diversas aplicaciones propias más allá de los requisitos de el proyecto. Proviene de una idea de servicio multifuncional, que en un futuro se puede ir expandiendo de manera más profesional

### 4.2.1 Servicio de Contacto

Como se ha podido leer, el servidor web ofrece un servicio de contacto, hasta el momento no era un requisito, cosa que nosotros veíamos bastante imprescindible, no obstante, en un punto quisimos ir más allá y de hecho queremos en un futuro ir más lejos agregando funcionalidades más genéricas y profesionales como, un call-center o un webchat.

### 4.2.2 Alta Seguridad

Gracias a las decisiones como escoger en su momento Django y a las ideas de hashéo encriptación y funcionalidades aplicadas, podemos decir que hasta el momento gracias también a las auditorias de seguridad, nuestro servidor web es un lugar seguro. No obstante, estamos alerta y seguimos testeando posibles fugas amenazas y depuración de Código

### 4.2.3 Customización

Llego un punto que decidimos cambiar completamente el diseño del proyecto, creemos que una parte muy importante es la interacción y la primera vista de el usuario, por ello el cambio fue en base a los usuarios y estándares de edición visual para una mejor interacción con el usuario.

Juntamente con unos sutiles códigos, se hacen grandes cambios de personalización de usuario tanto como el tema claro y oscuro como la edición de imagen de perfil y color.

Usamos por primera vez adrede el almacenamiento de cliente para guardar información de personalización. Gracias a ello el servidor se ahorra de usar mas datos, y posibles peligros al guardar imágenes o Código no deseados.

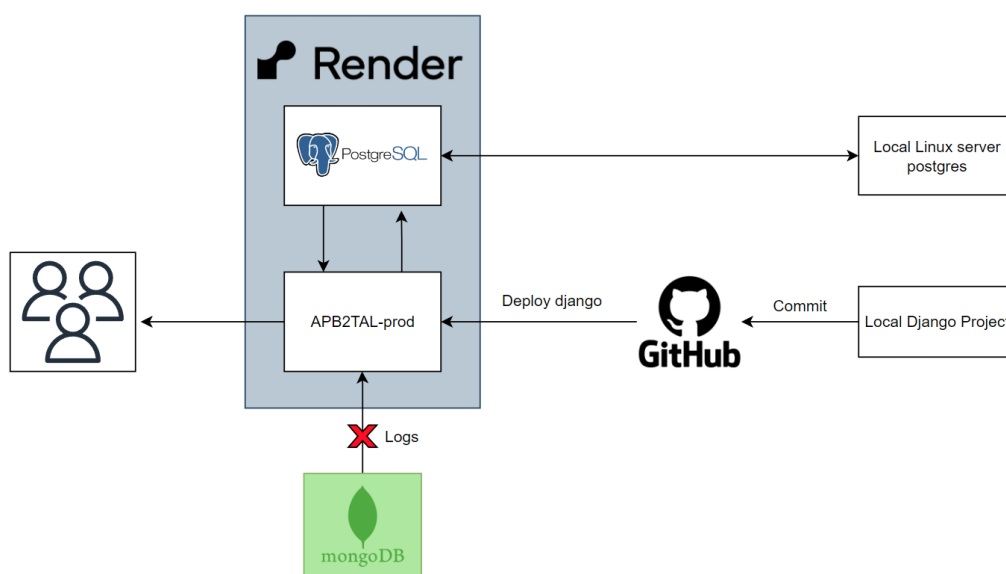
### 4.2.4 Producción

En primer lugar, la web debía ser un servidor hosteado por nosotros, esa era la idea, hacer un deploy de Django en apache o nginx, es bastante complejo se logro en su momento en nginx con gunicorn( Gunicorn es una herramienta o una interfaz que se encarga de todo lo que sucede entre el servidor web y su aplicación web), no obstante, la idea de publicarlo en internet era no tan ambiciosa.

Tras investigación tenemos render.com.

Render.com es una plataforma de la nube que ofrece servicios de alojamiento y despliegue de aplicaciones web, aplicaciones estáticas, despliegue de contenedores (Docker) y funciones serverless.

Estructura:



En primer lugar, creamos un repositorio de GitHub, con la configuración de archivos del proyecto de Django correcta:

Desde el local se hace un commit y el propio render hacer un deploy automático a la página web. Por otra parte, la base de datos tuvimos que pasar de SQLite a postgres, requisito de render.com. y nos podemos conectar a través de un external link proporcionado.

Las migraciones se hacen sola, tanto como la instalación de los requeriments, creamos un archivo llamado *build.sh* el cual se escriben los comandos que debe ejecutar, como la migración y la instalación de requeriments.

Mongo DB hasta el momento no está configurado, lo cual actualmente los usuarios no pueden brindarse del servicio de los logs.

Render.com aparte de hacer un deploy automático del proyecto de Django ofrece la publicación de un dominio no personalizado a internet.

Y para más información [clic aquí](#)

## 4.3 Aplicaciones Futuras

### 4.3.1 ¿Deuda técnica?

En cuanto a los requisitos pedidos el proyecto no tiene deuda técnica, lo que si tenemos dejado a tras son los métodos más fiables de utilizar y dar soporte a algunas funciones o métodos que sabemos que podrían ser más eficientes, actualmente existen situaciones que no controlamos en el proyecto y que pueden ser fatales para la producción, pero algunos integrantes han decidido llevar a cabo más allá el proyecto y solucionar los problemas.

### 4.3.2 Control de Usuarios

Uno de los desafíos más significativos que enfrentamos fue el control de usuarios. Este aspecto del proyecto resultó particularmente complejo debido a la necesidad de gestionar múltiples usuarios que pueden analizar archivos simultáneamente. A continuación, se detalla el enfoque adoptado para este problema utilizando Django.

#### Problemas Identificados

- Concurrencia de Usuarios:

Múltiples usuarios pueden estar realizando análisis de archivos al mismo tiempo, lo que puede causar problemas de concurrencia y rendimiento del sistema.

- Gestión de Sesiones:

Asegurar que cada usuario mantenga su propia sesión de trabajo sin interferir con las sesiones de otros usuarios.

- Control de Acceso y Permisos:

Implementar un sistema robusto de control de acceso para asegurar que los usuarios solo puedan acceder a los archivos y funcionalidades que les correspondan.

#### Posibles Soluciones

- Concurrencia de Usuarios

Para manejar la concurrencia de usuarios de manera eficiente, se pueden implementar las siguientes estrategias:

Colas de Tareas:

Utilización de colas de tareas con Celery y RabbitMQ. Celery permite gestionar tareas en segundo plano de manera eficiente. Cada análisis de archivo se coloca en una cola de tareas, asegurando que los análisis se procesen de manera asíncrona sin bloquear la aplicación web.

Escalabilidad Horizontal:

Configuración del servidor para soportar múltiples trabajadores de Celery, permitiendo que varios análisis se ejecuten en paralelo sin sobrecargar el sistema.

- Gestión de Sesiones

Para asegurar que cada usuario mantenga su sesión de manera aislada:

Autenticación Basada en Tokens:

Implementación de un sistema de autenticación basado en tokens utilizando Django Rest Framework (DRF) y JSON Web Tokens (JWT). Esto permite manejar las sesiones de usuarios de manera segura y escalable

- Control de Acceso y Permisos

Grupos y Permisos de Django:

Uso de los grupos y permisos integrados de Django para definir roles y permisos específicos para diferentes tipos de usuarios (administradores, usuarios regulares, etc.).

Middleware Personalizados:

Implementación de middlewares personalizados para verificar los permisos de los usuarios antes de permitirles acceder a ciertos recursos o realizar determinadas acciones.

Estas soluciones podrían manejar de manera eficiente el control de usuarios en el sistema:

- Los usuarios podrán subir y analizar archivos simultáneamente sin afectar el rendimiento del sistema.
- Cada usuario mantendrá su propia sesión de trabajo, evitando interferencias entre usuarios.
- Se asegurará que los usuarios solo puedan acceder a los archivos y funcionalidades autorizadas, mejorando la seguridad y la integridad del sistema.

## 5. Seguridad

### 5.1 Auditoria Web

INFORME AUDITORIA [clic aquí](#)

La auditoría web es un proceso crucial para evaluar la seguridad, rendimiento y cumplimiento de un proyecto web. En este contexto, se llevaron a cabo dos auditorías distintas en el proyecto de Django. La primera auditoría se realizó al final de la primera fase del proyecto, mientras que la segunda auditoría se llevó a cabo después de reiniciar y realizar mejoras significativas en el proyecto. A continuación, se detallan los hallazgos y mejoras de cada auditoría y se presentan las conclusiones derivadas de la comparación entre ambas.

#### 5.1.1 Primera auditoria

La primera auditoría se enfocó en identificar problemas iniciales y áreas de mejora en el proyecto Django. Durante esta fase, se examinaron varios aspectos clave, tales como la seguridad de la aplicación, el rendimiento de la base de datos, la eficiencia del código y el cumplimiento de las mejores prácticas de desarrollo web. Los hallazgos de esta auditoría revelaron varias vulnerabilidades y deficiencias que requerían atención inmediata.

#### 5.1.2 Segunda Auditoria

Tras la revisión y reinicio del proyecto, se realizó una segunda auditoría con el objetivo de evaluar las mejoras implementadas. Esta auditoría se centró en los mismos criterios que la primera, permitiendo una comparación directa de los progresos realizados. Se examinaron nuevamente la seguridad, el rendimiento y la calidad del código, además de verificar la implementación de las recomendaciones proporcionadas en la primera auditoría.

### 5.1.3 Comparación - Conclusiones

La comparación entre las dos auditorías mostró una notable mejora en varias áreas del proyecto. Las vulnerabilidades de seguridad identificadas en la primera auditoría fueron abordadas correctamente en la segunda fase del proyecto, mejorando significativamente la robustez de la aplicación. El rendimiento del sistema también mostró avances considerables, con tiempos de respuesta más rápidos y una mejor gestión de la base de datos. Además, la calidad del código se incrementó, reflejando una adherencia más estricta a las mejores prácticas de desarrollo.

Estas mejoras no solo incrementaron la eficiencia y seguridad del proyecto, sino que también sentaron una base sólida para su futuro desarrollo y escalabilidad que es precisamente lo que buscamos.

## 5.2 Aplicaciones de Seguridad

Tener en cuenta la seguridad en el proyecto era uno de los puntos más importantes. Hemos trabajado diligentemente para controlar todos los posibles fallos y errores que se pueden presentar en una página web. Con Django, hemos implementado varias medidas y aplicaciones de seguridad para garantizar la integridad, confidencialidad y disponibilidad del sistema.

### 5.2.1 Seguridad Aplicada

#### 5.2.1.1 *Validación y Sanitización de Datos*

Para prevenir inyecciones de código y otras vulnerabilidades relacionadas con la manipulación de datos, se implementaron prácticas estrictas de validación y sanitización de datos:

**Validación de entradas:** Todos los datos de entrada son validados rigurosamente utilizando formularios y modelos de Django y nuestros.

**Sanitización de salidas:** Utilizamos plantillas seguras personalizadas que escapan automáticamente las salidas para evitar ataques de inyección de HTML y JavaScript.

#### 5.2.1.2 *Configuración de Seguridad en Django*

Django ofrece múltiples configuraciones predeterminadas y características integradas que mejoran la seguridad de la aplicación. Algunas de las medidas que hemos implementado:

- Protección contra CSRF

Django viene con protección contra ataques de Cross-Site Request Forgery (CSRF) habilitada por defecto. Nos aseguramos de que todos los formularios incluyan tokens CSRF.

- Autenticación y Autorización

Implementamos un sistema robusto de autenticación y autorización utilizando los módulos de autenticación de Django y extendiéndolos según nuestras necesidades del proyecto.

Mas información [clic aquí](#)

### 5.2.2 Pfsense

Especificaciones Infraestructura de Red con pfSense [clic aquí](#)

Para garantizar un entorno seguro y controlado, hemos diseñado una estructura de red utilizando pfSense, un software de firewall de código abierto que proporciona una amplia gama de funcionalidades de seguridad y gestión de red. La configuración incluye dos usuarios en la LAN y un servidor en la DMZ donde se aloja nuestra aplicación Django. Esta configuración ha sido esencial para crear un entorno seguro y minimizar los riesgos asociados con el acceso no autorizado y las vulnerabilidades.

- Diseño de la Red

El diseño de la red se ha estructurado en tres segmentos principales:

LAN (Local Area Network): Esta red interna contiene los dos usuarios que necesitan acceder a los recursos internos. La LAN está protegida por el firewall pfSense, que regula el tráfico de entrada y salida, asegurando que solo el tráfico autorizado pueda transitar.

DMZ (Demilitarized Zone): La DMZ es una subred perimetral que alberga el servidor donde se ejecuta nuestra aplicación Django. Este servidor es accesible tanto desde la LAN como desde el exterior, pero está separado de la red interna principal para mitigar el riesgo en caso de un ataque.

WAN (Wide Area Network): Esta es la conexión externa a Internet. pfSense gestiona la conexión entre la WAN y los otros segmentos de red, aplicando políticas de seguridad estrictas para controlar el tráfico entrante y saliente.

- Beneficios de la Estructura de Red

Esta estructura de red proporciona varios beneficios clave:

**Seguridad Mejorada:** Al separar los usuarios y el servidor en diferentes segmentos de red, se limita el alcance de un posible ataque, protegiendo los activos más sensibles.

**Control de Acceso:** Las reglas de firewall y NAT aseguran que solo el tráfico autorizado puede acceder a los recursos necesarios, minimizando el riesgo de acceso no autorizado.

**Aislamiento de Servicios:** Al alojar el servidor Django en la DMZ, cualquier compromiso del servidor no se propagará fácilmente a la red interna, protegiendo otros recursos críticos.

## 6. Final

### 6.1 Conclusiones

A nuestro parecer, no solo creemos que hayamos cumplido satisfactoriamente los requisitos del proyecto, además creemos que hemos hecho un buen trabajo y proyecto.

Siempre hay cosas que mejorar, de hecho, como se explica en la memoria se pueden mejorar muchas cosas, pero dejando a lado el nivel técnico, sabemos que también se puede mejorar la gestión del proyecto, el tiempo, el equipo, las tareas y mucho más.

Sinceramente hemos aprendido mucho, no solo a nivel técnico como Django, Apis y muchas tecnologías y campos aplicados en el proyecto, se ha trabajado en equipo, en gestión y todo ello son aprendizajes que merecen mucho la pena tanto como en el mundo del aprendizaje como en el mundo laboral.

Concretamente, la web es una buena idea de proyecto, hacer un gestor y analizador de archivos en la nube, para los usuarios. En el mundo en el que nos movemos y avanzamos la seguridad debe crecer, por que las amenazas son reales y también crecen. La iniciativa propuesta, a nuestro parecer es un buen incentivo de ayudar a reflexionar durante el desarrollo, el futuro de los integrantes a nivel profesional y en algunos casos tal vez la manera de ver las cosas.

Terminamos agradeciendo a los tutores y profesores que nos han ayudado a resolver preguntas sin respuesta, y cuestiones difíciles de resolver, también a la gente externa que ha dado su opinión y aunque pequeñas sean sus ayudas, gracias a ellos grandes son nuestros avances.



## 6.2 Webgrafía

[Documentación Docker](#)

[Documentación Django](#)

[Documentación SQLite](#)

[Documentación MongoDB](#)

[Documentación JavaScript](#)

[Documentación AJAX](#)

[Deploy Render.com](#)

[Deploy Render.com Django](#)

[Documentación PostgreSQL](#)

[Documentación Fernet library](#)

[PFSense](#)

[Documentación de Python](#)

[Documentación Jinja](#)

[Documentación Bootstrap](#)

[Documentación Api VT](#)