# **Documentación API**

Contenido

[**Documentación API** 1](#_Toc133150241)

[**Métodos Login** 2](#_Toc133150242)

[ **Login** 2](#_Toc133150243)

[ **Obtener información del usuario logeado** 2](#_Toc133150244)

[**Métodos User** 3](#_Toc133150245)

[ **Obtener lista de usuarios** 3](#_Toc133150246)

[ **Obtener un usuario concreto** 4](#_Toc133150247)

[ **Crear usuario** 5](#_Toc133150248)

[ **Actualizar información usuario** 6](#_Toc133150249)

[ **Actualizar contraseña de usuario** 7](#_Toc133150250)

[ **Eliminar usuario** 7](#_Toc133150251)

[**Métodos Students** 8](#_Toc133150252)

[ **Obtener lista de estudiantes** 8](#_Toc133150253)

[ **Obtener estudiante por su correo** 9](#_Toc133150254)

[ **Crear un estudiante** 10](#_Toc133150255)

[ **Actualizar información de un estudiante** 11](#_Toc133150256)

[ **Eliminar un estudiante** 11](#_Toc133150257)

[**Métodos Documents** 12](#_Toc133150258)

[ **Obtener información de documentos por estudiante** 12](#_Toc133150259)

[ **Subir información de documentos por estudiante** 13](#_Toc133150260)

[ **Actualizar información de documentos por estudiante** 14](#_Toc133150261)

[ **Eliminar información de documentos por estudiante** 15](#_Toc133150262)

[**Esquemas** 16](#_Toc133150263)

[ TokenSchema 16](#_Toc133150264)

[ UserSchema: 16](#_Toc133150265)

[ StudentSchema: 16](#_Toc133150266)

## **Métodos Login**

* **Login**

**curl -X 'POST' \**

**'http://127.0.0.1:8000/login/' \**

**-H 'accept: application/json' \**

**-H 'Content-Type: application/x-www-form-urlencoded' \**

**-d 'grant\_type=&username={mail}.com&password={password}&scope=&client\_id=&client\_secret='**

Objetivo:

El objetivo de la función 'login' es autenticar a un usuario verificando su nombre de usuario y contraseña, y luego generar y devolver un token de acceso utilizando JWT (JSON Web Tokens) para posteriores peticiones a la API.

Entradas:

- form: una instancia de la clase OAuth2PasswordRequestForm que contiene las credenciales de acceso del usuario (nombre de usuario y contraseña).

- db: una instancia de la clase Session de SQLAlchemy que proporciona una conexión a la base de datos para buscar al usuario en la base de datos.

Flujo:

1. La función llama a la función 'search\_user' para buscar el usuario en la base de datos utilizando el nombre de usuario proporcionado.

2. Si se encuentra el usuario, la función verifica la contraseña proporcionada utilizando el método 'verify' de la clase CryptContext.

3. Si la contraseña es correcta, la función genera un token de acceso utilizando JWT y lo devuelve junto con el tipo de token.

Salidas:

- TokenSchema:

- access\_token: un Web Token JSON que contiene el nombre de usuario del usuario y la hora de caducidad.

- token\_type: una cadena que indica el tipo de token devuelto (portador).

* **Obtener información del usuario logeado**

**curl -X 'GET' \**

**'http://127.0.0.1:8000/login/me' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token...\**

Objetivo:

El objetivo de la función "me" es devolver los datos del usuario actualmente conectado. Es una petición GET que requiere autenticación y autorización.

Entradas:

- user: una instancia de la clase UserSchema que se obtiene de la función current\_user.

Flujo:

1. La función recibe una petición para obtener los datos del usuario actualmente logueado.

2. Llama a la función current\_user para comprobar si el usuario está autorizado a acceder a los datos.

3. Si el usuario está autorizado, la función devuelve los datos del usuario.

Salida:

- UserSchema instance: los detalles del usuario actualmente conectado.

## **Métodos User**

* **Obtener lista de usuarios**

**curl -X 'GET' \**

**'http://127.0.0.1:8000/user/' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token...**

Objetivo:

El objetivo de la función es recuperar una lista de todos los usuarios de la base de datos, pero sólo si el usuario actual tiene el rol de administrador.

Entradas:

- user: una instancia de UserSchema, obtenida de la función current\_user en el módulo authentication\_users.

- db: una instancia de Session, obtenida de la función get\_db en el módulo config.db.

Flujo:

1. Comprueba si el usuario actual tiene el rol de administrador. Si no es así, lanza una HTTPException con código de estado 401 y detalle "Usuario no autorizado".

2. Consultar en la base de datos todas las instancias de UsuarioDB.

3. Devuelve la lista de usuarios.

Salidas:

- Una lista de instancias de usuarios, recuperadas de la base de datos con el esquema usuario.

**[**

**{**

**"id": 0,**

**"usuario": "string",**

**"id\_rol": 0,**

**"estado": 0,**

**"created\_at": "2023-04-05T01:02:32.621Z"**

**}**

**]**

* **Obtener un usuario concreto**

**curl -X 'GET' \**

**'http://127.0.0.1:8000/user/{mail}' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token... \**

Objetivo:

El objetivo de la función es recuperar la información del usuario de la base de datos basándose en el nombre de usuario proporcionado y devolverla en forma de objeto UserSchema. La función también comprueba si el usuario actual tiene autorización para acceder a esta información.

Entradas:

- mail: una cadena que representa el mail de usuario del usuario para el que se desea recuperar la información

- db: un objeto Session que representa la sesión de base de datos

- usuario: un objeto UserSchema que representa al usuario actual que realiza la petición

Flujo:

1. La función recibe como entradas el nombre de usuario, la sesión de la base de datos y el usuario actual.

2. La función comprueba si el usuario actual tiene autorización para acceder a la información de usuario.

3. Si el usuario está autorizado, la función llama a la función search\_user para recuperar la información del usuario de la base de datos.

4. La información de usuario recuperada se devuelve en forma de objeto UserSchema.

Salida:

- Un objeto UserSchema que representa la información de usuario recuperada.

* **Crear usuario**

**curl -X 'POST' \**

**'http://127.0.0.1:8000/user/' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token... \**

**-H 'Content-Type: application/json' \**

**-d '{**

**"id": 0,**

**"usuario": "DYlQalK7C.DofPy-TQBe6i\_mnkRto.COyuA5GlXLtABw4xKTjaajcGoQhUjoR.l@luYbG7AmcUhEId2xwIMB26TxVQWawfDeUkUEqxscH8VMfwPieEC4t129NoqNQe5YxBLMTgPG5jgWLXovZ.iIk",**

**"password": "string",**

**"id\_rol": 2,**

**"estado": 1,**

**"created\_at": "2023-04-05T01:11:40.216Z"**

Objetivo:

El objetivo de la función es crear un nuevo usuario en la base de datos. Recibe los datos del usuario, verifica si el usuario está autorizado para crear un nuevo usuario, comprueba si el nombre de usuario ya existe, realiza el hash de la contraseña, crea un nuevo usuario en la base de datos y devuelve el usuario creado.

Entradas:

- datos\_usuario: datos del usuario a crear

- db: Session = Depends(get\_db) - dependencia de la sesión de la base de datos

- usuario: UserSchema = Depends(current\_user) - dependencia del usuario actual

Flujo:

1. Verificar si el usuario actual está autorizado para crear un nuevo usuario

2. Comprobar si el nombre de usuario ya existe en la base de datos

3. Hash de la contraseña

4. Crear un nuevo usuario en la base de datos

5. Devuelve el usuario creado

Salida:

- UserSchema - el usuario creado

- HTTPException - si el usuario actual no está autorizado o si el nombre de usuario ya existe

* **Actualizar información usuario**

**curl -X 'PUT' \**

**'http://127.0.0.1:8000/user/update/{id}\**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token...\**

**-H 'Content-Type: application/json' \**

**-d '{**

**"usuario": "cols5VDGAV4dOLHCMwRE0\_wii0@uspnJQNOJRLZ-PFEErKrngEdQaKZz1ThRaacEd.wtjh",**

**"password": "string",**

**"estado": int**

**}'**

Objetivo:

El objetivo de la función es actualizar los datos del usuario en la base de datos. La función recibe como entradas el ID de usuario, los nuevos datos de usuario y la sesión de la base de datos. También requiere que el usuario esté autenticado y autorizado para realizar la operación de actualización.

Entradas:

- id: número entero que representa el ID de usuario que se va a actualizar

- nuevos\_datos\_usuario: Objeto UserUpdateSchema que contiene los nuevos datos de usuario a actualizar

- db: Objeto de sesión SQLAlchemy que representa la sesión de base de datos

- user: Objeto UserSchema que representa al usuario autenticado y autorizado que realiza la operación de actualización

Flujo:

1. Comprobar si el usuario está autorizado para realizar la operación de actualización

2. Recuperar los datos del usuario de la base de datos utilizando el ID de usuario proporcionado

3. Si no se encuentra el usuario, lanzar una HTTPException con código de estado 404

4. Actualiza los datos del usuario con los nuevos datos proporcionados en el objeto UserUpdateSchema

5. Confirmar los cambios en la base de datos

6. Actualizar los datos de usuario de la base de datos

7. Devuelve los datos de usuario actualizados como un objeto UserSchema

Salida:

- Objeto UserSchema que representa los datos de usuario actualizados.

* **Actualizar contraseña de usuario**

**curl -X 'PUT' \**

**'http://127.0.0.1:8000/user/password' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token...\**

**-H 'Content-Type: application/json' \**

**-d '{**

**"password": "string"**

**}'**

Objetivo:

El objetivo de la función es actualizar la contraseña del usuario actual. Recibe la nueva contraseña como entrada, la hashea usando bcrypt, y actualiza la contraseña en la base de datos.

Entradas:

- nueva\_contraseña: un esquema que contiene la nueva contraseña.

- db: una dependencia de sesión de base de datos.

- user: una dependencia del esquema de usuario obtenida de la función current\_user.

Flujo:

1. Consultar la base de datos para obtener el objeto usuario.

2. Si no se encuentra el usuario, lanzar una HTTPException con un código de estado 404.

3. Si la nueva contraseña no es None, hashéala usando bcrypt y actualiza la contraseña del usuario en la base de datos.

4. Consigna los cambios en la base de datos y actualiza el objeto usuario.

5. Devuelve el objeto usuario actualizado.

Salida:

- El objeto usuario actualizado (userSchema).

* **Eliminar usuario**

**curl -X 'DELETE' \**

**'http://127.0.0.1:8000/user/{id}' \**

**-H 'accept: \*/\*' \**

**-H 'Authorization: Bearer token...\**

Objetivo:

El objetivo de la función es eliminar un usuario de la base de datos si el usuario está autorizado para ello. La función toma como entradas el id de usuario, la sesión de base de datos y el esquema de usuario y devuelve un código de estado indicando el éxito o fracaso de la operación.

Entradas:

- id: un entero que representa el id del usuario a eliminar

- db: un objeto de sesión de base de datos obtenido de la función get\_db

- user: un objeto UserSchema obtenido de la función current\_user

Flujo:

1. Comprueba si el usuario está autorizado a eliminar un usuario comprobando su rol

2. Obtener el usuario de la base de datos utilizando el id

3. Si no se encuentra el usuario, lanzar una HTTPException con un código de estado 404

4. Eliminar el usuario de la base de datos

5. Confirmar los cambios en la base de datos

Resultados:

- Si el usuario está autorizado y el usuario se encuentra en la base de datos, la función devuelve un código de estado de 204 indicando éxito.

- Si el usuario no está autorizado, la función lanza una HTTPException con un código de estado 401

- Si el usuario no se encuentra en la base de datos, la función genera una HTTPException con un código de estado 404

## **Métodos Students**

* **Obtener lista de estudiantes**

**curl -X 'GET' \**

**'http://127.0.0.1:8000/student/' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token...\**

Objetivo:

El objetivo de la función es recuperar una lista de todos los estudiantes de la base de datos. Se accede a la función a través de una petición HTTP GET y requiere la autenticación de un usuario con un ID de rol de 1.

Entradas:

- user: una instancia de la clase UserSchema que representa al usuario autenticado que realiza la petición.

- db: una instancia de la clase Session que representa la sesión de base de datos

Flujo:

1. La función recibe una petición HTTP GET al endpoint "/estudiante".

2. La función comprueba si el usuario autenticado tiene un ID de rol de 1. Si no es así, se lanza una excepción HTTP 401 No autorizado.

3. La función consulta en la base de datos todas las instancias del modelo EstudianteDB.

4. La función devuelve una lista de todos los estudiantes en la base de datos como instancias de la clase EstudianteDBSchema.

Salida:

- Una lista de todos los estudiantes en la base de datos con el esquema StudentSchema.

* **Obtener estudiante por su correo**

**curl -X 'GET' \**

**'http://127.0.0.1:8000/student/{correo}' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token...\**

Objetivo:

El objetivo de la función es recuperar los detalles de un estudiante de la base de datos basándose en su dirección de correo electrónico. La función también comprueba el rol del usuario y utiliza su dirección de correo electrónico si no es un administrador.

Entradas:

- correo: una cadena que representa la dirección de correo electrónico del estudiante a recuperar

- usuario: una instancia de UserSchema que representa al usuario actual que realiza la solicitud

- db: una instancia de Session que representa la sesión de la base de datos

Flujo:

1. La función comprueba el rol del usuario y utiliza su dirección de correo electrónico si no es administrador.

2. La función llama a la función search\_student para recuperar los detalles del estudiante de la base de datos.

3. 3. La función devuelve los datos del estudiante.

Resultados:

- Una instancia de StudentSchema que representa los detalles del estudiante recuperados.

* **Crear un estudiante**

**curl -X 'POST' \**

**'http://127.0.0.1:8000/student/' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token...\**

**-H 'Content-Type: application/json' \**

**-d '{**

**"id": 0,**

**"id\_curso": 0,**

**"correo": "string",**

**"nombres": "string",**

**"apellidos": "string",**

**"documento\_identidad": 0,**

**"edad": 0,**

**"telefono": 0,**

**"estado": "string"**

**}'**

Objetivo:

El objetivo de la función es crear un nuevo alumno en la base de datos. Recibe los datos del alumno a crear, verifica si el usuario que realiza la petición está autorizado, comprueba si el alumno ya existe en la base de datos, y finalmente añade el nuevo alumno a la base de datos.

Entradas:

- data\_student: un objeto StudentSchema que contiene los datos del alumno a crear.

- db: un objeto Session que representa la sesión de la base de datos.

- user: un objeto UserSchema que representa al usuario que realiza la petición.

Flujo:

1. Verificar si el usuario que realiza la petición está autorizado.

2. Convertir el objeto data\_student en un diccionario.

3. Consultar la base de datos para comprobar si ya existe un alumno con el mismo correo electrónico.

4. Si el alumno ya existe, lanzar una HTTPException con código de estado 409.

5. Si el alumno no existe, crear un nuevo objeto EstudianteDB con los datos del diccionario data\_student.

6. Añade el nuevo alumno a la base de datos.

7. Confirme los cambios en la base de datos.

8. Actualice el objeto student\_db para obtener los datos actualizados de la base de datos.

9. Devuelve el objeto student\_db.

Resultados:

- student\_db: un objeto StudentSchema que representa al estudiante recién creado.

* **Actualizar información de un estudiante**

**curl -X 'PUT' \**

**'http://127.0.0.1:8000/student/update/{correo}' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token...\**

**-H 'Content-Type: application/json' \**

**-d '{**

**"id\_curso": 0,**

**"correo": "string",**

**"nombres": "string",**

**"apellidos": "string",**

**"documento\_identidad": 0,**

**"edad": 0,**

**"telefono": 0,**

**"estado": "string"**

**}'**

Objetivo:

El objetivo de la función "actualizar\_alumno" es actualizar la información de un alumno en la base de datos. La función recibe el email del alumno a actualizar, los nuevos datos a actualizar y la información del usuario para autenticarse.

Entradas:

- correo: una cadena que representa el email del alumno a actualizar.

- nuevos\_datos\_alumno: un esquema que representa los nuevos datos a actualizar del alumno.

- db: una dependencia de la sesión de base de datos obtenida de la función "get\_db".

- user: un esquema que representa la información del usuario con fines de autenticación.

Flujo:

1. Comprobar si el usuario tiene el rol de administrador, si no es así, establecer el email al email del usuario.

2. Llamar a la función "buscar\_alumno" para obtener el alumno a actualizar.

3. Iterar sobre los nuevos datos y actualizar los atributos correspondientes del objeto alumno.

4. Consignar los cambios en la base de datos.

5. Refrescar el objeto alumno para obtener los datos actualizados.

6. Devuelve el objeto estudiante actualizado.

Resultados:

- student\_db: un esquema que representa el objeto estudiante actualizado.

* **Eliminar un estudiante**

**curl -X 'DELETE' \**

**'http://127.0.0.1:8000/student/{id-estudiante}' \**

**-H 'accept: \*/\*' \**

**-H 'Authorization: Bearer token...\**

Objetivo:

El objetivo de esta función es eliminar un alumno de la base de datos, dado el ID del alumno. La función también comprueba si el usuario que realiza la petición está autorizado para realizar esta acción.

Entradas:

- id: número entero que representa el ID del alumno a borrar

- db: Dependencia de sesión de SQLAlchemy obtenida de la función get\_db

- user: Dependencia UserSchema obtenida de la función current\_user

Flujo:

1. Comprobar si el usuario que realiza la petición está autorizado para realizar esta acción

2. Recupera al alumno de la base de datos utilizando el ID proporcionado

3. Si no se encuentra el alumno, lanzar una HTTPException con código de estado 404

4. Eliminar el estudiante de la base de datos

5. Confirmar los cambios en la base de datos

Resultados:

- Si la función se ejecuta correctamente, devuelve sin contenido con código de estado 204

- Si el usuario que realiza la petición no está autorizado, se lanza una HTTPException con código de estado 401

- Si el estudiante no se encuentra, se lanza una HTTPException con código de estado 404

## **Métodos Documents**

* **Obtener información de documentos por estudiante**

**curl -X 'GET' \**

**'http://127.0.0.1:8000/documents/{idStudent}' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token… \**

Objetivo:

El objetivo de la función es recuperar todos los documentos asociados a un alumno concreto identificado por su ID. La función también comprueba el rol del usuario y recupera el ID del alumno si el usuario no es administrador.

Entradas:

- id\_alumno: un entero que representa el ID del alumno cuyos documentos se quieren recuperar.

- user: instancia de UserSchema que representa al usuario actual.

- db: una instancia de Session que representa la sesión de la base de datos.

Flujo:

1. Comprobar el rol del usuario y recuperar el ID del alumno si el usuario no es administrador.

2. Consultar la base de datos para recuperar todos los documentos asociados al alumno identificado por el ID.

3. 3. Devolver los documentos recuperados.

Resultados:

- Una lista de instancias de DocumentsDBSchema representando los documentos recuperados.

* **Subir información de documentos por estudiante**

**curl -X 'POST' \**

**'http://127.0.0.1:8000/documents/' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token…\**

**-H 'Content-Type: application/json' \**

**-d '{**

**"id": int,**

**"id\_estudiante": int,**

**"id\_lista\_documentos": int,**

**"estado": " string ",**

**"url\_documento": "string"**

**}'**

Objetivo:

El objetivo de la función "subir\_documento" es subir un nuevo documento a la base de datos y asociarlo a un alumno. Si el usuario que sube el documento no es un administrador, la función buscará al estudiante asociado al usuario y utilizará su ID para asociarle el documento.

Entradas:

- "datos\_documento": un objeto DocumentsDBSchema que contiene los datos del nuevo documento que se va a subir.

- user": un objeto UserSchema que contiene información sobre el usuario que sube el documento

- db": objeto SQLAlchemy Session utilizado para interactuar con la base de datos.

Flujo:

1. Comprobar si el usuario que sube el documento es un administrador

2. Si no lo es, busca al alumno asociado al usuario y utiliza su ID para asociarle el documento

3. Crea un nuevo objeto DocumentoDB utilizando los datos de "documento\_datos"

4. Añade el nuevo documento a la base de datos utilizando el objeto Session de SQLAlchemy

5. Confirmar los cambios en la base de datos

6. Refresca el objeto DocumentoDB para asegurarte de que tiene los últimos datos de la base de datos

7. Devolver el objeto DocumentoDB como salida de la función

Salida:

- Un objeto DocumentsDBSchema que representa el documento recién cargado

* **Actualizar información de documentos por estudiante**

**curl -X 'PUT' \**

**'http://127.0.0.1:8000/documents/' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token…\**

**-H 'Content-Type: application/json' \**

**-d '{**

**"id": int,**

**"id\_estudiante": int,**

**"id\_lista\_documentos": int,**

**"estado": " string ",**

**"url\_documento": "string"**

**}'**

Objetivo:

El objetivo de la función es actualizar un documento en la base de datos. La función recibe los nuevos datos del documento, el usuario que realiza la petición y una sesión de la base de datos. Si el usuario no es administrador, la función busca el ID de alumno asociado al usuario y lo asigna al documento. A continuación, la función busca el documento en la base de datos y actualiza sus atributos con los nuevos datos. Finalmente, la función devuelve el documento actualizado.

Entradas:

- nuevos\_datos\_documento: un objeto DocumentsDBSchema que contiene los nuevos datos para el documento

- user: un objeto UserSchema que contiene información sobre el usuario que realiza la petición

- db: un objeto SQLAlchemy Session que representa la sesión de la base de datos

Flujo:

1. Comprobar si el usuario es un administrador. Si no lo es, busca el ID de estudiante asociado al usuario y asígnalo al documento.

2. Buscar el documento en la base de datos utilizando el ID del alumno y el ID del documento.

3. Si no se encuentra el documento, lanzar una HTTPException con un código de estado 404.

4. Actualizar los atributos del documento con los nuevos datos.

5. Consignar los cambios en la base de datos y actualizar el objeto documento.

6. Devuelve el documento actualizado.

Salida:

- Devuelve un objeto DocumentsDBSchema que representa el documento actualizado.

* **Eliminar información de documentos por estudiante**

**curl -X 'DELETE' \**

**'http://127.0.0.1:8000/documents/' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token…\**

**-H 'Content-Type: application/json' \**

**-d '{**

**"id\_estudiante": int,**

**"id\_lista\_documentos": int**

**}'**

Objetivo:

El objetivo de la función es eliminar un documento de la base de datos a partir del identificador del documento proporcionado en la entrada. La función también comprueba si el usuario tiene los permisos necesarios para borrar el documento y, en caso contrario, asocia el documento al alumno que realiza la petición.

Entradas:

- identificador\_documento: Un esquema que contiene el identificador del documento (id\_estudiante e id\_lista\_documentos) a borrar.

- db: Una dependencia de sesión de base de datos obtenida de la función get\_db.

- user: Una dependencia del esquema de usuario obtenida de la función current\_user.

Flujo:

1. Comprobar si el usuario tiene los permisos necesarios para eliminar el documento.

2. Si el usuario no tiene los permisos necesarios, asociar el documento al alumno que realiza la solicitud.

3. Consultar la base de datos para encontrar el documento a eliminar a partir del identificador del documento proporcionado en la entrada.

4. Si no se encuentra el documento, lanzar una HTTPException con código de estado 404.

5. Eliminar el documento de la base de datos.

6. Confirmar los cambios en la base de datos.

Resultados:

- Si el documento se elimina correctamente, la función devuelve un código de estado 204 (Sin contenido).

## **Métodos DocumentList**

* **Obtener la lista de documentos requeridos**

**curl -X 'GET' \**

**'http://127.0.0.1:8000/documentList/' \**

**-H 'accept: application/json' \**

**-H 'Authorization: Bearer token…**

Objetivo:

El objetivo de la función es recuperar una lista de todos los documentos de la base de datos y devolverla como respuesta al cliente. La función está diseñada para ser accedida a través de una petición HTTP GET y requiere autenticación del usuario.

Entradas:

- user: una instancia de la clase UserSchema que representa al usuario autenticado

- db: una instancia de la clase Session que representa la sesión de base de datos

Flujo:

1. Se accede a la función a través de una petición HTTP GET al endpoint '/documentList'.

2. El usuario actual se autentica mediante la función current\_user del módulo authentication\_users.

3. La sesión de la base de datos se obtiene utilizando la función get\_db del módulo config.db.

4. Se ejecuta una consulta sobre la sesión de base de datos para recuperar todos los documentos de la tabla ListaDocumentosDB.

5. La lista de documentos se devuelve como respuesta al cliente.

Salidas:

- lista\_documentos: una lista de todos los documentos recuperados de la base de datos.

**Esquemas**

* TokenSchema

**{**

**"access\_token": "token",**

**"token\_type": "bearer"**

**}**

* UserSchema:

**{**

**"id": int,**

**"usuario": "hOt5NOF8aRZRv3A5FLs9\_iPNpPAMjjm5NkivON89mjkrk3h51Q-zSzFtVpuIrmumjV8vigy..TVGb1CFs@-Jrx-zjnTQzCR-FDn.xi4-07pJvlIvz-iTqGhpMmnfOK.Gw",**

**"id\_rol": int,**

**"estado": int,**

**"created\_at": "2023-04-05T01:07:58.507Z"**

**}**

* StudentSchema:

**{**

**"id": int,**

**"id\_curso": int,**

**"correo": "string",**

**"nombres": "string",**

**"apellidos": "string",**

**"documento\_identidad": int,**

**"edad": int,**

**"telefono": int,**

**"estado": "string"**

**}**

* DocumentsDBSchema

**{**

**"id": 0,**

**"id\_estudiante": 0,**

**"id\_lista\_documentos": 0,**

**"estado": "string",**

**"url\_documento": "string"**

**}**