02-febrero-2025

Módulo 2 – apis rest

# introducción

En este proyecto se ha desarrollado una API utilizando FastAPI, con el objetivo de gestionar usuarios y administrar listas de canciones personalizadas. La API permite el registro, edición y eliminación de usuarios, así como la edición de canciones a sus listas personales. Para mejorar la funcionalidad, la API se conecta con Spotify, permitiendo obtener información detallada sobre las canciones agregadas.

# ENLACE A GITHUB

<https://github.com/lolafdezgUE/M2Act1.git>

# instalaciones y requisitos

Antes de ejecutar la API desarrollada, es necesario instalar las dependencias fastapi, pydantic, uvicorn y request:

Una vez instaladas las dependencias, para iniciar el servidor y ejecutar la API es necesario ejecutar el comando :

# definición de modelos

Para garantizar una organización estructurada en el desarrollo del proyecto, se han definido tres modelos o clases utilizando el módulo ***BaseModel*** de la librería ***pydantic***.

## Modelo de Canciones (song)

El modelo ***song*** representa la estructura básica de una canción, almacenando su nombre, el autor y el álbum al que pertenece.

## Modelo de Usuarios (user)

### El modelo ***user*** almacena la información esencial de cada persona registrada en el sistema. Incluye un nombre de usuario único (que funciona como ID), el nombre real, el correo electrónico y la contraseña. Además, opcionalmente, cada usuario puede contar con una lista de canciones asociadas, lo que le permite gestionar su biblioteca musical de manera personalizada.

## Modelo de Registro de Canciones (newSongRequest)

### El modelo ***newSongRequest*** facilita la incorporación de nuevas canciones a la lista de un usuario. Para ello se definen dos atributos clave: el nombre de usuario, que identifica a la persona a la que se le añadirá la canción, y el título de la canción que se desea agregar.

# endpoints generados

En el proyecto se han generado diferentes endpoints con diferentes funcionalidades.

## GET /users

### Este endpoint tiene como objetivo obtener un listado de todos los usuarios registrados en el sistema. Para ello, no es necesario enviar ningún parámetro, ya que simplemente devuelve una lista con todos los usuarios. La respuesta consistirá en un JSON que contiene un array de usuarios, donde cada usuario tiene su nombre de usuario, nombre completo, correo electrónico y las canciones asociadas. Si no hay usuarios registrados en el sistema se mostrará un mensaje indicador.

### Llamada y respuesta recibida:

### 

### Respuesta forzada al no haber usuarios:

### 

## GET /users/{usrName}

### Este endpoint permite obtener los detalles de un usuario específico a partir de su nombre de usuario (usrName). Solo es necesario proporcionar el nombre de usuario en la URL, y el sistema lo buscará en la “base de datos”. Si el usuario existe, se devolverá un objeto JSON con su información. En caso de que no se encuentre, el sistema responderá con un error 404 indicando que no se ha encontrado ningún usuario con el nombre proporcionado.

### Llamada y respuesta recibida:

### 

## POST /users/newUser

### Este endpoint se utiliza para registrar un nuevo usuario en el sistema. Para ello, el cliente debe enviar un objeto JSON con los datos del usuario, que incluyen su nombre de usuario (usrName), nombre completo (name), correo electrónico (email) y contraseña (password). El sistema primero validará que el nombre de usuario y el correo electrónico no estén ya registrados. Si alguno de estos datos ya existe, el sistema devolverá un error. Si los datos son válidos, el nuevo usuario será agregado al sistema y la respuesta incluirá un mensaje de éxito junto con la información del usuario recién creado.

### Llamada y respuesta recibida:

### 

### Error forzado intentando crear un usuario con el mismo nombre de usuario que otro ya existente:

### 

### Error forzado intentando crear un usuario con el mismo correo que el de otro usuario ya existente:

### 

## PUT /users/editUser

### Este endpoint permite editar la información de un usuario ya registrado. Para realizar la modiciación, el cliente debe enviar los datos actualizados del usuario, o sea, el nombre, el correo electrónico y/o la contraseña. El sistema identificará al usuario por su nombre de usuario (usrName) y actualizará la información proporcionada. De modo que, si la actualización es exitosa, la respuesta incluirá un mensaje confirmando los cambios y los nuevos datos del usuario. Sin embargo, si el nombre de usuario no existe o se ha modificado, se devolverá un error 404.

### Llamada y respuesta recibida:

### 

### Error forzado buscando un usuario inexistente:

### 

### Error forzado intentando editar el correo electrónico y poniendo uno que pertenece a otro usuario:

### 

## DELETE /users/{usrName}

### Este endpoint tiene como propósito eliminar a un usuario del sistema. El cliente debe proporcionar el nombre de usuario en la URL, y el sistema procederá a eliminar al usuario correspondiente. Si el usuario es encontrado, se eliminará del listado y se devolverá un mensaje confirmando la eliminación. En caso de que no se encuentre el usuario, el sistema responderá con un error 404.

### Llamada y respuesta recibida:

### 

### Error forzado buscando un usuario inexistente:

### 

## POST /users/newSong

### Este endpoint permite añadir una nueva canción a la lista de canciones de un usuario. El cliente debe proporcionar el nombre de usuario (usrName) y el nombre de la canción que desea añadir (newSong). El sistema realiza una búsqueda en la API de Spotify para obtener los detalles de la canción, en concreto, su autor y albúm. Si la canción ya está registrada en la lista del usuario, se devolverá un error indicando que la canción ya existe. Si la canción no está en la lista, se añadirá al usuario y se devolverá la lista de usuarios actualizada con las canciones correspondientes.

### Llamada y respuesta recibida:

### 

### Error forzado metiendo una canción que ya está registrada en el usuario:

### 

# conclusiones y observaciones

Este proyecto me ha permitido aprender y practicar varios aspectos esenciales en el desarrollo backend, desde los mecanismos de autenticación hasta el diseño de APIs, la integración con servicios externos y la gestión de usuarios. Además, me gustaría destacar las siguientes conclusiones en concreto.

## VALIDACIÓN DE ENTRADAS Y GESTIÓN DE ERRORES

Durante la realización de este proyecto, he comprobado que, antes de nada, es fundamental asegurarse de que todos los datos de entrada al sistema sean válidos. Por esta razón, en la configuración de los diferentes endpoints se han implementado verificaciones que garantizan que los datos sean correctos, que existan y que no se repitan. Además, si alguna de estas verificaciones detecta un problema, se gestiona el error de manera adecuada, informando al usuario de que algo no ha ido bien.

## USO FASTAPI

La realización de este proyecto ha sido una excelente oportunidad para familiarizarme con la librería **FastAPI**. Esta herramienta me ha permitido definir endpoints de forma ágil, clara y bien estructurada. Además, el uso de **Pydantic** para la validación de datos y la gestión de excepciones HTTP ha sido fundamental para evitar errores. Finalmente, la experiencia adquirida en la construcción de una API, en la implementación de las operaciones CRUD y en el manejo de los datos ha sido altamente enriquecedora.

## USO DE APIs EXTERNAS

La integración de APIs externas ha demostrado ser una gran ventaja, ya que permite reutilizar servicios ya desarrollados por terceros para enriquecer nuestros proyectos. Sin embargo, es importante tener en cuenta los límites de estos servicios externos, manejar correctamente los mecanismos de autenticación (como el token en este caso) y estar preparados para gestionar posibles errores, como la falta de datos (por ejemplo, cuando una canción es un single y no tiene álbum).