



PROYECTO SGE

2ª Evaluación

CFGS Desarrollo de Aplicaciones
Multiplataforma
Informática y Comunicaciones

**Desarrollo de Servicio API con FASTAPI y
Cliente Móvil**

Año: 2025

Fecha de presentación: 10/02/2025

Nombre y Apellidos: Erik Amo Toquero
Email: erik.amotoq@gmail.com

Contenido

Introducción.....	3
1.1 Estado del arte.....	3
1.2 Descripción general del proyecto.....	4
1.2.1 Objetivos	4
1.2.2 Entorno de trabajo (tecnologías de desarrollo y herramientas).....	4
1.3 Documentación técnica: análisis, diseño, implementación, pruebas y despliegue	5
1.3.1 Análisis del sistema (funcionalidades básicas de la aplicación)	5
1.3.2 Diseño de la base de datos	5
1.3.3 Implementación	6
1.3.4 Empleo de seguridad con PyJWT	17
1.3.5 Pruebas	19
1.3.6 Despliegue de la aplicación.....	21
1.4 Manuales.....	21
1.4.1 Manual de usuario:	21
1.4.2 Manual de instalación:.....	23
1.4.3 Conclusiones y posibles ampliaciones	29
Bibliografía	29

Introducción

La mayoría de aplicaciones actuales, ya sean multiplataforma como web, utilizan servicios en red para acceder a datos y poder mostrarlos, modificarlos, crearlos o borrarlos. En esta práctica se plantea crear un servicio API REST que supla a una aplicación multiplataforma para que, en vez de acceder a una base de datos de forma directa, se realicen peticiones a la API y esta sea la que llame a la base de datos

1.1 Estado del arte

La arquitectura de microservicios es un método de desarrollo de software basado en varios servicios o módulos ligeros que se ejecutan de forma independiente entre ellas. Dichos microservicios tienen conexiones sueltas, es decir, se comunican entre ellas pero trabajan de forma autónoma, sin depender de esas conexiones.

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

Debido a la vasta cantidad de librerías de Python, hay varios frameworks web que permiten crear una API reactiva con ese lenguaje. Predominan los frameworks de Flask y FastAPI.

- **Flask** es un framework web que permite a los desarrolladores realizar una aplicación web desde un simple archivo .py y no depende de una estructura específica
- **FastAPI** es un framework para desarrollar aplicaciones web con Python 3.6 de una forma rápida y ligera. Frente a otros frameworks, FastAPI destaca por su velocidad de carga, comparable con Node.js, su fácil aprendizaje y su documentación automática con OpenAPI.

Por las ventajas frente a Flask y el aprendizaje en clase, he decidido utilizar FastAPI para este proyecto.

1.2 Descripción general del proyecto

1.2.1 Objetivos

Frente a la aplicación en general, los objetivos son:

- Aprender a manipular archivos de distintos sistemas operativos
- 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 que maneje todas las operaciones CRUD necesarias.

Frente a la API de este proyecto, los objetivos son:

- Conectar la API a una base de datos y, mediante sus endpoints, realizar todas las operaciones CRUD que sean necesarias para el correcto funcionamiento de la aplicación.
- Dotar de seguridad a la API, teniendo que generar un token web

1.2.2 Entorno de trabajo (tecnologías de desarrollo y herramientas)

Ya que mi objetivo, aparte del desarrollo de la aplicación en Android, incluye el despliegue en red de la API y la base de datos, voy a necesitar dos servicios a mayores para poder usar mis servicios en red.

Lenguajes:

- El lenguaje elegido para la aplicación es **Dart**, un lenguaje tipado y open source utilizado por la mayoría de aplicaciones de Google, además de contar con el framework de Flutter, que permite generar interfaces gráficas siguiendo el modelo Material UI y además cuenta con una amplia biblioteca de dependencias para el uso del usuario.
- El lenguaje elegido para el servidor es **Python**, un lenguaje anárquico que, entre todos los usos que tiene debido a su fácil interpretación y sus amplias librerías, permite implementar RestAPIs con su librería FastAPI, e implementar conexiones a bases de datos mediante un *mapeo objeto-relacional* con su librería SQLAlchemy.
- El lenguaje de definición de bases de datos, al ser una base relacional, es **SQL**, y el gestor de bases de datos es **MySQL**.

Entornos:

- El entorno de desarrollo del api en Python y la aplicación en Dart es **Visual Studio Code**, un IDE que permite instalar plugins para facilitar el desarrollo en cualquier lenguaje de programación
- El entorno de implementación de la base de datos es **MySQL Workbench**, el Sistema Gestor de Bases de Datos oficial de MySQL, que permite acceder a bases de datos tanto locales como en red y editarlas en tiempo real
- **Servicios en Red**. Los servicios elegidos para desplegar la base de datos y las APIs en red son **Vercel** y **FreeDB.tech**
 - **Vercel**: Plataforma en red que permite desplegar APIs de forma gratuita. Permite desplegar una API guardada en un repositorio de GitHub, especificando la ruta del archivo que la inicia.
 - **FreeDB.tech**: Plataforma en red que permite crear y gestionar bases de datos gratuitamente, con ciertas limitaciones de espacio y consultas si no se realiza una suscripción.

1.3 Documentación técnica: análisis, diseño, implementación, pruebas y despliegue

1.3.1 Análisis del sistema (funcionalidades básicas de la aplicación)

Quitando todo lo relacionado a la ciberseguridad, las funciones en red básicas de la aplicación móvil son las siguientes:

- Buscar el dispositivo actual en la base de datos o crearlo si este no existe
- Crear, editar o borrar un usuario
- Listar dispositivos de un usuario
- Modificar parámetros de un dispositivo (Principalmente la fecha de último análisis)
- Listar internamente las firmas de una base de datos

Al hacer que la aplicación no se conecte a la base de datos que alberga toda esa información, obliga a la API a cumplir con dichas funciones, además de unas a mayores:

- Autorizar o denegar el acceso a los usuarios por medio de tokens
-

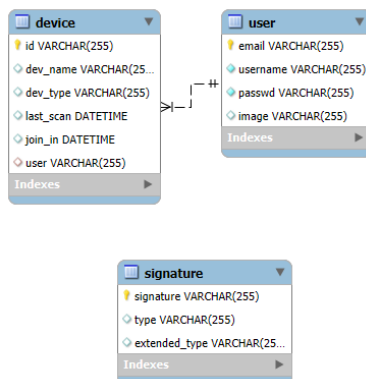
1.3.2 Diseño de la base de datos

Una vez definida la aplicación y sus usos, se define la base de datos.

La base de datos se conforma de las siguientes tablas:

- **Usuario (user):** Define el usuario, el cual se identifica con su correo electrónico y su contraseña (encriptada), además de poder usar una imagen de perfil.
- **Dispositivo (device):** Define los dispositivos que se conectan a la aplicación. Tiene un identificador que cambia en función del dispositivo (en algunos es la dirección MAC, en otros es el número de serie...), además de su nombre genérico, su tipo de SO, la fecha en la que se unió al servicio por primera vez y la fecha en la que se realizó un análisis por última vez.
- **Firma (signature):** Define las firmas guardadas en la base de datos. **No se encuentra relacionada con las otras dos tablas, ya que es totalmente independiente de éstas.** Guarda su firma (encriptada en Base64), y el tipo de malware que tiene, tanto en su formato extendido, como en simplemente el grupo al que pertenece (Virus, Troyano, Gusano, Spyware...)
- **Relación:**
 - **Un usuario puede tener varios dispositivos, mientras que un dispositivo solo puede pertenecer a un usuario.** Relación 1:N (One2Many y Many2One)

Esquema EER:



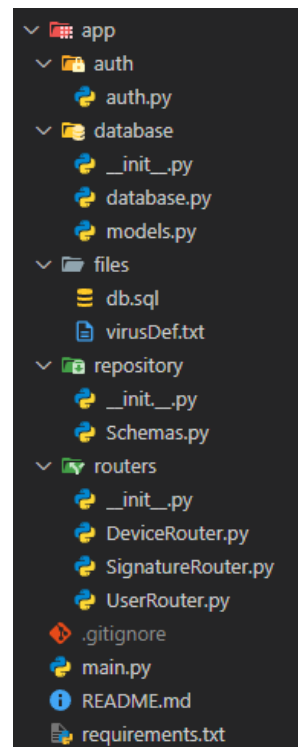
1.3.3 Implementación

La estructura de la API se definirá en varias carpetas dentro de la subcarpeta '**app**'. El archivo `main.py`, encargado de ejecutar el script para iniciar la APIRest, se encontrará en la raíz del proyecto, junto al archivo `requirements.txt`, el cual se utiliza para instalar las librerías que utiliza la API.

Estructura:

- **AUTH:** Guarda toda la información de seguridad de la API, dígame la verificación de autenticación y la obtención del Token de JWT.
- **Database:** Guarda tanto la conexión a la bases de datos (MySQL en este caso) como los modelos del ORM de la librería SQLAlchemy.
- **Files:** Guarda tanto el archivo sql de definición de la base de datos para crearlo en la base de datos designada, como un archivo de texto que guarda todas las firmas de virus recopiladas. Este archivo de texto está sacado de un proyecto de github.
- **Repository:** Guarda los archivos de esquemas, modelos similares a los de la base de datos que se utilizan para recibir los objetos de la API.
- **Routers:** Guarda los routers de la API, los distintos servicios que se unen a la raíz de la API y sirven para organizar el proyecto, consiguiendo una mejor legibilidad del código y detección de problemas.

Las librerías principales utilizadas en este proyecto son las siguientes:



- **FastAPI:** Permite al usuario generar una API, configurando sus distintos endpoints por medio de funciones.
- **MySQL Connector:** Conector a la base de datos de MySQL
- **SQLAlchemy:** ORM de las bases de datos SQL. Por medio de objetos en Python se puede editar información en la base de datos.
- **PyJWT:** Seguridad de APIs con Python. Genera JWTs (JSON Web Tokens), tokens que permiten acceder a los endpoints bloqueados por autenticación.
- **Uvicorn:** Permite probar la API en el host local.
- **Python-Jose:** Encriptación de datos con Python, utilizado para la encriptación SHA256.

1.3.3.1 Modelo, Esquema y Router de User

Para las clases que provienen de una base de datos, es necesario crear una clase que herede de `sqlalchemy.orm.Model`, mientras que las clases que utiliza de modelo la propia API heredan de `fastapi.BaseModel`.

En la clase del ORM de SQLAlchemy, se introducen los atributos de las columnas de la tabla “user” de MySQL, junto al atributo `__tablename__` con el nombre “user”

```
class User(Base):
    __tablename__ = "user"
    email = Column(VARCHAR(255), primary_key=True)
    username = Column(VARCHAR(255), unique=True)
    passwd = Column("passwd", VARCHAR(255))
    image = Column(VARCHAR(255))
```

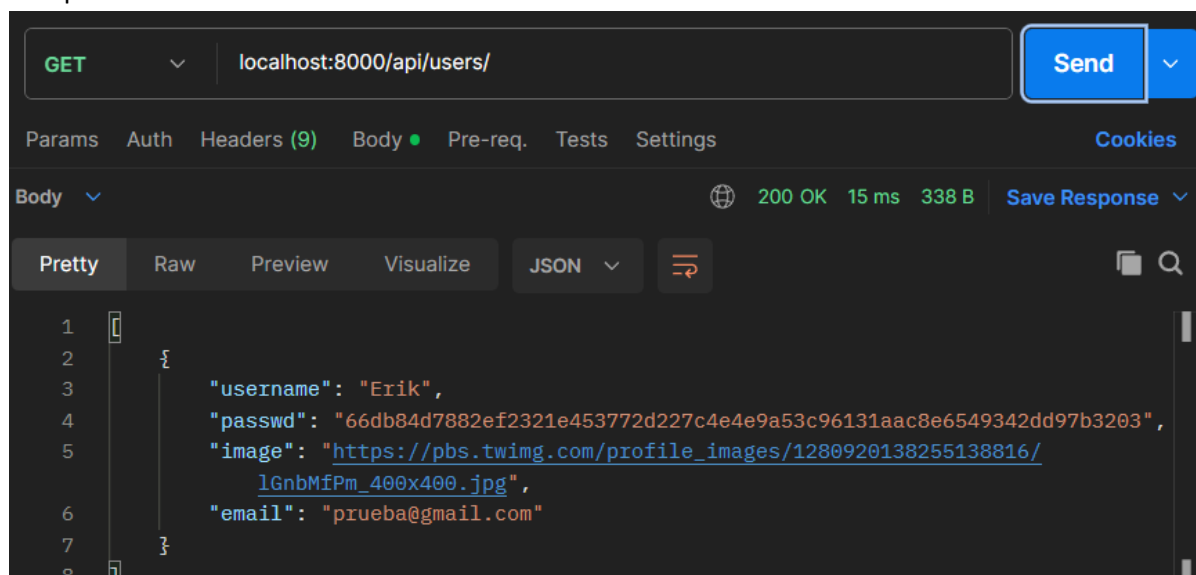
```
class User(BaseModel):
    email:str
    username:str
    passwd:str
    image:str
```

La raíz de los endpoints del usuario es `/api/users`, y de ahí se llama a las funciones según su terminación:

Listado de Usuarios: Devuelve una lista con todos los usuarios de la base de datos

```
@router.get("/")
def get_all_users(db:Session = Depends(get_db)):
    users = db.query(models.User).all()
    return users
```

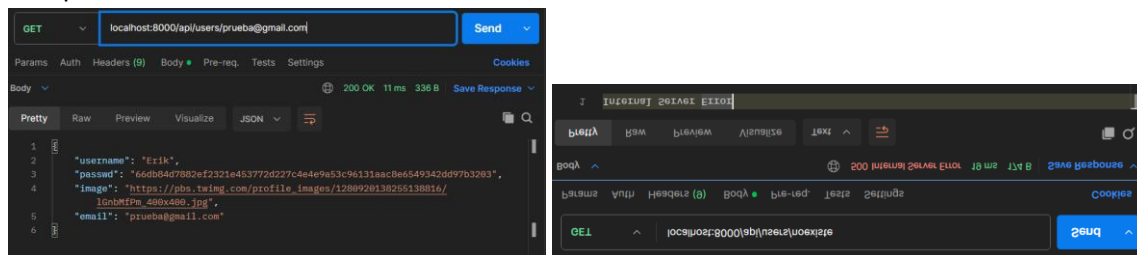
Comprobación:



Búsqueda de Usuario por medio de su correo electrónico o nombre de usuario: Recibe un dato por parámetro y si se encuentra un usuario cuyo correo electrónico o nombre de usuario coincida con ese dato lo devuelve

```
@router.get("/{string}")
def get_user_by_email_or_username(string:str, db:Session = Depends(get_db)):
    user = db.query(models.User).filter(or_(models.User.email == string, models.User.username == string)).first()
    if user:
        return user
    else:
        raise Exception("Error: Usuario no encontrado")
```

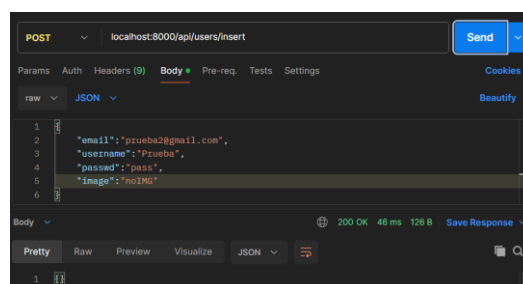
Comprobación:



Insertión de Usuario: Recibe por el cuerpo del API un usuario con su email, nombre de usuario, contraseña e imagen de perfil y se inserta en la bd.

```
@router.post("/insert")
def create_user(user:Schemas.User, db:Session = Depends(get_db)):
    usuario = models.User()
    usuario.email = user.email
    usuario.username = user.username
    usuario.passwd = user.passwd
    usuario.image = user.image
    db.add(usuario)
    db.commit()
    return usuario
```

Comprobación



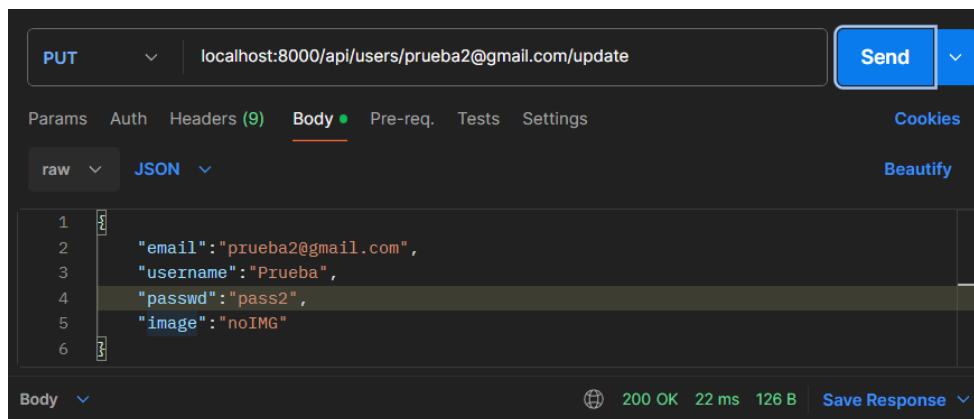
El resultado se verá en la base de datos:

	email	username	passwd
▶	prueba@gmail.com	Erik	66db84d7882ef2321e453772d227c4e4e9a53c9...
	prueba2@gmail.com	Prueba	pass

Edición de Usuario: Se recibe un correo electrónico por parámetro y un usuario por el cuerpo de la petición y modifica el usuario cuyo correo sea el previamente escrito.

```
@router.put("/{email}/update")
def update_user(email:str, user:Schemas.User, db:Session = Depends(get_db)):
    usuario = db.query(models.User).filter(models.User.email == email).first()
    if usuario:
        usuario.email = user.email
        usuario.passwd = user.passwd
        usuario.username = user.username
        usuario.image = user.image
        db.commit()
        return usuario
    else:
        raise Exception("Error: Usuario no encontrado")
```

Comprobación;



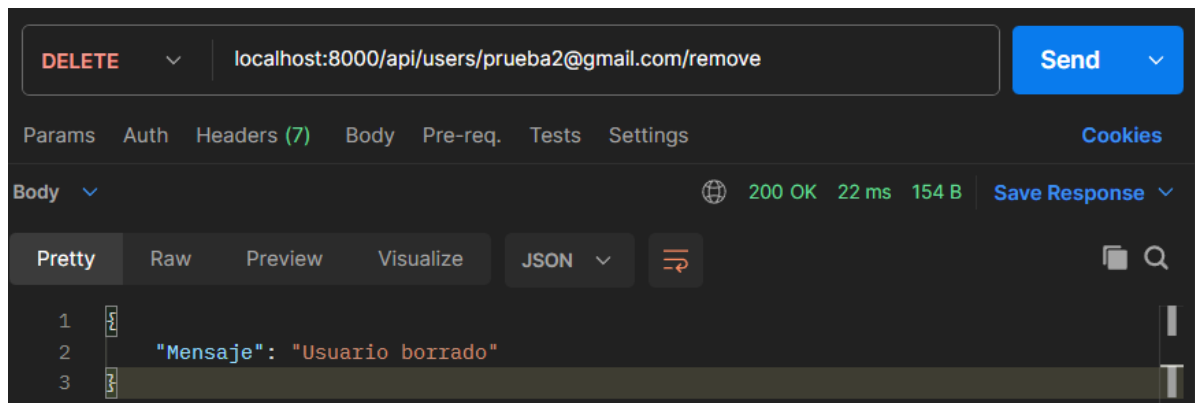
El resultado se verá en la base de datos:

	email	username	passwd
▶	prueba@gmail.com	Erik	66db84d7882ef2321e453772d227c4e4e9a53c9...
	prueba2@gmail.com	Prueba	pass2

Borrado de Usuario: Recibe un correo electrónico por parámetro y lo borra de la bd si existe.

```
@router.delete("/{email}/remove")
def delete_user(email:str, db:Session = Depends(get_db)):
    usuario = db.query(models.User).filter(models.User.email == email).first()
    if usuario:
        db.delete(usuario)
        db.commit()
        return {"Mensaje":"Usuario borrado"}
    else:
        raise Exception("Usuario no encontrado")
```

Comprobación:



Se verá en la base de datos que ha desaparecido el usuario

	email	username	passwd
▶	prueba@gmail.com	Erik	66db84d7882ef2321e453772d227c4e4e9a53c9...
*	NULL	NULL	NULL

1.3.3.2 Modelo, Esquema y Router de Device

En la clase del ORM de SQLAlchemy, se introducen los atributos de las columnas de la tabla “device” de MySQL, junto al atributo `__tablename__` con el nombre “device”

```
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"))

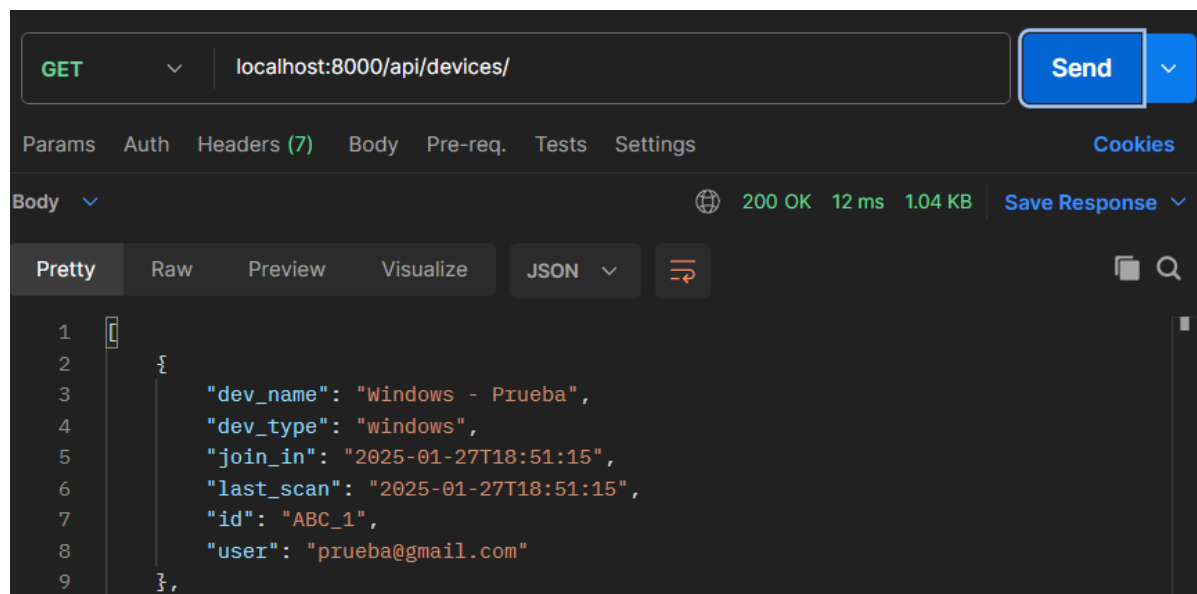
class Device(BaseModel):
    id:str
    dev_name:str
    dev_type:str
    last_scan:datetime
    join_in:datetime
    user:Optional[str] = None
```

El router de Device tiene como raíz `/api/devices`, de ahí descienden todas sus funciones:

Listado de Dispositivos: Devuelve todos los dispositivos de la base de datos

```
@router.get("/")
def get_all_devices(db:Session = Depends(get_db)):
    devList = db.query(models.Device).all()
    return devList
```

Comprobación:



The screenshot shows a REST client interface with the following details:

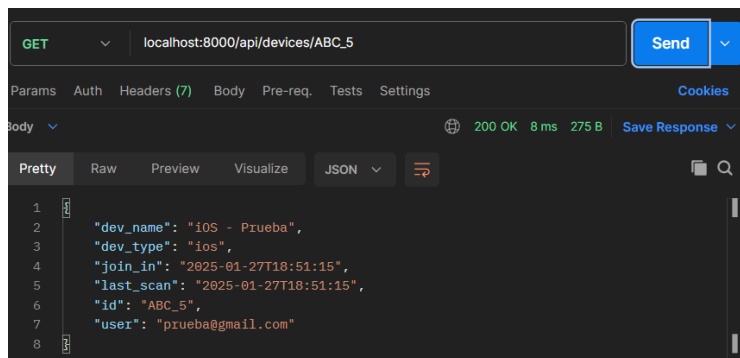
- Method:** GET
- URL:** localhost:8000/api/devices/
- Status:** 200 OK
- Time:** 12 ms
- Size:** 1.04 KB
- Response Body (JSON):**

```
[
  {
    "dev_name": "Windows - Prueba",
    "dev_type": "windows",
    "join_in": "2025-01-27T18:51:15",
    "last_scan": "2025-01-27T18:51:15",
    "id": "ABC_1",
    "user": "prueba@gmail.com"
  }
]
```

Búsqueda de Dispositivo por ID: Recibe un id de un dispositivo por parámetro y, si encuentra uno, lo devuelve.

```
@router.get("/{id}")
def get_devices_by_id(id:str, db:Session = Depends(get_db)):
    dev = db.query(models.Device).filter(models.Device.id == id).first()
    if dev:
        return dev
    else:
        raise Exception("Dispositivo no encontrado")
```

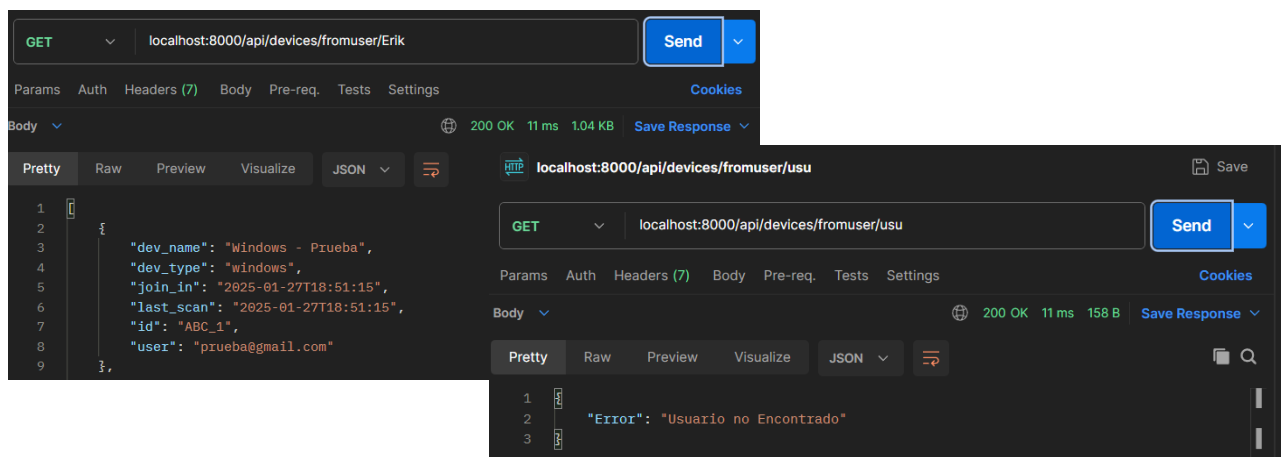
Comprobación:



Búsqueda de Dispositivo por Nombre o Email de Usuario: Busca un usuario por su nombre o email con un dato pasado por parámetro. Si lo encuentra, devuelve la lista de todos los dispositivos asociados a dicho usuario.

```
@router.get("/fromuser/{string}")
def get_all_devices_from_specified_user(string:str, db:Session = Depends(get_db)):
    user = db.query(models.User).filter(or_(models.User.email == string, models.User.username == string)).first()
    if user:
        devList = db.query(models.Device).filter(models.Device.user == user.email).all()
        return devList
    return {"Error": "Usuario no Encontrado"}
```

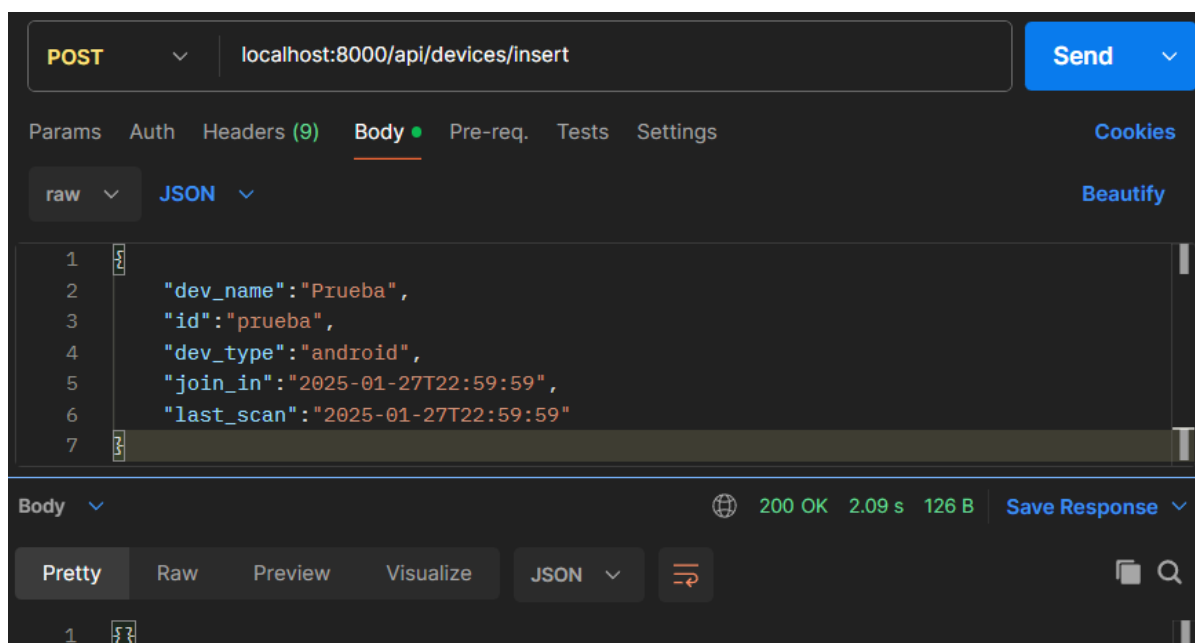
Comprobación:



Inserción de Dispositivo: Recibe un dispositivo por el cuerpo de la petición y lo inserta en la bd.

```
@router.post("/insert")
def insert_device(device:Schemas.Device, db:Session = Depends(get_db) ):
    dev = models.Device()
    dev.dev_name = device.dev_name
    dev.id = device.id
    dev.dev_type = device.dev_type
    dev.join_in = device.join_in
    dev.last_scan = device.last_scan
    if device.user:
        dev.user = device.user
    db.add(dev)
    db.commit()
    return dev
```

Comprobación:



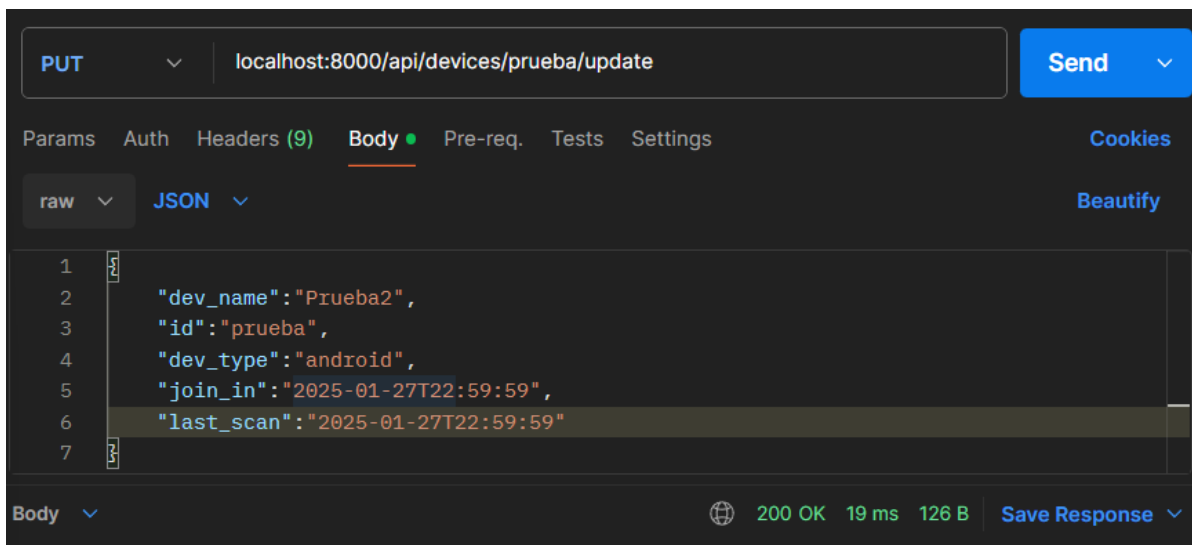
El resultado aparecerá en la bd. Si no se pone usuario, ese campo aparecerá nulo.

id	dev_name	dev_type	last_scan	join_in	user
ABC_1	Windows - Prueba	windows	2025-01-27 18:51:15	2025-01-27 18:51:15	prueba@gmail.com
ABC_2	Android - Prueba	android	2025-01-27 18:51:15	2025-01-27 18:51:15	prueba@gmail.com
ABC_3	MacOS - Prueba	macos	2025-01-27 18:51:15	2025-01-27 18:51:15	prueba@gmail.com
ABC_4	Linux - Prueba	linux	2025-01-27 18:51:15	2025-01-27 18:51:15	prueba@gmail.com
ABC_5	iOS - Prueba	ios	2025-01-27 18:51:15	2025-01-27 18:51:15	prueba@gmail.com
prueba	Prueba	android	2025-01-27 22:59:59	2025-01-27 22:59:59	NULL

Edición de Dispositivo: Recibe un id de un dispositivo y, si lo encuentra, edita su información con los datos pasados por el cuerpo de la petición.

```
@router.put("/{id}/update")
def update_device(id:str, device:Schemas.Device, db:Session = Depends(get_db)):
    dev = db.query(models.Device).filter(models.Device.id == id).first()
    if dev:
        dev.dev_name = device.dev_name
        dev.id = device.id
        dev.dev_type = device.dev_type
        dev.join_in = device.join_in
        dev.last_scan = device.last_scan
        dev.user = device.user
        db.commit()
        return dev
    else:
        raise Exception("Dispositivo no encontrado")
```

Comprobación:



The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** localhost:8000/api/devices/prueba/update
- Body (JSON):**

```
{
  "dev_name": "Prueba2",
  "id": "prueba",
  "dev_type": "android",
  "join_in": "2025-01-27T22:59:59",
  "last_scan": "2025-01-27T22:59:59"
}
```
- Response:** 200 OK, 19 ms, 126 B

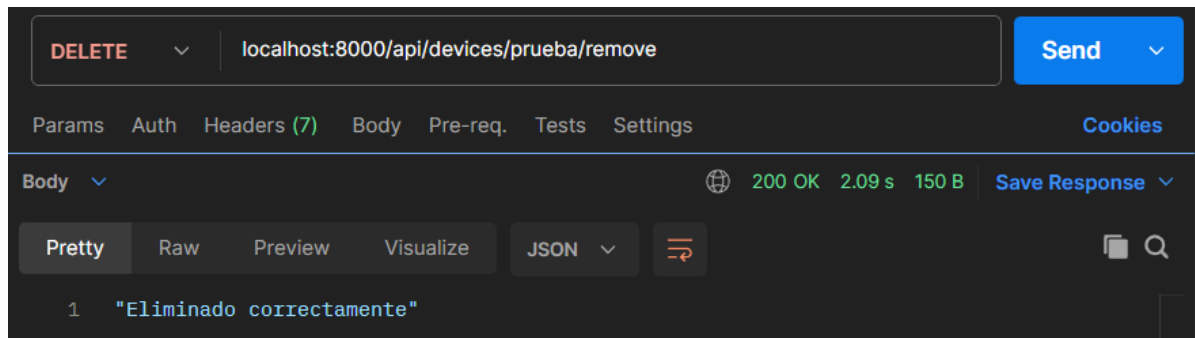
Se verá el resultado en la base de datos:

	id	dev_name
▶	ABC_1	Windows - Prueba
	ABC_2	Android - Prueba
	ABC_3	MacOS - Prueba
	ABC_4	Linux - Prueba
	ABC_5	iOS - Prueba
	prueba	Prueba2

Borrado de Dispositivo: Recibe un id por parámetro y, si encuentra un dispositivo con dicho identificador, lo borra de la base de datos

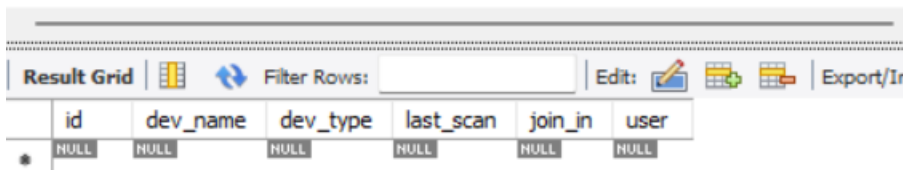
```
@router.delete("/{id}/remove")
def delete_device(id:str, db:Session = Depends(get_db)):
    dev = db.query(models.Device).filter(models.Device.id == id).first()
    if dev:
        db.delete(dev)
        db.commit()
        return "Eliminado correctamente"
    else:
        raise Exception("Dispositivo no encontrado")
```

Comprobación:



Si se busca en la base de datos, no aparecerá la ocurrencia

```
1 • SELECT * FROM m_antivirus_db.device where id = 'id';
```



	id	dev_name	dev_type	last_scan	join_in	user
*	NULL	NULL	NULL	NULL	NULL	NULL

1.3.3.3 Modelo, Esquema y Router de Signature

En la clase del ORM de SQLAlchemy, se introducen los atributos de las columnas de la tabla "signature" de MySQL, junto al atributo `__tablename__` con el nombre "signature"

```
class Signature(Base):
    __tablename__ = "signature"
    signature = Column(VARCHAR(255), primary_key=True)
    type = Column(VARCHAR(255))
    extended_type = Column(VARCHAR(255))

class Signature(BaseModel):
    signature:str
    type:str
    extended_type:str

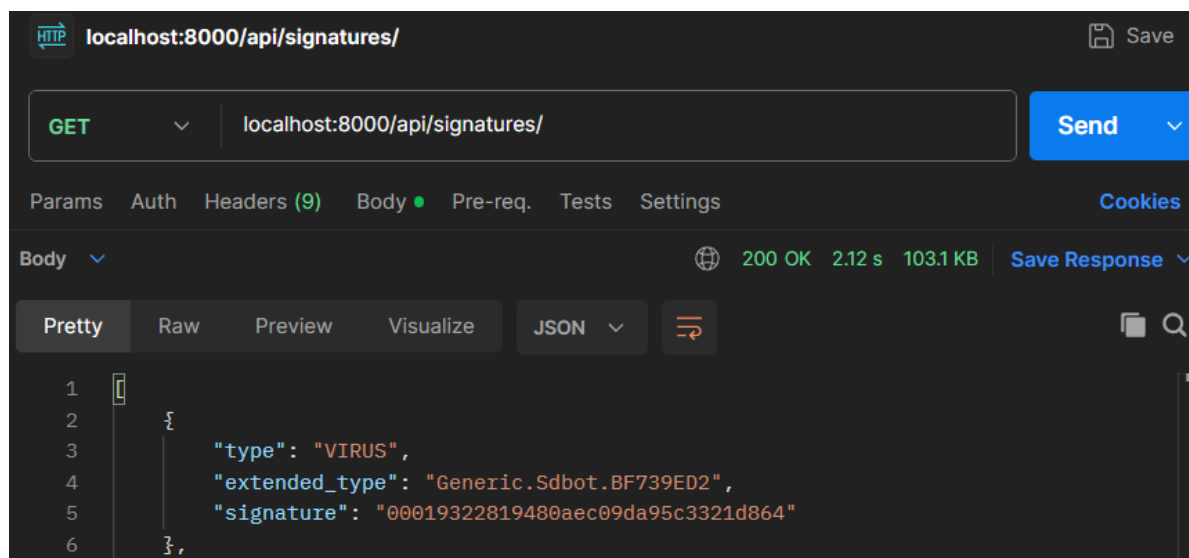
router = APIRouter(prefix="/api/signatures", tags=["Signatures"])
```

De momento, este router solo tiene un método útil en la aplicación, debido a que no hay implementadas funciones de desarrollador para introducir nuevas firmas, editar las ya existentes o eliminarlas directamente desde la API.

Listado de Firmas: Devuelve las firmas guardadas de la base de datos.

```
@router.get("/")
def get_all_signatures(db:Session = Depends(get_db)):
    return db.query(Signature).all()
```

Comprobación:



1.3.4 Empleo de seguridad con PyJWT

La librería que utiliza Python para generar JWT es PyJWT. Aun teniendo eso, es necesario configurar la autenticación, o cual se hará con el módulo de seguridad de FastAPI (fastapi.security):

Se necesitarán dos clases de dicho módulo: OAuth2PasswordBearer, lo cual se utilizará para crear un esquema de autorización por el que se cifrarán las peticiones de la API, y OAuth2PasswordRequestForm, que servirá para recibir por parámetro un formulario de autenticación de usuarios.

Para empezar, creé una clase de autenticación que no dejará de funcionar como un router, por lo que fue necesario hacer un router y un esquema de autenticación que se fijara en el endpoint donde se consigue el token:

```
router = APIRouter(prefix="/api", tags=["Token Control"])
oauth2_scheme = OAuth2PasswordBearer("/api/token")
@router.post("/token") # /api/token
def login():
    [...]
```

A continuación, creé una función que recibiera por parámetro un diccionario y generara un token. Para ella, elegí una clave secreta la cual se iba a cifrar y un algoritmo de cifrado:

```
SECRET_KEY = "Clave_Magik_AV"
ALGORITHM = "HS256"
def create_token(data: dict):
    token = jwt.encode(data, SECRET_KEY, algorithm=ALGORITHM)
    return token
```

Ahora, rellené el método login para la respuesta del usuario. El formulario de OAuthPasswordRequestForm tiene varios campos, interesando en este caso el nombre de usuario y contraseña. Para facilitar las cosas, creé una norma para esta API: Todos los usuarios registrados en la aplicación pueden generar un token para la API. No es lo más seguro del mundo, pero para un proyecto pequeño sirve.

Por ende, la aplicación haría lo siguiente: Buscar un usuario con el correo rellenado en el campo username del formulario. Si lo encontraba, comprobaría que la contraseña del formulario (hasheada) coincidía con la del usuario en la base de datos. Si es así, se genera la token y se devuelve la información de la token al usuario.

```
@router.post("/token")
def login(form_data: OAuth2PasswordRequestForm = Depends(), db:Session = Depends(get_db)):
    user = db.query(User).filter(and_(User.email == form_data.username,
    User.passwd ==
    hashlib.sha256(form_data.password.encode()).hexdigest())).first()
    if user:
        token = create_token(data={"sub":user.email})
        return {"access_token": token, "token_type":"bearer"}
```

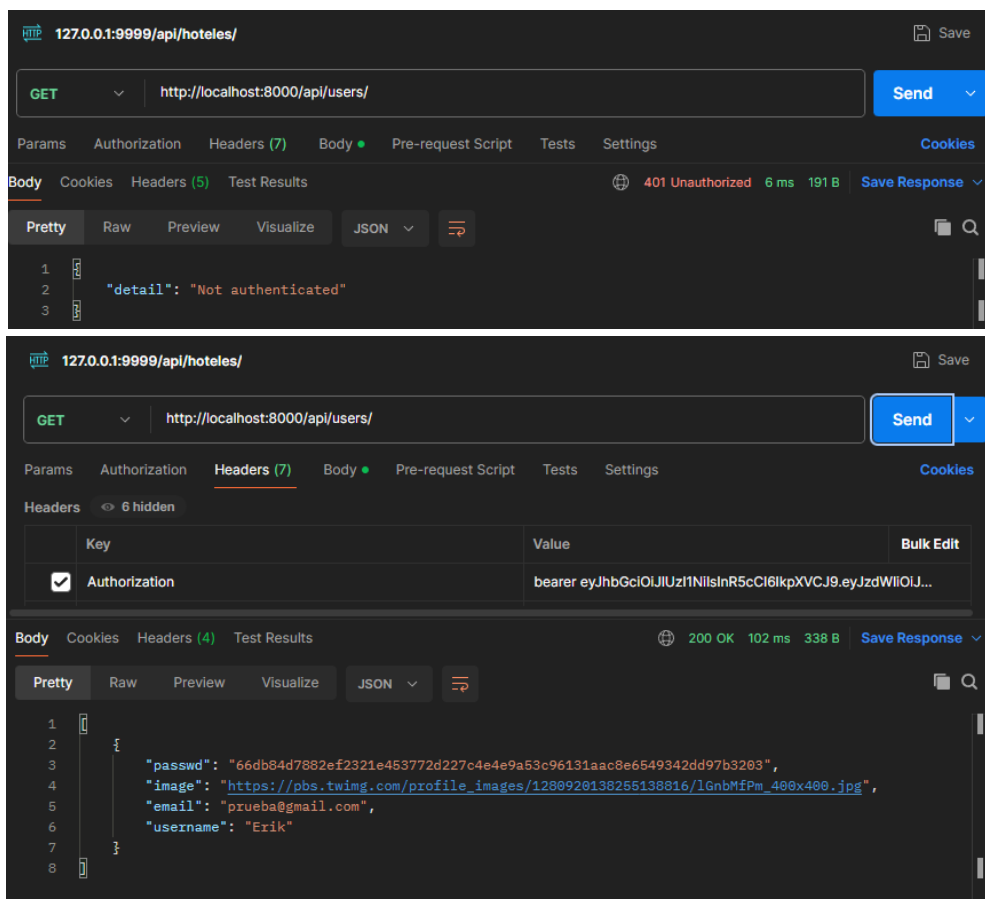
Lo único que queda es configurar la API para que un usuario sin token no tenga acceso a los endpoints no deseados. En el caso de este servicio web, para el correcto funcionamiento de la aplicación, están habilitados los siguientes endpoints:

- **/api/devices/{id}**: El identificador de un dispositivo es tan complejo que es muy difícil conseguir su identificador si no lo tienes de antemano
- **/api/devices/insert**: La aplicación debe ser capaz de guardar el dispositivo nada más iniciarse, cosa que no es posible si se requiere una token desde el principio
- **/api/signatures/**: Es una información no dañina y está sacada de sitios de código abierto (GitHub), por lo que su copia o uso es irrelevante
- **/api/users/{string}**: Aunque se pueda adivinar su nombre o correo electrónico, la contraseña sigue saliendo hasheada.
- **/api/users/insert**: Se repite la historia, se hashea la contraseña y se hacen las comprobaciones en la app de firmas, por lo que intentar crear un usuario aquí solo dará lugar a fallos y usuarios inaccesibles desde la API.

Para realizar la autorización en el resto de endpoints, se añade la dependencia del Token a la función del endpoint. Por ejemplo, como se hace en esta función:

```
@router.get("/")
def get_all_users(db:Session = Depends(get_db), token:str =
Depends(auth.oauth2_scheme)):
    users = db.query(models.User).all()
    return users
```

Comprobación de la autenticación:



The top screenshot shows a GET request to `http://localhost:8000/api/users/` resulting in a `401 Unauthorized` response. The response body is `{"detail": "Not authenticated"}`.

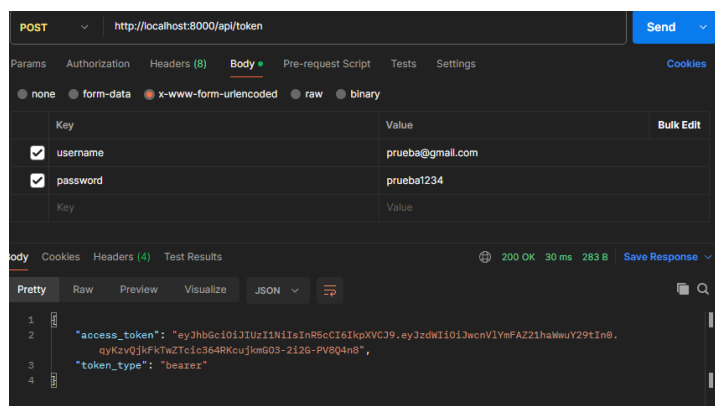
The bottom screenshot shows the same GET request, but with the `Authorization` header set to `bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIIOiJ...`. The response is `200 OK` with a JSON body containing user details:

```
{
  "passwd": "66db84d7882ef2321e453772d227c4e4e9a53c96131aac8e6549342dd97b3203",
  "image": "https://pbs.twimg.com/profile_images/1280920138255138816/1GnbMfPm_400x400.jpg",
  "email": "prueba@gmail.com",
  "username": "Erik"
}
```

1.3.5 Pruebas

1.3.5.1 Prueba de obtención de WebToken (API)

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.

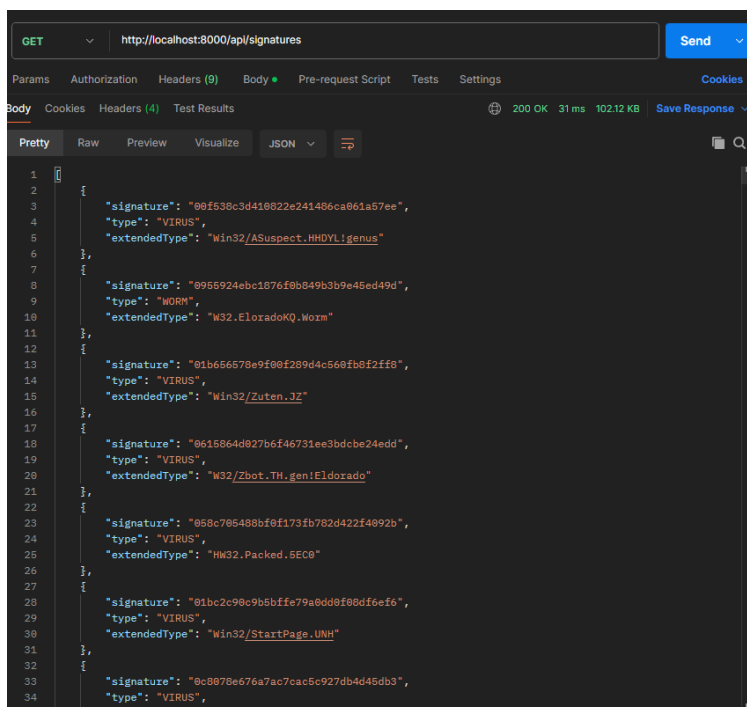


Esta clave será guardada tanto en Postman como en la propia aplicación (a modo de pruebas, luego se encargará de recoger ese token automáticamente) para su futuro uso

1.3.5.2 Prueba de Obtención de firmas (API)

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

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



1.3.5.3 Prueba de Introducción de Dispositivo (App)

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:8000/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:

flutter: {"id":"7BB60C47-4C13-4E43-8B39-9295B1A984AB"},"dev_name":"erik.amotoq","dev_type":"w
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
```

1.3.5.4 Prueba de Introducción de Usuario en Dispositivo (App)

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:8000/api/users/Prueba
flutter:

flutter: {"id":"7BB60C47-4C13-4E43-8B39-9295B1A984AB"},"dev_name":"erik.amotoq","dev_type":"w
indows","join_in":"2025-01-28 09:47:07.604843","last_scan":"2025-01-28 09:47:07.604843","use
r":"prueba@gmail.com"}
flutter:

flutter: | #0 APIReaderUtils.putData (package:magik_antivirus/utills/DBUtils.dart:121:14)
flutter: | #1 <asynchronous suspension>
flutter:

flutter: | El put del item {id: {7BB60C47-4C13-4E43-8B39-9295B1A984AB}, dev_name: erik.amot
oq, dev_type: windows, join_in: 2025-01-28 09:47:07.604843, last_scan: 2025-01-28 09:47:07.604
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
```



1.3.6 Despliegue de la aplicación

La API se encuentra desplegada en la plataforma **Vercel**, la base de datos en **FreeDB** y la aplicación se podrá descargar en la sección **Releases** de GitHub.

Aun así, durante el desarrollo he encontrado problemas con dicho despliegue de la API en red, ya que Vercel redireccionaba los y Dart no encontraba ninguna forma de leer eso correctamente, por lo que de momento lo único que puede estar más o menos disponible en red de forma medianamente asequible es la base de datos.

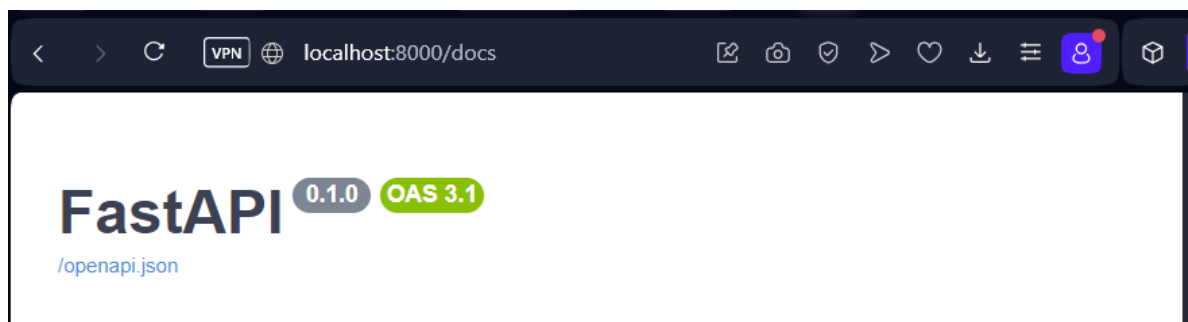
1.4 Manuales

1.4.1 Manual de usuario:

Estas instrucciones se tratan básicamente del uso de la API desde la documentación de FastAPI, una documentación que se crea automáticamente con la creación de funciones en los routers-

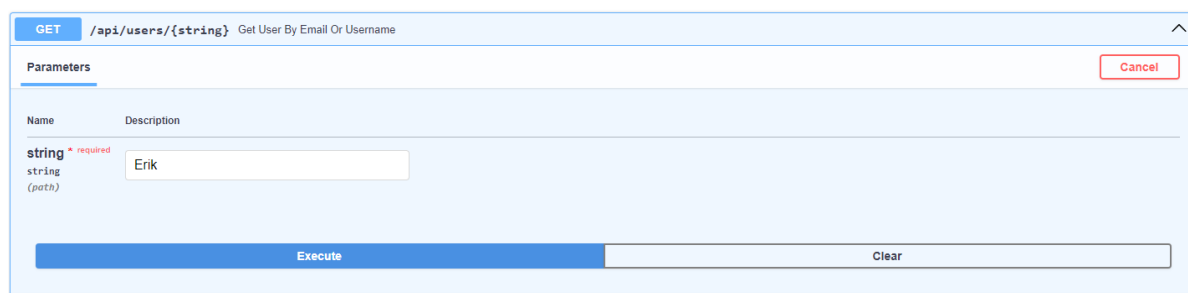
1.4.1.1 Acceso a la Documentación

Para acceder a la documentación del API, hay que ir a su host y añadir el endpoint /docs:



Dentro de la documentación, aparecen las distintas peticiones que se pueden hacer, algunas ellas protegidas por JWT.

Para utilizar las peticiones sin proteger habrá que hacer clic en uno de los endpoints, dar a “Try Out”, introducir las variables si es necesario y dar a “execute”.



En la parte inferior aparecerá el código de respuesta de la petición junto la información que devuelve, el endpoint completo de la petición realizada, etc.

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8000/api/users/Erik' \
  -H 'accept: application/json'
```

Request URL

http://localhost:8000/api/users/Erik

Server response

Code	Details
200	<p>Response body</p> <pre>{ "username": "Erik", "password": "66db84f7892ef2721e453772d27c4e4e953-96131acc8e6549342dd97b3283", "image": "https://pbs.twimg.com/profile_images/1280920138255138816/1GnbMFPm_400x400.jpg", "email": "prueba@gmail.com" }</pre> <p>Response headers</p> <pre>content-length: 210 content-type: application/json date: Fri, 31 Jan 2025 20:48:47 GMT server: uvicorn</pre>

1.4.1.2 Autenticación

Como se ha mencionado previamente, hay una serie de endpoints que aparecen con un candado a su derecha. Ese candado marca que estos están protegidos por medio de un mecanismo de autenticación por usuario y contraseña. Se podrá entrar y ejecutar sin problema el punto de acceso, pero no dará ningún tipo de información y saltará el *error 401: Unauthorized*.

GET /api/users/ Get All Users

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8000/api/users/' \
  -H 'accept: application/json'
```

Request URL

http://localhost:8000/api/users/

Server response

Code	Details
401 Unauthorized	<p>Error: Unauthorized</p> <p>Response body</p>

Para acceder a estos endpoints, es necesario verificar la identidad del usuario. Esto se hace dando al botón Authorize de la parte superior y rellenando un formulario de autenticación. Los datos que hay que introducir son las credenciales que tenga el usuario en la propia aplicación. Cuando se rellene el formulario y se verifique la autenticidad del usuario, se podrá acceder desde la documentación a todos los métodos

Available authorizations

Scopes are used to grant an application different levels of access to data on behalf of the end user. Each API may declare one or more scopes. API requires the following scopes. Select which ones you want to grant to Swagger UI.

OAuth2PasswordBearer (OAuth2, password)

Authorized

Token URL: /api/token
Flow: password
username: Erik
password: *****
Client credentials location: basic
client_secret: *****

LogoutClose

1.4.2 Manual de instalación:

El despliegue de la API se puede hacer o en local o en red, pero de momento por problemas explicados previamente da menos problemas desplegarlo de forma local:

1.4.2.1 Despliegue de Base de Datos Local

Es necesario instalar MySQL Server de forma local en el dispositivo o por medio de un contenedor de Docker. En el caso de hacerlo por medio de una instalación local en Windows:

Acceder a la [página de descargas de MySQL](#) y pulsar **MySQL Community (GPL) Downloads**

MySQL Enterprise Edition

MySQL Enterprise Edition includes the most comprehensive set of advanced features, management tools and technical support for MySQL.

[Learn More »](#)

[Customer Download from My Oracle Support \(MOS\) »](#)

[Trial Download from Oracle edelivery »](#)

[Developer Download from Oracle OTN »](#)

MySQL NDB Cluster CGE

MySQL NDB Cluster is a real-time open source transactional database designed for fast, always-on access to data under high throughput conditions.

- MySQL NDB Cluster
- MySQL NDB Cluster Manager
- Plus, everything in MySQL Enterprise Edition

[Learn More »](#)

[Customer Download from My Oracle Support \(MOS\) »](#)

[Trial Download from Oracle edelivery »](#)

[MySQL Community \(GPL\) Downloads »](#)



Elegir la opción **MySQL Installer for Windows** y descargar cualquiera de las dos versiones del instalador.

MySQL Installer 8.0.41



Note: MySQL 8.0 is the final series with MySQL Installer. As of MySQL 8.1, use a MySQL product's MSI or Zip archive for installation. MySQL Server 8.1 and higher also bundle MySQL Configurator, a tool that helps configure MySQL Server.

Select Version:

8.0.41

Select Operating System:

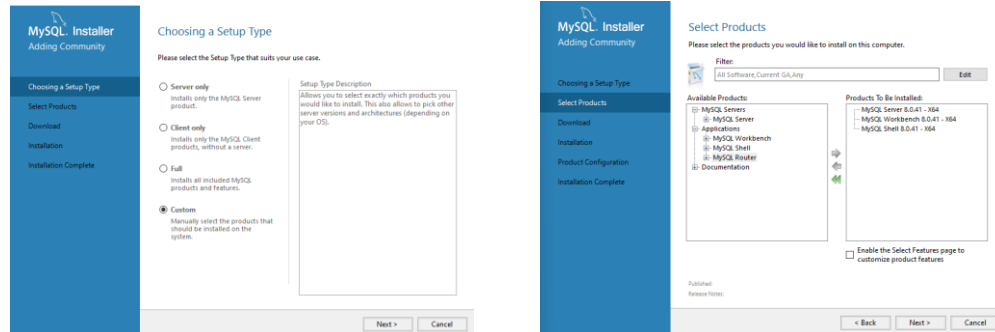
Microsoft Windows

Windows (x86, 32-bit), MSI Installer	8.0.41	2.1M	Download
(mysql-installer-web-community-8.0.41.0.msi)	MD5: 22ed92c892160254fbf0f93d811360c2 Signature		
Windows (x86, 32-bit), MSI Installer	8.0.41	352.2M	Download
(mysql-installer-community-8.0.41.0.msi)	MD5: c2e89b80cf89c2214e5ecb9f91b77f10 Signature		



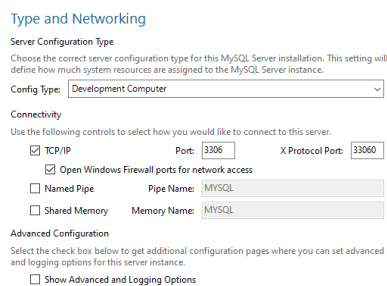
We suggest that you use the [MD5 checksums and GnuPG signatures](#) to verify the integrity of the packages you download.

En la interfaz de instalación, cambiar la instalación a **Personalizada** (Custom) y elegir las últimas versiones de MySQL Server (el servidor), MySQL Workbench (La interfaz gráfica de MySQL) y MySQL Shell (La terminal para poder ejecutar sentencias)

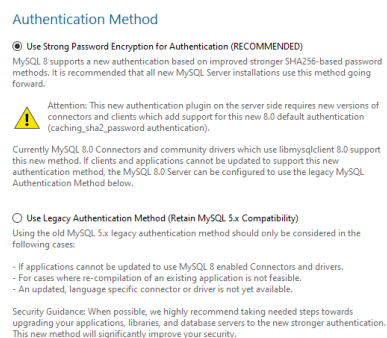


Cuando todo se instale correctamente será necesario configurar el servidor:

En la pestaña de tipo de servidor, dejar todo como está:



En el método de autenticación, dejar todo como está:



En la pestaña de cuentas y roles, añadir una contraseña para el usuario “root”, el superadministrador de la base de datos

Accounts and Roles

Root Account Password
Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password:

Repeat Password:

Las siguientes pantallas se quedan por defecto como lo prepara el instalador, y para finalizar se aplican los cambios en la última:

Windows Service

☒ Configure MySQL Server as a Windows Service

Windows Service Details
Please specify a Windows Service name to be used for this MySQL Server instance. A unique name is required for each instance.

Windows Service Name:

☒ Start the MySQL Server at System Startup

Run Windows Service as ...
The MySQL Server needs to run under a given user account. Based on the security requirements of your system you need to pick one of the options below.

☒ **Standard System Account**
Recommended for most scenarios.

☐ Custom User
An existing user account can be selected for advanced scenarios.

Server File Permissions

MySQL Installer can secure the server's data directory by updating the permissions of files and folders located at:

C:\ProgramData\MySQL\MySQL Server 8.0\Data

Do you want MySQL Installer to update the server file permissions for you?

☒ Yes, grant full access to the user running the Windows Service (if applicable) and the administrators group only. Other users and groups will not have access.

☐ Yes, but let me review and configure the level of access.

☐ No, I will manage the permissions after the server configuration.

Apply Configuration
The configuration operation has completed.

Configuration Steps | Log

- ✓ Writing configuration file
- ✓ Updating Windows Firewall rules
- ✓ Adjusting Windows service
- ✓ Initializing database (may take a long time)
- ✓ Updating permissions for the data folder and related server files
- ✓ Starting the server
- ✓ Applying security settings
- ✓ Updating the Start menu link

The configuration for MySQL Server 8.0.41 was successful.
Click Finish to continue.

Cuando esté todo instalado, habrá que acceder a la base de datos desde MySQL Workbench

Manage Server Connections

MySQL Connections
Local instance MySQL80

Connection Name:

Connection: Remote Management System Profile

Connection Method: Method to use to connect to the RDBMS

Parameters | SQL | Advanced

Hostname: Port: Name or IP address of the server host - and TCP/IP port.

Username: Name of the user to connect with.

Password: Clear The user's password. Will be requested later if it's not set.

Default Schema: The schema to use as default schema. Leave blank to select it later.

New Delete Duplicate Move Up Move Down Test Connection Close

Ahora, será ejecutar el archivo **db.sql** que se encontrará en el proyecto del API para cargar toda la base de datos local.

Ahora, en el proyecto de Python es necesario cambiar la URL de la base de datos:

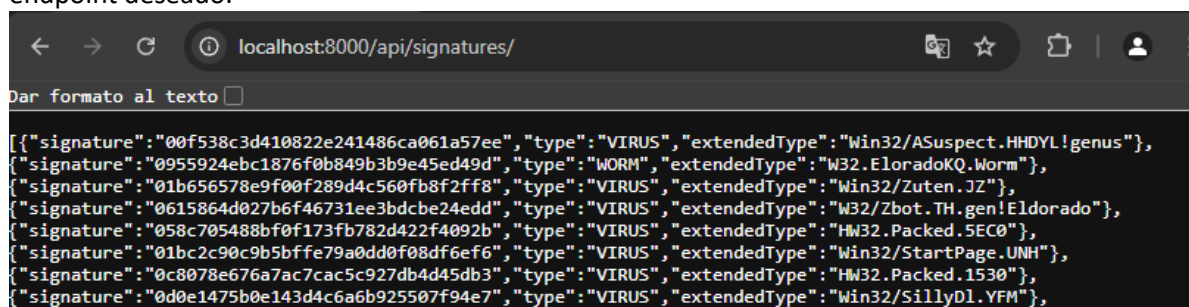
```
# URL de la BD
SQLALCHEMY_DATABASE_URL =
"mysql+mysqlconnector://root:{password}@localhost:3306/m_antivirus_db"
```

1.4.2.2 Despliegue de API Local

Con la API ya creada, es tan sencillo como iniciar el archivo main desde una terminal o desde el propio proyecto en el IDE que se esté utilizando:

```
PS D:\DAM 2\TFG\tfg_antivirus_api> & "D:/DAM 2/TFG/tfg_antivirus_api/venv/Scripts/python.exe" "d:/DAM 2/TFG/tfg_antivirus_api/main.py"
INFO: Started server process [12136]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

Para acceder a la API, se utilizará la url localhost:{puerto asignado} como host y la dirección al endpoint deseado.

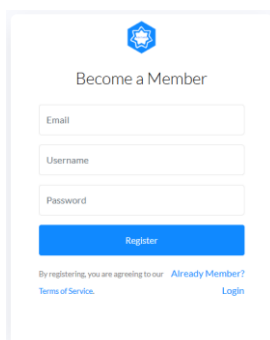


En el caso de acceder al api, hará falta ir a DBUtils, el archivo de Dart que guarda todo lo necesario para el acceso a los datos externos. En la clase APIUtils, habrá que cambiar la variable apiRESTLink al host actual

```
//Utils del API que se vaya a leer
class APIReaderUtils {
  //Enlace estático al API Rest
  ///
  ///Da igual el endpoint del api que sea, ya que esto se repite en todos.
  static String apiRESTLink = "localhost:8000";
```

1.4.2.3 Despliegue de la Base de Datos en FreeDB

Lo primero es registrarse en la web freedb.tech, poniendo una serie de credenciales para el usuario, además de otros parámetros que pidió más adelante para saber el uso de la base de datos.



Become a Member

Email

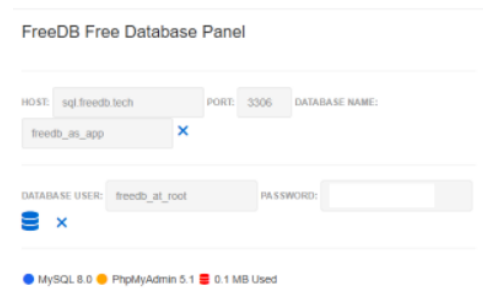
Username

Password

Register

By registering, you are agreeing to our [Terms of Service](#) [Already Member? Login](#)

Acto seguido, hay que crear una base de datos y un usuario con acceso a esta. Una vez están ambos creados de forma automática por el programa, se puede conectar a la base de datos para gestionarla o bien importando las credenciales en MySQL Workbench, o bien accediendo desde phpmyadmin, un servicio gestor de bases de datos online. Para mayor comodidad y mejor respuesta del cliente, lo hice es recomendable MySQL Workbench.



Parameters	SSL	Advanced
Hostname:	sql.freedb.tech	Port: 3306
Username:	freedb_at_root	
Password:	Store in Vault ... Clear	
Default Schema:		

Name or IP address of the server host - and TCP/IP port.

Name of the user to connect with.

The user's password. Will be requested later if it's not set.

The schema to use as default schema. Leave blank to select it later.

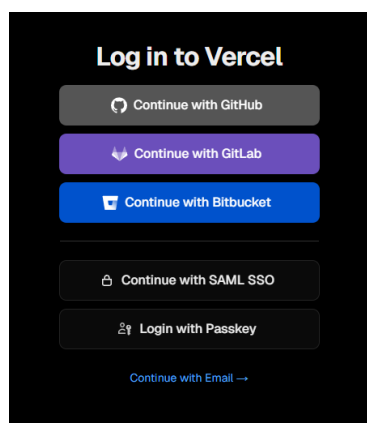
Una vez se importe el script SQL de la base de datos y se cree toda la estructura en la base de datos, hay que cambiar los parámetros de la conexión a la base de datos en Python. Hay que tener en cuenta que, además de SQLAlchemy, Python necesita el plugin mysql-connector-python para poder conectarse a una base de datos de MySQL.

```
# URL de la BD
SQLALCHEMY_DATABASE_URL =
f"mysql+mysqlconnector://{user}:{password}@sql.freedb.tech:3306/{dbname}"
```

1.4.2.4 Despliegue de la API en Vercel

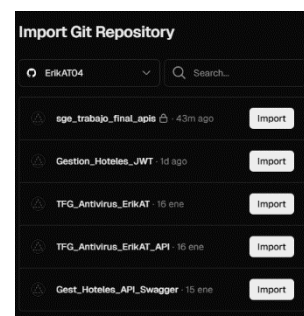
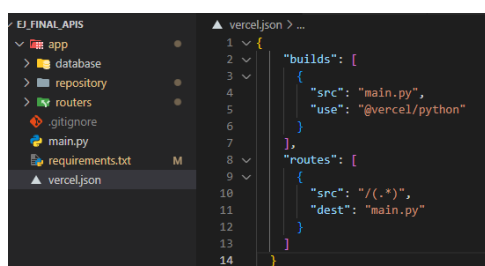
Para desplegar la API en Vercel, lo primero fue crear un repositorio para ello. Como la API va a tener datos sensibles como el usuario y la contraseña para acceder directamente a la base de datos, lo mejor es crear un repositorio privado. Vercel no tendrá problema en utilizarlo como API de todos modos.

Tras crear el repositorio, hay que registrarse en Vercel usando el perfil de Github. De este modo, se cargará automáticamente una lista con todos los repositorios, sean o no públicos.

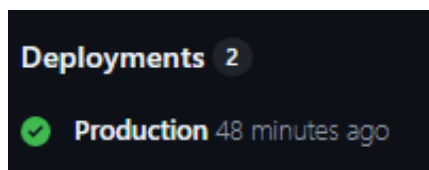
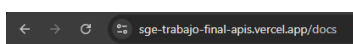


Una vez en la pantalla principal de Vercel, introducir el proyecto al servicio de APIs dando al botón 'Add New' -> 'Project'. Aparecerán ahí todos los proyectos del perfil elegido de GitHub, ordenados por fecha de última actualización. Al dar a importar, se creará en el menú de Vercel una pantalla para poder ver información de la API, aunque en este momento aparecerá un error de servidor porque no está configurada.

Para terminar de configurar la API, hace falta añadir un archivo llamado **vercel.json** a la raíz del proyecto del API, el cual tendrá la siguiente estructura:



En cuanto se haga un push a Github con este cambio, se reiniciará el servicio de Vercel y se tendrá acceso a la API sin problema, además de que se podrá ver el estado del despliegue en el propio repositorio.



Si se ha hecho algún cambio en el proyecto y se hace push, se reconstruirá la API automáticamente.

Ahora, para probar la API en red, hay que cambiar el host de APIUtils en el proyecto

```
///Utils del API que se vaya a leer
class APIReaderUtils {
    ///Enlace estático al API Rest
    ///
    ///Da igual el endpoint del api que sea, ya que esto se repite en todos.
    static String apiRESTLink = "enlace_host.vercel.app";
```

El único inconveniente encontrado con Vercel es que, a la hora de usarlo en la aplicación, no se hacían correctamente las peticiones Post y Put por una supuesta redirección, por lo que el proyecto de Dart no puede utilizar esta API, pero se puede acceder a ella sin problema desde herramientas como exploradores o incluso Postman.

1.4.3 Conclusiones y posibles ampliaciones

He cumplido los objetivos propuestos con esta aplicación, que al final eran poder manipular una base de datos en red con las propuestas operaciones CRUD y aplicarla a una aplicación multiplataforma, con el plus de desplegarlas en red para su uso fuera del área local.

Como mejora, me habría gustado poder completar el despliegue en red correctamente, ya que la aplicación en Vercel y otros servicios dan problemas a la hora de realizar peticiones distintas a GET, ya que redireccionan a otro endpoint y da error en la aplicación en Dart. Otra posible mejora es añadir en la aplicación un apartado para desarrolladores donde poder añadir firmas de alguna forma, y así implementar en la API el control completo CRUD de las firmas.

Bibliografía

[Definición de la arquitectura de microservicios](#)

[Información sobre base de datos online \(freedb\)](#)

[Información sobre despliegue de API en Vercel](#)

[Información sobre despliegue de API con FASTApi en Vercel Server](#)

[Información sobre Flask](#)

[Repositorio con el archivo de Firmas](#)

[Enlace al proyecto de la aplicación en GitHub](#)

[Enlace al proyecto del API en GitHub](#)