DOCUMENTACION LEC API

Guillermo López Sanz

INDICE

[1. INTRODUCCIÓN 2](#_Toc129554848)

[2. EXPLICACIÓN DE MÉTODOS 3](#_Toc129554849)

[2.1 INICIO 3](#_Toc129554850)

[2.2 MÉTODOS GET 4](#_Toc129554851)

[2.3 MÉTODOS POST 11](#_Toc129554852)

[2.4 PUT 13](#_Toc129554853)

[2.5 DELETE 14](#_Toc129554854)

[3. ERRORES 15](#_Toc129554855)

# INTRODUCCIÓN

**1.1 ¿Para qué es la API?**

La idea para la API es poder gestionar equipos, resultados, y jugadores de la LEC (League of Legends European Championship) en fase regular para la temporada 2023. En ella podremos acceder y modificar la base de datos incluida en el archivo (con sqlite) para poder actualizar los campos necesarios.

Es una API desarrollada con Python (usando flask, requests y sqlite).

**1.2 ¿Qué se puede hacer con la API?**

Hay muchas opciones desarrolladas para gestionar los datos de la LEC con la API, las cuales voy a enu merar a continuación y más tarde explicaré en detalle. Son las siguientes:

* Iniciar la BBDD
* GET equipos.
* GET equipo.
* GET jugadores.
* GET posiciones.
* GET resultados.
* INSERT jugador.
* DELETE jugador.
* UPDATE jugador.
* UPDATE equipo.
* INSERT resultado.
* Vista clasificación
* Vista equipo.
* Reset BBDD.

**1.3 Introducción a la LEC.**

La LEC es la liga europea del videojuego League of Legends, una de las más prestigiosas a nivel mundial.

El League of Legends (más conocido como LoL) es un videojuego en el que, sin entrar en mucho detalle, se enfrentan dos equipos de cinco jugadores en el que cada jugador tiene una posición. Estas posiciones son TOP, JUNGLE, MID, ADC, SUPPORT (añadidas en la tabla de posiciones, la cual no se puede modificar salvo que se edite a mano la bbdd, ya que no varían en el juego).

En fase regular, los equipos juegan un partido a la semana entre ellos, sumando una victoria si ganan, y una derrota si pierden. Al final de temporada, el equipo con más victorias y menos derrotas es el ganador.

# EXPLICACIÓN DE MÉTODOS

## 2.1 INICIO

Hay un método implementado para crear la bbdd. Lo primero es crear el entorno virtual en nuestra carpeta. Para ello hay que ejecutar el siguiente comando dentro de la carpeta donde vayamos a tener el archivo app.py:

*python3 -m venv env*

Después activaremos el entorno virtual con el siguiente comando:

*call env/Scripts/activate*

En la carpeta donde se encuentre el archivo.py debe haber un archivo txt requirements.txt en el que introduciremos las siguientes librerías para instalarlas en el proyecto: Flask, requests y flask-cors. Instalamos mediante el siguiente comando:

*pip install -r requirements.txt*

Una vez instaladas, ejecutamos la aplicación con el siguiente comando:

*flask –app app.py –debug run*

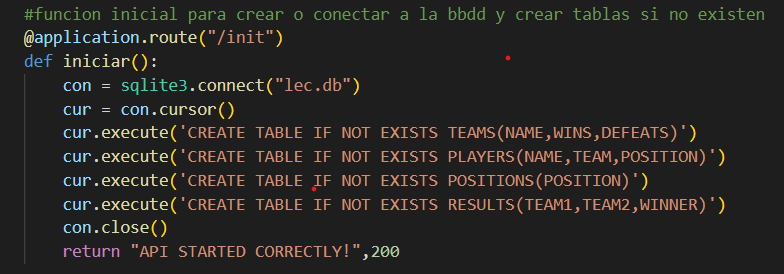
Este comando activará el servidor y te dará el endpoint, el cual necesitarás para escribir cada ruta para ejecutar los métodos.

**2.1.1 INIT (NO ES NECESARIA, YA ADJUNTO BBDD CON DATOS CON LA API)**

Init es un método que si no existe la bbdd lec.db te la crea y crea las tablas si no existen.

Se ejecuta con la siguiente ruta (en mi caso mi endpoint es http://127.0.0.1:5000):

*http://127.0.0.1:5000/init*



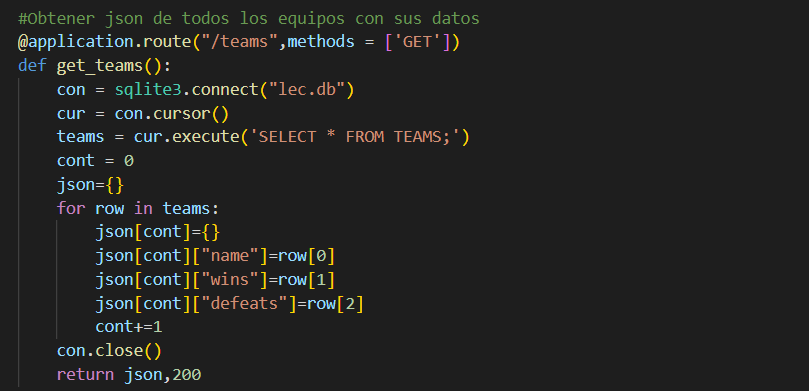
## 2.2 MÉTODOS GET

El método GET se utiliza como petición por parte del front recursos al back. Los métodos GET implementados en la API son los siguientes:

**2.2.1 GET TEAMS.**

Este método devuelve un json con todos los equipos que hay en la BBDD. El método se ejecuta con la siguiente ruta:

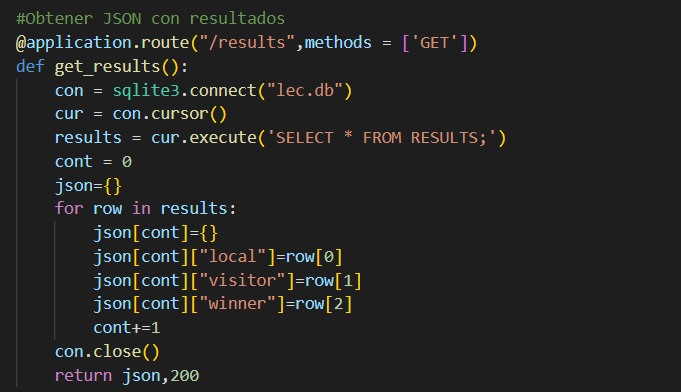
*http://127.0.0.1:5000/teams*



**2.2.2 GET RESULTS**

Este método devuelve un json con todos los resultados que hay en la BBDD. El método se ejecuta con la siguiente ruta:

*http://127.0.0.1:5000/results*



**2.2.3 GET PLAYERS**

Este método devuelve un json con todos los jugadores que hay en la BBDD. El método se ejecuta con la siguiente ruta:

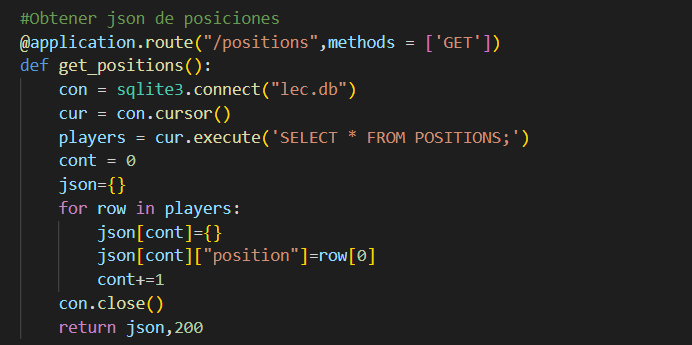
*http://127.0.0.1:5000/players*

**

**2.2.4 GET POSITIONS**

Este método devuelve un json con todas las posiciones que hay en la BBDD. El método se ejecuta con la siguiente ruta:

*http://127.0.0.1:5000/positions*

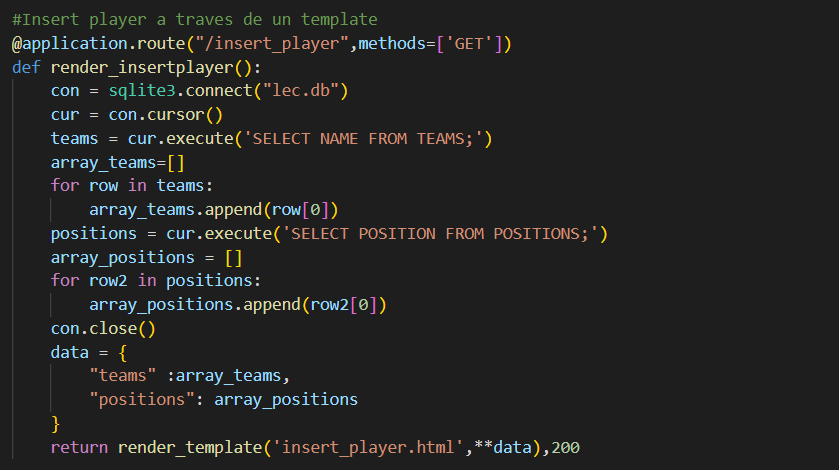


**2.2.5 CARGAR TEMPLATE INSERT PLAYER**

El método insert player es un método POST, pero ese POST se envía desde un template que se carga cuando llamamos a la siguiente ruta:

*http://127.0.0.1:5000/insert\_player*

Esta ruta va a cargar obtener los equipos y posiciones que existen en la bbdd para cargar el template con estos datos, evitando así que el usuario pueda introducir datos de equipos o posiciones que no están en la bbdd.



El template generado se ve de la siguiente manera:



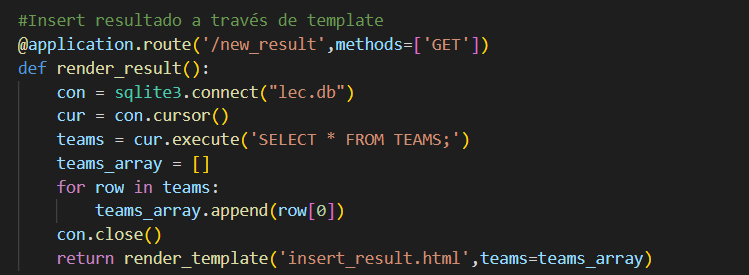
Donde en –SELECT TEAM—están todos los equipos y en –SELECT POSITION—las posiciones.

Al pulsar el botón INSERT, se ejecutará la misma ruta pero esta vez con el método POST.

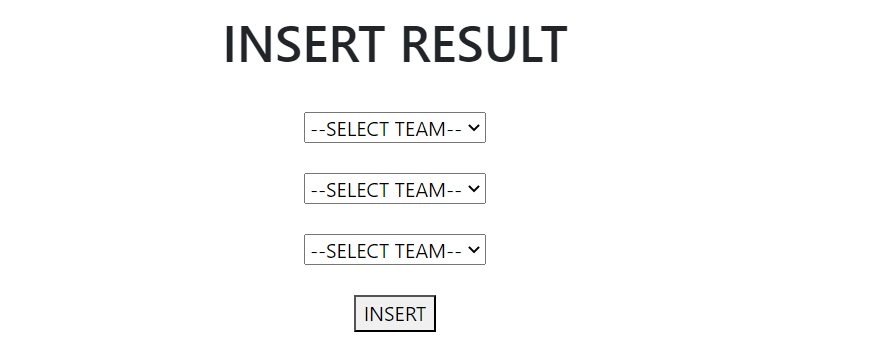
**2.2.6 CARGAR TEMPLATE DE INSERT RESULT**

De la misma manera que el método anterior uso este método. Cuando llamas a la siguiente ruta con el método GET se llama al template al cual se le pasan los equipos de la bbdd para poder generar resultados de la forma team1, team2, winner:

*http://127.0.0.1:5000/new\_result*



Se ve de la siguiente manera:



En cada SELECT aparecen todos los equipos que hay en la BBDD.

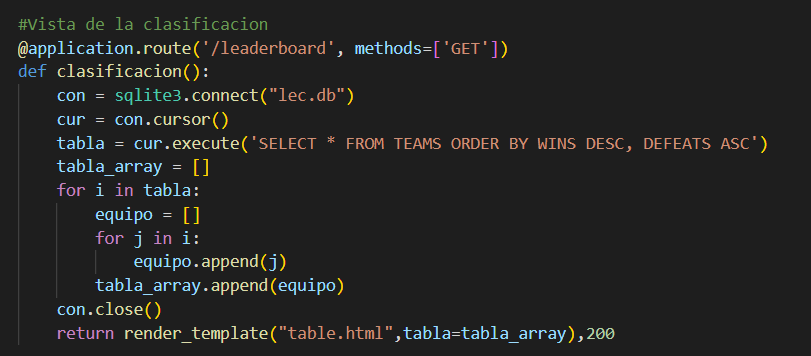
Al pulsar el botón INSERT se ejecuta la misma ruta pero cambiando el método a POST.

**2.2.7 VISTA DE CLASIFICACIÓN:**

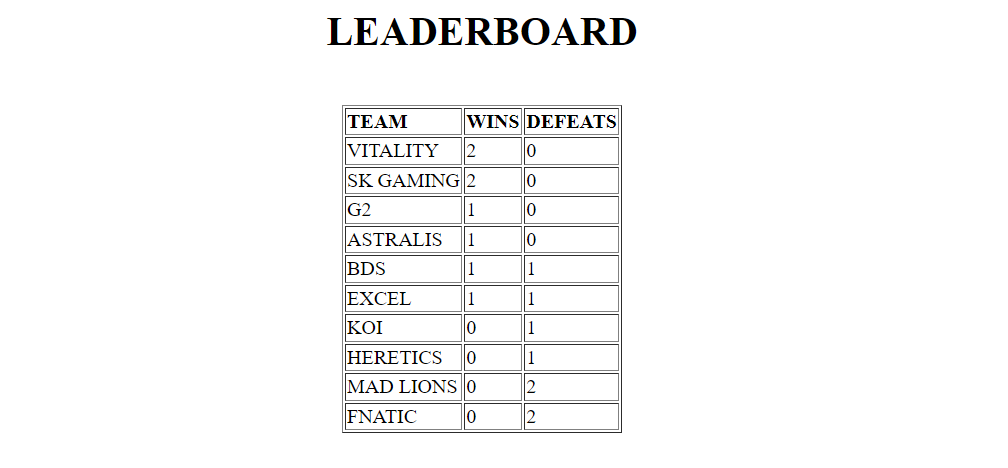
Método que muestra una tabla con la clasificación de la LEC en base a los resultados registrados en la BBDD. Es un template al que se le pasan los datos de cada equipo y se genera una tabla con estos datos.

Se ejecuta con la siguiente ruta:

*http://127.0.0.1:5000/leaderboard*



La vista de la clasificación es la siguiente:



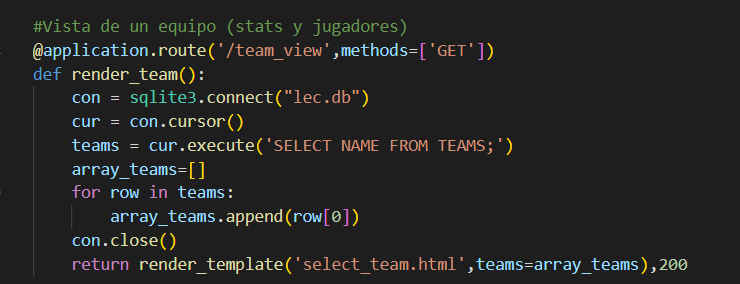
Los equipos están ordenados por las victorias de manera descendente y derrotas de manera ascendente.

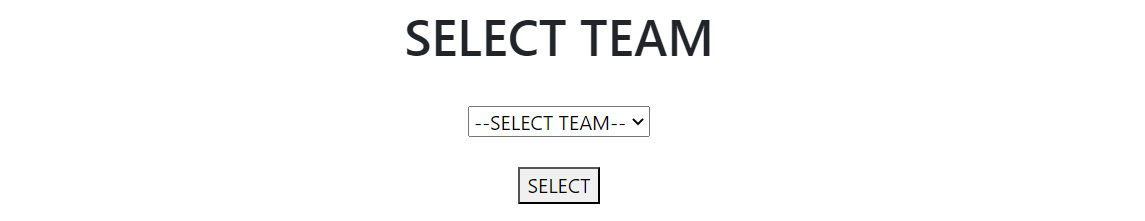
**2.2.8 TEAM VIEW**

Este método es un para cargar un template con una tabla con las stats de un equipo y otra tabla con sus jugadores. El equipo se elige a través de un template que tiene un select con los equipos que hay en la bbdd.

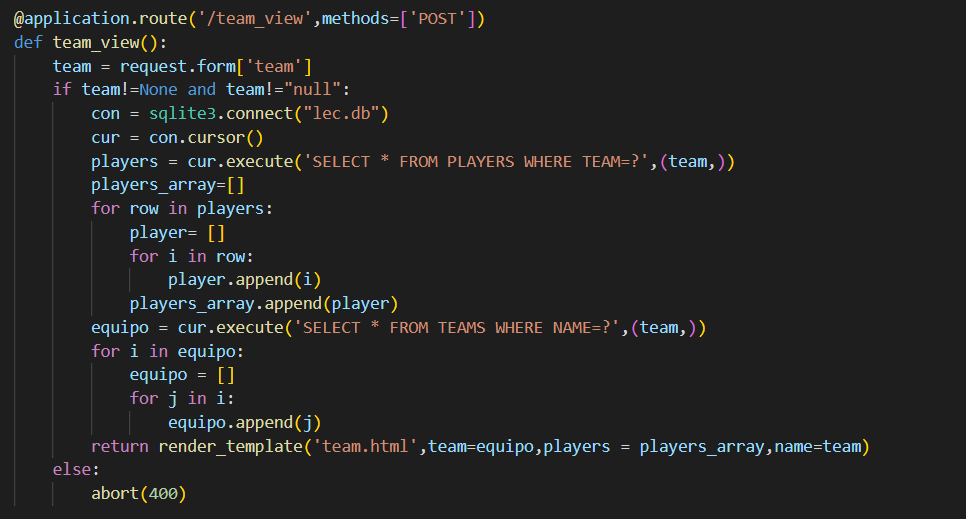
Ruta: *http://127.0.0.1:5000/team\_view*

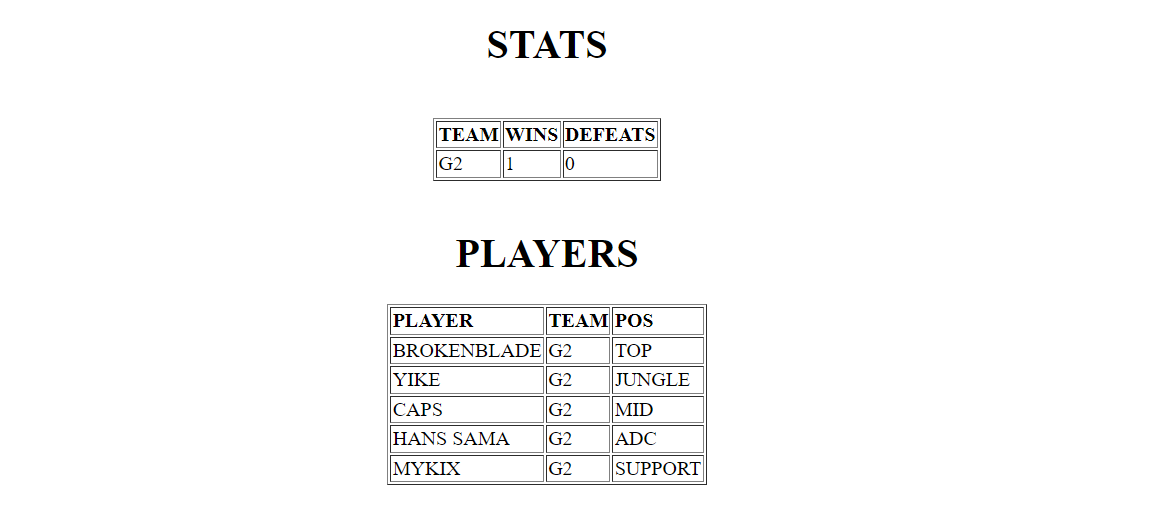
Al igual que los anteriores métodos, al pulsar el botón SELECT se vuelve a llamar la ruta en método POST y se muestra el template con la tabla.





Aunque el team\_view al pulsar el SELECT es un POST, voy a mostrar el resultado y el método en este apartado ya que es un POST sencillo de un equipo solo y al final el resultado es un GET de los datos de un equipo, pero que al necesitar pasarse el equipo como parámetro, es un método POST.





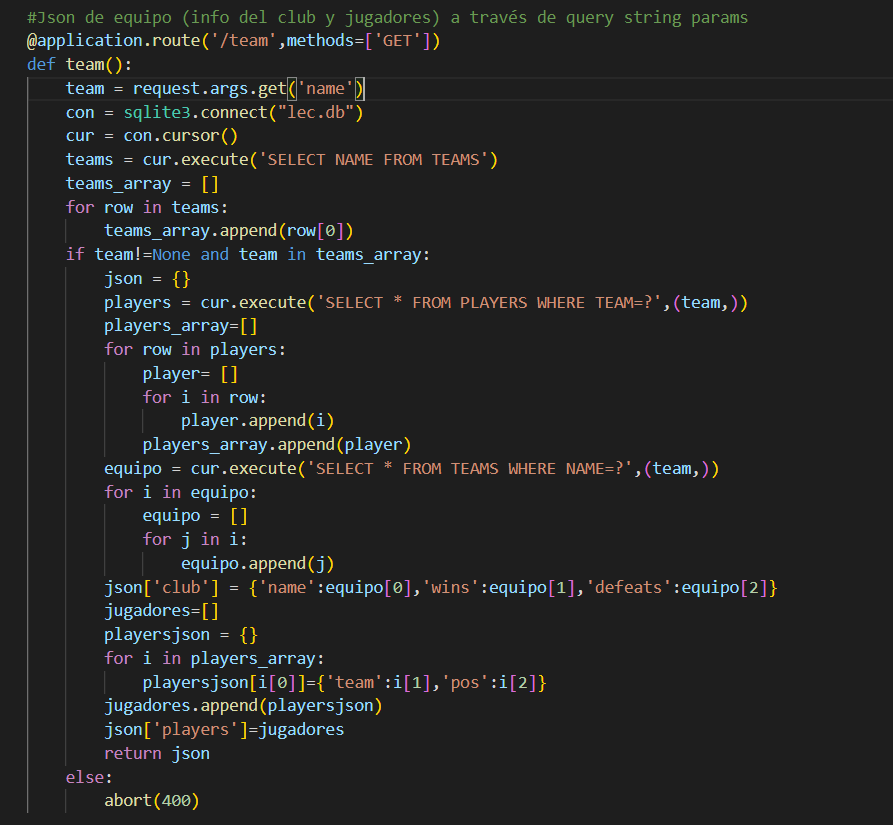
Ejemplo para el equipo G2.

**2.2.9 GET TEAM**

Método para a través de query string param “name” devuelve un json con los datos de un equipo si este existe en la bbdd. Un ejemplo de ejecución sería:

Ruta: *http://127.0.0.1:5000/team?name=G2*

Este método devolvería un json con todos los datos del equipo G2.



## 2.3 MÉTODOS POST

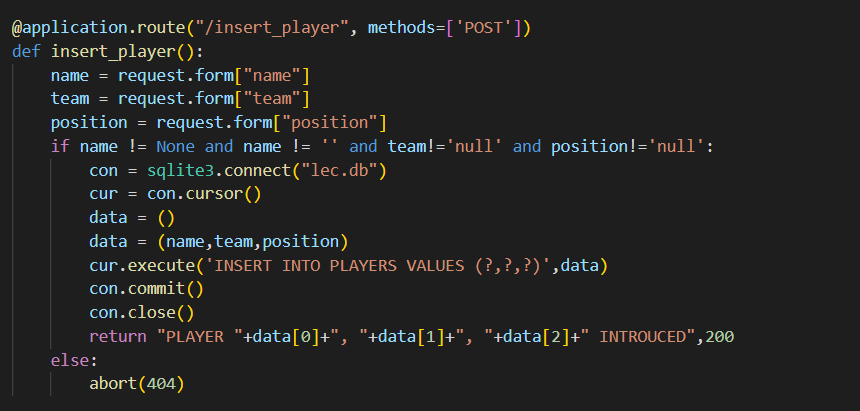
El método HTTP POST envía datos al servidor. El tipo del cuerpo de la solicitud es indicada por la cabecera Content-Type.

**2.3.1 POST INSERT PLAYER**

Obtiene los datos del template generado con el método GET de la ruta:

*http://127.0.0.1:5000/insert\_player*

Recibe los datos del POST y con ellos hace las comprobaciones necesarias para comprobar si se puede insertar el jugador, y si se cumplen lo inserta en la bbdd.

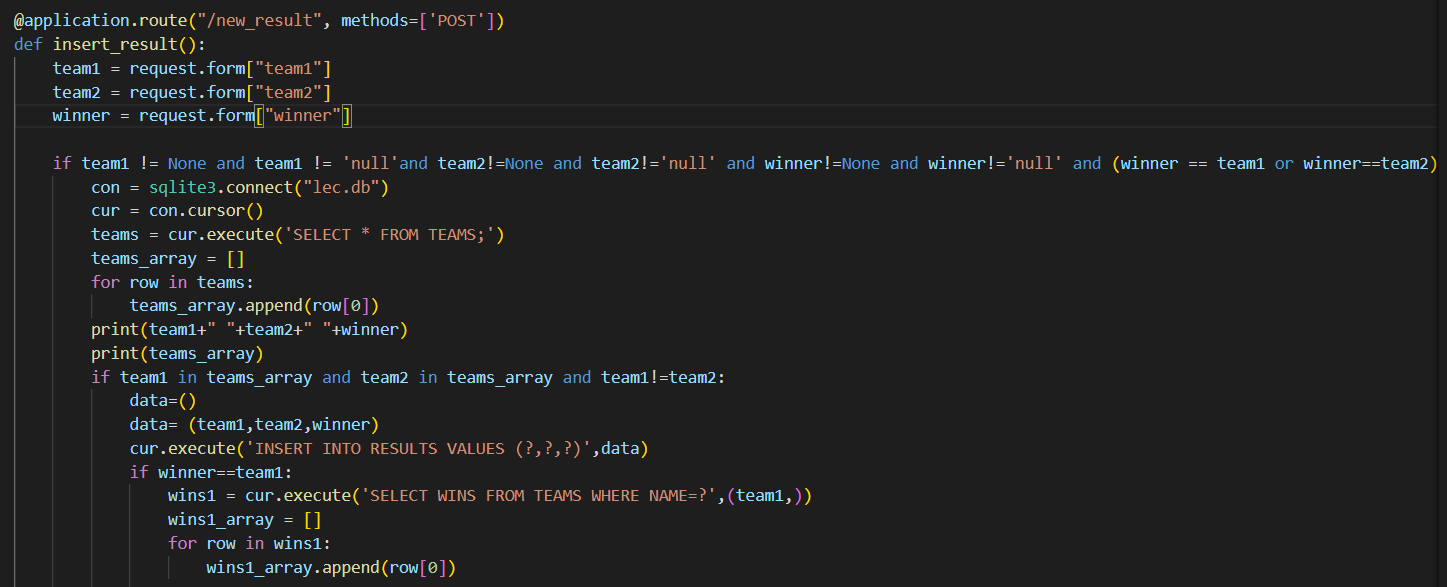


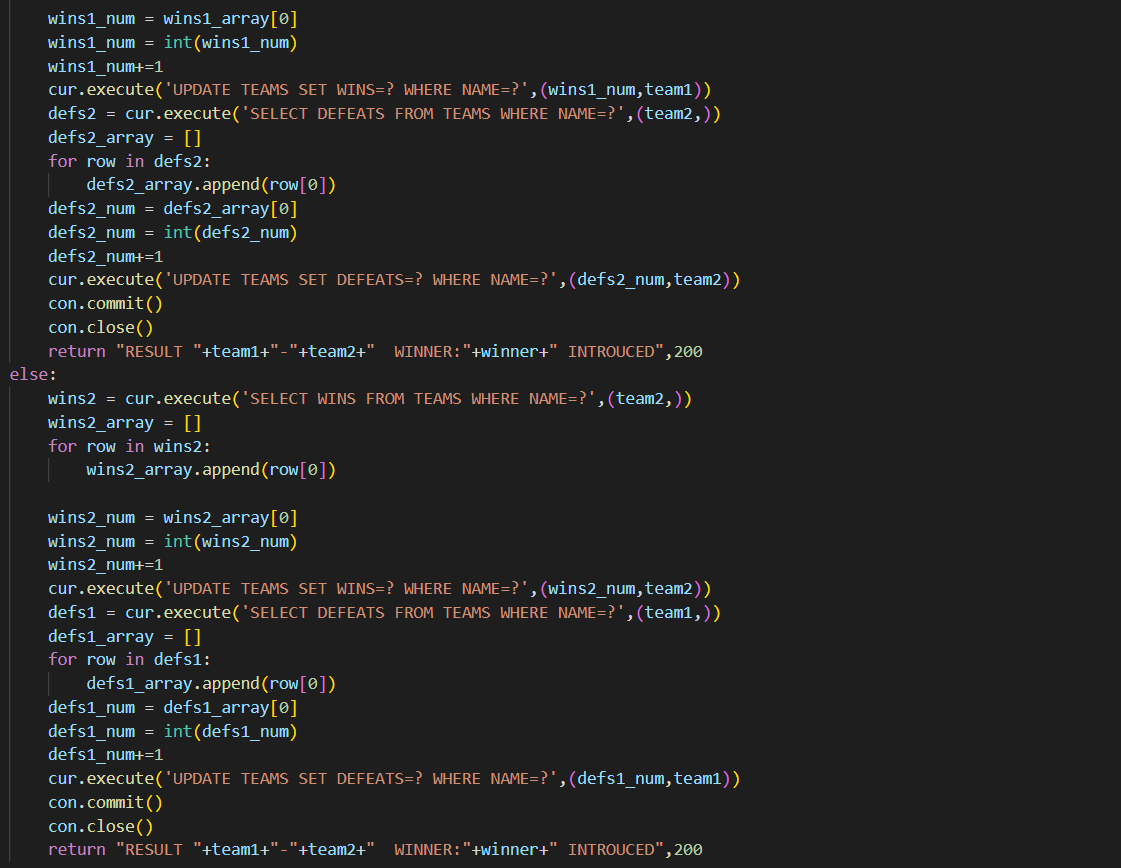
**2.3.2 POST de NEW RESULT**

Realiza un insert en la bbdd de un nuevo resultado con los datos recuperados del template generado en el método get de la ruta:

*http://127.0.0.1:5000/new\_result*

Tiene en cuenta que los equipos 1 y 2 no sean iguales y que el equipo ganador sea uno de los dos equipos seleccionados en los selects de equipo 1 y equipo 2.







## 2.4 PUT

La petición HTTP PUT crea un nuevo elemento o reemplaza una representación del elemento de destino con los datos de la petición.

**2.4.1 MODIFY PLAYER**

A través de postman o una herramienta similar, si ejecutamos la ruta con query string params de “name”,”team”,”position” se modificará el jugador de nombre name, con la posición o equipo pasado por parámetros si estos existen en la bbdd. El parámetro obligatorio es name y team y position pueden pasarse los dos, o uno de ellos.

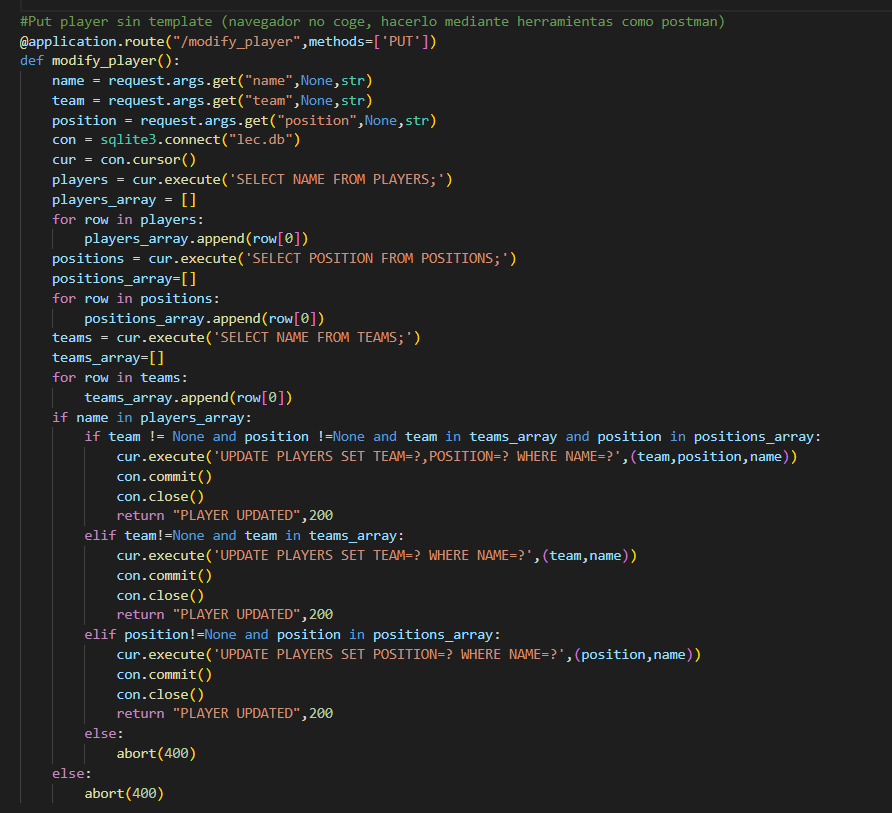
Ejemplos de ejecución:

*http://127.0.0.1:5000/modify\_player?name=BROKENBLADE&team=KOI&position=JUNGLE*

*http://127.0.0.1:5000/modify\_player?name=BROKENBLADE&team=KOI*

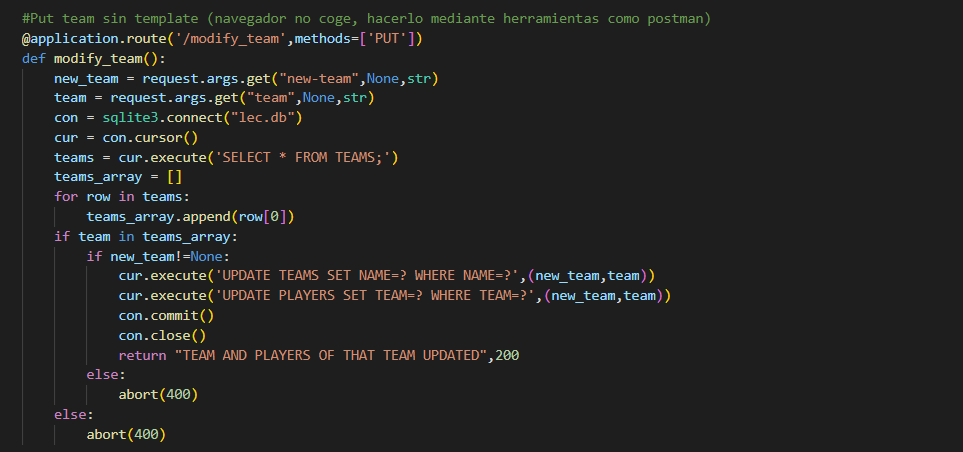
*http://127.0.0.1:5000/modify\_player?name=BROKENBLADE &position=JUNGLE*

Cualquiera de estas tres rutas ejecutarían correctamente el método.

****

**2.4.2 PUT TEAM**

A través de postman o una herramienta similar, si ejecutamos la ruta con query string params de “new-team”,”team”, se modificará el equipo de nombre team, con el nombre pasado en new-team. Ambos parámetros son obligatorios y team tiene que existir en la BBDD.

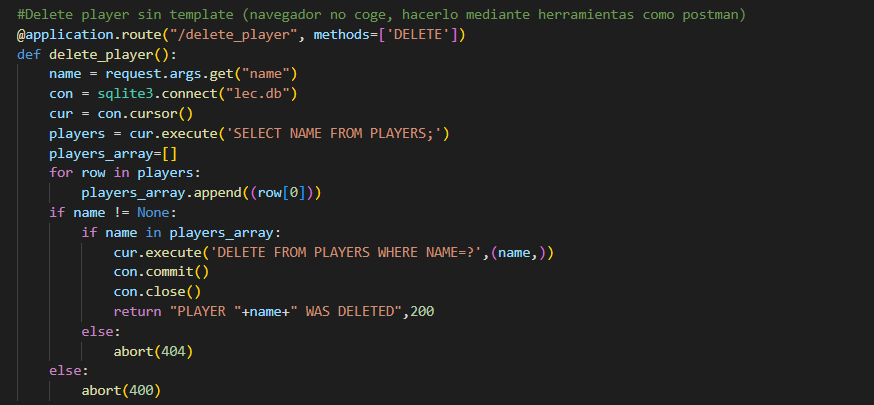
****

## 2.5 DELETE

El método DELETE borra un recurso en específico.

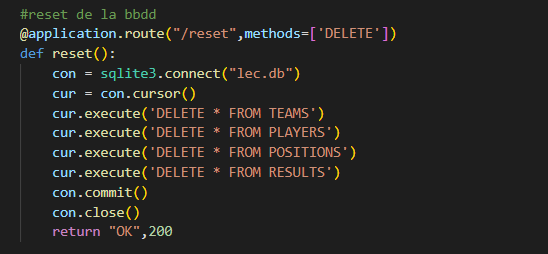
**2.5.1 DELETE PLAYER**

A través de postman o una herramienta similar, si ejecutamos la ruta con query string params de “name”,””, se eliminará el jugador de nombre name en la bbdd.



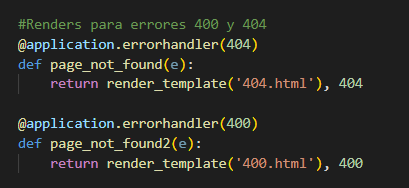
**2.5.2 RESET**

Método para vaciar la bbdd salvo la tabla de posiciones que nunca va a cambiar. No se recomienda utilizar pues habría que volver a meter todos los datos manualmente o ejecutar una query con todas las “subquerys” para rellenar la bbdd.



# ERRORES

Se ejecutan en situaciones como cuando no existe la ruta, se pasan como parámetros datos imposibles… Hay dos diseñados en la API.



Si se ejecuta el abort(err) siendo err 404 o 400, se ejecuta el template indicando que se trata de un error 400. El template es el siguiente:

