APB2TAL

Índice

1. Introducción

1.1 Presentación

1.1.1 Proyecto

1.1.2 Grupo

1.1.3 Memoria

1.2 Fases

1.2.1 Planteamiento Fase 0

El Proyecto empezó a mitades de octubre después del mini proyecto, lo cual se han trabajado aproximadamente 7 meses desde su inicio, con un total de 3 participantes.

El proyecto ha constado de 3 etapas, Fase 0, Fase 1 y Fase 2

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

- Crear un servidor web. (lliure elecció)
- Crear una base de dades relacional. (lliure elecció).
- Crear una base de dades no relacional. (lliure elecció).
- Crear mockups de la pàgina web amb les següents

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.

Así que en la primera fase de proyecto Tuvimos que afrontar el proyecto con

- Crear un servidor web -> Django
- Crear una base de dades relacional -> SQlite (proporcionada por django)
- Crear una base de dades no relacional -> MongoDB

Estos fueron los requisitos de la fase 0

- Autenticació d'usuaris.
- Creació d'usuaris.
- Pujar fitxers i carpetes.
- Verificar amb Virustotal els fitxers.
- Mostrar els resultats.

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 postrar 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 els fitxers amb usuaris o departaments.
- Registre de fitxers pujats per l'usuari i eliminats per contenir programari maliciós.
- Descarregar de fitxers.
- Panell administratiu on validar un usuari i assignar-lo a un departament.
- Registre de tots els fitxers pujats i eliminats pels usuaris.

“Ni idea por dónde comenzar.” *- el grupo*

A partir de este punto, vimos que esto iba y podría ser algo grande por ello, una nueva herramienta entra en juego.

GitHub

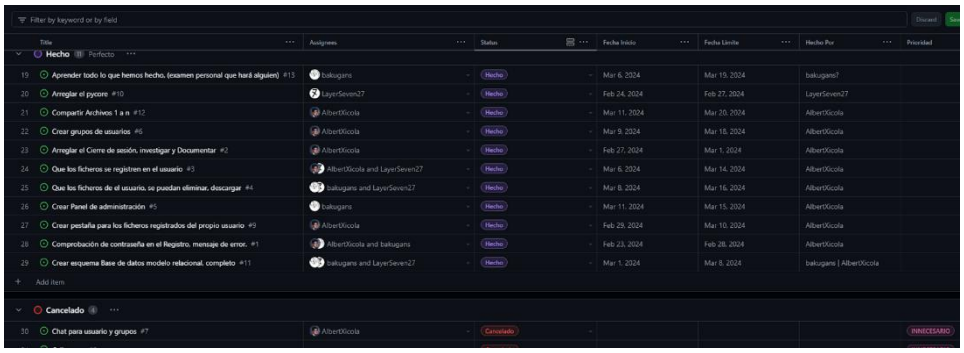
Es una plataforma online de desarrollo de software que se usa para almacenar, supervisar y trabajar con proyectos de software. Facilita el intercambio de archivos de código y trabajar en proyectos colaborativos de código abierto.

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*.

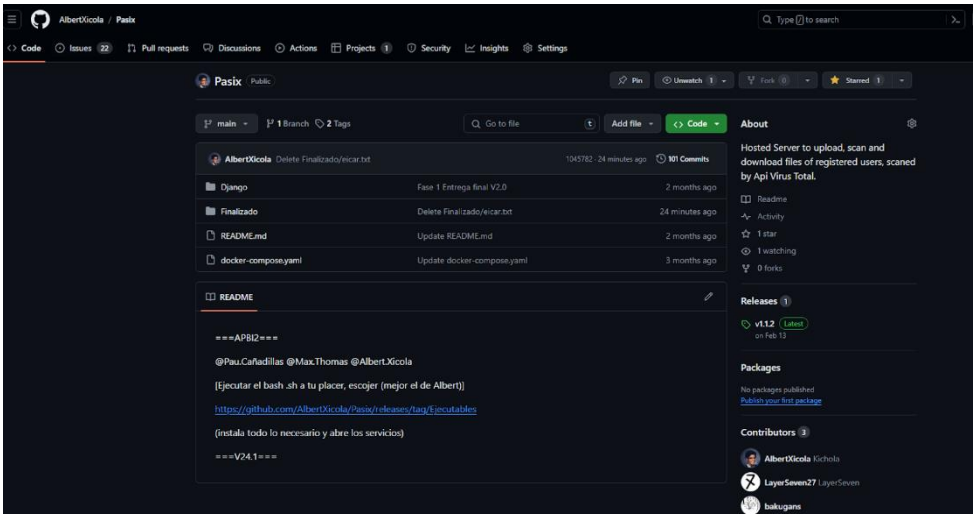
A partir de este momento el trabajo era más versátil, cada uno tenía pequeñas tareas a realizar y datos que aportar a la documentación.

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 del mismo.

Proyecto en ese momento:



Repositorio en ese momento:



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

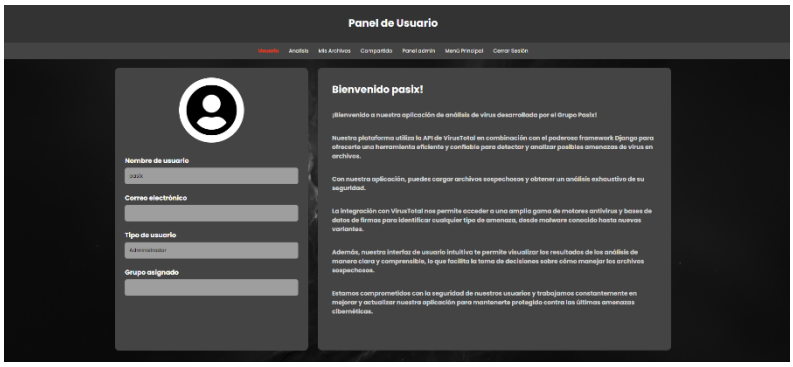
Hasta este punto ya en febrero, seguimos en la búsqueda y “tecleo” de código del proyecto, nos repartimos el trabajo con lo que mejor se le daba a cada uno.

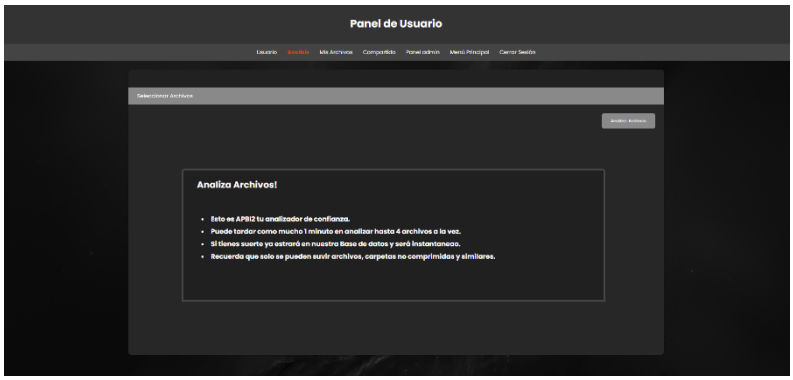
Pudimos realizar los puntos pedidos, excepto algunos, Registros de los archivos eliminados (logs en término corto), un panel administrativo, a parte de el de *django* predefinido y a pesar de cumplir con el resto, aun así, el proyecto chirriaba, como se dice, cogido con pinzas.

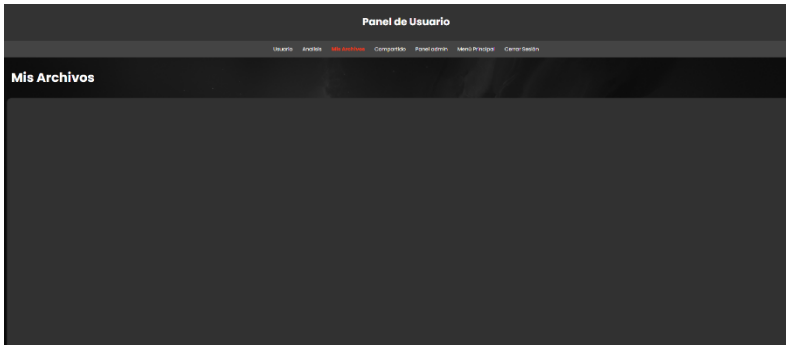
¿Funcionaba? **Si.** ¿A la perfección sin ningún tipo de error? **No.**

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 .







Aparentemente visualmente era agradable a nuestro parecer, de hecho no funcionaba tan tan tan mal, o eso creíamos, por eso...

1.2.3 Planteamiento Fase 2

Última fase, la más crucial, la más definitiva, la más todo.

¿Que nos queda?

- Aplicar un xifrat de fitxers mentre el fitxer ha sigut verificat però no descarregat.
- Aplicar seguretat a la xarxa, configurant un tallafoc com pfSense per davant del servidor.
- Auditar la web de vulnerabilitats web.
- Pensar i aplicar un parell de millores a aplicar al producte.
- + Deute tècnic

Llegados a este punto, y haciendo un pensamiento positivo, terrorífico 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.

2. Proyecto APB2TAL

2.1 ¿Que usamos y por qué?

2.1.1 Django

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. Es como una caja de herramientas muy completa y eficiente que te ayuda a construir sitios 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.

2.1.1.4 Arquitectura Modelo-Vista-Template (MTV)

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

- Modelo (Model):

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.

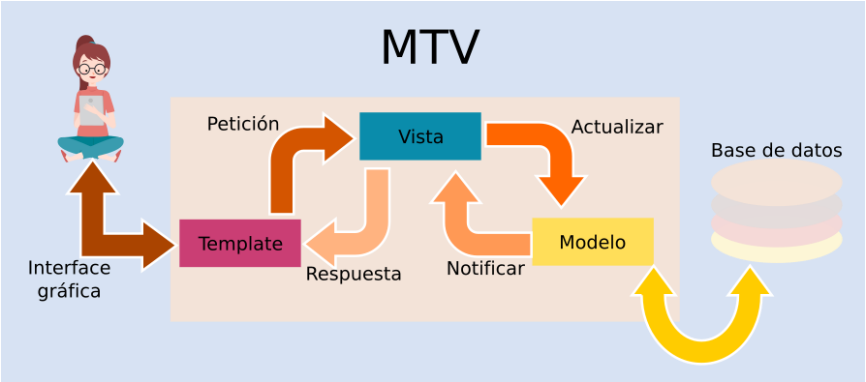
- Vista (View):

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.

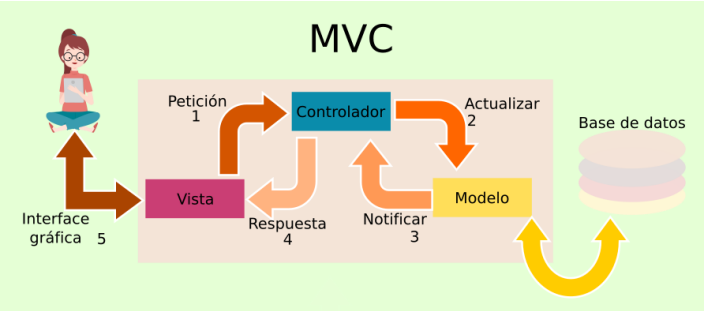
- Vista (View):

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 otras funcionalidades listas para usar.

- 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.

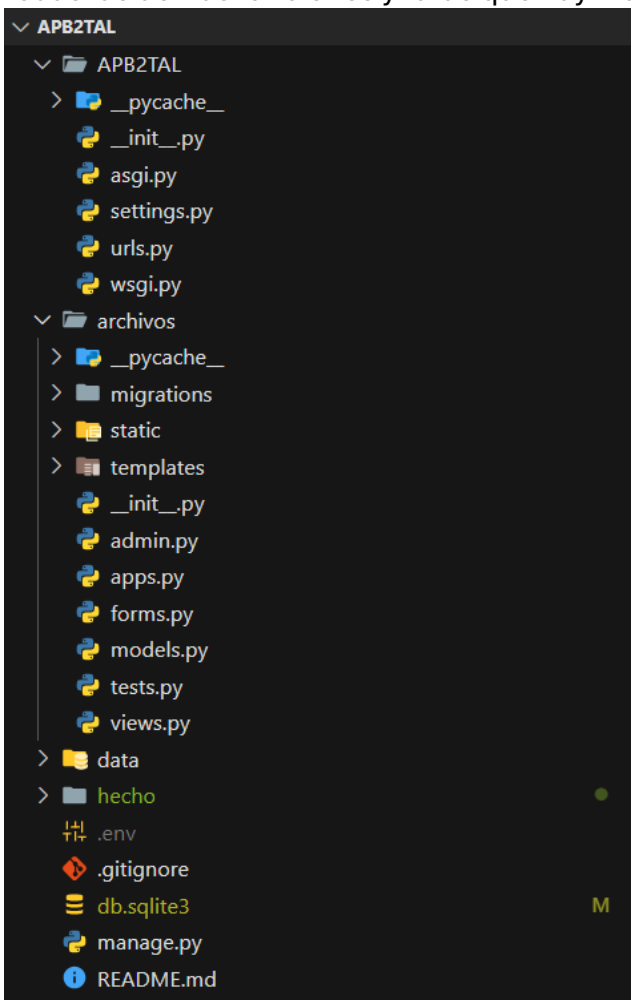
Las aplicaciones en Django son componentes modulares que te permiten organizar tu proyecto en partes funcionales. Cada aplicación es una entidad autónoma que puede incluir modelos, vistas, plantillas, archivos estáticos y otros elementos necesarios para cumplir con una funcionalidad específica. Esto 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 nuclio 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/                 <-      Applicacion (solo hay una, archivos que están los datos web)
        __init__.py         <-      inicializador Python de la app
        admin.py            <-      información de la ruta de amdministracion
        apps.py             <-      Contiene la configuración de la aplicación
        models.py           <-      Modelos de la base de datos que commplie 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 7 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.

archivos_adquisicion	Tabla para asociar archivo con usuario
archivos_archivo	Información de los archivos registrados
archivos_compartido	Asociar los archivos de un usuario a usuario compartido
archivos_mensajes	Datos de mensaje de usuario a usuario
auth_group	Grupos de el proyecto
auth_user	Usuarios del proyecto
auth_user_group	Usuarios asociados a grupos

Estas son las tablas que se utilizan de normal las cuatro 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 9 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 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.

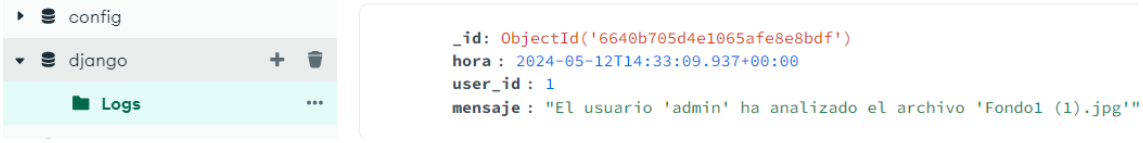
MongoDB permite almacenar datos con un esquema dinámico, lo que significa que cada registro de log no necesita cumplir con un formato predefinido. Esto facilita la captura de diferentes tipos de registros, incluso si varían en estructura o contenido.

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 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 so, 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 como 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

Tecnologías – 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 miembros del equipo. 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

2.2 Plan general de la estructura del proyecto

2.2.1 Diagramas

3. Funcionamiento Proyecto

3.1 Usuarios

3.1.1 Inicio de sesión

3.1.2 Registro

3.1.3 Perfil

3.2 Grupos

3.3 Administrador

3.3.1 Administrar usuarios

3.3.1.1 Aceptar usuario

3.3.1.2 Modificar usuario

3.3.1.3 Banear usuario

3.3.2 Administrar Grupos

3.3.2.1 Crear Grupo

3.3.2.2 Eliminar Grupo

3.3.2.3 Asignar Grupo

3.3.3 Django Admin

3.4 Archivos

3.4.1 Subir archivo

3.4.1.1 Hashear

3.4.1.2 Preguntar en la BD

3.4.2 Analizar archivo

3.4.2.1 API Virus Total

3.4.2.2 Guardar en BD

3.4.2.3 Encriptar

3.4.3 Resultado Archivo

3.4.3.1 Resultado

3.4.3.2 Descargar

3.4.3.3 Eliminar

3.4.3.4 Compartir