DOCUMENTACIÓN API 150 V.02

INTRODUCCIÓN

Esta es la documentación creada sobre el commit 45 de la API 150, creada por los alumnos de 2ºDAM, Adrián Blanco Martínez y Gabriel Sánchez Amorín, teniendo como supervisor técnico a Diego García Ordás.

A lo largo del presente documento, se explican las funciones del proyecto, así como su forma de uso correcto.

TABLA DE CONTENIDO

Contenido

ESTRUCTURA DE FICHEROS

La estructura de ficheros del api es la siguiente:

Api150

- Api
  - Acts
    - Delete.php
    - Insert.php
    - List.php
    - listOne.php
    - update.php
  - Auth
    - createUser.php
    - login.php
    - logout.php
  - Greetings
    - Delete.php
    - Insert.php
    - List.php
    - listOne.php

        - update.php
    - Hymns
        - Delete.php
        - Insert.php
        - List.php
        - listOne.php
        - update.php
    - Materials
        - Delete.php
        - Insert.php
        - List.php
        - listOne.php
        - update.php
    - Phrases
        - Delete.php
        - Insert.php
        - List.php
        - listOne.php
        - update.php
    - Pjenvironments
        - Delete.php
        - Insert.php
        - List.php
        - listOne.php
        - update.php
    - Pray
        - Delete.php
        - Insert.php
        - List.php
        - listOne.php
        - update.php
    - Story
        - Delete.php
        - Insert.php
        - List.php
        - listOne.php
        - update.php
    - visits
        - Delete.php
        - Insert.php
        - List.php
        - listOne.php
        - update.php
- Config
    - Database.php
- Objects
    - DAO.php
    - Sesión.php

    - User.php
- Test
    - testApi.php
- útil
    - act.php
    - ambiente.php
    - commonFunctions.php
    - Greetings.php
    - Historia.php
    - Hymn.php
    - Logger.php
    - Material.php
    - Phrase.php
    - Pray.php
    - uploadFilesByURL.php
    - visit.php

EXPLICACIÓN Y JUSTIFICACIÓN DE LA ESTRUCTURA DE FICHEROS

En nuestra API, la estructura de ficheros cuelga de un directorio raíz denominado api150.

En el siguiente nivel encontramos los directorios de api, config, objects, test y útil.

El directorio api contendrá los endpoint necesarios para realizar las funciones con la base de datos.

Cada uno de los directorios de los endpoint contendrá 5 ficheros php: delete, insert, list, listOne y update.

A excepción del endpoint Auth, que contiene createUser, login y logout.

El directorio config contendrá todo lo relativo al objeto Database.

El directorio objects contendrá el DAO y una serie de ficheros necesarios para el login y logout.

El directorio test contiene el fichero de testeo de la API (Previsto para la siguiente versión de la API).

El directorio útil contiene los ficheros necesarios para hacer funcionar de forma correcta los endpoint, creando los mismos una representación en objeto de los elementos que se introducen o extraen de la base de datos, así como el fichero con la lógica de funcionamiento de la subida de ficheros por URL.

FUNCIONAMIENTO

ACTOS

DELETE

Recibe por el método DEL un json en el body de la petición los datos “id” y “token”.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Tras esto, comprueba que la id recibida no esté vacía. Si la id está vacía arrojará un error.

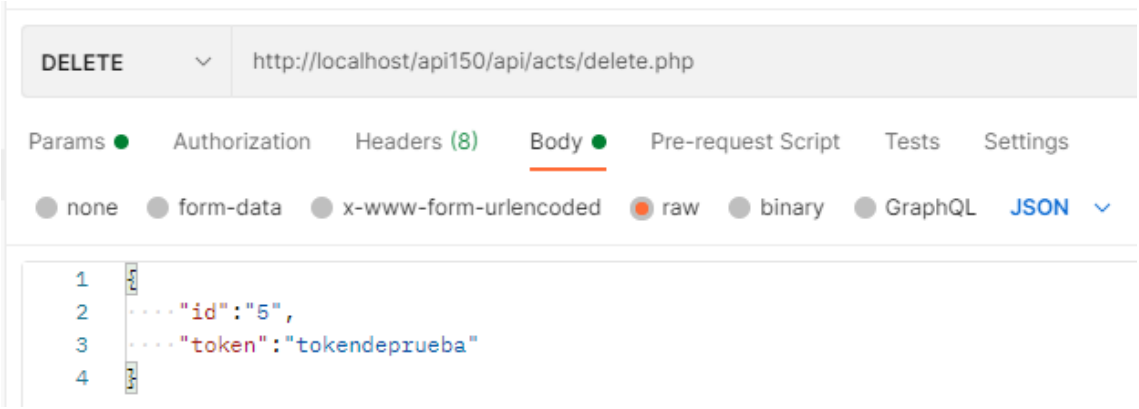
En caso contrario, procede a comprobar que, efectivamente, existe un registro asignado a esa id en la base de datos. Si esto es cierto, se elimina el elemento.

Si no existe, se arroja un error.

Método: DEL

Parámetros: id, token

Respuestas: 200, 406, 400, 401, 403.



INSERT

Recibe por el método POST un json en el body de la petición los datos titulo, fecha, enUso, categoría, token y medios[url, tipo].

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que no hay ningún acto que corresponda a ese titulo.

Si el acto ya existe arroja un error, si no, para a comprobar si en la petición se han incluido medios.

Si no hay medios, realiza la inserción de manera normal.

En caso contrario, comprueba que el formato de los medios es correcto, inserta los medios, inserta el acto, y crea las relaciones internas en la base de datos.

Si uno o más de los medios ya existen, no se incluirá una nueva referencia, sino que se relacionará con el medio ya existente.

Si algo falla en este proceso, se recibe el único error no común de la API.

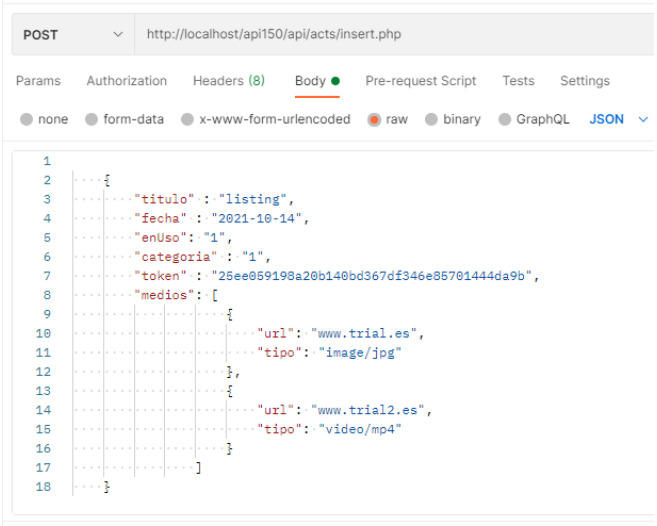
Consiste en un error 418 (“El servidor se rehúsa a hacer café con una tetera”).

Es un error no controlado, que requiere la presencia de un administrador de la base de datos. No puede tratarse desde código.

Método: POST

Parámetros: titulo, fecha, enUso, categoría, token y medios[url, tipo].

Respuestas: 200, 201, 406, 400, 401, 403, 503.



LIST

Es el endpoint más sencillo de utilizar.

Solo necesita ser llamado para mostrar todos los datos contenidos en la tabla de actos.

Debido a necesidades de limpieza, no muestra las relaciones entre los actos y los medios.

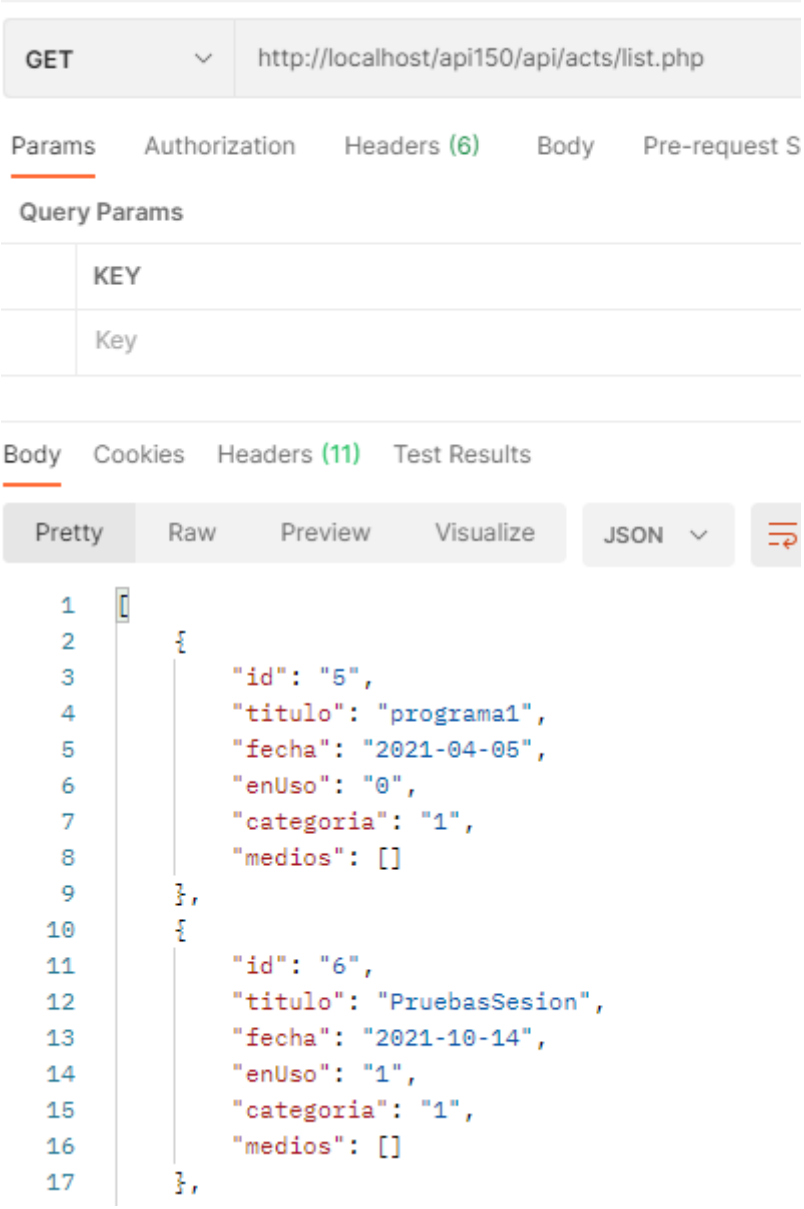
Es decir, este endpoint muestra únicamente los datos de los actos, sin entrar en los medios que contiene.

Para eso tenemos el siquiente endpoint.

Método: GET

Parámetros:

Respuestas: Lista completa o lista vacía.



LISTONE

Es el endpoint preparado para listar un acto concreto, además de todas sus relaciones.

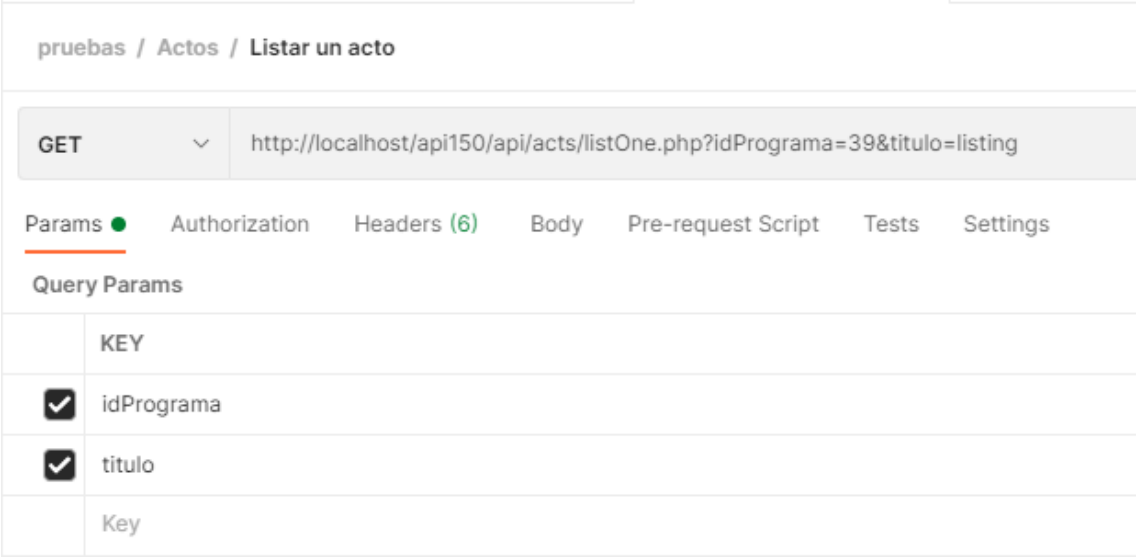
Recibe por el método GET los parámetros idPrograma y título.

Está preparado para funcionar con solo uno de ellos, cualquiera, o con ambos.

Al ser llamado, comprueba que exista, por lo menos uno de los parámetros necesarios para realizar la consulta, y devuelve el resultado.

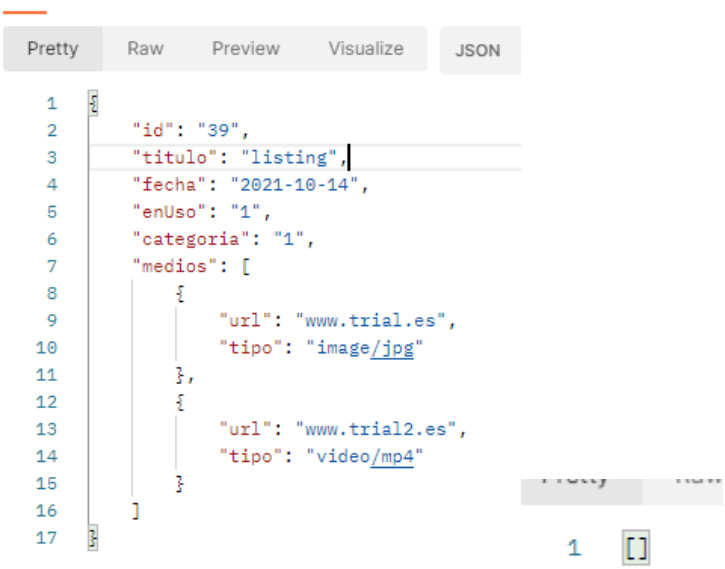
Si no se incluye ningún parámetro, devuelve un error.

Método: GET



Parámetros: idPrograma, titulo

Respuestas: 400.



UPDATE

Recibe por el método PUT los datos idPrograma, nuevoTitulo, nuevaFecha, enUso, token, y en un json en el body de la petición medios[url, tipo].

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que existe un acto que corresponda a esa id.

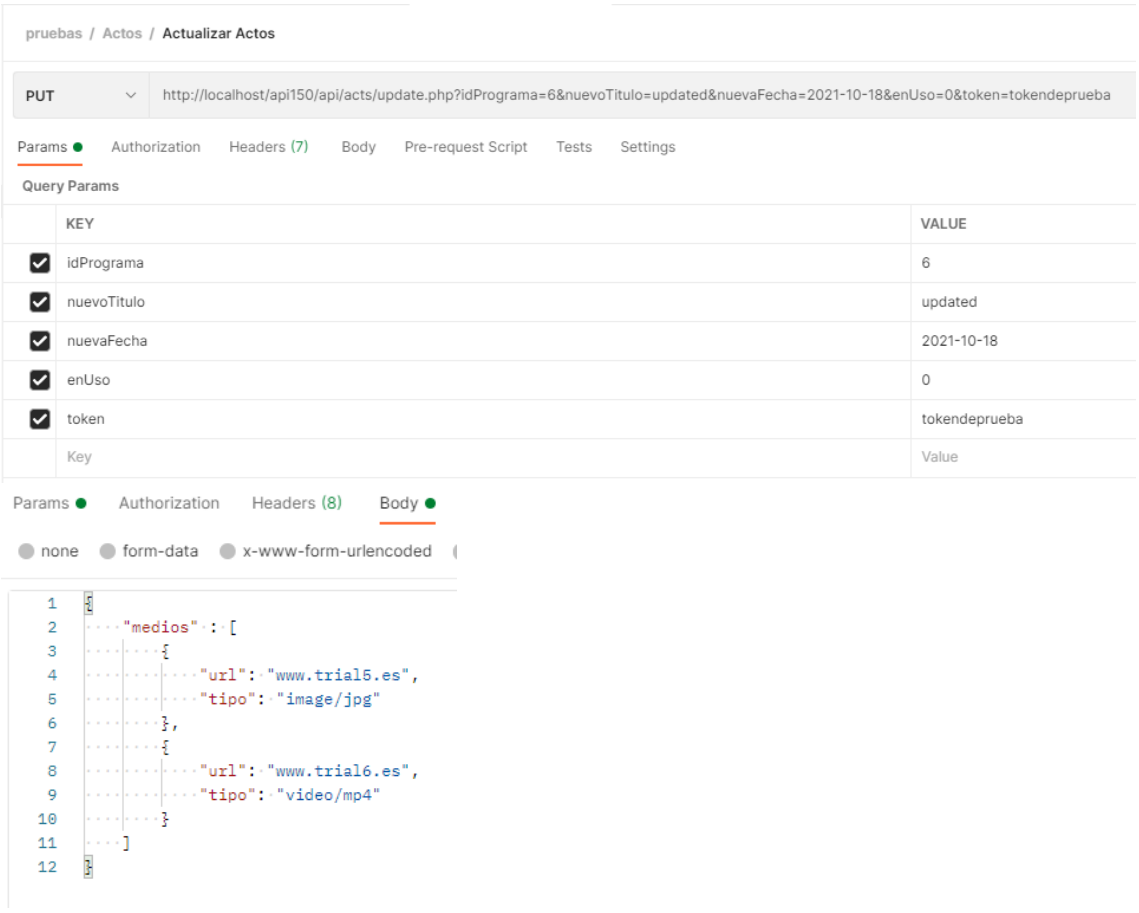
Si el acto no existe, arroja un error, en caso contrario, actualiza el acto.

Se ejecuta una actualización de los medios en la capa inferior, transparente al usuario.

Método: PUT

Parámetros: idPrograma, nuevoTitulo, nuevaFecha, enUso, token, medios[]

Respuestas: 403, 401, 400, 406, 200.



AUTH

CREATEUSER

Recibe por el método POST un json en el body de la petición los datos username, password, mail, rol y token.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

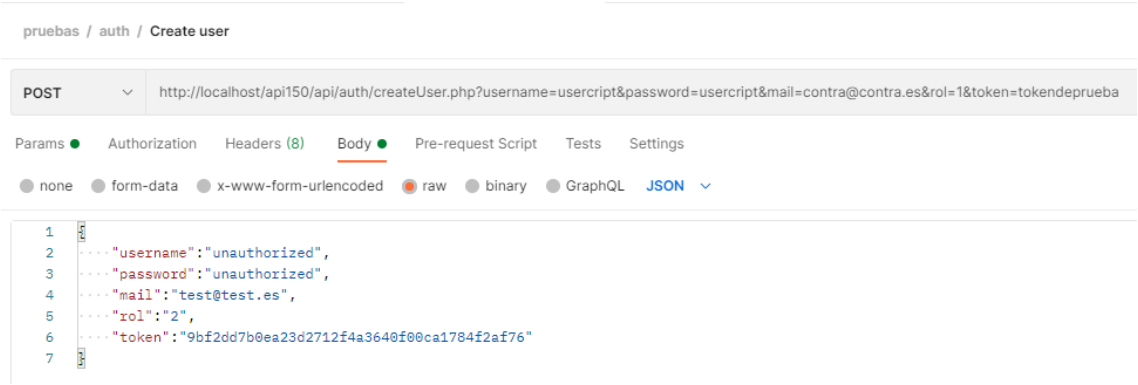
Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, crea el usuario.

Método: POST

Parámetros: username, password, mail, rol y token.

Respuestas: 403, 401, 400, 406, 200.



LOGIN

Recibe por el método POST un json en el body de la petición los datos username y password.

Comprueba que los datos no están vacíos.

Si esto es correcto, consulta en la base de datos a ver si existe ese usuario.

Si el usuario existe, genera un token que se incluye en la tabla de sesión.

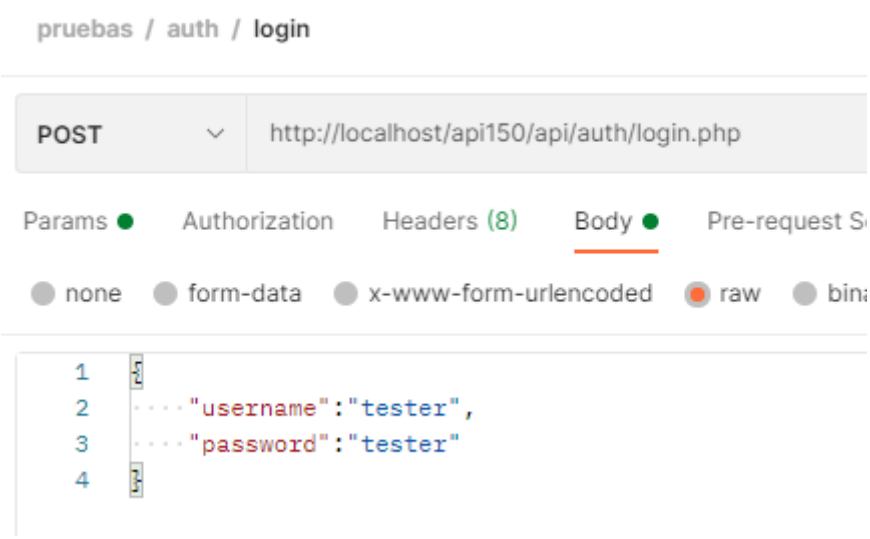
Si el usuario no existe, devuelve un token vacío.

Tras esto, se devuelve el token, que será el elemento que se enviará al resto de peticiones. Se usará a modo de acreditación para realizar operaciones en la aplicación.

Método: POST

Parámetros: username, password

Respuestas: 200.



LOGOUT

Recibe por el método POST un json en el body de la petición el token.

Comprueba que los datos no están vacíos.

Si esto es correcto, consulta en la base de datos a ver si existe ese token en la tabla de sesión.

Si el token existe, se cambia su expireDate al tiempk actual, de manera que la sesión caduca y será necesario un nuevo login para realizar cualquier nueva operación.

Si el token no se encuentra en la tabla de sesión, se devuelve un error 404.

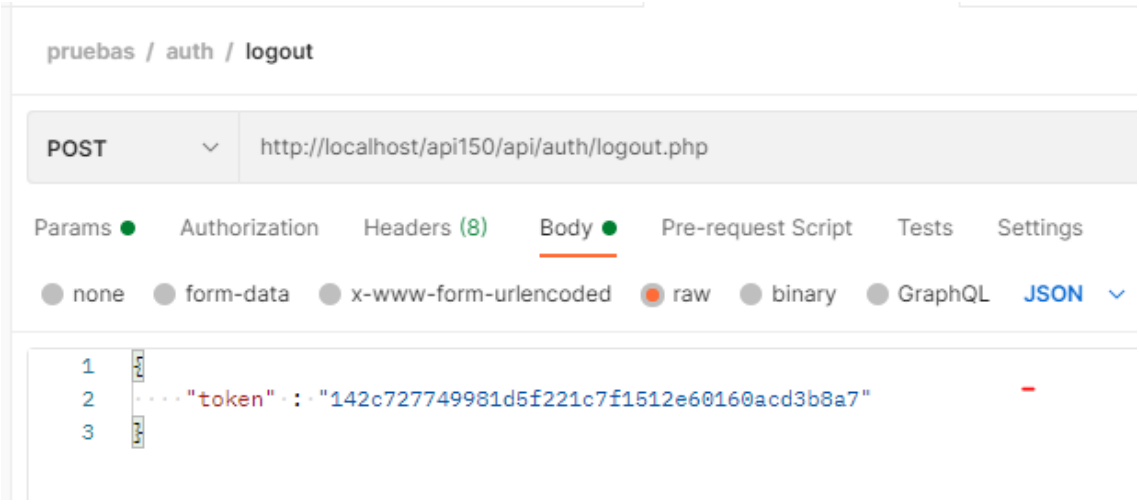
Esto es prácticamente imposible que ocurra, puesto que para acceder al logout hay que haber pasado antes por el login.

LA VISUALIZACIÓN DE ESTE ERROR SOLO PUEDE SIGNIFICAR QUE HAY PROBLEMAS EN EL CÓDIGO DEL BACKEND, UN FALLO EN LA BASE DE DATOS, O UN FALLO DE TRANSMISIÓN DEL TOKEN EN EL FRONTEND.

Método: POST

Parámetros: token

Respuestas: 200, 400.



GREETINGS

DELETE

Recibe por el método DEL un json en el body de la petición los datos “idSaludo” y “token”.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Tras esto, comprueba que la id recibida no esté vacía. Si la id está vacía arrojará un error.

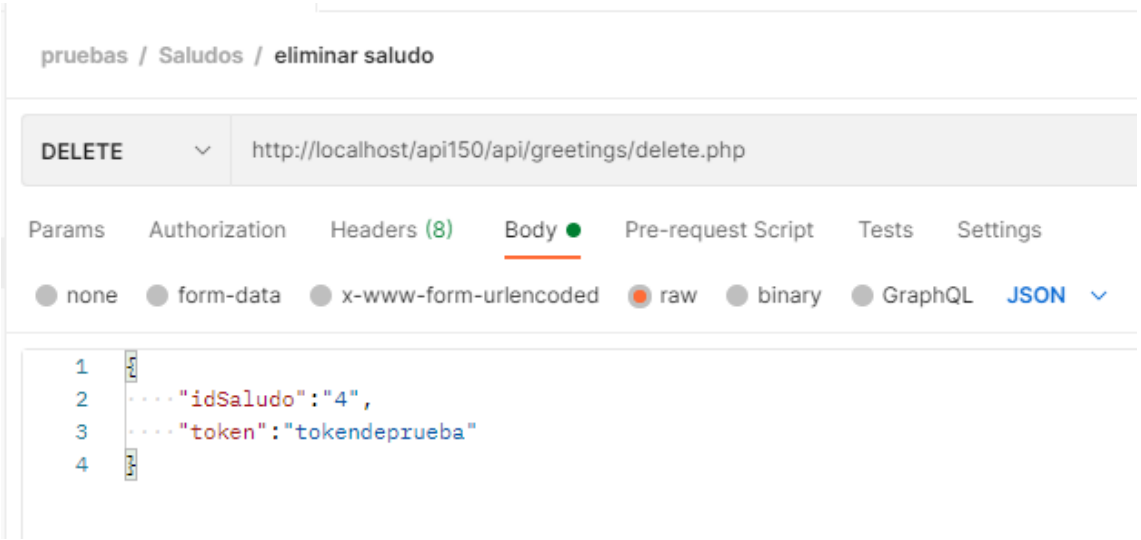
En caso contrario, procede a comprobar que, efectivamente, existe un registro asignado a esa id en la base de datos. Si esto es cierto, se elimina el elemento.

Si no existe, se arroja un error.

Método: DEL

Parámetros: idSaludo, token

Respuestas: 200, 406, 400, 401, 403.



INSERT

Recibe por el método POST un json en el body de la petición los datos titulo, descripción, texto, enUso, token, medios[url, tipo].

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que no hay ningún saludo que corresponda a ese titulo.

Si el saludo ya existe arroja un error, si no, para a comprobar si en la petición se han incluido medios.

Si no hay medios, realiza la inserción de manera normal.

En caso contrario, comprueba que el formato de los medios es correcto, inserta los medios, inserta el saludo, y crea las relaciones internas en la base de datos.

Si uno o más de los medios ya existen, no se incluirá una nueva referencia, sino que se relacionará con el medio ya existente.

Si algo falla en este proceso, se recibe el único error no común de la API.

Consiste en un error 418 (“El servidor se rehúsa a hacer café con una tetera”).

Es un error no controlado, que requiere la presencia de un administrador de la base de datos. No puede tratarse desde código.

Método: POST

Parámetros: titulo, descripción, texto, enUso, token, medios[url, tipo].

Respuestas: 200, 201, 406, 400, 401, 403, 503.

pruebas / Saludos / **insetar saludo**

POST http://localhost/api150/api/greetings/insert.php?titulo=titu2&descripcion=desc1&texto=kslrlfhjsefi&token=tokendeprueba2

Params ● Authorization Headers (8) Body ● Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "titulo":"postedMedios2",
    "descripcion":"postedMedios",
    "texto":"posted",
    "enUso":"0",
    "token":"0be63959315ab2b05e272a4cd6df598777e96239",
    "medios": [
            {
                "url": "www.trial.es",
                "tipo": "image/jpg"
            },
            {
                "url": "www.trial2.es",
                "tipo": "video/mp4"
            }
        ]
}
```

LIST

Solo necesita ser llamado para mostrar todos los datos contenidos en la tabla de saludos.

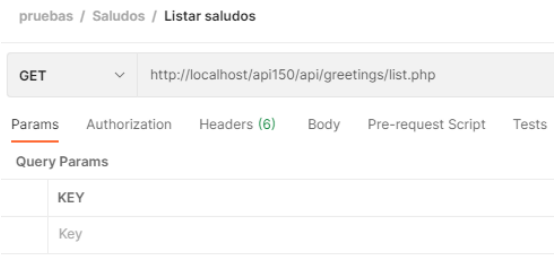
Debido a necesidades de limpieza, no muestra las relaciones entre los actos y los medios.

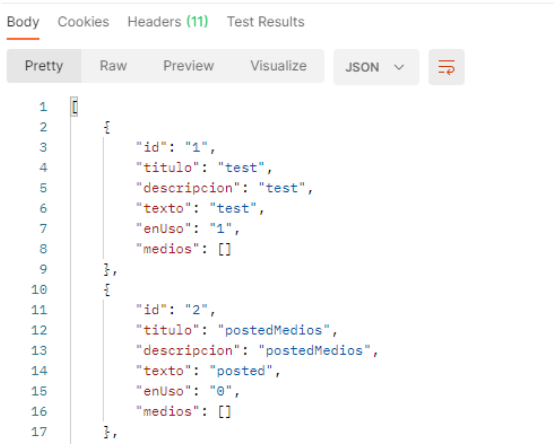
Es decir, este endpoint muestra únicamente los datos de los saludos, sin entrar en los medios que contiene.

Para eso tenemos el siquiente endpoint.

Método: GET

Parámetros:





Respuestas: Lista completa o lista vacía.

LISTONE

Es el endpoint preparado para listar un saludo concreto, además de todas sus relaciones.

Recibe por el método GET los parámetros idSaludo y título.

Está preparado para funcionar con solo uno de ellos, cualquiera, o con ambos.

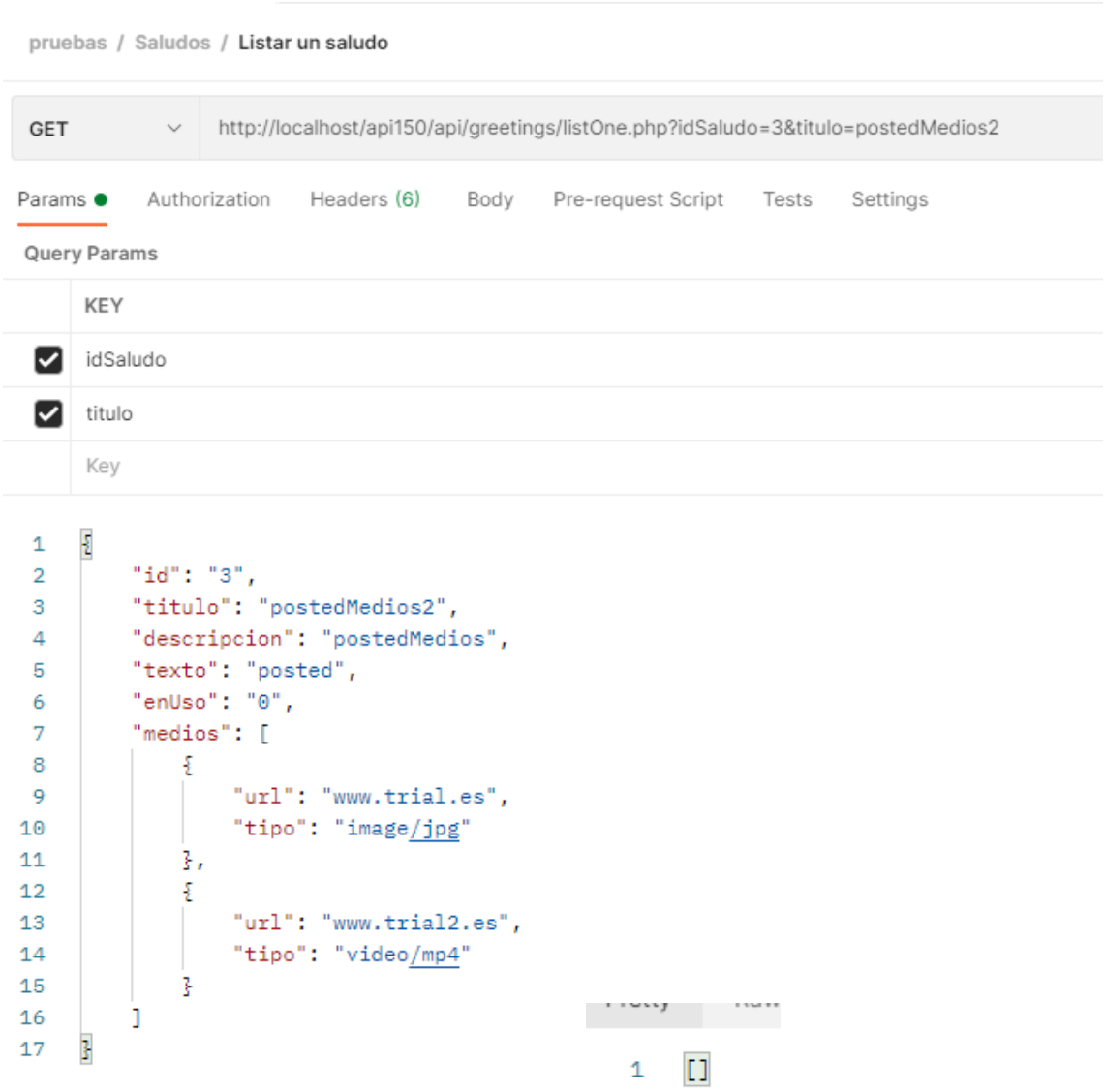
Al ser llamado, comprueba que exista, por lo menos uno de los parámetros necesarios para realizar la consulta, y devuelve el resultado.

Si no se incluye ningún parámetro, devuelve un error.

Método: GET

Parámetros: idSaludo, titulo

Respuestas: 400.



UPDATE

Recibe por el método PUT un json en el body de la petición los datos idSaludo, nuevoTitulo, nuevaDescripcion, nuevoTexto, enUso, token, medios[url, tipo].

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que existe un acto que corresponda a esa id.

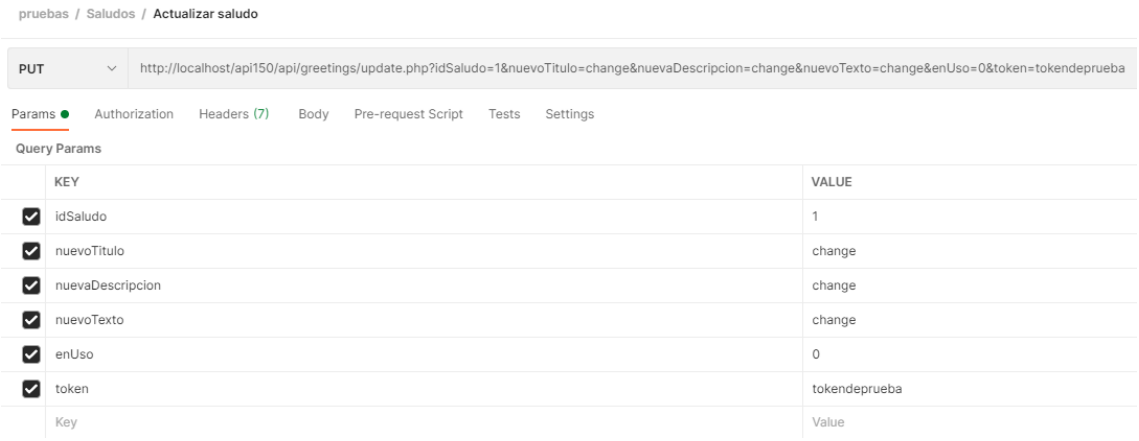
Si el acto no existe, arroja un error, en caso contrario, actualiza el acto.

Se ejecuta una actualización de los medios en la capa inferior, transparente al usuario.

Método: PUT

Parámetros: idSaludo, nuevoTitulo, nuevaDescripcion, nuevoTexto, enUso, token, medios[url, tipo].

Respuestas: 403, 401, 400, 406, 200.



HYMNS

DELETE

Recibe por el método DEL un json en el body de la petición los datos “idHimno” y “token”.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Tras esto, comprueba que la id recibida no esté vacía. Si la id está vacía arrojará un error.

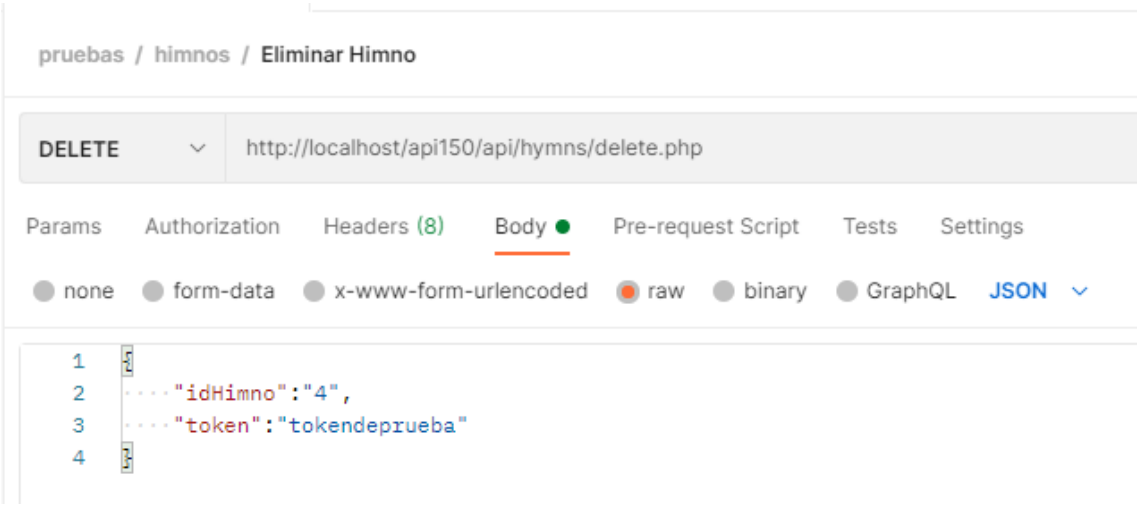
En caso contrario, procede a comprobar que, efectivamente, existe un registro asignado a esa id en la base de datos. Si esto es cierto, se elimina el elemento.

Si no existe, se arroja un error.

Método: DEL

Parámetros: idHimno, token

Respuestas: 200, 406, 400, 401, 403.



INSERT

Recibe por el método POST un json en el body de la petición los datos titulo, letra, enUso, token, medios[url, tipo].

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que no hay ningún himno que corresponda a ese titulo.

Si el himno ya existe arroja un error, si no, para a comprobar si en la petición se han incluido medios.

Si no hay medios, realiza la inserción de manera normal.

En caso contrario, comprueba que el formato de los medios es correcto, inserta los medios, inserta el himno, y crea las relaciones internas en la base de datos.

Si uno o más de los medios ya existen, no se incluirá una nueva referencia, sino que se relacionará con el medio ya existente.

Si algo falla en este proceso, se recibe el único error no común de la API.

Consiste en un error 418 (“El servidor se rehúsa a hacer café con una tetera”).

Es un error no controlado, que requiere la presencia de un administrador de la base de datos. No puede tratarse desde código.

Método: POST

Parámetros: titulo, letra, enUso, token, medios[url, tipo].

Respuestas: 200, 201, 406, 400, 401, 403, 503.

pruebas / himnos / **insertar himno**

POST http://localhost/api150/api/hymns/insert.php

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "titulo":"postedMedios6",
    "letra":"posted4",
    "enUso":"1",
    "token":"6329839f1619570ce52ccece3f3b6946c40bcead",
    "medios": [
            {
                "url": "www.trial.es",
                "tipo": "image/jpg"
            },
            {
                "url": "www.trial2.es",
                "tipo": "video/mp4"
            }
        ]
}
```

LIST

Solo necesita ser llamado para mostrar todos los datos contenidos en la tabla de himnos.

Debido a necesidades de limpieza, no muestra las relaciones entre los himnos y los medios.

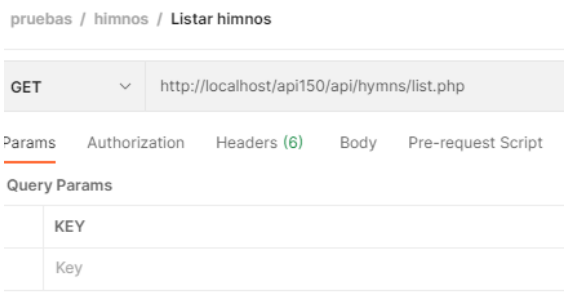
Es decir, este endpoint muestra únicamente los datos de los actos, sin entrar en los medios que contiene.

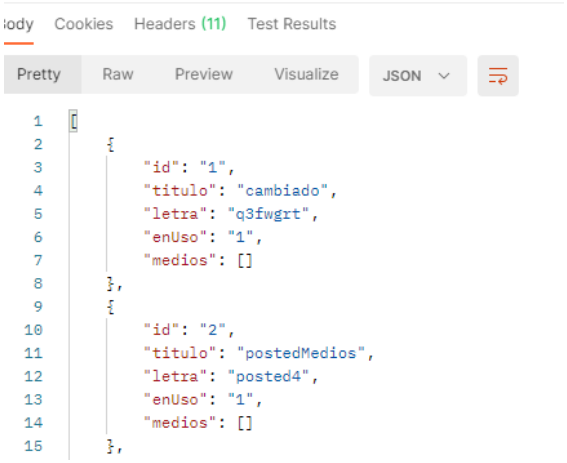
Para eso tenemos el siquiente endpoint.

Método: GET

Parámetros:

Respuestas: Lista completa o lista vacía.





LISTONE

Es el endpoint preparado para listar un himno concreto, además de todas sus relaciones.

Recibe por el método GET los parámetros idSaludo y título.

Está preparado para funcionar con solo uno de ellos, cualquiera, o con ambos.

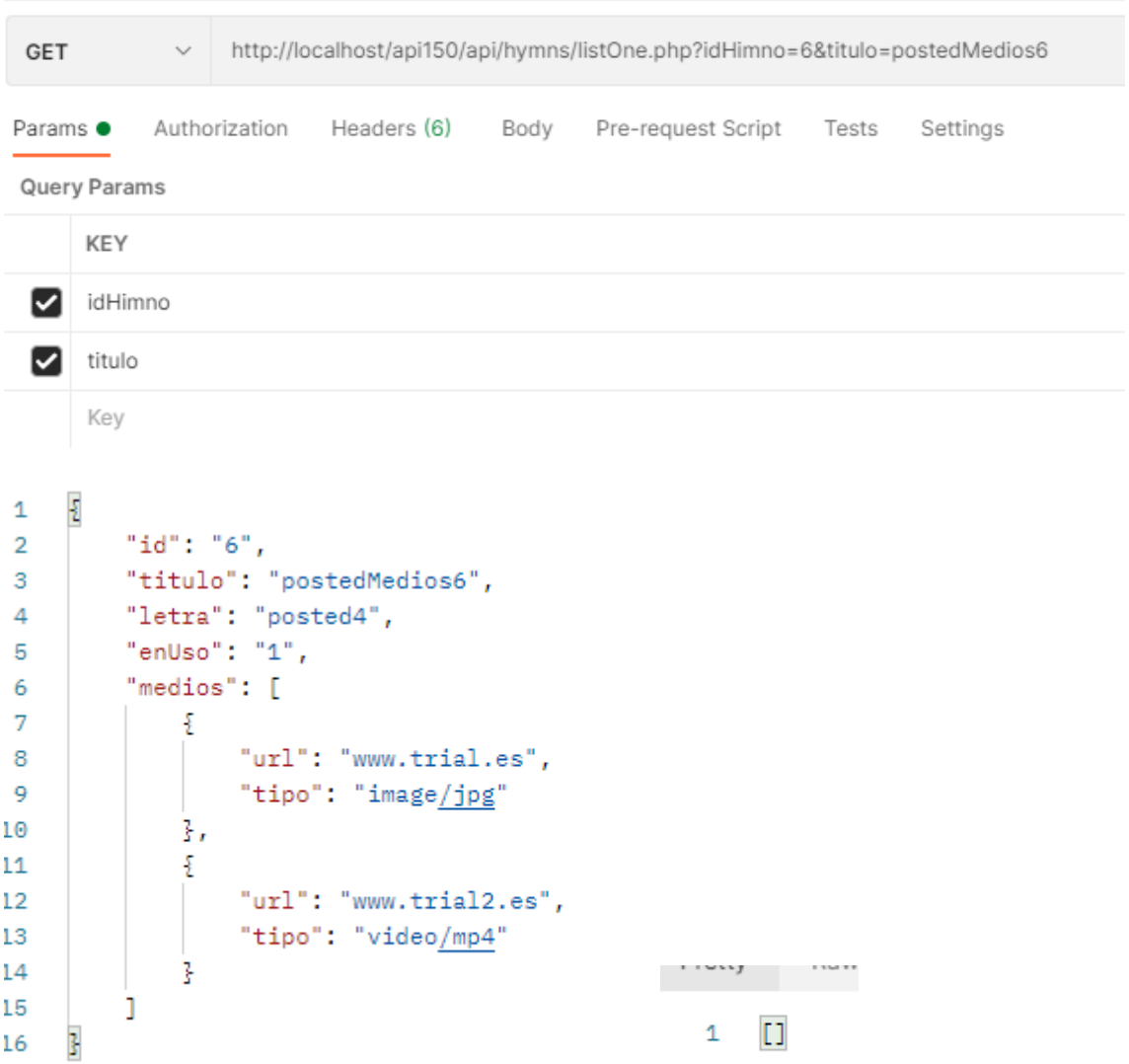
Al ser llamado, comprueba que exista, por lo menos uno de los parámetros necesarios para realizar la consulta, y devuelve el resultado.

Si no se incluye ningún parámetro, devuelve un error.

Método: GET

Parámetros: idSaludo, titulo

Respuestas: 400.



UPDATE

Recibe por el método PUT un json en el body de la petición los datos idHimno, nuevoTitulo, nuevaLetra, enUso, token, y en un json en el body de la petición medios[url, tipo].

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que existe un acto que corresponda a esa id.

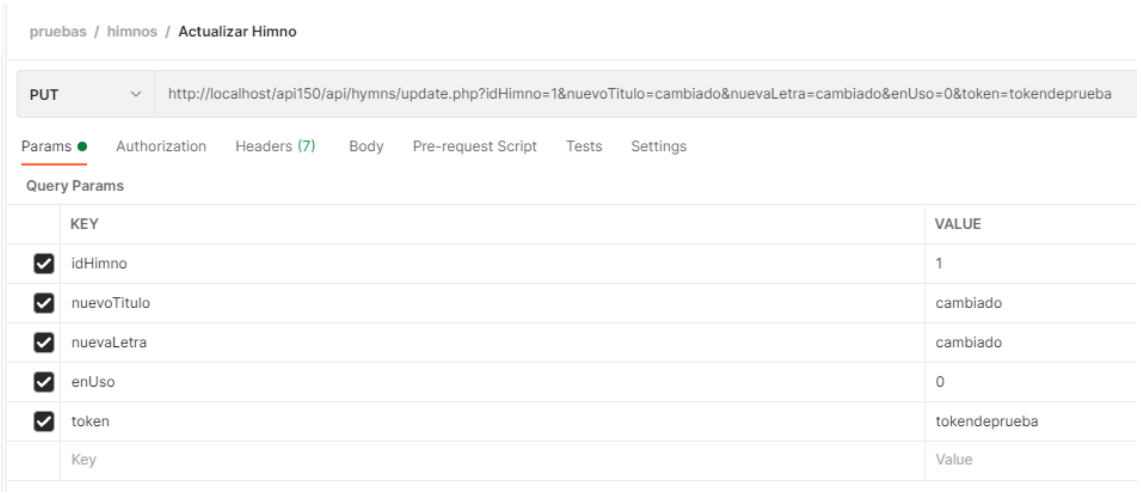
Si el acto no existe, arroja un error, en caso contrario, actualiza el acto.

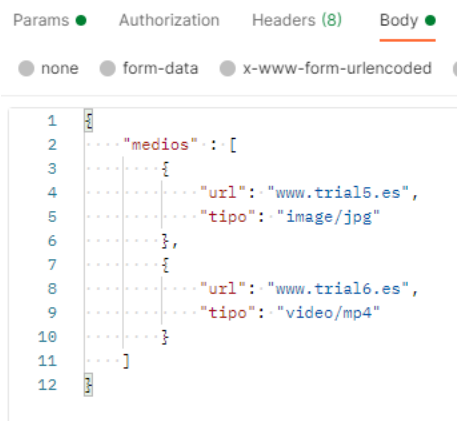
Se ejecuta una actualización de los medios en la capa inferior, transparente al usuario.

Método: PUT

Parámetros: idHimno, nuevoTitulo, nuevaLetra, enUso, token, medios[url, tipo].

Respuestas: 403, 401, 400, 406, 200.





MATERIALS

DELETE

Recibe por el método DEL un json en el body de la petición los datos “id_Medio” y “token”.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Tras esto, comprueba que la id recibida no esté vacía. Si la id está vacía arrojará un error.

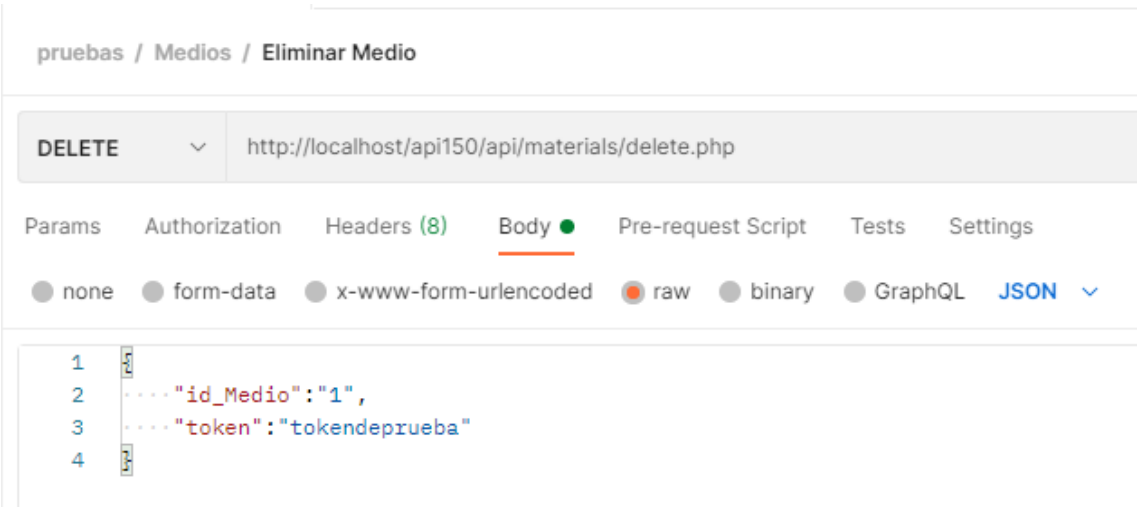
En caso contrario, procede a comprobar que, efectivamente, existe un registro asignado a esa id en la base de datos. Si esto es cierto, se elimina el elemento.

Si no existe, se arroja un error.

Método: DEL

Parámetros: id_Medio, token

Respuestas: 200, 406, 400, 401, 403.



INSERT

Recibe por el método POST un json en el body de la petición los datos token, url y tipo.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

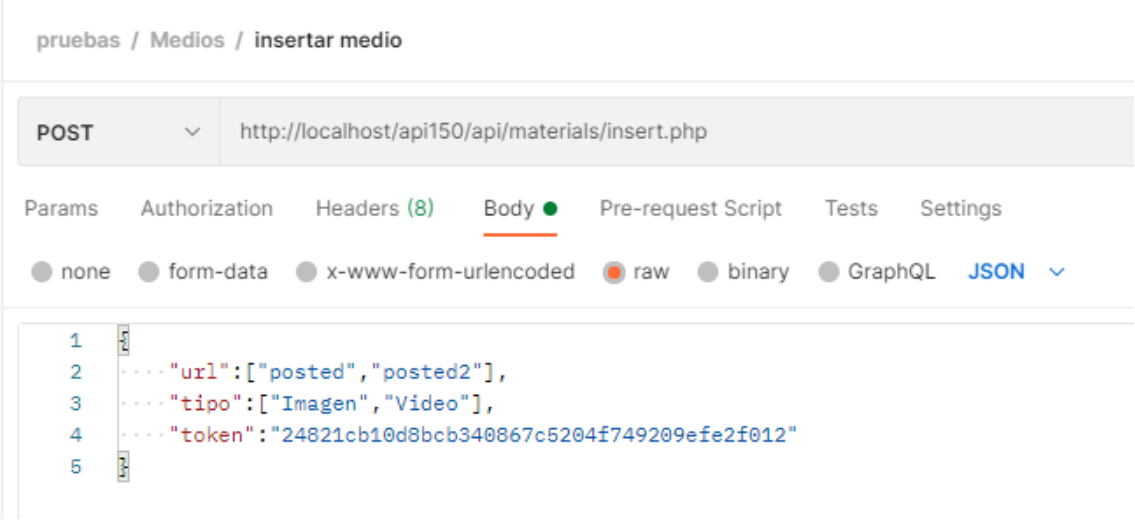
Si los datos son correctos, comprueba que no hay ningún medio que corresponda a esa URL.

Si el medio ya existe arroja un error.

Método: POST

Parámetros: token, url, tipo

Respuestas: 200, 201, 406, 400, 401, 403, 503.



LIST

Solo necesita ser llamado para mostrar todos los datos contenidos en la tabla de medios.

Método: GET

Parámetros:

Respuestas: Lista completa o lista vacía.


```
[
    {
        "id": "1",
        "url": "www.google.es/search=
        "tipo": "1"
    },
    {
        "id": "12",
        "url": "posted",
        "tipo": "1"
    },
    {
        "id": "13",
        "url": "posted2",
        "tipo": "2"
    },
```


LISTONE

Es el endpoint preparado para listar un medio concreto.

Recibe por el método GET los parámetros idMedio y URL.

Está preparado para funcionar con solo uno de ellos, cualquiera, o con ambos.

Al ser llamado, comprueba que exista, por lo menos uno de los parámetros necesarios para realizar la consulta, y devuelve el resultado.

Si no se incluye ningún parámetro, devuelve un error.

Método: GET

Parámetros: idMedio, URL

Respuestas: 400.

pruebas / Medios / **Listar un material**

GET http://localhost/api150/api/materials/listOne.php?idMedio=1&URL=www.google.es/search=algo

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	KEY
☑	idMedio
☑	URL


```
[
    {
        "id": "1",
        "url": "www.google.es/search=algo",
        "tipo": "1"
    }
]
```


1 []

UPDATE

Recibe por el método PUT los datos idMedio, nuevaURL, token.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

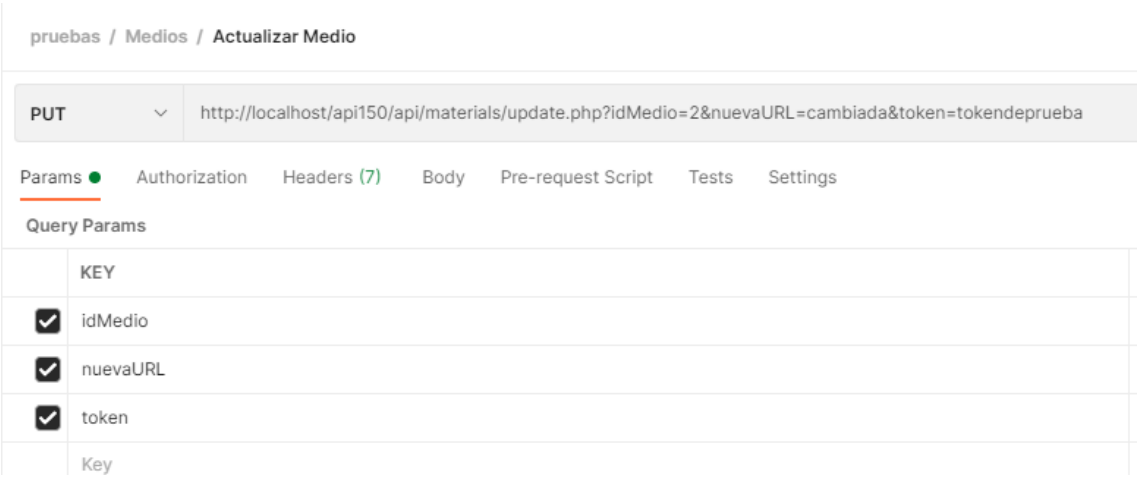
Si los datos son correctos, comprueba que existe un medio que corresponda a esa id.

Si el medio no existe, arroja un error, en caso contrario, actualiza el medio.

Método: PUT

Parámetros: idMedio, nuevaURL, token.

Respuestas: 403, 401, 400, 406, 200.



PHRASES

DELETE

Recibe por el método DEL un json en el body de la petición los datos “idfrase” y “token”.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Tras esto, comprueba que la id recibida no esté vacía. Si la id está vacía arrojará un error.

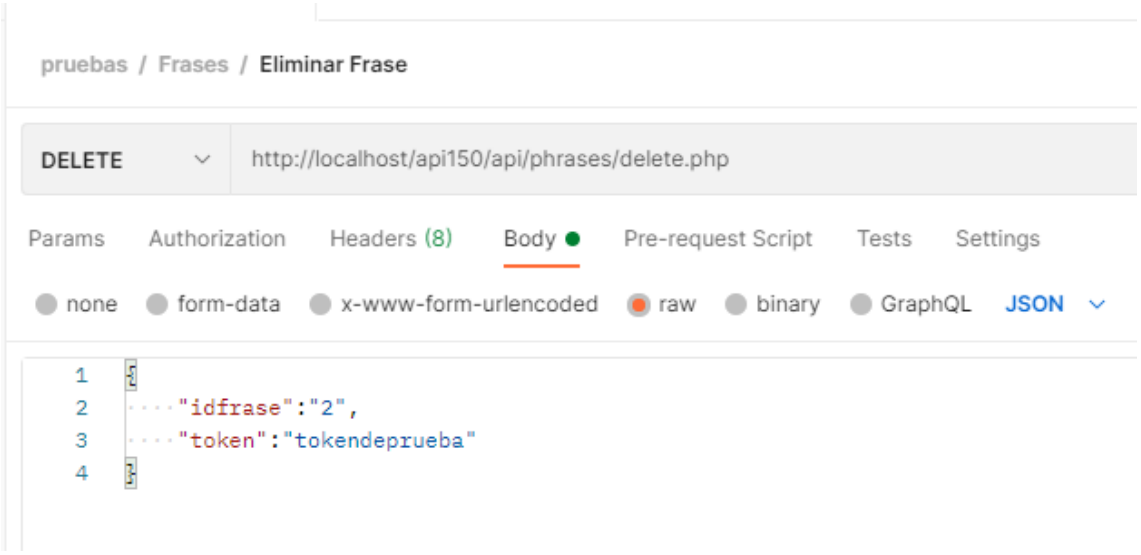
En caso contrario, procede a comprobar que, efectivamente, existe un registro asignado a esa id en la base de datos. Si esto es cierto, se elimina el elemento.

Si no existe, se arroja un error.

Método: DEL

Parámetros: idfrase, token

Respuestas: 200, 406, 400, 401, 403.



INSERT

Recibe por el método POST un json en el body de la petición los datos texto, fecha, enUso, token.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que no hay ninguna frase que corresponda a ese texto.

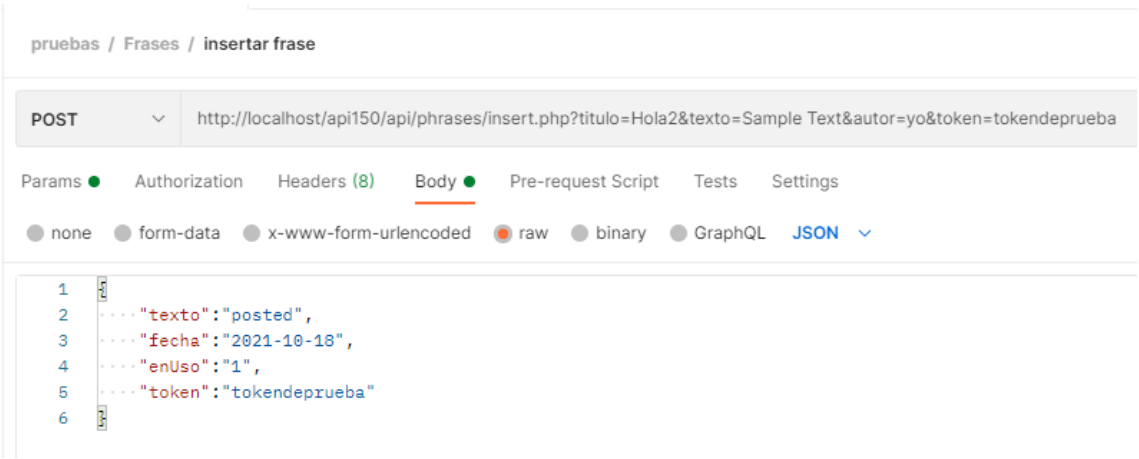
Si la frase ya existe arroja un error.

En caso contrario, inserta la frase.

Método: POST

Parámetros: texto, fecha, enUso, token.

Respuestas: 200, 201, 406, 400, 401, 403, 503.



LIST

Solo necesita ser llamado para mostrar todos los datos contenidos en la tabla de frases.

Método: GET

Parámetros:

Respuestas: Lista completa o lista vacía


```
[
    {
        "id": "1",
        "texto": "test",
        "fecha": "2021-05-19",
        "enUso": "0"
    }
]
```


.

LISTONE

Es el endpoint preparado para listar una frase en concreto.

Recibe por el método GET los parámetros idFrase y fecha.

Está preparado para funcionar con solo uno de ellos, cualquiera, o con ambos.

Al ser llamado, comprueba que exista, por lo menos uno de los parámetros necesarios para realizar la consulta, y devuelve el resultado.

Si no se incluye ningún parámetro, devuelve un error.

Método: GET

Parámetros: idFrase, fecha

Respuestas: 400.

pruebas / Frases / **Listar una frase**

GET http://localhost/api150/api/phrases/listOne.php?idFrase=1&fecha=2021-05-19

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
☑ idFrase	1
☑ fecha	2021-05-19
Key	Value


```
[
    {
        "id": "1",
        "texto": "test",
        "fecha": "2021-05-19",
        "enUso": "0"
    }
]
```


```
1 []
```

UPDATE

Recibe por el método PUT los datos idFrase, nuevoTexto, nuevaFecha, enUso, token.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

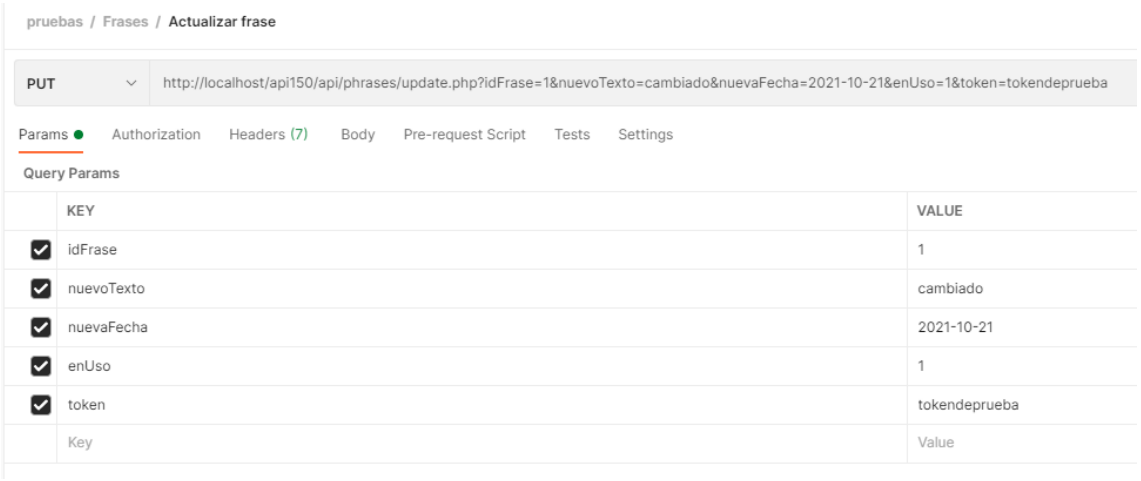
Si los datos son correctos, comprueba que existe una frase que corresponda a esa id.

Si la frase no existe, arroja un error, en caso contrario, actualiza la frase.

Método: PUT

Parámetros: idFrase, nuevoTexto, nuevaFecha, enUso, token

Respuestas: 403, 401, 400, 406, 200.



PJENVIRONMENTS

DELETE

Recibe por el método DEL un json en el body de la petición los datos “idAmbiente” y “token”.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Tras esto, comprueba que la id recibida no esté vacía. Si la id está vacía arrojará un error.

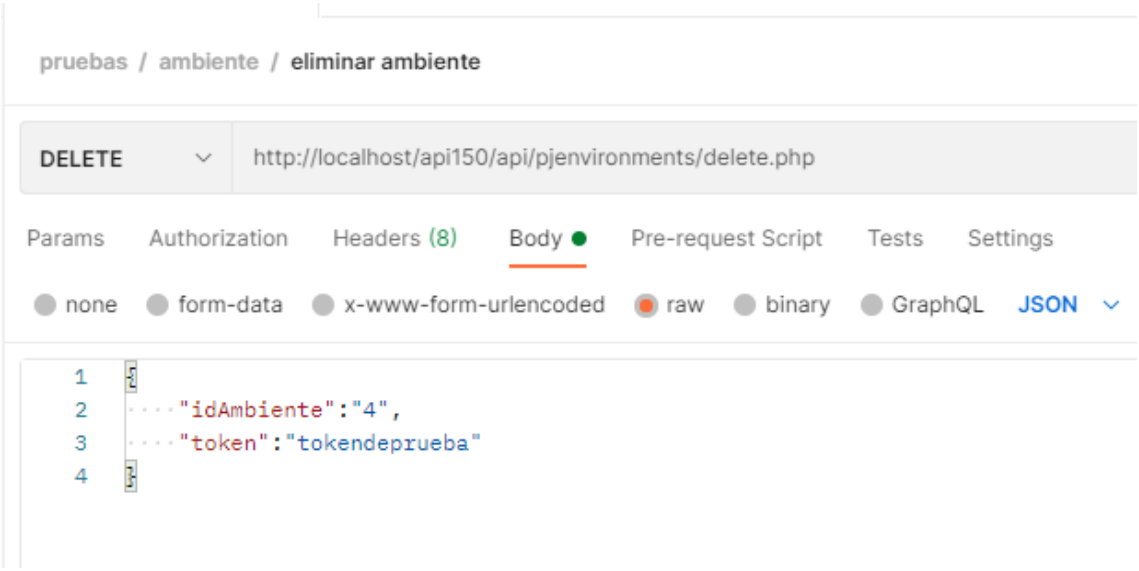
En caso contrario, procede a comprobar que, efectivamente, existe un registro asignado a esa id en la base de datos. Si esto es cierto, se elimina el elemento.

Si no existe, se arroja un error.

Método: DEL

Parámetros: idAmbiente, token

Respuestas: 200, 406, 400, 401, 403.



INSERT

Recibe por el método POST un json en el body de la petición los datos tituloAmbiente, descripcionAmbiente, enUso, token, medios[url, tipo]

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que no hay ningún ambiente que corresponda a ese titulo.

Si el ambiente ya existe arroja un error, si no, pasa a comprobar si en la petición se han incluido medios.

Si no hay medios, realiza la inserción de manera normal.

En caso contrario, comprueba que el formato de los medios es correcto, inserta los medios, inserta el himno, y crea las relaciones internas en la base de datos.

Si uno o más de los medios ya existen, no se incluirá una nueva referencia, sino que se relacionará con el medio ya existente.

Si algo falla en este proceso, se recibe el único error no común de la API.

Consiste en un error 418 (“El servidor se rehúsa a hacer café con una tetera”).

Es un error no controlado, que requiere la presencia de un administrador de la base de datos. No puede tratarse desde código.

Método: POST

Parámetros: tituloAmbiente, descripcionAmbiente, enUso, token, medios[url, tipo]

Respuestas: 200, 201, 406, 400, 401, 403, 503.

pruebas / ambiente / **insertar ambiente**

POST http://localhost/api150/api/pjenvironments/insert.php

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "tituloAmbiente":"postedMedios2",
    "descripcionAmbiente":"posted4",
    "enUso": "0",
    "token":"b2c6ec323f86981654d9753390c348599760883b",
    "medios": [
            {
                "url": "www.trial.es",
                "tipo": "image/jpg"
            },
            {
                "url": "www.trial2.es",
                "tipo": "video/mp4"
            }
        ]
}
```

LIST

Solo necesita ser llamado para mostrar todos los datos contenidos en la tabla de ambientes.

Debido a necesidades de limpieza, no muestra las relaciones entre los ambientes y los medios.

Es decir, este endpoint muestra únicamente los datos de los actos, sin entrar en los medios que contiene.

Para eso tenemos el siquiente endpoint.

Método: GET

Parámetros:

Respuestas: Lista completa o lista vacía.


```
[
    {
        "id": "3",
        "titulo": "test",
        "descripcion": "test",
        "enUso": "1",
        "medios": []
    },
    {
        "id": "4",
        "titulo": "postedMedios",
        "descripcion": "posted4",
        "enUso": "0",
        "medios": []
    },
    {
        "id": "5",
        "titulo": "postedMedios2",
        "descripcion": "posted4",
        "enUso": "0",
        "medios": []
    }
```


LISTONE

Es el endpoint preparado para listar un ambiente, además de todas sus relaciones.

Recibe por el método GET los parámetros idAmbiente y título.

Está preparado para funcionar con solo uno de ellos, cualquiera, o con ambos.

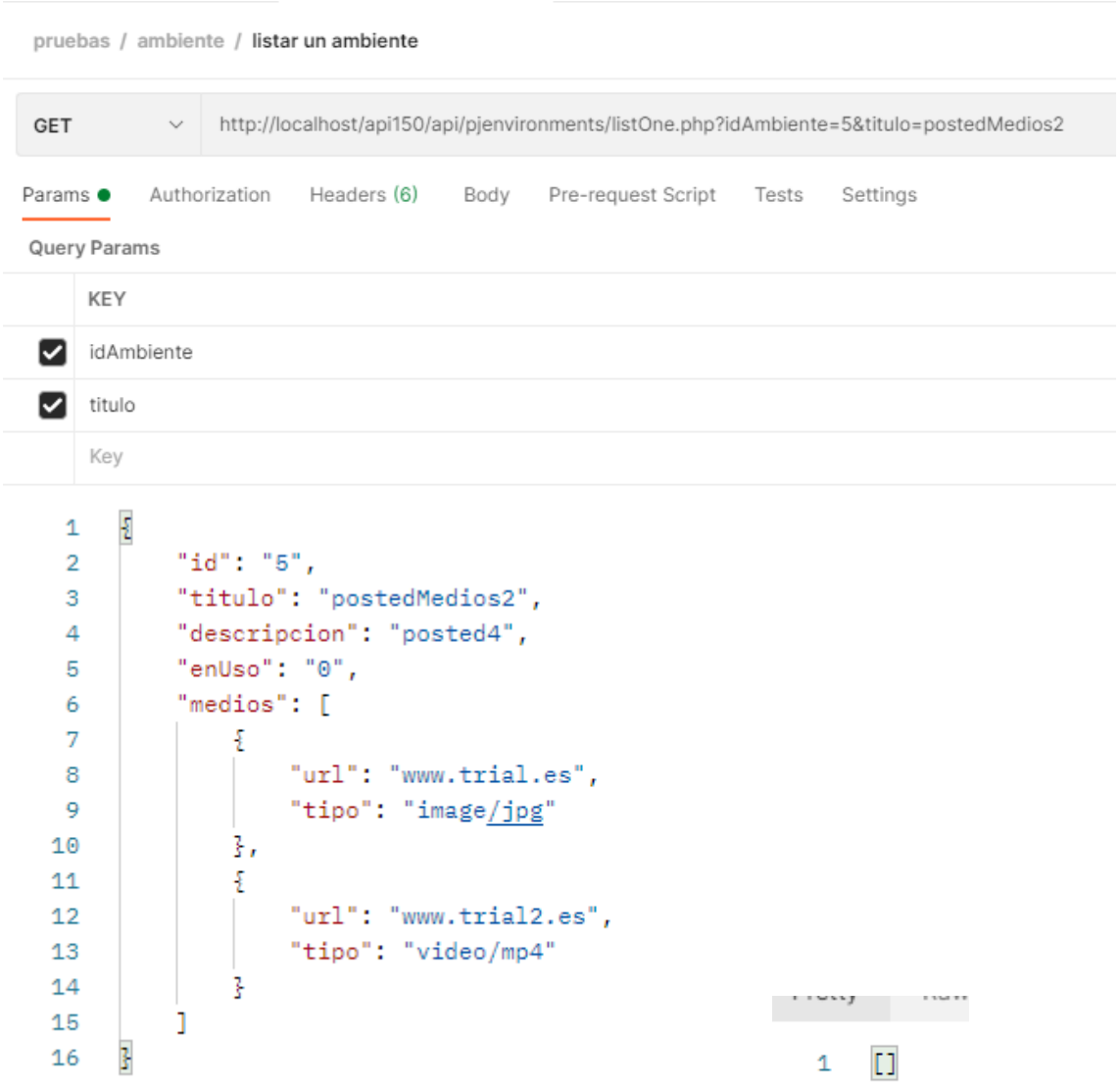
Al ser llamado, comprueba que exista, por lo menos uno de los parámetros necesarios para realizar la consulta, y devuelve el resultado.

Si no se incluye ningún parámetro, devuelve un error.

Método: GET

Parámetros: idAmbiente, titulo

Respuestas: 400.



UPDATE

Recibe por el método PUT los datos idAmbiente, nuevoTitulo, enUso, token, y en un json en el body de la petición medios[url, tipo].

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que existe un ambiente que corresponda a esa id.

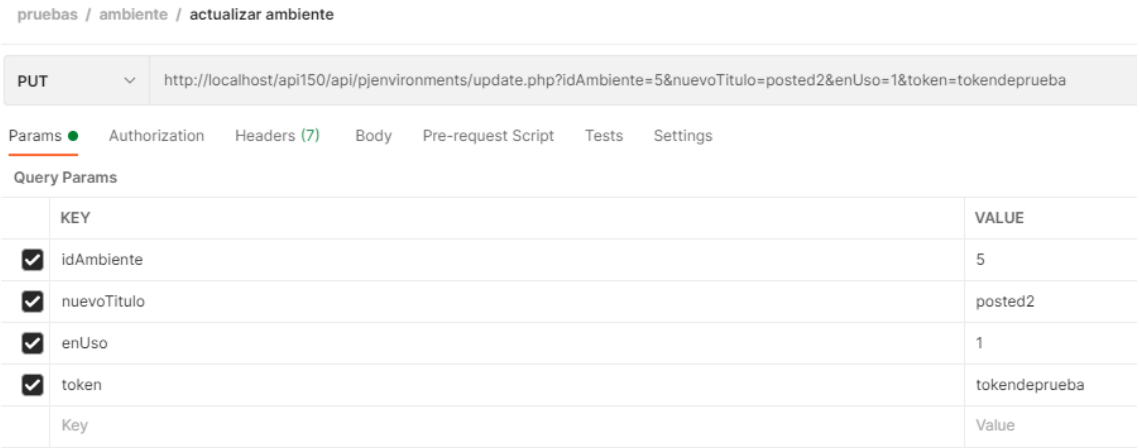
Si el ambiente no existe, arroja un error, en caso contrario, actualiza el ambiente.

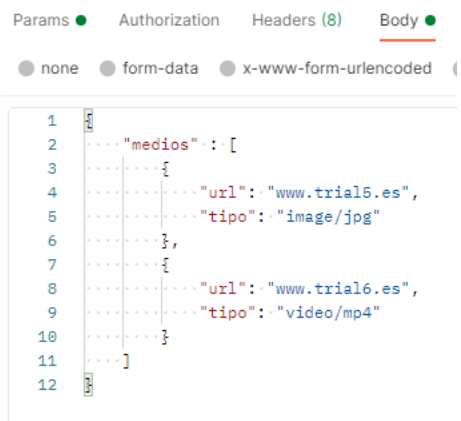
Se ejecuta una actualización de los medios en la capa inferior, transparente al usuario.

Método: PUT

Parámetros: idAmbiente, nuevoTitulo, enUso, token, medios[url, tipo].

Respuestas: 403, 401, 400, 406, 200.





PRAY

DELETE

Recibe por el método DEL un json en el body de la petición los datos “idOracion” y “token”.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Tras esto, comprueba que la id recibida no esté vacía. Si la id está vacía arrojará un error.

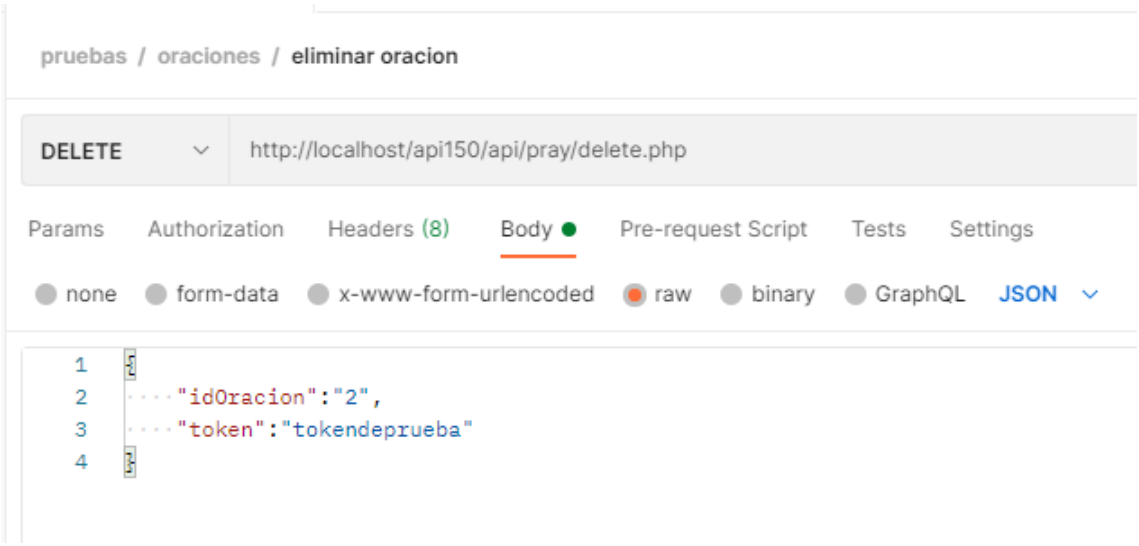
En caso contrario, procede a comprobar que, efectivamente, existe un registro asignado a esa id en la base de datos. Si esto es cierto, se elimina el elemento.

Si no existe, se arroja un error.

Método: DEL

Parámetros: idOracion, token

Respuestas: 200, 406, 400, 401, 403.



INSERT

Recibe por el método POST un json en el body de la petición los datos titulo, texto, enUso, token

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

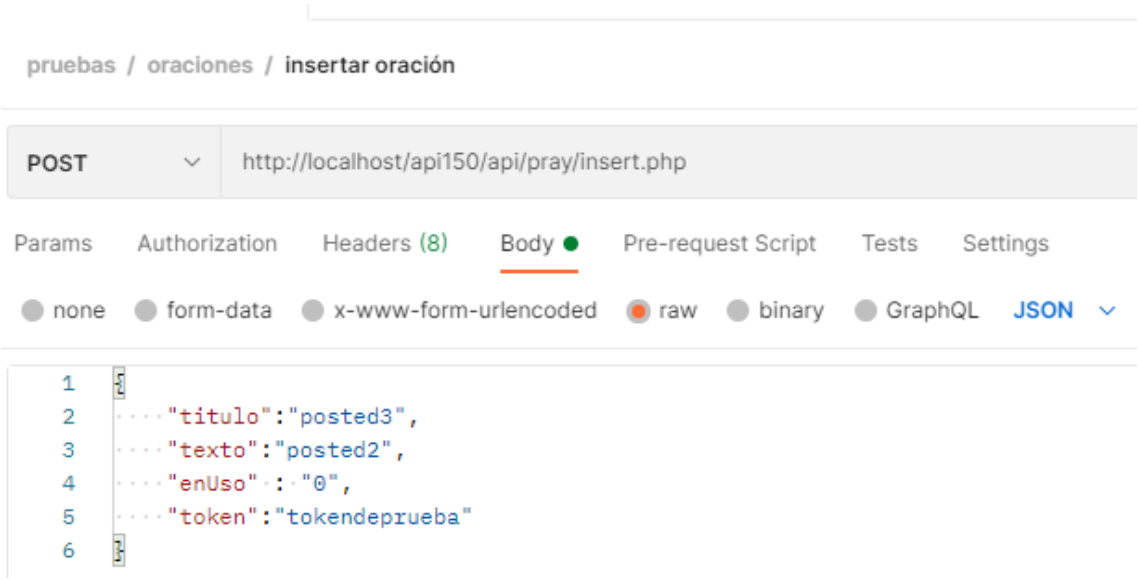
Si los datos son correctos, comprueba que no hay ninguna oración que corresponda a ese titulo.

Sila oración ya existe arroja un error, si no, la inserción se realiza de forma normal.

Método: POST

Parámetros: titulo, texto, enUso, token

Respuestas: 200, 201, 406, 400, 401, 403, 503.



LIST

Solo necesita ser llamado para mostrar todos los datos contenidos en la tabla de oraciones.

Método: GET

Parámetros:

Respuestas: Lista completa o lista vacía.

LISTONE

Es el endpoint preparado para listar una oración en concreto, además de todas sus relaciones.

Recibe por el método GET los parámetros idOracion y título.

Está preparado para funcionar con solo uno de ellos, cualquiera, o con ambos.

Al ser llamado, comprueba que exista, por lo menos uno de los parámetros necesarios para realizar la consulta, y devuelve el resultado.

Si no se incluye ningún parámetro, devuelve un error.

Método: GET

Parámetros: idOracion, titulo

Respuestas: 400.

pruebas / oraciones / **listar una oracion**

GET http://localhost/api150/api/pray/listOne.php?idOracion=1

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	KEY
☑	idOracion
☐	titulo
	Key


```
[
    {
        "id": "1",
        "titulo": "test",
        "texto": "test",
        "enUso": "1"
    }
]
```


1 []

UPDATE

Recibe por el método PUT los datos idOracion, nuevoTitulo, nuevoTexto, enUso, token.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

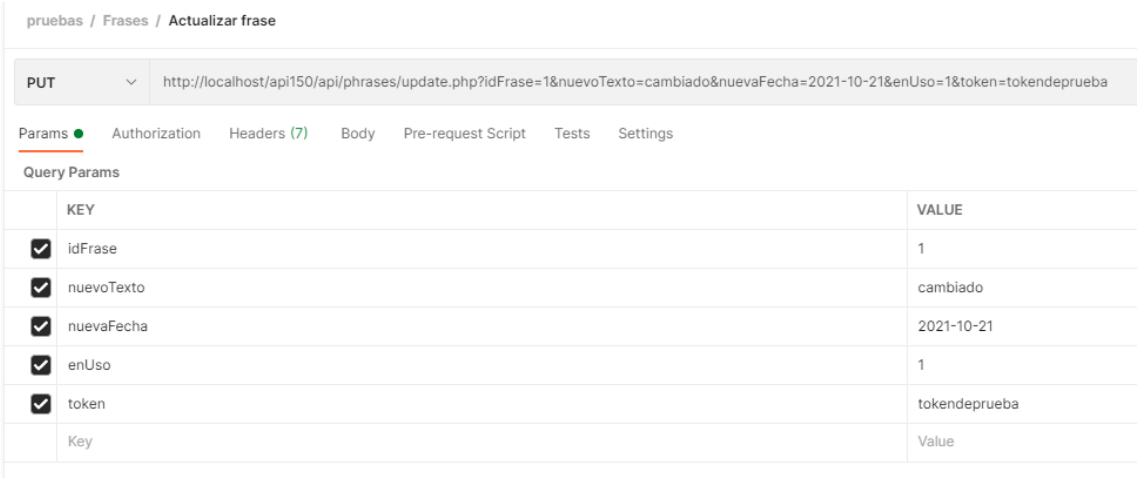
Si los datos son correctos, comprueba que existe una oración que corresponda a esa id.

Si la oración no existe, arroja un error, en caso contrario, actualiza la oración.

Método: PUT

Parámetros: idOracion, nuevoTitulo, nuevoTexto, enUso, token

Respuestas: 403, 401, 400, 406, 200.



STORY

DELETE

Recibe por el método DEL un json en el body de la petición los datos “idHistoria” y “token”.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Tras esto, comprueba que la id recibida no esté vacía. Si la id está vacía arrojará un error.

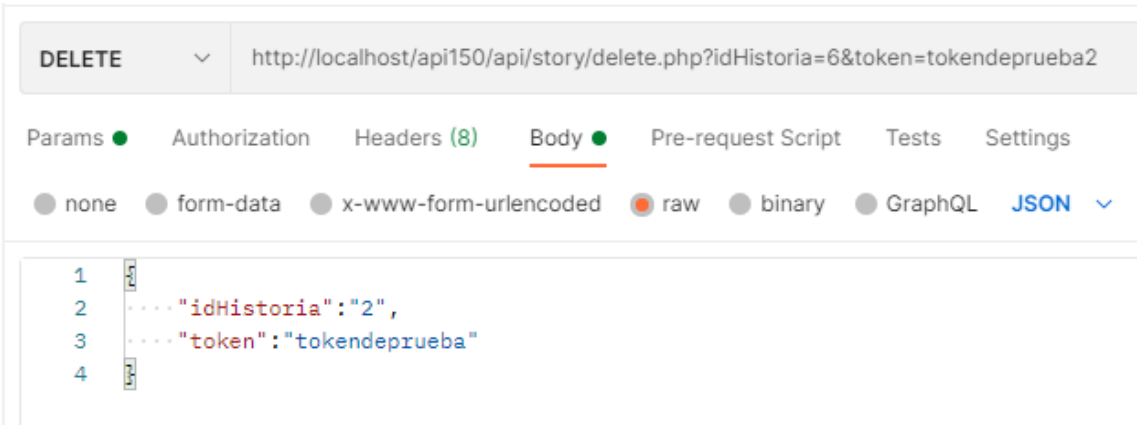
En caso contrario, procede a comprobar que, efectivamente, existe un registro asignado a esa id en la base de datos. Si esto es cierto, se elimina el elemento.

Si no existe, se arroja un error.

Método: DEL

Parámetros: idHistoria, token

Respuestas: 200, 406, 400, 401, 403.



INSERT

Recibe por el método POST un json en el body de la petición los datos tituloHistoria, subtituloHistoria, descripcion, enUso, token, medios[url, tipo]

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que no hay ninguna historia que corresponda a ese titulo.

Si la historia ya existe arroja un error, si no, pasa a comprobar si en la petición se han incluido medios.

Si no hay medios, realiza la inserción de manera normal.

En caso contrario, comprueba que el formato de los medios es correcto, inserta los medios, inserta el himno, y crea las relaciones internas en la base de datos.

Si uno o más de los medios ya existen, no se incluirá una nueva referencia, sino que se relacionará con el medio ya existente.

Si algo falla en este proceso, se recibe el único error no común de la API.

Consiste en un error 418 (“El servidor se rehúsa a hacer café con una tetera”).

Es un error no controlado, que requiere la presencia de un administrador de la base de datos. No puede tratarse desde código.

Método: POST

Parámetros: tituloHistoria, subtituloHistoria, descripcion, enUso, token, medios[url, tipo]

Respuestas: 200, 201, 406, 400, 401, 403, 503.

pruebas / historias / **insertar historia**

POST http://localhost/api150/api/story/insert.php

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
{
    "tituloHistoria":"trial3",
    "subtituloHistoria":"trial",
    "descripcion":"trial",
    "enUso" : "0",
    "token":"11361d942715992a4d4feee9cc92a71260248b26",
    "medios": [
        {
            "url": "www.trial.es",
            "tipo": "image/jpg"
        },
        {
            "url": "www.trial2.es",
            "tipo": "video/mp4"
        }
    ]
}
```

LIST

Solo necesita ser llamado para mostrar todos los datos contenidos en la tabla de historias.

Debido a necesidades de limpieza, no muestra las relaciones entre los historias y los medios.

Es decir, este endpoint muestra únicamente los datos de los actos, sin entrar en los medios que contiene.

Para eso tenemos el siquiente endpoint.

Método: GET

Parámetros:

Respuestas: Lista completa o lista vacía.


```
[
    {
        "idHistoria": "1",
        "titulo": "cambiado2",
        "subtitulo": "cambiado2",
        "descripcion": "cambiado2",
        "enUso": "0",
        "medios": []
    },
    {
        "idHistoria": "2",
        "titulo": "posted5",
        "subtitulo": "posted5",
        "descripcion": "posted5",
        "enUso": "0",
        "medios": []
    },
```


LISTONE

Es el endpoint preparado para listar una historia en concreto, además de todas sus relaciones.

Recibe por el método GET los parámetros idHistoria y título.

Está preparado para funcionar con solo uno de ellos, cualquiera, o con ambos.

Al ser llamado, comprueba que exista, por lo menos uno de los parámetros necesarios para realizar la consulta, y devuelve el resultado.

Si no se incluye ningún parámetro, devuelve un error.

Método: GET

Parámetros: idHistoria, titulo

Respuestas: 400.

pruebas / historias / **listar una historia**

GET | http://localhost/api150/api/story/listOne.php?idHistoria=1&titulo=trial2

Params ● | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings

Query Params

KEY	
☑	idHistoria
☑	titulo
	Key


```
{
    "idHistoria": "1",
    "titulo": "cambiado2",
    "subtitulo": "cambiado2",
    "descripcion": "cambiado2",
    "enUso": "0",
    "medios": [
        {
            "url": "www.trial5.es",
            "tipo": "image/jpg"
        },
        {
            "url": "www.trial6.es",
            "tipo": "video/mp4"
        }
    ]
}
```


1 []

UPDATE

Recibe por el método PUT los datos idHistoria, nuevoTitulo, nuevoSubtitulo, nuevaDescripcion, enUso, token, y en un json en el body de la petición medios[url, tipo].

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que existe una historia que corresponda a esa id.

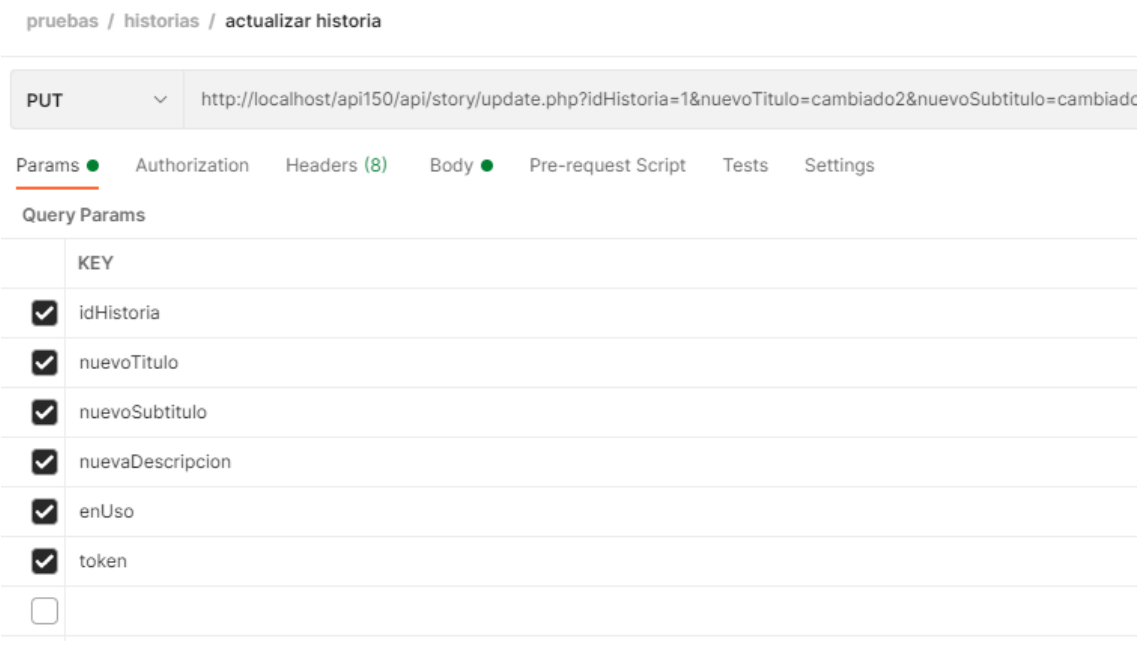
Si la historia no existe, arroja un error, en caso contrario, actualiza la historia.

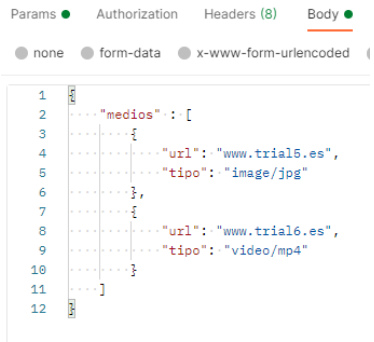
Se ejecuta una actualización de los medios en la capa inferior, transparente al usuario.

Método: PUT

Parámetros: idHistoria, nuevoTitulo, nuevoSubtitulo, nuevaDescripcion, enUso, token, medios[url, tipo].

Respuestas: 403, 401, 400, 406, 200.





VISITS

DELETE

Recibe por el método DEL un json en el body de la petición los datos “idVisita” y “token”.

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Tras esto, comprueba que la id recibida no esté vacía. Si la id está vacía arrojará un error.

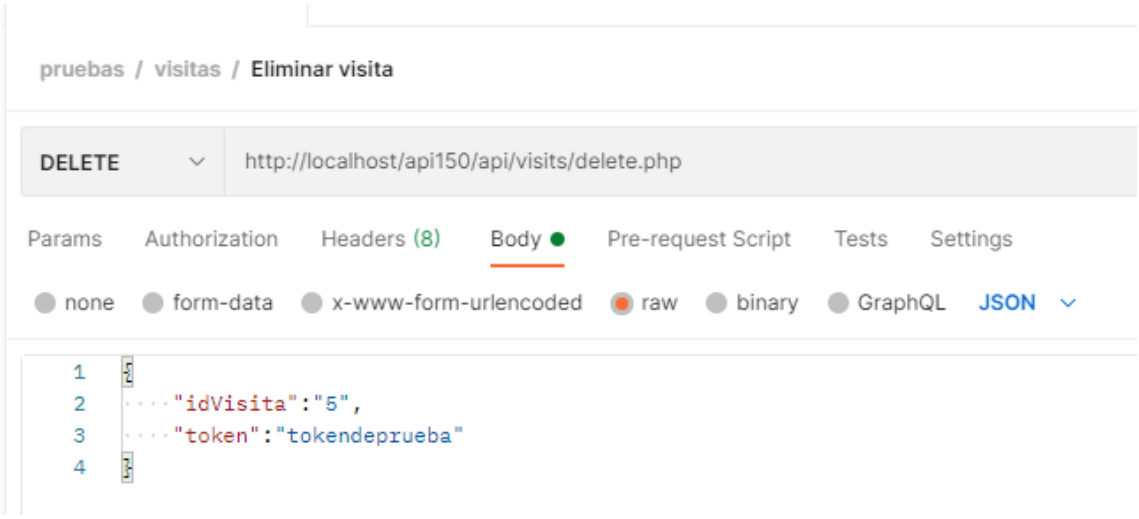
En caso contrario, procede a comprobar que, efectivamente, existe un registro asignado a esa id en la base de datos. Si esto es cierto, se elimina el elemento.

Si no existe, se arroja un error.

Método: DEL

Parámetros: idVisita, token

Respuestas: 200, 406, 400, 401, 403.



INSERT

Recibe por el método POST un json en el body de la petición los datos tituloVisita , token medios[url, tipo].

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que no hay ninguna visita que corresponda a ese titulo.

Si la visita ya existe arroja un error, si no, pasa a comprobar si en la petición se han incluido medios.

Si no hay medios, realiza la inserción de manera normal.

En caso contrario, comprueba que el formato de los medios es correcto, inserta los medios, inserta el himno, y crea las relaciones internas en la base de datos.

Si uno o más de los medios ya existen, no se incluirá una nueva referencia, sino que se relacionará con el medio ya existente.

Si algo falla en este proceso, se recibe el único error no común de la API.

Consiste en un error 418 (“El servidor se rehúsa a hacer café con una tetera”).

Es un error no controlado, que requiere la presencia de un administrador de la base de datos. No puede tratarse desde código.

Método: POST

Parámetros: tituloHistoria, subtituloHistoria, descripcion, enUso, token, medios[url, tipo]

Respuestas: 200, 201, 406, 400, 401, 403, 503.

POST http://localhost/api150/api/visits/insert.php

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "tituloVisita":"postedMedios2",
    "token":"32c86b19032e566a9313b724b58b9ca20b678fa8",
        "medios": [
            {
                "url": "www.trial.es",
                "tipo": "image/jpg"
            },
            {
                "url": "www.trial2.es",
                "tipo": "video/mp4"
            }
        ]
}
```

LIST

Solo necesita ser llamado para mostrar todos los datos contenidos en la tabla de visitas.

Debido a necesidades de limpieza, no muestra las relaciones entre las visitas y los medios.

Es decir, este endpoint muestra únicamente los datos de los actos, sin entrar en los medios que contiene.

Para eso tenemos el siquiente endpoint.

Método: GET

Parámetros:

Respuestas: Lista completa o lista vacía.


```
[
    {
        "id": "1",
        "titulo": "posted",
        "medios": []
    },
    {
        "id": "2",
        "titulo": "postedMedios",
        "medios": []
    },
    {
        "id": "3",
        "titulo": "postedMedios2",
        "medios": []
    }
]
```


LISTONE

Es el endpoint preparado para listar una visita en concreto, además de todas sus relaciones.

Recibe por el método GET los parámetros idVisita y título.

Está preparado para funcionar con solo uno de ellos, cualquiera, o con ambos.

Al ser llamado, comprueba que exista, por lo menos uno de los parámetros necesarios para realizar la consulta, y devuelve el resultado.

Si no se incluye ningún parámetro, devuelve un error.

Método: GET

Parámetros: idVisita, titulo

Respuestas: 400.

pruebas / visitas / **listar una visita**

GET | http://localhost/api150/api/visits/listOne.php?idVisita=3&titulo=postedMedios2

Params ● | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings

Query Params

	KEY
☑	idVisita
☑	titulo


```
{
    "id": "3",
    "titulo": "postedMedios2",
    "medios": [
        {
            "url": "www.trial.es",
            "tipo": "image/jpg"
        },
        {
            "url": "www.trial2.es",
            "tipo": "video/mp4"
        }
    ]
}
```


1 []

UPDATE

Recibe por el método PUT los datos idVisita, nuevoTitulo, token, y en un json en el body de la petición, medios[url, tipo].

Comprueba que el token recibido tiene permisos de administrador, está correctamente logueado y es un token válido.

A continuación, comprueba que la sesión no está expirada.

Una vez comprobado esto, comprueba que los datos recibidos son válidos, y que no hay ninguno vacío.

Si los datos son correctos, comprueba que existe una visita que corresponda a esa id.

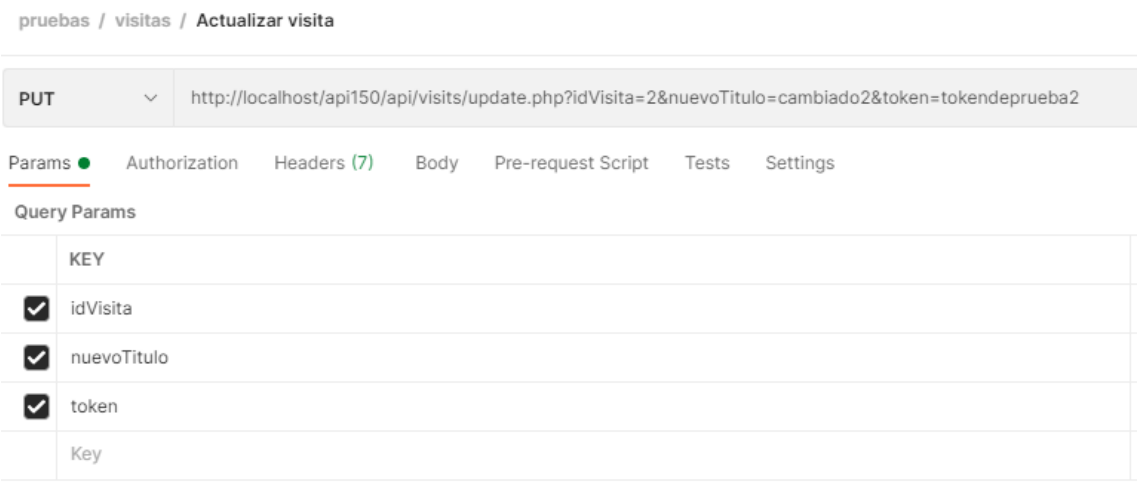
Si la visita no existe, arroja un error, en caso contrario, actualiza la visita.

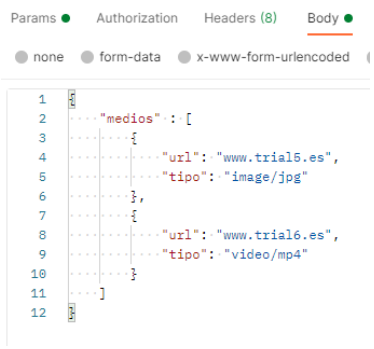
Se ejecuta una actualización de los medios en la capa inferior, transparente al usuario.

Método: PUT

Parámetros: idVisita, nuevoTitulo, token, medios[url, tipo].

Respuestas: 403, 401, 400, 406, 200.





REPOSITORIO DE LA API

https://github.com/TecnoLettuce/api150v.02