Imagen que contiene Forma

Descripción generada automáticamente

**PROYECTO SGE**

**2ª EVALUACIÓN**

|  |
| --- |
| CFGS Desarrollo de Aplicaciones Multiplataforma  Informática y Comunicaciones |

|  |
| --- |
| **Library of Ohara (Python API Version)** |
| ***Año:*** *2025*  ***Fecha de presentación:*** *11/02/2025* |

|  |  |
| --- | --- |
| **Nombre y Apellidos: Jaime González Bravo**  **Email: jaime.gonbra@educa.jcyl.es** |  |

Índice

[Introducción: 3](#_Toc190162978)

[Estado del Arte: 3](#_Toc190162979)

[Arquitectura de Microservicios (API): 3](#_Toc190162980)

[API (Interfaz de Programación de Aplicaciones): 3](#_Toc190162981)

[Estructura de una API: 3](#_Toc190162982)

[Formas de crear una API (FastAPI): 4](#_Toc190162983)

[Descripción general de proyecto: 4](#_Toc190162984)

[Objetivo: 4](#_Toc190162985)

[Entorno de Trabajo: 4](#_Toc190162986)

[Documentación técnica: 4](#_Toc190162987)

[Análisis del sistema: 4](#_Toc190162988)

[Diseño de la BD: 5](#_Toc190162989)

[Implementación: 7](#_Toc190162990)

[Pruebas: 14](#_Toc190162991)

[Despliegue de la Aplicación: 20](#_Toc190162992)

[Manuales 21](#_Toc190162993)

[Manual de usuario: 21](#_Toc190162994)

[Manual de instalación: 22](#_Toc190162995)

[Conclusiones: 22](#_Toc190162996)

[Bibliografía: 22](#_Toc190162997)

# Introducción:

El proyecto consiste en un (intento) de API funcional en el que se trabajen con una base de datos relacional (en este caso PosgreSQL) dando servicio a diferentes funcionalidades que se verán en el mismo.

Como resumen, quería comprobar mis conocimientos de Python para probarme a mi mismo y conseguir una API parecida a la que quiero hacer en mi TFG.

# Estado del Arte:

## Arquitectura de Microservicios (API):

Es un método de desarrollo de aplicaciones de software que funciona como un conjunto de pequeños servicios que se ejecutan de manera independiente y autónoma proporcionando una funcionalidad de negocio completa. Los microservicios se comunican entre sí a través de APIs y cuentan con un sistema de almacenamiento propio (una base de datos). De esta forma, se facilita el uso y desarrollo de estas, ya no es una aplicación de gran tamaño sino varios mini proyectos que se encargan de una cosa en específico.

## API (Interfaz de Programación de Aplicaciones):

es un conjunto de protocoles que se usa para diseñar e integrar el software de las aplicaciones. Permiten que tus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Por tanto, simplifica el desarrollo y permite ahorrar tanto tiempo como dinero.

## Estructura de una API:

La estructura básica de una API suele seguir ciertas convenciones para permitir la interacción de clientes y servidores.

Como protocolo, se utiliza el de HTTP/HTTPS que es el estándar para todas las comunicaciones, siendo el HTTPS el mejor ya que es más seguro.

Los métodos más utilizados son:

GET: petición para obtener la información del servidor.

POST: petición para enviar datos (se suele utilizar para crear nuevos recursos).

PUT: petición para enviar datos (se suele utilizar para actualizar recursos).

PATCH: similar que el PUT, pero para un solo campo.

DELETE: petición para eliminar un recurso.

Las partes más características de una API son:

El esquema que hace referencia al protocolo (https://).

El dominio que es la dirección del servidor donde se encuentra la API (localhost:8000)

La ruta que es específica para cada recurso que la API maneja (/api/usuario/all)

Los parámetros, que ya sean los que se emplean en la propia ruta o bien, los parámetros que se utilizan en los cuerpos de las peticiones (body).

## Formas de crear una API (FastAPI):

FastAPI es uno de los frameworks más populares y rápidos a la hora de crear APIs en Python debido a su rendimiento y facilidad. Para poder utilizarlo, requieres instalar el servidor ASGI (servidor de aplicaciones asíncronas) como el denominado Uvicorn. Además, este framework, implementa la documentación interactiva del api conocido como Swagger que explicará y nos dejará probar todas las APIs que hayamos creado.

# Descripción general de proyecto:

## Objetivo:

La idea principal del proyecto es crear una API similar a la que quiero utilizar para el TFG utilizando los conocimientos aprendidos durante las clases. Además de así practicar con Python.

## Entorno de Trabajo:

Como Gestor de base de datos, he utilizado PosgreSQL ya que tiene buena sinergia trabajando con Python y, además, utilizo Docker para levantar el servicio. De esta manera, puedo ejecutar y lanzar el servicio en cualquier momento.

Para poder ver las tablas y los cambios realizados, utilizo PGAdmin que es la interfaz de usuario que se conecta a PosgreSQL para mostrar todos los campos de las tablas.

Como lenguaje de programación, he utilizado Python debido a su fácil uso además de Visual Studio Code como IDE para poder ejecutar el código. Como framework empleado para la creación de la API, empleo el ya denominado FastAPI.

# Documentación técnica:

## Análisis del sistema:

Debido a la falta de tiempo y en algunos casos, falta de conocimientos, no dispongo de aplicación móvil ni de sistema de autenticación. Lo que el usuario puede llegar a hacer es la de creación, modificación visualización y eliminación de Usuarios, Libros, Géneros y Autores. Es bastante básico, pero es lo que me ha dado tiempo a realizar.

## Diseño de la BD:

Esta es la idea de la cual partí

Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.

Finalmente, acabo así.

Texto

El contenido generado por IA puede ser incorrecto.

Como se puede ver en la imagen, lo más complicado fue el tema de las relaciones entre género, autor y libro respectivamente. Porque, aunque solo se guarde en la base de datos en libro los ids del autor y del género, cada vez que visualizo uno autor o un género, muestro también los libros a los que hace referencia dicha relación. (créeme, ha sido lo más complicado).

## Implementación:

Texto

El contenido generado por IA puede ser incorrecto.

Aunque al principio parece todo muy enrevesado, es una organización “simple”.

Por un lado, tenemos la carpeta db con todo lo relacionado con la base de datos. En ella, tenemos el fichero models con el modelo de base de datos y el de database que configura todo lo necesario para la conexión a base de datos.

Texto

El contenido generado por IA puede ser incorrecto.

Utilizo la biblioteca de sqlachemy que proporciona herramientas para interactuar con bases de datos relacionales de manera eficiente y flexible (parecido a un ORM).

Texto

El contenido generado por IA puede ser incorrecto.

Siguiendo por orden, llega la carpeta modelos, en el que están como su nombre indica, las clases con los modelos que se van a utilizar para el correcto uso de la aplicación. Dichas clases son una imagen y semejanza con el de la base de datos añadiendo algunas cosas.

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

Como podemos ver en la imagen, tenemos la clase usuario que es idéntica a la de la bd y la clase UsuarioDTO que me sirve a mi a la hora de insertar un nuevo usuario o modificar uno ya existente. Utilizo pydantic que es una biblioteca de Python que se utiliza principalmente para la validación de datos y para la creación de modelos de datos de manera sencilla, eficiente y segura.

En el fichero género, también tenemos una clase Genero que es idéntica al de la bd haciendo referencia a la lista con los libros relacionado y la del DTO para facilitarme el insertado y el modificado de datos.

Texto

El contenido generado por IA puede ser incorrecto.

El fichero de Autor es muy parecido al de género. En él, tenemos tanto la clase autor que hace referencia al de la base de datos y el DTO.

Texto

El contenido generado por IA puede ser incorrecto.

Por ultimo y no menos importante, tenemos el fichero de Libro con las clases de libro y libroDTO. En dicha clase, tenemos el id que hace referencia tanto al autor y al género al que pertenece dicho libro.

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

Siguiendo con la primera imagen, tendremos el archivo repository en el que se encapsula la lógica de acceso a la base de datos. En él, se debería de meter toda la funcionalidad que realiza nuestra API (cosa interesante de saber antes de realizar este proyecto). Para simplificar, todo ese código lo tengo en los routers.

Los routers están todos juntos y representan tanto la dirección como lo que van a realizar dichos microservicios. Para resumir, solo mostraré uno ya que el resto es bastante parecido la verdad.

Como podemos ver en la imagen, inicia con el router que sirve para indicarle a la aplicación que este fichero “controla” todas las acciones de autores. Es solo una manera de organizar y no tener todo en un mismo fichero. Tal y como hemos visto con anterioridad, tenemos cada uno de los métodos de las APIs (get,post,put,delete).

Como primer método, es el método get all que recibe todos los autores de la bd. (curiosidad, en este método siempre me mostraban los libros, en los otros siempre me daban problemas.) también está el de getByID en el que me busca en base de datos un autor que tenga dicho id que pasamos por parámetro. Si no existe, le enviamos al usuario un mensaje de que no existe dicho autor y si existe, le pasamos el autor.

Texto

El contenido generado por IA puede ser incorrecto.

Como vemos en la imagen, tenemos el metodo post para añadir un autor nuevo.

Para ello, compruebo si existe un autor con dicho nombre. En caso de que exista, no permite insertarlo, en caso contrario, lo añade a la bd y muestra el mensaje de añadido.

Texto

El contenido generado por IA puede ser incorrecto.

Como vemos en la imagen, tenemos el metodo patch para modificar los campos de un autor. Para ello, compruebo si existe el autor con dicho id. En caso de que no exista, no lo modifica y manda un mensaje de error. En caso de que si, los campos del autor se sobrescriben con los que hemos enviado en el body de la operación.

En el metodo delete para borrar un autor. Al igual que el update, buscamos si existe dicho autor y en caso de que si, lo borra y como tengo en la base de datos el campo de delete-cascade (en teoria) deberia de borrar los libros a los que hace referencia dicho autor.

Por ultimo, tenemos una función para comprobar si existe un autor con dicho nombre. Devolverá true si existe o false en caso contrario.

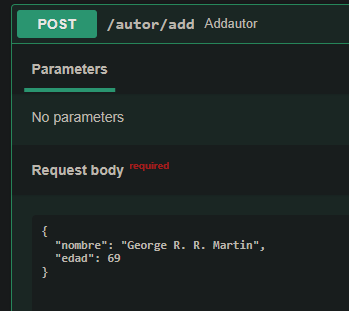
Texto

El contenido generado por IA puede ser incorrecto.

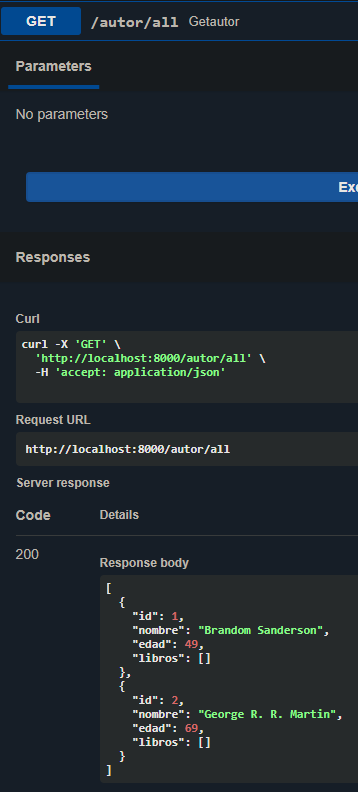
Por último, tenemos la guinda del pastel, la función que me devuelve el autor según su id. (es la que más quebraderos de cabeza me ha dado.) como podemos ver, buscamos el autor por id, y nos devuelve el autor, aquí tenía problemas porque no se me mostraba la lista de libros o me la mostraba vacía. Para solucionarlo, tuve que instanciar el libro y ponerle todos sus valores.

## Pruebas:

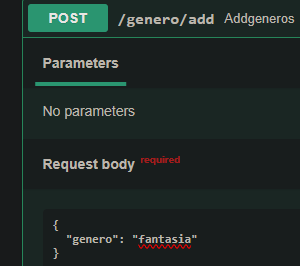
Insertar un nuevo autor:

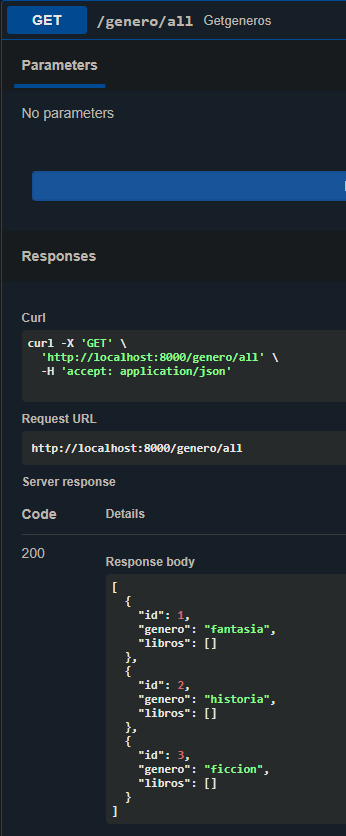


Le damos a ejecutar y nos insertará dicho autor (siempre y cuando, los datos estén bien.)

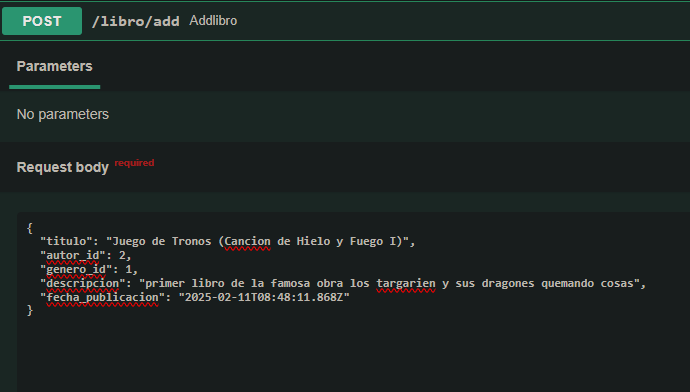


Después, creamos un genero



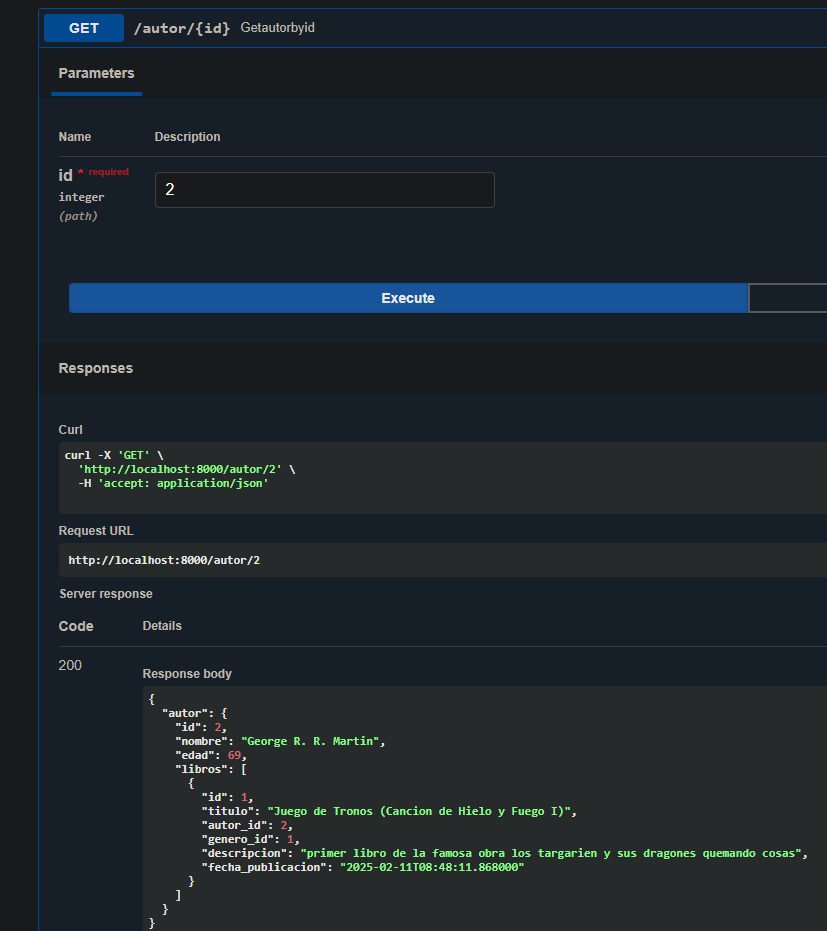


Y, por último, crearemos un libro al que le pondremos poner el genero y el autor correspondiente.

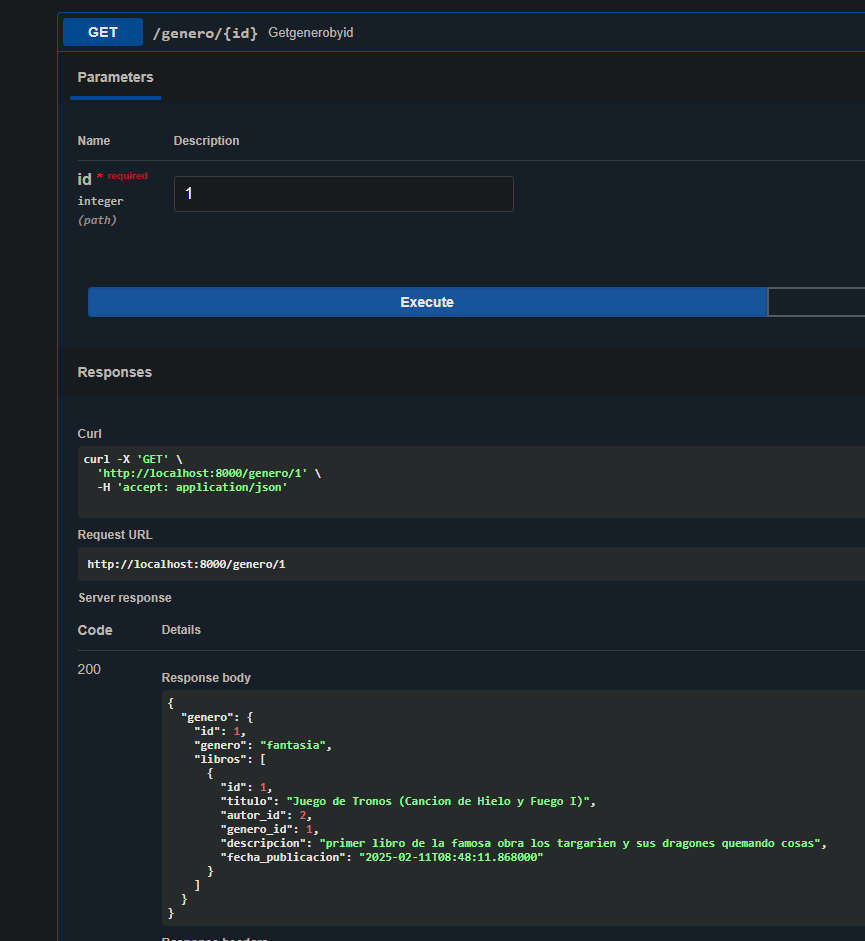


Con esto, insertaríamos un libro con dicho autor y género.

Ahora, si mostramos los datos de dicho autor, podremos comprobar que efectivamente se agrega el libro al autor



Pasara lo mismo con el genero (en mi caso, fantasia)



Esto sin duda es lo que mas me ha llevado con diferencia y no veas la de horas invertidas solo para que pueda ver las listas. Por este motivo, no he podido añadir el tema de autorización ya que no me daba tiempo.

## Despliegue de la Aplicación:

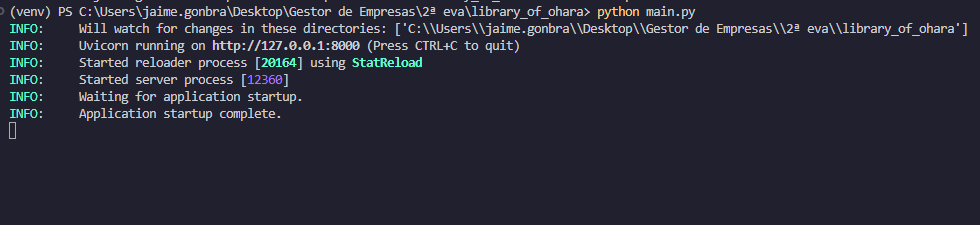
En mi caso, solo es la API y como utilizo Docker, pues levanto los contenedores que tengo en el Docker (similar a como lo tenía en el del manage).

Esto me levanta tanto el servicio de PostgreSQL donde esta la bd como pgAdmin en el puerto 80.

Por último, para correr la aplicación que corre la API, utilizo primero el comando para activar el entorno virtual



Y el de ejecutar la aplicación.



Ya cuando todo este corriendo, podemos ir a la url que vemos ahí o aquí:

[http://localhost:8000/docs#/](http://localhost:8000/docs%23/) solo que cambiando una vez que le hayamos pulsado los caracteres “%23” por el carácter “#” y funciona.

# Manuales

## Manual de usuario:



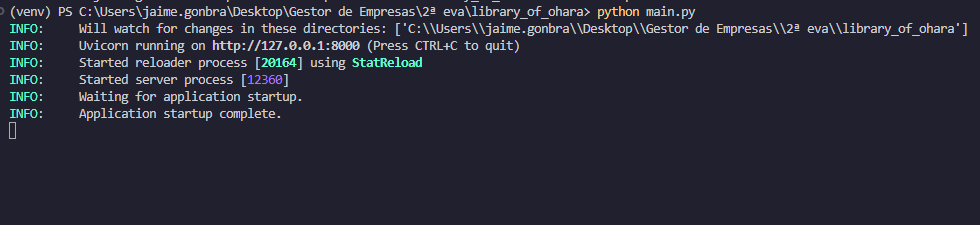
Como se puede ver en la imagen, aquí tenemos todas las operaciones que se pueden llegar a realizar utilizando este proyecto. Dichas acciones llegan desde crear, modificar, eliminar y visualizar ya sean todos los datos como solo por su id de libros, géneros, autores y usuarios.

## Manual de instalación:

En mi caso, solo es la API, para ejecutarla solo tendremos que activar el entono virtual para poder añadir todas las dependencias que requiere el proyecto.



Después, ponemos el comando para ejecutar la aplicación. el de ejecutar la aplicación.



# Conclusiones:

Cristina, este proyecto ha sido una montaña rusa de emociones, en algunos casos todo bien porque funcionaban las cosas, pero una vez que quería tener todo lo de aquí en mi casa o viceversa, fue una completa depresión. En fin, que ya está hecho y no quiero volver a tocar Python hasta dentro de unos años por lo menos.

Dos posibles ampliaciones que se podrían haber dado son:

Inclusión de la tabla usuariosLibros que como su nombre indica, seria para tener registrado todos los libros que un usuario ha leído. En bd seria una relación n: n entre usuarios y libros.

Introducir el jwt en el proyecto para darle seguridad al proyecto que, aunque sí que lo hemos dado en AAD, se hacerlo en Java, pero no en Python.

# Bibliografía:

Teoría de microservicios: [decidesoluciones.es](https://decidesoluciones.es/arquitectura-de-microservicios/%23:~:text=La%20arquitectura%20de%20microservicios%20es%20un%20m%C3%A9todo%20de%20desarrollo%20de,una%20funcionalidad%20de%20negocio%20completa.)

Teoría de API: [redhat.com](https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces)

Enlace al proyecto en GitHub: [Library of Ohara](https://github.com/KonoDIODa13/proyectoPython)