



PROYECTO FINAL

CFGS Desarrollo de Aplicaciones
Multiplataforma
Informática y Comunicaciones

Magik Antivirus

Tutor individual: Cristina Silván Pardo

Tutor colectivo: Cristina Silván Pardo

Año: 2025

Fecha de presentación: 22/05/2025

Nombre y Apellidos: Erik Amo Toquero
Email: erik.amotoq@educa.jcyl.es



Tabla de contenido

1	Identificación proyecto	4
2	Organización de la memoria	4
3	Descripción general del proyecto	4
3.1	Objetivos	4
3.2	Cuestiones metodológicas	5
3.2.1	Detección Basada en Huellas Digitales (Hash-Based Detection)	5
3.2.2	APIs.....	5
3.3	Entorno de trabajo	6
3.3.1	Herramientas de Desarrollo	6
3.3.2	Lenguajes de Programación	6
4	Descripción general del producto.....	7
4.1	Visión general del sistema:.....	7
4.2	Métodos, técnicas y arquitecturas	8
4.3	Despliegue de la aplicación	10
4.3.1	Despliegue de la Base de Datos en Red (FreeDB)	10
4.3.2	Despliegue de la API (Vercel)	10
4.3.3	Despliegue de la Aplicación (GitHub Releases)	11
5	Planificación y presupuesto	12
6	Documentación Técnica.....	13
6.1	Especificación de requisitos	13
6.2	Análisis del sistema	13
6.3	Diseño del sistema.....	14
6.3.1	Diseño de la Base de Datos	14
6.3.2	Diseño de la Interfaz de usuario.	16
6.3.3	Diseño de la Aplicación.	20
6.4	Implementación:	21

6.4.1	Entorno de desarrollo.	21
6.4.2	Estructura del código.	22
6.4.3	Cuestiones de diseño e implementación reseñables.....	33
6.5	Pruebas.....	37
6.5.1	Pruebas de API	37
6.5.2	Pruebas de la aplicación.....	38
7	Manuales de usuario	41
7.1	Manual de usuario.....	41
7.2	Manual de instalación	41
7.2.1	Android.....	41
7.2.2	Windows	42
7.2.3	Linux	42
8	Conclusiones y posibles ampliaciones	43
9	Bibliografía	44

1 Identificación proyecto

Este proyecto trata del desarrollo de un software antivirus teóricamente funcional en el mayor número de sistemas operativos posible. Esta aplicación será capaz de analizar los archivos del sistema en el que se encuentra instalada, buscando archivos que, según una base de datos de huellas o hashes, puedan suponer una amenaza para el usuario. Estos archivos serán puestos en cuarentena, imposibilitando su uso al usuario salvo si se restauran manualmente.

2 Organización de la memoria

La memoria se organiza en 6 apartados principales:

1. Descripción del Proyecto: Donde se habla del proyecto en general, los objetivos que busca cumplir y los medios que se van a utilizar para llegar al producto final
2. Planificación del Proyecto: Se hará un breve resumen de qué hay que hacer, cómo se hará y el tiempo estimado que se espera tardar en cada apartado.
3. Análisis de Sistema: Se analizarán todos los requisitos necesarios para que la aplicación cumpla su función correctamente
4. Diseño y Desarrollo de la Aplicación: Resumirá de inicio a fin el proceso de desarrollo.
5. Pruebas de Uso: Se harán pruebas ligeras en distintas partes de la aplicación y se explicará su fin
6. Manual de Usuario e Instalación: El manual de instrucciones para que el usuario pueda utilizar correctamente la aplicación

3 Descripción general del proyecto

3.1 Objetivos

Aprender a manipular archivos de los distintos sistemas operativos, creándolos, editándolos y borrándolos

Ser capaz de desplegar una aplicación que, con una única versión de código, pueda operar correctamente en Android, iOS, MacOS, Windows y Linux.

Poder conectar una aplicación con una base de datos por medio de un servicio API, haciendo a través de ésta todas las operaciones CRUD necesarias.

Gestionar los datos de usuario, así como sus dispositivos conectados y sus preferencias en la aplicación de forma local.

Optimizar los recursos de un proceso con tanto peso como es el análisis de ficheros de un sistema operativo.

3.2 Cuestiones metodológicas

3.2.1 Detección Basada en Huellas Digitales (Hash-Based Detection)

La detección basada en huellas digitales es uno de los métodos de identificación de malware *conocido* más utilizados. Es importante destacar el conocido, ya que este método no sirve para amenazas no detectadas o registradas previamente por otros sistemas.

Una *huella* o *hash* representa el código interno encriptado de un archivo. Cuando se registra un nuevo fichero como malware, su código es encriptado, normalmente siguiendo la función hash **MD5** o **SHA256**, y su resultado es guardado en una base de datos de *hashes conocidos*, donde los sistemas de protección posteriormente recogerán esa huella para analizar otros archivos en busca de coincidencias.

En cuanto a las aplicaciones que aplican este método de reconocimiento, necesitan encriptar el código del fichero y compararlo con los existentes en la base de datos a la que se acceda. Si se encuentra una coincidencia, significa que exactamente ese mismo fichero ha sido previamente registrado como malware, y debe ser puesto en cuarentena.

Frente a otros sistemas de detección, como son el de por análisis de anomalías o firmas, este presenta la ventaja de que es más rápido, debido a que simplemente se encripta un archivo y se busca una coincidencia en una base de datos, pero cuenta con dos desventajas principales:

Solo detecta malware conocido, es decir, si hay algún archivo infectado que no se detecte como tal en la base de datos, el antivirus no lo confinará.

Si un archivo maléfico cambia su código, sea para cambiar su estructura o se añada un solo carácter más, cambiará su hash por completo, por lo que volverá a ser indetectable para el sistema.

3.2.2 APIs

Los servicios Web son aplicaciones que permiten la comunicación con distintos dispositivos electrónicos en red. Esto se hace por el protocolo HTTP: Un protocolo de transferencia de archivos de hipertexto por la red. Se basa en que el cliente hace una **petición** (un mensaje para hacer alguna instrucción en red) al servidor, y este devuelve una respuesta en forma de código.

El rango de 200 a 299 sirve para marcar que la petición ha sido aceptada por el servidor

- El rango de 300 a 399 marca un error en la autorización del servicio
- El rango de 400 a 499 sirve para marcar un error en la petición
- Más allá de 500, la respuesta marca un error interno del servidor

Las APIs son un contrato que permite a los desarrolladores interactuar con una aplicación. Una RESTful API es una API que se adapta al uso que le quiera dar el desarrollador. Esta se forma de EndPoints, puntos de interacción cliente-servidor, donde el cliente realiza una petición de obtención o manipulación de datos y el servidor le devuelve información en formato JSON.

3.3 Entorno de trabajo

3.3.1 Herramientas de Desarrollo

Las herramientas utilizadas para llevar a cabo el proceso de desarrollo de esta aplicación han sido:

- **Visual Studio Code** como entorno de desarrollo principal, debido a su versatilidad para trabajar con múltiples lenguajes de programación y su amplia variedad de extensiones que permiten facilitar el desarrollo de código en cualquier lenguaje.
- **Figma** como herramienta de creación de wireframes, que permite diseñar de forma gráfica una maqueta interactiva de la aplicación que se espera conseguir
- **Vercel** como plataforma de despliegue del servicio web a utilizar. Vercel permite desplegar las aplicaciones directamente desde los repositorios de una cuenta de GitHub, y tiene la ventaja de que se actualiza automáticamente cada vez que el repositorio cambia su versión.
- **FreeDB** como plataforma de base de datos relacional en red. Es una buena alternativa para que proyectos con fines de aprendizaje tengan una base de datos a la que puedan acceder en varios dispositivos de forma totalmente gratuita. Para este proyecto, como actualmente solo cuenta con fines educativos y no lucrativos, además de no guardar datos vulnerables de los usuarios, es la plataforma que me he dispuesto a utilizar.
- **VMWare** como software de virtualización. Al ser un proyecto de desarrollo multiplataforma, la aplicación debe ser funcional y testeada en distintos sistemas operativos, por lo que se utilizarán máquinas virtuales para testear en distintos sistemas.
- **SQLite** como herramienta de creación de Bases de Datos Locales. Permite guardar información en un archivo del dispositivo. Se caracteriza con su alta velocidad de lectura y escritura de información, pero tiene la desventaja de que la base de datos funciona únicamente en el dispositivo local.
- **MySQL** como Sistema Gestor de Base de Datos en Red. MySQL permite su acceso y su gestión de forma remota utilizando un usuario que tenga acceso a sus bases de datos.

3.3.2 Lenguajes de Programación

Junto con los programas utilizados para realizar este proyecto, se han utilizado los siguientes lenguajes:

DART como lenguaje de programación de la aplicación. Es un lenguaje tipado que permite la programación JIT y permite cierta portabilidad con todos los dispositivos. Dentro de DART, se encuentra su framework más conocido para el desarrollo de interfaces, **Flutter**. Se caracteriza por la posibilidad de instalar dependencias fácilmente para optimizar la creación de código, además de utilizar por defecto los Widgets de Material UI o Cupertino para generar sus interfaces gráficas.

Python como lenguaje de programación de la API. Python es un lenguaje no tipado, sencillo y con una curva de aprendizaje muy agradable para cualquier usuario. Puede realizar cualquier tipo de tareas gracias a su amplia cantidad de librerías.

4 Descripción general del producto

4.1 *Visión general del sistema:*

Antes de definir la vista general, hay que tener claro lo que se busca conseguir con este proyecto: La idea es desarrollar una aplicación que puede crear y borrar archivos que están guardados como potencial malware en una base de datos. Fuera de eso también se quiere tener un sistema de inicio de sesión, donde el usuario pueda introducir sus datos y poder ver los dispositivos vinculados a su cuenta.

Teniendo esto claro, se dan los siguientes puntos:

Límites del Sistema

La aplicación depende directamente del sistema operativo y los límites que éste tenga, ya que necesita acceder a su sistema de ficheros, además de que requiere conexión a internet para ser utilizado. Otro límite a tener en cuenta son los propios recursos del dispositivo. El análisis recursivo de archivos y carpetas consume bastante memoria, y si no se gestiona correctamente puede dar problemas a nivel de la aplicación y del propio dispositivo.

Funcionalidades Básicas

- Acceder a una base de datos externa para recoger hashes de malware.
- **Gestionar usuarios.** (crear, buscar, actualizar y borrarlos de una base de datos)
- Crear, leer y borrar archivos del sistema operativo.

Funcionalidades Extra

- Permitir borrar o recuperar archivos de cuarentena a elección del usuario
- Permitir al usuario crear carpetas por las que no quieran que el análisis entre
- Personalizar la interfaz del usuario:
 - Cambio entre temas claro y oscuro
 - Cambiar la **paleta de colores principal** entre varias opciones
- Dar la opción de tener la aplicación en **varios idiomas**, convirtiéndola en una aplicación multilingüe.

Usuarios

Solo se contempla un tipo de usuario, debido a que no se puede administrar ningún dato vulnerable desde dentro de la aplicación y no hay ninguna función exclusiva que requiera algún tipo de pago.

Sistemas con los que la aplicación interactúa

- Bases de datos en red (por medio de un servicio web)
- Internet (debido a que las imágenes de usuario son enlaces directos a internet)
- El propio sistema operativo (específicamente su sistema de archivos)

4.2 Métodos, técnicas y arquitecturas

La estructura principal de la aplicación se basa en la arquitectura **MVVM** (Model-View-ViewModel). El código se organiza en tres bloques:

- **Vistas:** Son todas las pantallas y widgets con los que cuenta la aplicación. En este bloque sólo aparece el código con el que el usuario interactúa directamente desde su dispositivo.
- **Modelos:** Son las clases destinadas a guardar datos de la aplicación. El usuario no puede editar sus propiedades manualmente, sólo puede ver y manejar su información en la vista.
- **ViewModel:** Son las clases que controlan el estado de las vistas, y contienen funciones que actúan sobre los modelos y cómo aparecen estos en las vistas. Estas clases son controladas en Flutter como Proveedores o Providers. Son clases que se ligán al contexto de la aplicación y, por medio de sus funciones, guardan y actualizan valores que se pueden utilizar a lo largo de todo el ciclo de vida de la aplicación.

A parte del MVVM como arquitectura principal, hay varios tipos de clases a tener en cuenta:

- **Objetos de Acceso a Datos (DAO):** Se caracterizan, como su nombre indica, en guardar funciones para acceder a las bases de datos y realizar sus operaciones CRUD. En el caso de esta aplicación, todas estas clases implementan una interfaz creada llamada DAO, que se caracteriza por tener dos tipos genéricos, uno que marca el objeto modelo de cada tabla de la base de datos, y otro, de tipo primitivo, que representa la clave primaria de los datos. Las funciones de inserción, actualización y eliminación devuelven una booleana que marcará si la operación ha tenido éxito o no.
- **Archivos de idioma:** La aplicación está diseñada para cumplir con los criterios de internacionalización. Es decir, se adapta a distintos idiomas, cambiando los textos según el que el usuario seleccione.

```
abstract class DAOInterface<T, V>{  
  //Función de creación en BD  
  Future<bool> insert(T item) async{  
    return true;  
  }  
  //Función de actualización en BD  
  Future<bool> update(T item) async{  
    return true;  
  }  
  //Función de obtención en BD  
  Future<T?> get(V value) async{  
    return null;  
  }  
  //Función de listado en BD  
  Future<List<T>> list() async{  
    return List.empty();  
  }  
  //Función de borrado en BD  
  Future<bool> delete(T item) async{  
    return true;  
  }  
}
```

Para gestionar esto, Dart utiliza la librería **l10n**, que permite definir textos en varios idiomas mediante archivos .arb (Application Resource Bundle).

Por cada idioma soportado, se crea un archivo de recursos con el nombre “app_{código del idioma}.arb”, y su estructura es similar a un archivo JSON. En estos archivos se definen pares clave-valor: la clave debe estar presente en cada lo archivo de idioma y su valor será la frase traducida correspondiente.

- **Clases de Utilidad (Servicios):** Estas clases se utilizan para iniciar servicios o funciones de la aplicación, y también podrán contener atributos estáticos para que se pueda acceder a ellos desde cualquier lado de la aplicación.

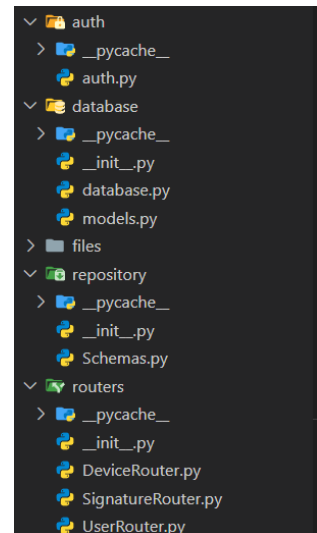
En cuanto a la API, esta sigue el modelo de la Arquitectura en Capas. Esta se caracteriza por ser una de las más comunes a la hora de crear una API, ya que divide y diferencia cada capa de la infraestructura de esta.

El código se divide en 3 grupos:

- **Routers:** Representan la capa de *presentación*, y definen las funciones que se llaman desde distintas peticiones HTTP en función de su ruta definida. Sirven para recibir las peticiones del usuario, acceder a la capa de servicios y devolver la respuesta de dicha capa.
- **Servicios:** Representan la capa de *lógica* o *servicios*, y definen las funciones y objetos necesarios para acceder a la capa de datos. En este caso, como solo hay un servicio, que es el de acceso a la base de datos, he llamado a la carpeta directamente “database”, que guarda el archivo database, con los métodos de apertura, cierre y llamada a la base de datos.
- **Modelos:** Representa la capa de *modelos*, y en este caso se divide en dos subgrupos:
 - **Modelos de la API (Schemas):** Por medio de la librería PyDantic, estos modelos sirven para validar y manipular los datos que se reciben y envían. Definen la estructura que los datos de entrada deben seguir en las llamadas y cómo va a devolver la API estas mismas.
 - **Modelos de la Base de Datos (Models):** Por medio de la librería SQLAlchemy, y siguiendo la estructura SQL del gestor MySQL, estos modelos sirven para representar cada tabla de la base de datos a la que se va a acceder y permiten con más facilidad realizar las operaciones CRUD. Todos los atributos tienen el atributo `__tablename__`, que representa el nombre de la tabla a la que SQLAlchemy tiene que acceder, y cada atributo (de tipo Column), guarda el tipo de dato de cada columna de la tabla.

Aparte de esta estructura, también he implementado un sistema de **autenticación**, basado en JSON Web Token. La autenticación está directamente ligada a la base de datos. El usuario recibirá su Token para acceder a todas las funciones de la API usando su nombre de usuario y contraseña.

Con todas estas estructuras mencionadas, el aspecto final de la aplicación sería el que se muestra a la derecha, donde los modelos de la base de datos se guardan con su servicio, la autenticación se encuentra en una carpeta independiente y los esquemas están en una carpeta de repositorios. Cada carpeta cuyas clases van a ser utilizadas desde otros sitios cuenta con un archivo `__init__.py` para iniciar sus clases automáticamente al ejecutar el archivo `main.py`



4.3 Despliegue de la aplicación

El despliegue de toda la aplicación se divide en 3 apartados distintos:

4.3.1 Despliegue de la Base de Datos en Red (FreeDB)

La base de datos se encuentra alojada en FreeDB, un servicio web que permite crear bases de datos relacionales con el gestor MySQL de forma totalmente gratuita. Si esta aplicación tuviera algún tipo de fin lucrativo o fuera más profesional en ese aspecto, esta opción no sería muy viable, pero como en un principio de momento sólo tiene fines educativos, es una buena alternativa para poder acceder a todos los usuarios, dispositivos y los hashes de archivos desde cualquier lugar.

Para crear la base de datos, fue necesario crear y verificar una cuenta gratuita, con la que nos daba acceso a su servicio. Después, utilizando un script SQL personalizado (y guardado en la raíz del proyecto de la aplicación para tomar de referencia), creé las tablas 'User', 'Device' y 'Hash', e inserté todas las ocurrencias de las huellas, además de cuentas y dispositivos de prueba.

A partir de esto, todo lo que hizo falta fue cambiar el enlace a la base de datos en la API de la local a la nueva en red.

Hay que tener en cuenta la cantidad tan limitada de consultas con las que se cuenta, por lo que será totalmente necesario optimizar las consultas desde la API y desde la aplicación lo máximo posible.

4.3.2 Despliegue de la API (Vercel)

La API que accede a la base de datos está desplegada en Vercel, un servicio que da la posibilidad de cargar páginas y aplicaciones web directamente desde un proyecto de GitHub, además de que se recarga el servicio de forma automática tras actualizar el proyecto.

Para preparar el despliegue, fue necesario crear un archivo json llamado "vercel.json" y añadir las rutas para iniciar el archivo de Python, además de dejar en la raíz de forma obligatoria el archivo *requirements* con todas las librerías que utiliza la API.

Para desplegar la aplicación hizo falta subir la API a GitHub, hacer registro en Vercel con esa cuenta y seleccionar el repositorio a subir, del resto se encargará el propio servicio.

Cuando todo esté subido, Vercel proporcionará una url con el formato "**nombre-repositorio.vercel.app**". Ese es el host que hay que introducir para hacer todas las llamadas desde la aplicación.

Los repositorios de GitHub cuentan con un apartado de Lanzamientos o **Releases**, donde el creador y sus colaboradores puede subir archivos para que los demás usuarios puedan descargarlos. Normalmente se utilizan para desplegar las versiones nuevas de sus aplicaciones.

Como mi aplicación estará disponible para Linux, Windows y Android, construiré las aplicaciones de la siguiente forma:

- Para terminar, desplégue el lanzamiento de la última versión, subiendo los archivos de los sistemas operativos Linux, Windows y Android con sus respectivos nombres para identificarlos correctamente.

pág. 11

5 Planificación y presupuesto

La planificación será la siguiente:

- Lo principal será realizar un **Wireframe** de la aplicación para tener una idea principal de cómo quiero que sea el producto final, con sus distintas pantallas y widgets.
- Lo siguiente es comenzar a diseñar el **modelo UML** de todas las clases que iré creando, con los atributos que vea necesarios
- En función del modelo UML, habrá que diseñar los modelos de las bases de datos que se vayan a utilizar, teniendo en cuenta las relaciones que pueda haber entre estas y si es necesario o no tenerlas en red.
- Cuando se tenga el modelo EER de las bases de datos, habrá que trabajar en diseñar cada base de datos por separado
- Cuando se tenga todo, habrá que pasar a empezar a hacer código para la aplicación
 - Por una parte, hay que hacer la API que se utilizará para acceder a la base de datos, ya que Flutter tiene problemas en general con acceder a bases de datos en red por sí solo.
 - Por otra parte, también se empezará a crear la primera versión de la aplicación, comenzando con las clases Modelo, Controlador y las clases de Acceso a Datos, y luego ya pasar a las vistas.
- Según se vaya realizando la aplicación, habrá que ir cambiando partes de las ideas principales por incongruencias, mejoras o nuevas necesidades en el código.
- Una vez esté el primer prototipo, con todo lo esencial hecho y totalmente funcional, habrá que ir haciendo ampliaciones en la aplicación.

Con todo esto, deduzco que necesitaré cerca de 100 horas para realizar y completar mi trabajo final.

Lo bueno es que no hay presupuesto a gastar, debido a que ya se cuenta con todas las herramientas necesarias para hacer el código y se va a buscar que los servicios a mayores sean gratuitos. En caso de comercializar esta aplicación, si se viera que estos servicios gratuitos pueden tener algún problema de vulnerabilidad, habría que cambiarlo.

6 Documentación Técnica

6.1 Especificación de requisitos

Los requisitos mínimos para asegurar el funcionamiento de la aplicación se dividen en requisitos funcionales y no funcionales, dependiendo de si se habla de qué hace la aplicación o cómo funciona ésta respectivamente:

- Requisitos Funcionales
 - Debe ser capaz de analizar los archivos del sistema operativo en el que se encuentra instalado
 - Debe conectarse a una base de datos que guarde hashes de software malicioso.
 - Debe contar con un directorio donde almacenar los archivos en cuarentena, además de alguna forma dentro de la aplicación para borrarlos o restaurarlos.
- Requisitos No Funcionales
 - La aplicación estará disponible y funcional para, al menos, Windows, Linux y Android.
 - Tendrá una interfaz gráfica intuitiva y accesible para los usuarios.
 - Debe poder acceder a internet y tener los permisos necesarios del dispositivo para funcionar correctamente.
 - Estará optimizada, buscando no utilizar muchos recursos del dispositivo mientras realiza su análisis.

6.2 Análisis del sistema

La aplicación tiene el funcionamiento de cualquier sistema antivirus que funciona analizando huellas digitales de archivos de una base de datos. Su funcionamiento se divide en dos partes: El uso del usuario, basado en el FrontEnd, y el trabajo interno de la aplicación, reflejado en el BackEnd:

- Gestión Visual e Interfaz de Usuario
 - La aplicación ofrece una interfaz al usuario donde puede observar todas las funciones del antivirus, además de controlar su cuenta de usuario.
 - El usuario puede controlar si se está ejecutando un escaneo en ese momento o no, además de poder iniciarlo desde la raíz de la aplicación, desde una carpeta específica, o incluso detenerlo.
 - El usuario puede añadir o borrar carpetas a las que no quiere que la aplicación acceda
 - El usuario puede elegir si borrar o restaurar archivos confiscados por el software.
- Funcionamiento Interno de la Aplicación
 - Accede a los archivos internos del sistema de forma recursiva desde la raíz o una carpeta que previamente haya elegido el usuario, lee los bytes de cada archivo y los encripta en md5. Busca en una base de datos alguna coincidencia con el código encriptado y, si encuentra una coincidencia, encripta ese archivo y lo guarda en un directorio de archivos en cuarentena.
 - Durante el análisis, si la aplicación encuentra la ruta a alguna carpeta a la que se le haya denegado el acceso, simplemente la saltará y continuará analizando el siguiente elemento.
 - Si el usuario decide restaurar un archivo. El programa buscará la ruta en su base de datos local, desencriptará el código y lo reescribirá en un fichero en la ruta de origen.

6.3 Diseño del sistema



6.3.1 Diseño de la Base de Datos

Hay dos bases de datos en toda la aplicación, independientes la una de la otra, y cada una con una estructura distinta:

6.3.1.1 Base de Datos Local

Esta base de datos tiene como objetivo guardar datos específicos del dispositivo, los cuales son:

- Carpetas de Acceso Prohibido: Las carpetas a las que el usuario no quiere que el antivirus entre para analizar. Se guarda su nombre y su ruta, además de un identificador de cada una para que la base de datos sepa cuál editar o borrar en caso de haber varias iguales.
- Archivos en Cuarentena: Son los archivos que han dado algún tipo de problema a la hora de ser analizados (El código encriptado coincidía con uno de los guardados de la base de datos). Se guarda el nombre y ruta del archivo antes de ser puesto en cuarentena, el nuevo nombre (encriptado para evitar su uso), y la ruta donde se encuentra en cuarentena.

forbFolders		files	
	id INTEGER		id INTEGER
	name TEXT		name TEXT
	route TEXT		route TEXT
			newName TEXT
			newRoute TEXT
			malwareType TEXT
			quarantineDate DATE

Powered by yFiles

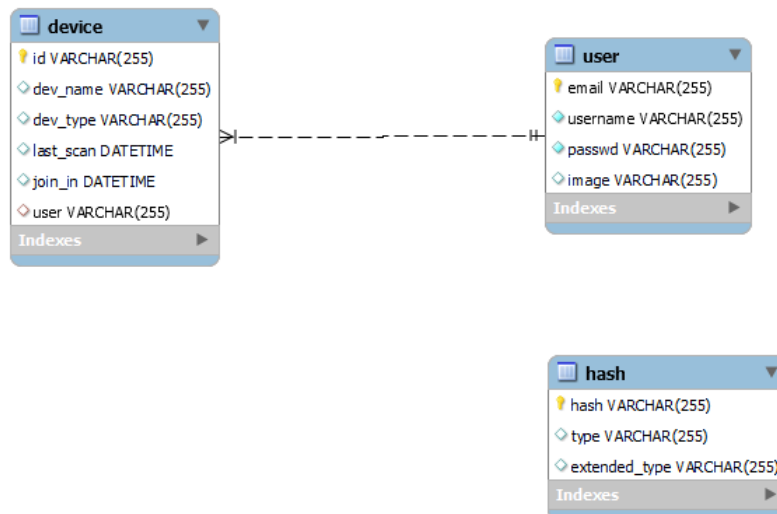
No hay ningún tipo de conexión entre estas dos tablas, pero SQLite es la mejor alternativa para guardar datos en una base de datos en el propio dispositivo y así no tener que guardar este tipo de información en red.

6.3.1.2 Base de Datos en Red

Esta tiene como fin guardar información que se quiere guardar y reflejar en todos los dispositivos.

Consta de tres tablas:

- Dispositivo (Device): Guarda los dispositivos con los que se ha accedido a la aplicación en algún momento. De estos guarda su identificador encriptado (El identificador varía en función del dispositivo. Por ejemplo, en Windows y Linux es el identificador de la máquina física, mientras que en Android es el número de serie), el nombre del dispositivo, el sistema operativo (dev_type), la fecha en la que se unió por primera vez a la aplicación, la fecha de su último escaneo y el usuario que tiene su sesión iniciada actualmente.
- Usuario (User): Guarda los usuarios que tienen acceso a la aplicación. De estos se guarda su correo electrónico, su nombre de usuario, su contraseña y el enlace a una imagen para poner en su perfil.
- Huella (Hash): Guarda el código encriptado de las aplicaciones que han sido previamente detectadas como software malicioso. Se guarda su huella como clave primaria (No hay dos malwares con la misma huella), el tipo de malware del que se trata y un tipo extendido (un nombre clave de éste). Esta tabla no guarda ninguna relación con las otras dos que aparecen en la base de datos, y lo más óptimo sería guardarla en una base de datos aparte, pero para este proyecto, por hacerlo más sencillo, decidí guardarlo en la misma base de datos y acceder a ello desde la misma API.



6.3.2 Diseño de la Interfaz de usuario.

Para el diseño de esta aplicación se utilizó Figma, una herramienta que, entre otras funciones, permite crear maquetas visuales de interfaces de aplicación. Estas maquetas pueden incluso contener lógica y navegación pese a ser una herramienta que no necesita escribir código. Fue utilizada para idear el diseño principal de la aplicación tal y como se presenta a continuación:

6.3.2.1 Login

En el menú de Inicio de Sesión, el usuario verá una Card con distintos campos: Dos TextFields y dos botones.

Cuando el usuario pulsa el botón de inicio de sesión, el programa verificará los campos de usuario y contraseña:

- Si están vacíos, dará error
- Si tienen datos, buscarán un usuario en la base de datos que coincida con el email o el nombre dado.
 - Si no hay ocurrencias en la base de datos, mostrará un error de que el usuario no existe
 - Si hay ocurrencias, mira a ver si la contraseña es correcta
 - Si no es correcta, mostrará un error de contraseña
 - Si es correcta, enviará al usuario a la próxima página: **La Vista Principal**



The mockup shows a login interface with a dark blue header containing the text 'Iniciar Sesión'. Below the header is a white card with the title 'Introduzca sus Credenciales'. The card contains two text input fields: 'Usuario o Correo' and 'Contraseña'. Below these fields are two buttons: a dark blue 'Iniciar Sesión' button and a white 'Registrarse' button with a dark blue border. The card is set against a dark blue background.

6.3.2.2 Pantalla Principal

En esta pantalla, el usuario verá de forma constante el título de la pestaña en la que se encuentra y una barra de navegación inferior con la que puede ir a distintas páginas de la aplicación.

Dependiendo de qué pestaña esté seleccionada en el menú inferior, el contenido del cuerpo (body) de la vista será distinta



6.3.2.2.1 Analizar

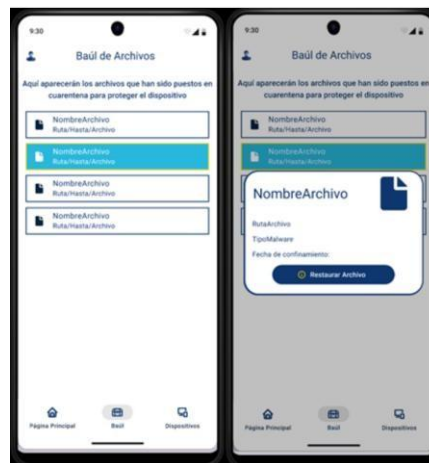
En la vista de análisis, el usuario de primeras verá un botón grande donde pone “Analizar” y debajo un texto que marca la última vez que se ha escaneado el dispositivo. Una vez le dé a este botón, el programa empezará a escanear todos los archivos del equipo en el que se encuentre, empezando desde la raíz del sistema de ficheros. Para hacer más amena la espera al usuario, éste verá un círculo girando y un texto que avisa de qué parte del proceso está llevando a cabo el programa, además decir que **se puede abandonar esta pestaña sin ningún problema**, ya que no afectará al funcionamiento del análisis.



6.3.2.2.2 Baúl

En esta sección, el usuario verá una lista de ficheros con nombre y ruta. Estos ficheros son los que, tras analizar el equipo, no han pasado por los requisitos de seguridad del programa y, para proteger al usuario, los ha puesto en cuarentena.

Si el usuario pulsa sobre uno de ellos, aparecerá un pop up con información del archivo, como la ruta en la que se encontraba, el malware detectado y otros. Si pulsa al botón de Restaurar archivo, este desaparecerá de la lista y el programa lo sacará de su cuarentena.



6.3.2.2.3 Dispositivos

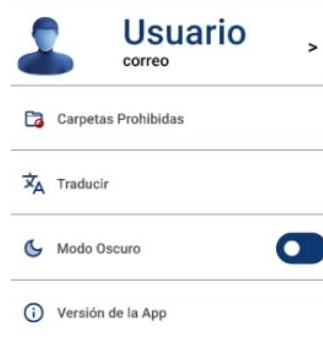
Esta vista es meramente informativa. Muestra al usuario todos los dispositivos que tiene vinculados a su cuenta. Puede ver el tipo de sistema operativo que tiene, el nombre del dispositivo y tanto la fecha de su último escaneo como la fecha de registro en esta cuenta.



6.3.2.3 Menú Lateral

En la barra superior del menú principal, aparece un icono con la imagen del perfil del usuario. Si se pulsa ahí, aparecerá el Drawer con distintos widgets:

- El primero es la cabecera del Drawer, que muestra la foto de perfil del usuario, su nombre y su correo. Si el usuario pulsa en el widget, accederá al menú de usuario
- El segundo apartado es un tile con la imagen y el texto de carpetas prohibidas. Si se pulsa, el programa navegará hacia la vista de carpetas prohibidas
- El tercero es un tile con una imagen y un texto de “Traducir”, al final del tile aparecerá un DropDown que mostrará distintos idiomas a traducir. Cuando el usuario elija un idioma, todos los textos del programa cambiarán al idioma elegido
- El cuarto es un tile para cambiar el modo de claro a oscuro y viceversa. Con el switch que hay al final del tile, el usuario puede alternar entre los modos claro y oscuro
- El quinto es un tile que, si el usuario lo pulsa, accederá a una ventana con toda la información de la versión actual del programa



6.3.2.4 Menú de Usuario

Cuando se entre aquí, el usuario podrá ver todos sus datos: El nombre, email, su fecha de unión, el número de dispositivos conectados y todos los escaneos que ha hecho en total a lo largo de su experiencia con el programa.

Aparte de esto, tiene una selección de botones para cambiar varios parámetros, como el nombre y la contraseña. **La foto de perfil se puede cambiar si se pulsa en el icono de la imagen.**

Además de esos cambios, se puede cerrar sesión y borrar la cuenta con otros botones que se ven a continuación.



6.3.2.5 Carpetas Prohibidas

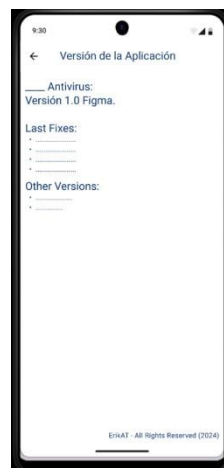
Cuando el usuario entra en esta pantalla, se carga una lista con todas las carpetas por las que, por el motivo que sea, no interesa que el programa pase. Si el usuario pulsa al botón de añadir carpeta, cargará el explorador de ficheros del sistema operativo y permitirá seleccionar la carpeta.

Las carpetas aparecen listadas según cuándo se añadieron (van por un id auto incrementado). El usuario puede borrar las carpetas de la lista dando al icono de borrar en el tile de la carpeta.



6.3.2.6 Información de versiones

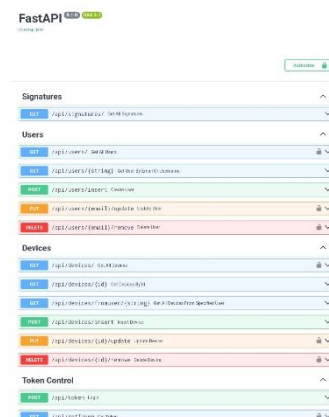
Cuando el usuario pulsa en el Drawer la opción de "Versión de la App", cargará una ventana con los datos de la versión actual, además de cargar la información de versiones anteriores. Es meramente informativo, el usuario no tiene que hacer nada en esta ventana



6.3.3 Diseño de la Aplicación.

6.3.3.1 API

La API, al ser un servicio web para hacer peticiones a una base de datos, no cuenta con una interfaz gráfica. Sin embargo, gracias a la auto implementación de Swagger en la dependencia de FastAPI, se puede acceder a una interfaz predeterminada en la red con todas las rutas y métodos HTTP. En esta página web se puede también ver las peticiones que están bloqueadas por un medio de autenticación, pudiendo conseguir acceso desde la interfaz a estas peticiones dando al botón “Authorize” e introduciendo las credenciales de usuario oportunas.



6.3.3.2 Aplicación Móvil

Al contrario que la API, la aplicación destinada a los usuarios tiene una interfaz visual distribuida en distintas pantallas conectadas entre ellas:

- La aplicación comienza con una **pantalla de carga** que carga los datos iniciales de la aplicación por detrás. La siguiente pantalla puede ser o la de inicio de sesión o la pantalla principal, dependiendo de si se ha iniciado sesión o no previamente.
- La pantalla de Inicio de Sesión tiene dos botones: Uno de inicio de sesión, donde envía al usuario a la página principal si sus credenciales son correctas, y uno de registro, que abre un diálogo para introducir datos para crear la cuenta. Si todo está correcto, redirecciona al usuario a la página principal
- En la pantalla principal, hay tres fragmentos que se pueden visualizar:
 - o Pantalla de Análisis, con dos botones para analizar desde la raíz o desde un directorio respectivamente. Si el análisis está activo, aparece una rueda de carga y un botón para cancelar el análisis
 - o Baúl de archivos: Aparecen los archivos locales que se encuentran en cuarentena. Se puede seleccionar de uno en uno para borrarlos o restaurarlos, o se pueden seleccionar varios a la vez para hacer la misma operación en todos.
 - o Dispositivos: Aparece una lista de dispositivos vinculados a la cuenta. Si se mantiene pulsado en uno que no sea el dispositivo actual, se da la opción de desvincularlo.
- Desde el menú superior se puede abrir un menú de navegación lateral, que da a elegir entre gran variedad de pantallas, además de las opciones de cambiar idioma y tema:
 - o Opciones de Usuario, con todo tipo de operaciones de cambio de datos, incluyendo un cierre de sesión y una cuenta
 - o Carpetas Prohibidas, donde el usuario puede introducir o borrar carpetas que no quiere que sean analizadas.
 - o Versiones de la aplicación, donde se lleva un control de qué ha sido añadido en cada versión.

6.4 Implementación:

6.4.1 Entorno de desarrollo.

El entorno de Desarrollo que he utilizado para hacer todo el proyecto ha sido Visual Studio Code, ya que permite bastante flexibilidad para trabajar en cualquier lenguaje. Para trabajar más cómodamente tanto en Dart como en Python, he descargado sus dos extensiones oficiales, las cuales ofrecen soporte para el lenguaje, corrección de errores y snippets para hacer la programación más sencilla.

En cuanto a los lenguajes utilizados para la aplicación, Dart y Python, me he servido de bastantes dependencias para generar la aplicación. A continuación, hablaré de las principales dependencias de cada uno de los lenguajes, dejando de lado las que quizá sean más estéticas y menos necesarias para el producto final:

- Python
 - **FastAPI:** Framework utilizado para construir la API.
 - **PyDantic:** Librería para verificar las correctas propiedades de los datos de entrada y salida.
 - **MySQL Connector:** Conector de bases de datos de MySQL en Python.
 - **SQLAlchemy:** Librería que permite seguir la estructura ORM de cualquier gestor de bases de datos SQL en Python.
 - **Python-Jose:** Librería que permite cifrar tokens.
 - **PyJWT:** Librería que codifica y decodifica JSON Web Tokens en Python.
- Dart
 - **Flutter:** Framework de desarrollo de interfaces gráficas de aplicaciones con Dart.
 - **SQLite & SQLite_Commons_FFI:** Permite acceder a bases de datos locales de SQLite en Dart.
 - **Provider:** Permite gestionar el estado de Flutter y compartir recursos entre pantallas.
 - **File_Picker:** Añade la posibilidad de seleccionar archivos del sistema.
 - **Shared_Preferences:** Guarda datos de forma persistente en el dispositivo en formato par clave-valor.
 - **Permission_Handler:** Permite gestionar los permisos del sistema operativo.
 - **Crypto:** Permite hacer operaciones de encriptación en muchos formatos distintos.
 - **Logger:** Permite generar mensajes en el Log de la aplicación para verlos en modo desarrollo.
 - **Device_Info_Plus & Device_Name:** Acceden a la información del hardware y software interno del dispositivo.
 - **Flutter_Background_Service:** Permite ejecutar acciones en la aplicación aunque esta se encuentre en segundo plano.
 - **Cancelable_Compute:** Añade a los métodos compute, utilizados para crear hilos, una función para poder detenerlos de forma más sencilla.

6.4.2 Estructura del código.

La estructura de todo el código se divide entre la estructura que ha tenido la API y la aplicación.

6.4.2.1 Estructura de la API

Como ya se ha dicho previamente, la API se basa en la arquitectura por capas. Estas capas se reflejan en los modelos, los routers y el servicio de base de datos, además del servicio de autenticación.

6.4.2.1.1 Estructura de los Modelos

Los modelos son los objetos tanto de la base de datos como de la propia API, estos últimos creados con Pylance para evitar errores en la transferencia de datos entre cliente y servidor. Ambas clases sólo cuentan con la declaración de sus atributos, no cuentan con ningún tipo de función.

Modelos Pylance (Schemas):

Todos los modelos implementan la heredan de la clase abstracta BaseModel, proveniente de la librería de Pylance. Las declaraciones de sus objetos siguen el formato 'nombre:tipo = defecto', además de poder añadir Optional para decir que el atributo puede ser nulo.

Ejemplo con Device:

```
class Device(BaseModel):
    id:str
    dev_name:str
    dev_type:str
    last_scan:datetime
    join_in:datetime
    user:Optional[str] = None
```

Modelos SQLAlchemy (Models):

Todos ellos tienen en común que heredan de la clase abstracta Model, proveniente de la librería de SQLAlchemy, y sus declaraciones son nombre = Column({tipo de dato de columna}, [primary_key = boolean, name = str, ForeignKey('id de la tabla de referencia')]). También cabe recalcar que todas las tablas declaradas deben tener el atributo __tablename__, con el nombre de la tabla a la que hacen referencia, y todos los atributos de la clase tienen que aparecer reflejadas como columnas en la base de datos, como muestra el modelo a continuación:

```
class Device(Base):
    __tablename__ = "device"
    id = Column(VARCHAR(255), primary_key=True)
    dev_name = Column(VARCHAR(255))
    dev_type = Column(VARCHAR(255))
    last_scan = Column(DateTime)
    join_in = Column(DateTime)
    user = Column(VARCHAR(255), ForeignKey("user.email"))
```

6.4.2.1.2 Estructura de los Routers

Cada router tiene un objeto creado de tipo `APIRouter`, el cual contiene un prefijo con el cual empiezan todas las llamadas a la API que se encuentren guardadas en ese fichero. Además, para ordenarlos en la interfaz de Swagger, guardan el atributo `tags`, que muestra en la documentación de la API este nombre en la colección de todas las llamadas con este prefijo.

```
router = APIRouter(prefix="/api/devices", tags=["Devices"])
```

A partir de ahí, la estructura de todas las funciones API comienzan de la misma forma:

1. Se define la llamada a la API con `@router.get("prefijo de la API, metiendo variables entre llaves")`
2. Se define la función, poniendo en parámetros tanto los `BaseModel` que se reciban por el cuerpo de la API (si los hay), como los valores del prefijo, y además llamadas tanto a la declaración de la base de datos y un método de autenticación.
3. Se realiza el resto de la función.

Un ejemplo visual puede ser la función de obtención de un usuario por su correo y su contraseña:

```
@router.put("/{email}/update")
def update_user(email:str, user:Schemas.User, db:Session = Depends(get_db), token:str =
Depends(auth.oauth2_scheme)):
    usuario = db.query(models.User).filter(models.User.email == email).first() # Usuario
en la BD
    if usuario: # Si el usuario existe, se actualizan los valores utilizando el cuerpo de
la petición
        usuario.email = user.email
        usuario.passwd = user.passwd
        usuario.username = user.username
        usuario.image = user.image
        db.commit() # Se actualizan los valores
        return usuario # Se devuelve el usuario
    else: # Si no encuentra el usuario, se lanza una excepción
        raise Exception("Usuario no encontrado")
```

La API recibe por la url el email del usuario y por el cuerpo de la API el esquema con los datos a cambiar de éste. Luego, se conecta a la base de datos (`db:Session = Depends(get_db)`) y verifica si el token que ha pasado el usuario es correcto (`token = Depends(auth.oauth2:scheme)`).

Si no da problemas ni la conexión a la base de datos ni la autenticación del Token, se recorre el código de la API, en este caso verificando que exista el usuario y cambiando su información en la base de datos si existe.

6.4.2.1.3 Servicio de Base de Datos

Lo primero que se hace en el fichero de acceso a la base de datos es declarar la URL para acceder a la base de datos. Esta URL contiene para empezar el método de conexión (la dependencia utilizada junto a SQLAlchemy, en este caso mysql y mysql-connector), seguida del nombre de usuario y contraseña de un administrador de la base de datos, el host donde se encuentra la base de datos y el nombre de ésta.

```
DBUSER = "" # Nombre de usuario de la BD
DBPASSWORD = "" # Contraseña de la BD
DB_HOST = "" # Host en el que se encuentra la BD
DB_NAME = "" # Nombre de la BD a la que se va a conectar
SQLALCHEMY_DATABASE_URL =
f"mysql+mysqlconnector://{DBUSER}:{DBPASSWORD}@{DB_HOST}/{DB_NAME}"
```

Después, se crea el motor de arranque de la base de datos, un generador de sesiones y la base declarativa que, de forma implícita, va a cargar los modelos del ORM.

```
# Motor de conexion de la bd
engine = create_engine(SQLALCHEMY_DATABASE_URL)
# Crea el generador de sesiones de la BD
sessionLocal = sessionmaker(bind=engine, autocommit=False, autoflush=False)
# Modelos de la BD
Base = declarative_base()
```

Para terminar, se declara una función encargada de crear una sesión de la base de datos, unirla al hilo principal y, cuando se termine de utilizar, se cierra la sesión.

```
def get_db():
    db = sessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Esta última función es la que va a ser utilizada en las llamadas de la API, declarándola en los parámetros de las funciones dentro de Depends, dando a entender a la llamada que, si esta función falla, debe lanzar una excepción y, con ello, el error 500: Internal Server Error.

6.4.2.1.4 Autenticación

La autenticación no deja de ser un router, pese a servir para otorgar a los usuarios una forma de acceder a los recursos de la API. Por ello, empieza con un objeto APIToken, aunque sigue con un objeto Bearer, utilizado para marcar a la API que esa llamada a la API sirve para crear tokens.

```
router = APIRouter(prefix="/api", tags=["Token Control"])
oauth2_scheme = OAuth2PasswordBearer("/api/token")
```

Como estamos utilizando la librería jwt, la encriptación de los archivos utiliza una clave secreta de encriptación y un algoritmo para encriptar los archivos. Puesto que es uno de los que más hemos utilizado, elegí como algoritmo el SHA256, o HS256 para jwt.

```
SECRET_KEY = ""
ALGORITHM = "HS256"
```

La siguiente función declarada fue la de creación de tokens, que recibe un diccionario con información y genera la token con la clave secreta y el algoritmo seleccionados.

```
# Función de creación de tokens
def create_token(data: dict):
    token = jwt.encode(data, SECRET_KEY, algorithm=ALGORITHM)
    return token
```

Ahora, se diseñó esta función que recibe datos de un formulario en formato x-www-url-encoded, y devuelve el token si encuentra un usuario y su contraseña en la base de datos. Además, se diseñó una función para conseguir la token directamente a través de esta verificación previa:

```
@router.post("/token")
def login(form_data: OAuth2PasswordRequestForm, db:Session = Depends(get_db)):
    pass_encoded = hashlib.sha256(form_data.password.encode()).hexdigest()
    # Se encripta la contraseña y se usa en el filtro de la query
    user = db.query(User).filter(and_(User.email == form_data.username, User.passwd ==
    pass_encoded)).first()
    if user:
        token = create_token(data={"sub":user.email})
        return {
            "access_token": token,
            "token_type":"bearer"
        }
@router.get("/getToken")
def get_token(token:str = Depends(oauth2_scheme)):
    return token
```

Para terminar, hacía falta decidir qué funciones iban a estar protegidas por el protocolo de autenticación, debido a que no se instalaba de por sí solo en toda la API. Aquellas que estuvieran protegidas por JWT deberían llamar a la función get_token, asegurando su correcta inicialización con Depends.

6.4.2.2 Estructura de la Aplicación

La estructura de la aplicación es más compleja que la de la API, ya que tiene bastantes más factores:

6.4.2.2.1 Inicio de la aplicación

El inicio de la aplicación, mostrado en el método principal del archivo main, es el siguiente:

1. La aplicación se asegura de que todos los widgets de flutter carguen correctamente. Esto se hace cuando el inicio de la ejecución de la aplicación cuenta con métodos asíncronos, como pasa en este ejemplo.
2. **Se cargan las preferencias** de la aplicación, debido a que los proveedores de datos las necesitan para funcionar correctamente.
3. **Comienza la carga de todo el contenido de la aplicación** en uno de los proveedores. Se realiza antes de iniciar la aplicación para asegurar que lo primero que se ve en la aplicación no es una pantalla de negro esperando a que todo cargue en el hilo principal, sino una pantalla de carga mostrando lo que actualmente se está realizando.
4. **Inicia la aplicación**, con todos los proveedores preparados.

```
void main() async {  
  WidgetsFlutterBinding.ensureInitialized(); //Asegura que los widgets se inicializan  
correctamente  
  await AppEssentials.getPreferences(); // Carga las preferencias del usuario en la  
aplicación.  
  UserDataProvider provider = UserDataProvider();  
  provider.loadAssets(); //Comienza la carga de todos los datos de la aplicación.  
  runApp(MultiProvider( //Inicia la interfaz gráfica.  
    providers: [  
      ChangeNotifierProvider(create: (context) => provider),  
      ChangeNotifierProvider(create: (context) => StyleProvider()),  
      ChangeNotifierProvider(  
        create: (context) => AnalysisProvider(),  
      ),  
      ChangeNotifierProvider(create: (context) => LanguageNotifier())  
    ],  
    child: MainApp(),  
  ));  
}
```

6.4.2.2.2 Funcionamiento del análisis

El escaneo de archivos se realiza por medio de un conjunto de funciones, donde destaca la función recursiva **scanDir()**. Esta función se llama a sí misma cada vez que, a la hora de analizar, se encuentra una carpeta nueva, y contiene parámetros a mayores de los que utiliza en su interior, como son la lista de archivos que han sido detectados como malware, y la lista de hashes para detectarlos.

Su funcionamiento es el siguiente: Comienza comprobando si la carpeta actual pertenece a las carpetas cuyo acceso ha sido restringido por el usuario. De no ser así, la aplicación lista todos los recursos guardados en su interior.

- Si es un directorio, la función se llama a sí misma, esta vez pasando el nuevo directorio por parámetro.
- Si es un fichero, la función llama a **scanFile()**, encargada de analizar ese archivo específico.

```
static Future<void> scanDir(Directory dir, List<String> hashList,
    List<String> folders, List<Map<String, String>> files) async {
    if (!folders.contains(dir.path)) {
        try {
            await for (var f in dir.list(recursive: false)) {
                if (f is File) {
                    await scanFile(f, hashList, files);
                } else if (f is Directory) {
                    await scanDir(f, hashList, folders, files);
                }
            }
        } catch (e) {}
    }
}
```

La otra función principal es **scanFile()**, que funciona de la siguiente forma:

Comienza leyendo los bytes del archivo, y posteriormente lo encripta con la función hash md5. Si encuentra en la lista de hashes alguna coincidencia, añade a la lista de archivos a poner en cuarentena al archivo, indicando el hash resultado.

```
static Future<void> scanFile(
    File f, List<String> hashList, List<Map<String, String>> files) async {
    try {
        var bytes = await f.readAsBytes();
        String s = crypto.md5.convert(bytes).toString();
        Logger().d("Hash: $s");
        if (hashList.contains(s)) {
            Logger().d("Archivo ${f.path} es un virus");
            files.add({
                "path": f.path,
                "hash": s,
            });
        }
    } catch (e) {}
}
```

6.4.2.2.3 Estructura de las clases de Vistas

Las vistas son todas las pantallas, subpantallas y widgets de la aplicación. Pueden ser widgets sin estado (Stateless Widgets, su aspecto no varía) o widgets con estado (Stateful Widgets y States, con aspecto y contenido variables con el tiempo).

La estructura, independientemente de si tienen estado es igual:

- Declaración de todos los atributos que vaya a utilizar la vista.
- **Método sobrecargado build()**, en el cual se cargan todos los widgets en función del contexto de la aplicación y se preparan para ser mostrados en el dispositivo.
- **Resto de funciones** que vayan a aparecer en la vista actual, incluyendo el método **initState()** en los widgets con estado, que permite precargar información y alterar el estado al iniciar la vista. Esto se utiliza mucho para cargar objetos de bases de datos.

6.4.2.2.4 Estructura de las clases Modelo

Los modelos son las distintas clases que se van a utilizar en la aplicación. Salvo unas excepciones que se explicarán posteriormente, estas clases únicamente contienen la declaración de sus atributos y el constructor. Por ejemplo, esta es la estructura de la clase de las carpetas de acceso restringido (ForbFolder):

```
class ForbFolder {  
    int? id; ///Identificador  
    String name; ///Nombre de la carpeta  
    String route; ///Ruta de la carpeta  
    ForbFolder({this.id, required this.name, required this.route});  
}
```

6.4.2.2.5 Estructura de las clases ViewModel (Proveedores)

Los proveedores sirven para compartir recursos y tener permanencia de datos a lo largo de todas las pantallas de la aplicación. Para hacer esto, cuentan con unos escuchadores que se pueden actualizar (o notificar) cuando hay un cambio en éstos.

La estructura de los proveedores es:

- **Declaración de variables del proveedor**, exactamente igual que la declaración en las clases normales
- **Creación** de funciones que editen las variables previas y notifiquen a los escuchadores.

Un ejemplo puede ser el de la variable User, del proveedor UserDataProvider:

```
User? thisUser;  
void changeUser(User user) {  
    thisUser = user;  
    notifyListeners();  
}
```

6.4.2.2.6 Estructura de Clases DAO y APIContent

Aparte de las clases modelo, se utilizan las clases DAO, las cuales sirven para hacer todas las operaciones CRUD de las bases de datos, independientemente si son locales o en red. Para simplificar su creación, definí una interfaz DAO con todos los métodos que debían heredar de esta.

Esta interfaz utiliza dos tipos genéricos: T, que representa el tipo de objeto asociado a la tabla (por ejemplo, en una clase DAO de dispositivos, T sería la clase Device), y V, que representa el tipo de dato primitivo correspondiente al valor de la clave primaria de cada tabla.

```
abstract class DAOInterface<T, V>{  
    ///Función de creación en BD  
    Future<bool> insert(T item) async{  
        return true;  
    }  
    ///Función de actualización en BD  
    Future<bool> update(T item) async{  
        return true;  
    }  
    ///Función de obtención en BD  
    Future<T?> get(V value) async{  
        return null;  
    }  
    ///Función de listado en BD  
    Future<List<T>> list() async{  
        return List.empty();  
    }  
    ///Función de borrado en BD  
    Future<bool> delete(T item) async{  
        return true;  
    }  
}
```

Hay modelos que intercambian información con la API, y esta información se cambia consecutivamente entre mapas de pares clave valor e instancias de cada modelo. Por ello, definí una interfaz para implementar dos funciones: Una para transformar el objeto en mapa y la otra para transformar el mapa en una instancia de la clase.

```
abstract class APIContent{  
    ///Función de paso de objeto a mapa  
    Map<String, String?> toAPI(){  
        return <String, String?>{};  
    }  
    ///Función de paso de mapa a objeto  
    void toItem(Map<String, String> map){  
    }  
}
```

6.4.2.2.7 Estructura de Servicios

Principalmente hay 4 clases de Servicios en la aplicación, creadas con distintos métodos estáticos para facilitar su acceso y su uso a lo largo de todo el ciclo de vida la aplicación.

6.4.2.2.7.1 SQLiteUtils

Contiene las funciones de creación y acceso de las bases de datos locales. Además, contiene un objeto estático de tipo Database, que guarda la conexión a la base de datos, al que puede acceder todas las clases DAO que afectan al gestor SQLite.

Cuenta con dos funciones principales:

- **LoadDB()** -> Inicia el sistema de SQLite de flutter y accede al archivo de la base de datos.
- **StartDB()** -> Crea todas las tablas de la base de datos local en caso de que no se hayan creado previamente.

6.4.2.2.7.2 APIReaderUtils

Contiene las funciones para hacer de forma genérica peticiones GET, POST, SET y DELETE. Además, guarda dos Strings estáticas, una con el host de la API (apiRESTLink), y otra que guarda el token de autenticación. Al principio este último está vacío, pero desde la aplicación se llama a un método para obtener un token y guardarlo para su futuro uso.

Todas las funciones reciben por parámetros la URL de la petición HTTP y, si se envía algún objeto por el cuerpo de la petición, una instancia de un objeto de la interfaz APIContent para convertir en mapa:

- **getToken(String username, String password)** -> Guarda la Token con el usuario y la contraseña.
- **getData(Uri url)** -> Realiza la petición HTTP GET con la URL asignada
- **postData(Uri url, APIContent item)** -> Convierte el objeto en un mapa legible para la API y realiza una petición HTTP POST con la URL asignada y el mapa como cuerpo.
- **putData(Uri url, APIContent item)** -> Convierte el objeto en un mapa legible para la API y realiza una petición HTTP PUT con la URL asignada y el mapa como cuerpo.
- **deleteData(Uri url)** -> Realiza la petición HTTP DELETE con la URL asignada

6.4.2.2.7.3 AppEssentials

Esta clase, además de contar con métodos que se deben iniciar al inicio de la aplicación, tiene otros atributos y funciones que se ven utilizados tanto en otras pantallas como incluso en los subprocesos de la app.

- Atributos
 - **Prefs (SharedPreferences)** -> Las preferencias guardadas en el caché del dispositivo.
 - **EmailRegExp (RegExp)** -> Expresión regular utilizada en la verificación del DNI.
 - **QuarantineDirectory (Directory)** -> Directorio donde se va a guardar los archivos en cuarentena.
 - **ChosenLocale (Locale)** -> Idioma guardado en las preferencias de la aplicación
 - **IsLightMode (Boolean)** -> Booleana que indica si el modo es claro u oscuro, guardado en las preferencias de la aplicación.
 - **Dev (Device)** -> Dispositivo actual.
 - **Color** -> Color principal de la paleta de colores, guardado en las preferencias de la aplicación.
- Funciones
 - **getPreferences()** -> Carga la instancia de las preferencias de la aplicación y asigna sus respectivos valores
 - **newPreferences()** -> Inicia unas preferencias por defecto si en la aplicación no existen previamente.
 - **changeLang(String lang)** -> Guarda en las preferencias el idioma actual.
 - **saveColorPreferences(Color color)** -> Guarda en las preferencias los valores R, G y B del color elegido actualmente
 - **registerThisDevice()** -> Recopila información del dispositivo y llama a la API para verificar si ese dispositivo existe actualmente. Si es así, actualiza valores, y si no, crea la instancia en la base de datos.
 - **changeTheme(bool isLight)** -> Cambia el tema guardado en las preferencias en función de la booleana isLight
 - **loadHashes()** -> Carga y guarda de forma estática todas las huellas de la base de datos.
 - **getDevicesList(User user)** -> Recoge todos los dispositivos pertenecientes a un usuario de la base de datos.
 - **getOutOfQuarantine(QuarantinedFile s)** -> Borra el archivo de cuarentena, lo crea de vuelta en el repositorio de origen y borra la instancia de la base de datos local.
 - **eraseFile(QuarantinedFile s)** -> Borra el archivo de la carpeta de cuarentena y lo borra de la base de datos, pero no lo restaura.

6.4.2.2.7.4 ScanIsolate

Esta clase contiene todo lo relacionado a las operaciones que se realizan en el subproceso de análisis. Los subprocesos en Dart preferentemente tienen que ser estáticos, porque así no dependen en ningún momento del estado de la aplicación ni de ninguna clase específica donde éste se inicie, y así se evitan errores de ejecución.

Esta clase tiene dos atributos: **compute**, que guarda el subproceso en ejecución, y **receivePort**, que almacena un puerto de que permite comunicar el hilo principal con los subprocesos.

Además de los atributos, esta clase cuenta con los siguientes métodos estáticos:

- **startAnalysis(String? customDir, BuildContext context)** -> Dependiendo de si se ha pasado o no un directorio o no, pone de directorio raíz del análisis uno personalizado o la raíz del dispositivo (C:\ en Windows, /storage/emulated/0 en Android y / en el resto de sistemas compatibles). También procede a cargar la lista de directorios prohibidos y guarda sus rutas junto al directorio principal y el puerto de conexión en la lista de argumentos que recibirá el hilo.
Una vez está todo esto hecho, comienza el subproceso llamando a *scanDirInThread*, pasando los argumentos de los que dispone, y genera un listener en el puerto para que, cuando reciba todos los archivos infectados, los ponga en cuarentena.
- **scanDirInThread(List<dynamic> args)** -> Esta función es la que se inicia al comenzar el subproceso, y procede a guardar cada argumento o grupo de argumentos en sus respectivas variables (El directorio principal es el argumento 0, el puerto de conexión es el argumento 1 y del argumento 2 al final son los directorios restringidos).
- Tras esto, el programa carga los hashes de la API y comienza el proceso de análisis desde el directorio principal. Una vez termina, envía todos los archivos infectados al hilo padre para que los ponga en cuarentena.
- **scanDir(Directory dir, List<Map<String, String>> files, List<String> hashes, List<String> folders)** -> Esta función es la que va a tener cierta recursividad. Primero, se comprueba si el directorio padre no está dentro de los archivos prohibidos. Y si no lo está, comienza a analizar todos los recursos que tiene en su interior
 - Si el objeto que analiza es un directorio, recorrerá esta misma función, esta vez pasando por parámetros el directorio actual.
 - Si el objeto que analiza es un fichero, llama a la función **scanFile()**
- **scanFile(File f, List<String> hashes, List<Map<String, String>> files)** -> La aplicación lee y guarda los bytes del archivo y los encripta siguiendo el patrón md5. Acto seguido, busca alguna ocurrencia en la lista de hashes generada previamente. Si encuentra alguna ocurrencia, se guarda tanto la ruta al archivo como el patrón de la huella en la lista de archivos a confiscar.
- **setAllOnQuarantine(List<Map<String, String>> message)** -> Se llama desde el listener del puerto del hilo principal. Esta función ejecuta para cada mapa del mensaje el método *putInQuarantine*.
- **putInQuarantine(String fpath, String hash)** -> Borra el archivo de su directorio, lo encripta y lo guarda en una carpeta designada por el dispositivo. Acto seguido, guarda en la base de datos la información del archivo antes y después de ser puesto en cuarentena.
- **cancelScan(BuildContext context)** -> Si el proceso sigue en ejecución, lo detiene completamente.

6.4.3 Cuestiones de diseño e implementación reseñables.

6.4.3.1 Personalización de tema

Algo que quiero recalcar de mi aplicación es la personalización de la aplicación, debido a que, más allá de los modos claro y oscuro con los que cuenta, el usuario también tiene la oportunidad de cambiar el tema principal de la aplicación, pudiendo variar entre 7 colores distintos.

Esta personalización se ha logrado de la siguiente forma:

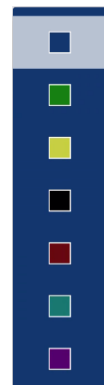
Comencé por crear una paleta de colores en mi fichero de estilos de la aplicación, donde todos los colores que quería que variaran fueran un “fundido” con el color principal. También, fuera de esta paleta, creé una variable que guardara y manejara ese color.

```
Color mainColor = AppEssentials.color;
Map<String, Color> get palette => {
    "appLight": Color.alphaBlend(Color.fromRGBO(255, 255, 255, 0.7), mainColor),
    "appMain": mainColor,
    "appDark": Color.alphaBlend(Color.fromRGBO(0, 0, 0, 0.1), mainColor),
    [...]
};
```

Al tener get al inicio de la declaración, esta variable pasa a ser un getter, por lo que es completamente dinámica y no requiere actualizar nada más que el color principal para que cambie correctamente. En ambos temas, claro y oscuro, cada color de cada widget dependerá completamente de este mapa, y de este modo, si tenemos algo vinculado a los tonos principal, claro u oscuro, cambiará en función de qué color tenga elegido el usuario.

Lo siguiente fue añadir los colores a la aplicación, que se realiza con un menú desplegable con todas las opciones asequibles. Cuando el usuario elija una opción, no sólo se cambiará en su aplicación, sino que se guardará en el caché de la aplicación. Esto se ha conseguido con la dependencia **shared_preferences**, que permite guardar tipos primitivos en el caché de la aplicación. Esta aplicación guarda el color principal dividido en tres valores enteros, coincidiendo con los valores RGB.

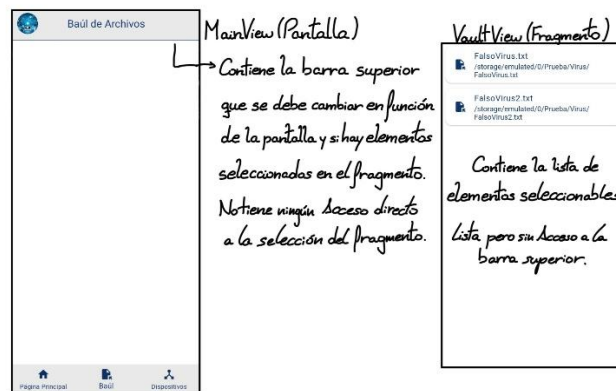
Como cuando se carga la aplicación por primera vez no se cuenta con un color seleccionado, se carga directamente el azul por defecto.



6.4.3.2 Selección múltiple de ficheros en cuarentena

Mientras hacía la aplicación, pese a que me gustara la idea de que tuvieras que entrar en el menú contextual de cada fichero para decidir si borrarlos o restaurarlos, se me hacía un poco pesado tener que hacerlo con cada fichero que se pusiera en cuarentena. Por ello fue que decidí crear un sistema donde se pudieran seleccionar varios ficheros a la vez, aunque hubo un ligero problema con esta implementación al principio:

Las pantallas del menú principal se dividen en 2 partes: La pantalla (Scaffold), donde se encuentran únicamente la barra superior y el menú de navegación inferior, y un fragmento que varía en función del menú seleccionado. Uno de esos fragmentos es el que carga de la base de datos local los archivos en cuarentena, y permite borrarlos o restaurarlos. El problema empieza cuando en la pantalla principal tengo la barra superior que debería cambiar si selecciono algún valor del fragmento, en el fragmento tengo la información a seleccionar, y, lo más importante, no tengo una forma clara de acceder a los datos del fragmento desde la pantalla ni viceversa.



La solución que encontré fue utilizar los Provider de Flutter para conectar la información de las dos pantallas por el contexto. Creé una lista auxiliar en uno de los proveedores, la cual se actualizaría con la cantidad de elementos seleccionados en el fragmento del baúl de archivos.

Si esta lista tenía información, la barra superior cambiaría para decir al usuario la cantidad de archivos seleccionados y dos botones: Uno para restaurar y otro para borrar. La lista de archivos cambiaría su icono de archivo a un botón de selección que aparece seleccionado si el elemento se encuentra tanto en la lista de la pantalla como en la lista del proveedor. Si se da a borrar o restaurar, se hace desde la pantalla principal la operación de restauración o eliminación y se notifica al estado para actualizarse.

Ahora también quedaba la parte de borrar la lista del proveedor sin cambiar datos, por si simplemente dejaba de hacer falta que tuviera datos. Esto se puede hacer por medio de dar al botón de salir o de cambiar de pantalla. Ambos harán que se borre la lista completa del proveedor y así notifique a la pantalla y el fragmento, este último en el caso de que siga abierto.



6.4.3.3 Introducción de programación paralela

Al principio, el análisis de ficheros se realizaba sobre el hilo principal de la aplicación, aunque fuera de forma asíncrona al resto de la interfaz y otras funciones. En teoría cumplía su función correctamente, ya que no interfería en ninguna otra acción que hiciera el usuario, pero si el análisis llegaba a un fichero suficientemente pesado, como puede ser un video de larga duración, necesitaba recoger más memoria para leer sus bytes, la suficiente como para evitar que incluso la interfaz funcionara correctamente. Para solucionar este fallo, decidí implementar programación paralela en la aplicación.

Los subprocesos de las aplicaciones en Dart funcionan de una forma muy distinta a como funcionan en Java o Kotlin. No es tan sencillo como crear un hilo que cuelgue del principal y pasarle los objetos que interesen tener en ese subproceso.

Dart crea subprocesos “aislados” (o Isolates) para realizar procesos de forma paralela a la aplicación. Estos Isolates ejecutan su función como si fuera una llamada a un método “main” en un ejecutable distinto. Desde este no se puede acceder a ningún atributo declarado en el hilo padre, independientemente de si estos son o no estáticos, sólo se puede utilizar lo que se reciba en los argumentos de la función, y estos sólo pueden ser de tipo primitivo o serializable (listas y mapas).

He decidido gestionar hilos por medio del método **compute**, el cual, por medio de la función que le dé por parámetros y los argumentos, crea automáticamente un Isolate que recorre la función asignada con los parámetros que se asignan.

Ahora había otro problema que solucionar: La gestión de los ficheros una vez se encontraban virus.

Antes de implementar hilos, insertaba los ficheros en cuarentena directamente usando funciones estáticas conectadas a la base de datos de SQLite, Sin embargo, esta opción dejó de ser posible debido a que las bases de datos locales no funcionan correctamente intentando acceder desde subprocesos.

Por ello, decidí optar por utilizar **puertos de comunicación**. Estos puertos están incluidos en la librería interna de **dart:isolate**, y permite transmitir mensajes entre los hilos principales y sus subprocesos, con el único inconveniente de que los mensajes solo pueden ser de tipo primitivo o serializable. La solución para utilizar estos datos fue enviar al subproceso en sus argumentos el puerto de envío, encargado de guardar en un mapa los ficheros que habían sido bloqueados y con qué huella, y en el proceso principal generar un *listener* con el puerto de recepción para procesar los datos del hilo y, ahora sí, poner los ficheros en cuarentena y guardar la información en la base de datos.

Para rematar con la gestión de hilos, decidí también añadir la opción de detener manualmente el análisis. Esto lo hice con otra dependencia: **cancelable_compute**. Esta dependencia permite guardar el hilo iniciado con **compute** en una variable y detenerla a conciencia con el método **cancel**. Lo único necesario fue añadir un botón que detuviera el análisis cuando este se estuviera ejecutando.

6.4.3.4 Análisis de una carpeta en específico

Mientras hacía las pruebas del análisis de archivos, me di cuenta de que a veces era muy pesado tener que analizar absolutamente todo el dispositivo cada vez que quería hacer alguna prueba, y que, si un usuario quería analizar una carpeta o un grupo de estas, sería una buena alternativa poder elegir de alguna otra forma el inicio del análisis. Por ello, decidí habilitar un botón que dejara elegir la carpeta para comenzar el análisis.

Como la elección de las carpetas de acceso prohibido, utilicé la dependencia **file_picker** de Flutter para poder seleccionar una carpeta específica del dispositivo. Esta dependencia tiene una función llamada `getDirectoryPath`, encargada de abrir una ventana en el dispositivo para seleccionar un directorio del sistema de archivos. Si se selecciona una, devuelve la ruta a ese directorio como `String`, mientras que, si no se selecciona nada, devuelve 'null'. La idea es que, si se selecciona un directorio, este sea analizado.

Antes de hacer este cambio, la función de análisis de ficheros funcionaba utilizando siempre el directorio raíz (diferente dependiendo del dispositivo), pero con la posibilidad de hacerlo desde un directorio específico, ideé la posibilidad de pasar a la función un `path` como `String` por parámetro y que, si dicho `String` está vacío, se utilice el directorio raíz para analizar:

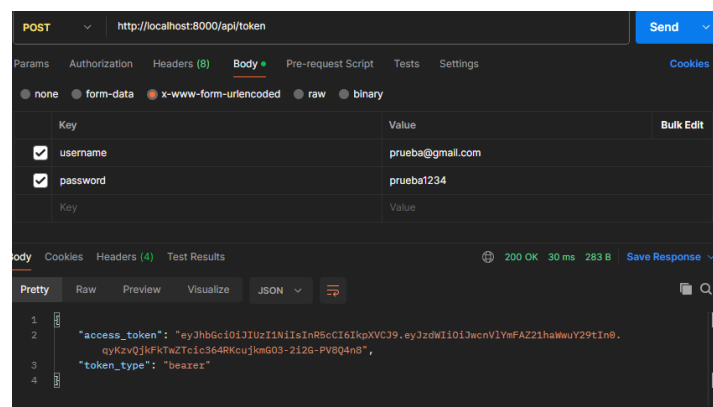
```
void startAnalysis(String? customDir, BuildContext context) async {  
  String mainDirectory = customDir ??  
    ((Platform.isAndroid)  
      ? "/storage/emulated/0"  
      : (Platform.isWindows)  
        ? "C:\\\\"  
        : "/");  
  [...]  
}
```

6.5 Pruebas.

6.5.1 Pruebas de API

6.5.1.1 Prueba de obtención de WebToken

Para obtener la Token es necesario contar con un usuario ya registrado en la base de datos. Por ello, se cuenta con un usuario de correo electrónico “prueba@gmail.com” y la contraseña “prueba1234”. Estos parámetros se enviarán con formato form-urlencoded.

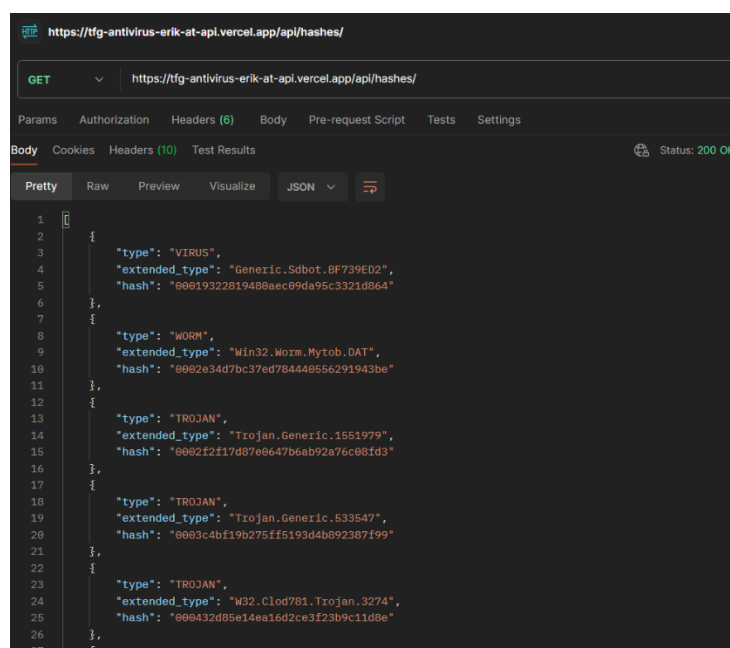


Esta clave será guardada tanto en Postman para hacer pruebas desde el servicio, pero esta misma clave se consigue por detrás en la aplicación por medio de una función activada al iniciar sesión:

6.5.1.2 Prueba de Obtención de Hashes

La obtención de huellas no precisa de verificación con token, ya que sirve para cargar las huellas guardadas en la bd, no permite editar nada fuera de ahí.

Para ello, hay que acceder a `/api/hashes/`



6.5.2 Pruebas de la aplicación

6.5.2.1 Prueba de Introducción de Dispositivo

Cuando la aplicación carga en un dispositivo, recibe sus parámetros por medio de un método get de la API. Si el dispositivo acaba de abrir la aplicación por primera vez, se atrapa una excepción de objeto no encontrado en la base de datos, y se inserta dicho dispositivo en la base de datos por medio de un método post de la API.

```
flutter: | http://localhost:8080/api/devices/%7B7BB60C47-4C13-4E43-8B39-9295B1A984AB%7D
flutter: |
flutter: |
flutter: | #0 DeviceDAO.get (package:magik_antivirus/DataAccess/DeviceDAO.dart:51:16)
flutter: | #1 <asynchronous suspension>
flutter: |
flutter: | FormatException: Unexpected character (at character 1)
flutter: | noBody
flutter: | ^
flutter: |
flutter: | {"id":{"7BB60C47-4C13-4E43-8B39-9295B1A984AB"},"dev_name":"erik.amotoq","dev_type":"w
flutter: | indows","join_in":"2025-01-28 09:47:07.604843","last_scan":"2025-01-28 09:47:07.604843","use

File "D:\DAM 2\TFG\tfg_antivirus_api\app\routers\DeviceRouter.py", line 21, in get_devices_by_id
raise Exception("Dispositivo no encontrado")
Exception: Dispositivo no encontrado
INFO: 127.0.0.1:52392 - "POST /api/devices/insert HTTP/1.1" 200 OK
```


6.5.2.2 Prueba de Introducción de Usuario en Dispositivo

El usuario está creado y el dispositivo también, pero si miramos el dispositivo en la bd (o en la API), veremos que el campo 'user' está vacío. Desde la aplicación, será necesario iniciar sesión o registrarse para rellenar este campo

En cuanto se rellenen los campos de texto de nombre de usuario/correo electrónico y la información sea correcta, se editará el dispositivo, añadiendo el correo del usuario al campo en la bd.

```
flutter: | http://localhost:8080/api/users/Prueba
flutter: |
flutter: | {"id":{"7BB60C47-4C13-4E43-8B39-9295B1A984AB"},"dev_name":"erik.amotoq","dev_type":"w
flutter: | indows","join_in":"2025-01-28 09:47:07.604843","last_scan":"2025-01-28 09:47:07.604
flutter: | r":"prueba@gmail.com"}
flutter: |
flutter: | #0 APIReaderUtils.putData (package:magik_antivirus/utis/DBUtils.dart:121:14)
flutter: | #1 <asynchronous suspension>
flutter: |
flutter: | El put del item {id: {7BB60C47-4C13-4E43-8B39-9295B1A984AB}, dev_name: erik.amot
flutter: | oq, dev_type: windows, join_in: 2025-01-28 09:47:07.604843, last_scan: 2025-01-28 09:47:07.604
flutter: | 843, user: prueba@gmail.com} ha dado el codigo 200
flutter: |
```

```
INFO: 127.0.0.1:52903 - "GET /api/users/Prueba HTTP/1.1" 200 OK
INFO: 127.0.0.1:52904 - "PUT /api/devices/%7B7BB60C47-4C13-4E43-8B39-9295B1A984AB%7D/update HTTP/1.1" 200 OK
```



6.5.2.3 Prueba de Inicio de Sesión Automático

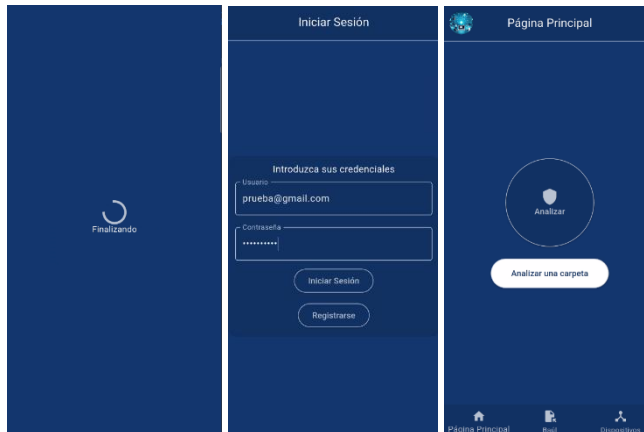
Hay fragmentos de código dedicados a cargar y guardar datos del dispositivo, y entre esos datos puede encontrarse el correo electrónico del usuario que, en la base de datos, dicta que tiene su sesión iniciada. Por ello, he decidido basar el inicio de sesión automático en ese correo electrónico:

```
await AppEssentials.registerThisDevice();
if (AppEssentials.dev!.user != null) {
    User? u = (await UserDAO().get(AppEssentials.dev!.user!));
    if (u != null) {
        changeUser(u);
    }
}
```

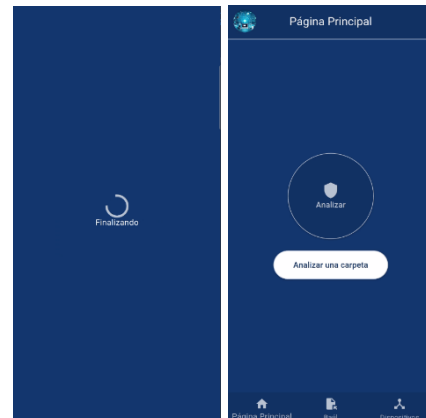
Este bloque, que se encuentra en el proveedor de datos del usuario, se encarga de cargar el dispositivo en la base de datos, además de que, si el dispositivo se encontraba ya en la base de datos, y resulta que éste tenía ya un usuario registrado, recoge ese usuario de la base de datos por su correo electrónico y lo guarda en el proveedor.

La prueba se hace de la siguiente forma: Se abre la aplicación y se inicia sesión. Acto seguido, se abre la aplicación de vuelta a ver si funciona:

Primer Acceso a la Aplicación:



Segundo Acceso a la Aplicación:



6.5.2.4 Prueba de Cierre de Sesión Remoto

Aprovechando el inicio de sesión automático, implementé también la opción de cerrar sesión en un dispositivo desde otro distinto. Como se guarda el correo del usuario en cada dispositivo, y muestro una lista de todos los dispositivos del usuario, planeé una forma de desvincular cuentas manteniendo pulsado el dispositivo a quitar.

Para hacer la prueba, como no tenía más dispositivos en el momento, lo hice observando directamente la base de datos en red:

id	dev_name	dev_type	last_scan	join_in	user
0a9116bcbdbadd0a681dea07367968a11f178f8...	Pixel 7 Pro	android	2025-04-29 19:01:03	2025-04-29 18:58:16	prueba@gmail.com

En la cuenta de prueba, aparece este dispositivo registrado. En la interfaz visual se puede ver su widget. Al pulsarlo, da la opción de borrarlo de la lista. Si se acepta, dando al botón de desvincular, desaparecerá el dispositivo tanto de la lista de la aplicación como de la base de datos en red:

id	dev_name	dev_type	last_scan	join_in	user
0a9116bcbdbadd0a681dea07367968a11f178f8...	Pixel 7 Pro	android	2025-04-29 19:01:03	2025-04-29 18:58:16	NULL

Con la prueba hecha en el apartado anterior, si se dispusiera actualmente de ese dispositivo con la aplicación instalada, necesitaría guardar sus datos de nuevo.



7 Manuales de usuario

7.1 Manual de usuario

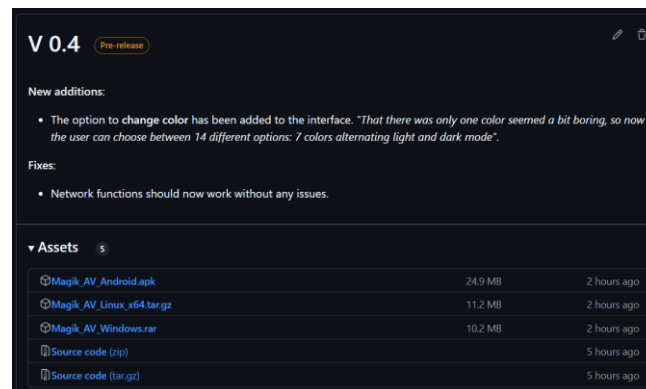
El manual de usuario se encuentra en el otro pdf realizado para este proyecto: **Amo_Toquero_Erik_Manual_ProyectoFinal_DAM25.pdf**

7.2 Manual de instalación

La instalación en cada dispositivo parte de la misma base:

Ir al proyecto de GitHub del antivirus e irse al apartado “Releases” (o acceder al siguiente enlace: https://github.com/ErikAT04/TFG_Antivirus_ErikAT/releases).

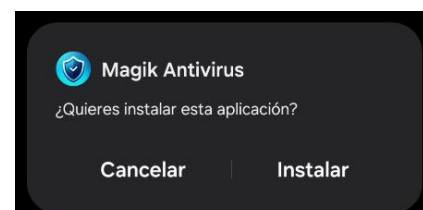
Una vez allí, deberán ir al lanzamiento más reciente y descargar el archivo propio de su sistema operativo:



La instalación de la aplicación es sencilla, pero habrá cambios en función del sistema operativo en el que se haga.

7.2.1 Android

En Android, será necesario descargar el archivo APK (Magik_AV_Android.apk). Una vez descargado, en el explorador de archivos habrá que buscar el fichero e instalarlo con la herramienta de instalación de paquetes por defecto de Android. Es posible que pida al usuario permiso para instalar archivos de origen desconocido. Si es así, bastará con aceptarlo.



7.2.2 Windows

En Windows, habrá que descargar el archivo encriptado en RAR (Magik_AV_Windows.rar). Cuando se descargue, se debe descomprimir el archivo con alguna herramienta (WinRAR o el propio explorador de archivos de Windows) en la dirección donde el usuario va a querer tener la aplicación.

Dentro de la carpeta, se podrán ver varios archivos y ficheros:

- .dart_tool: Guarda en su interior la base de datos local
- data: Guarda los assets, paquetes y otros recursos necesarios para que funcione correctamente la aplicación
- flutter_windows.dll: Extensión de la aplicación necesaria para su correcto funcionamiento
- magik_antivirus.exe: Ejecutable de la aplicación
- permission_handler.dll: Extensión de la aplicación que maneja los permisos de ésta.

Es importante no sacar el ejecutable de esta carpeta, sino hacer algún acceso directo para abrir la aplicación con más facilidad si el usuario lo prefiere.

7.2.3 Linux

En Linux la instalación cambia un poco en un punto:

Hay que descargar el archivo encriptado en formato tar.gz (Magik_AV_Linux_x64.tar.gz) y descomprimirlo en la carpeta deseada.

La aplicación no funcionará correctamente si no se tienen las librerías de SQLite (el gestor de la base de datos local) **instaladas:**

- Para instalarlas, hay que ejecutar el siguiente comando con opciones de administrador:
sudo apt install sqlite3 libsqlite3-dev

Dentro de la carpeta, aparecen varios archivos:

- ._dart_tool: Guarda en su interior la base de datos local
- data: Guarda los assets, paquetes y otros recursos necesarios para que funcione correctamente la aplicación
- lib: Guarda las librerías externas que utiliza la aplicación para funcionar.
- magik_antivirus: El ejecutable de la aplicación. Al igual que en Windows, es importante no sacar este archivo de la carpeta para que todo funcione correctamente.

8 Conclusiones y posibles ampliaciones

Con el proyecto ya prácticamente terminado, puedo decir que he cumplido y con mejoras el propósito inicial del proyecto:

He logrado crear una aplicación antivirus *teóricamente* funcional, que permite iniciar sesión, gestionar los dispositivos de los usuarios y, más importante, es capaz de analizar todo el sistema de archivos de un dispositivo sin incapacitar el uso de la misma aplicación o de otras al usuario, además de brindar la posibilidad de guardar y sacar de cuarentena los ficheros detectados como virus.

En cuanto a mi aprendizaje, he aprendido a crear y desplegar servicios en red completamente funcionales, a recorrer los archivos internos en los distintos dispositivos desde un programa y a manejar programación paralela con Dart.

Como **posibles ampliaciones**, sé varios puntos que me habría gustado introducir, pero, por falta de tiempo y recursos, he tenido que dejar de lado para sacar adelante el producto actual:

- Crear **notificaciones** que aparecieran cada vez que un archivo ha sido puesto en cuarentena.
- Buscar alguna forma de evitar que los archivos que han salido de cuarentena vuelvan a entrar. La idea era muy buena, pero no había formas suficientemente sencillas de hacerlo realidad, porque no se podía cancelar un hash de la base de datos en caso de que volviera a aparecer otra vez la misma amenaza, ni se podía editar el hash, y tampoco era buena idea guardar la ubicación del archivo, porque significaría no poder moverlo de esa carpeta.
- Crear un servicio de análisis instantáneo, donde el sistema está siempre observando los ficheros, y si se encuentra un archivo nuevo descargado de internet se analiza automáticamente. No era una idea difícil, pero había que hacer servicios en segundo plano para que pudiera funcionar y se complicaba bastante para hacerlo en todas las plataformas.
- Añadir la opción de actualizar automáticamente las versiones de las aplicaciones. Esto se puede hacer mediante GitHub Actions, con una acción personalizada llamada *Flutter Actions*. Traté de implementar esta funcionalidad para poder actualizar con más facilidad la aplicación cada vez que había un fallo, pero cuando llegaba a la construcción de Linux, debido a la estructura de los archivos y distintos problemas de compatibilidad, siempre saltaba un error de lectura de las clases.

9 Bibliografía

Definición de Huellas Digitales:

<https://encyclopedia.kaspersky.com/glossary/hash/>

Detección por Huellas Digitales:

<https://systemweakness.com/malware-hash-analysis-identifying-threats-with-cryptographic-precision-8d87499c50c4>

Virus_Detector: Repositorio de GitHub con el archivo de huellas utilizado.

https://github.com/sujon1990/virus_detector

Documentación de FreeDB: Herramienta utilizada para crear la base de datos en red:

<https://freedb.tech/dashboard/faq.php>

Documentación de Vercel: Herramienta utilizada para desplegar la API:

<https://vercel.com/docs/deployments>

Guía de Despliegue de API con FastAPI en Vercel:

<https://dev.to/abdadeel/deploying-fastapi-app-on-vercel-serverless-18b1>

Explicación de la **Arquitectura en Microservicios** oficial de Intel:

<https://www.intel.la/content/www/xl/es/cloud-computing/microservices.html>

Uso de Hilos en Flutter y Dart:

<https://docs.flutter.dev/perf/isolates>

Creación y automatización de **GitHub Actions con Flutter** (Investigación para futuras implementaciones):

<https://github.com/marketplace/actions/flutter-action>