API REST JAVA-SPARK CON MARIADB

El mismo rest que se construyó con la librería Jersey, ahora lo hacemos pero con Spark, haremos uso de la misma BD y las mismas capas MVC del ejemplo ya realizado anteriormente con Jersey.

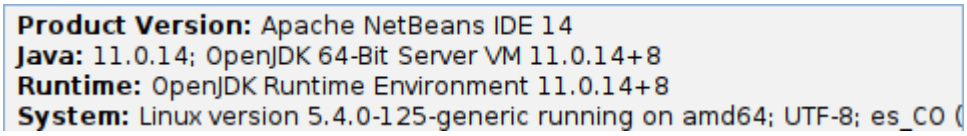
A mi la verdad me parece más fácil trabajar con Spark, pues no hay que batallar con anotaciones, el código queda más simple y entendible, y con la ventaja de que no se requiere de una aplicación o contenedor web, se puede ejecutar desde la línea de comandos, ya que el jetty viene incluido en las mismas dependencias. Spark tiene algo muy bueno y es que utiliza la sintaxis funcional de Java 8, y se puede probar con httpie o curl. Para este ejemplo se construyó una aplicación desktop sencilla, cada quien que la optimice, mejore y se divierta que de eso se trata.

Este ejemplo es un simple CRUD para una tabla de registro de Usuarios.

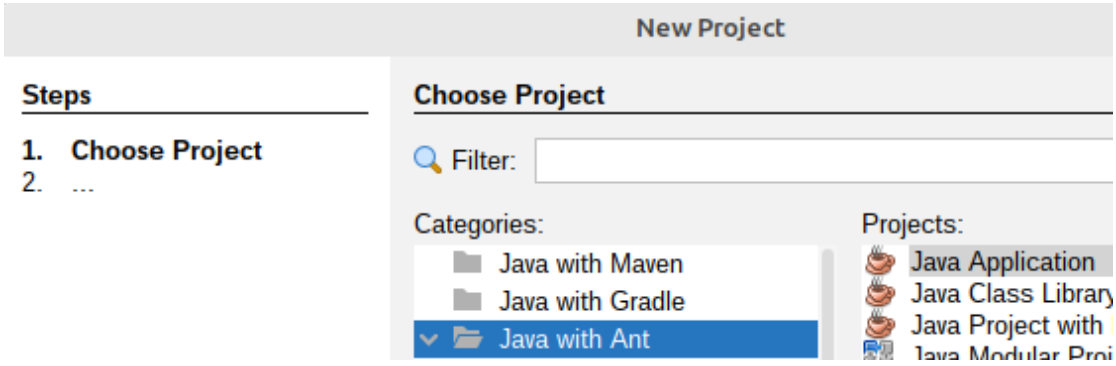
Vamos inicialmente a crear nuestra Base de Datos para probar el API REST, ésta BD está muy sencilla, pues solo tiene una tabla con tres columnas, pero por ahora es solo lo que requerimos para probar. La BD yo la llamé: ApiRest_DB, se puede renombrar con el nombre que se desee, pero deberás luego modificar la cadena de conexión, en la clase dispuesta para ello, y que se encuentra alojada en la capa DAL. Crearemos una estructura llamada: Usuarios y tres procedimientos almacenados, para gestionar todo directamente en la BD, y agilizar las respuestas solicitadas al API, pues si bien el json es ligeramente más eficiente que el xml, todavía los servicios web pecan de ser algo lentos, la versión de MariaDB que yo instalé es: Ver 15.1 Distrib 10.3.34-MariaDB.

CREACIÓN PROYECTO EN JAVA-NETBEANS

Una vez hayas creado la BD, procederemos a crear el API en Java, para este caso utilicé NetBeans 14 y JDK 11, sobre sistema operativo Linux.

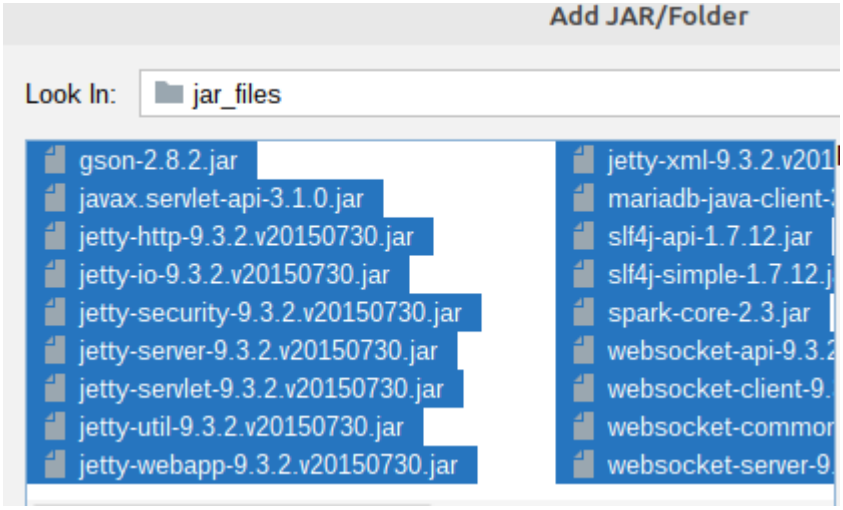


El primer paso es dar click en File y seleccionamos New Project, y allí elegimos proyecto tipo: Java Application.



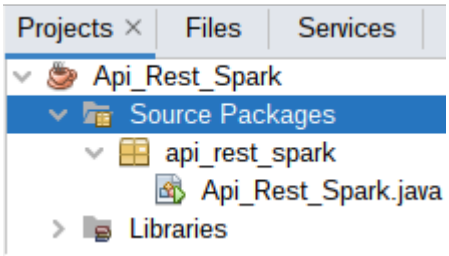
Le colocamos el nombre al api y le damos Finish al asistente de creación.

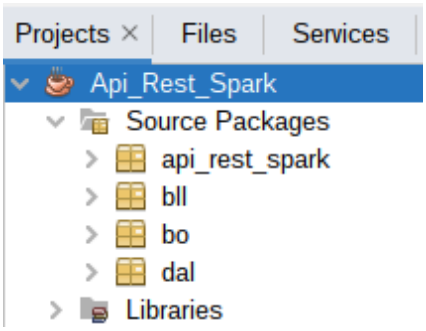
En el explorador de proyectos, ahora te ubicas en Libraries, y vamos a importar las librerías que se encuentran en el mismo repositorio que descargaste de github, allí hay una carpeta llamada: jar_files, que está comprimida con zip, la descomprimes y traes todo el contenido (19 librerías)



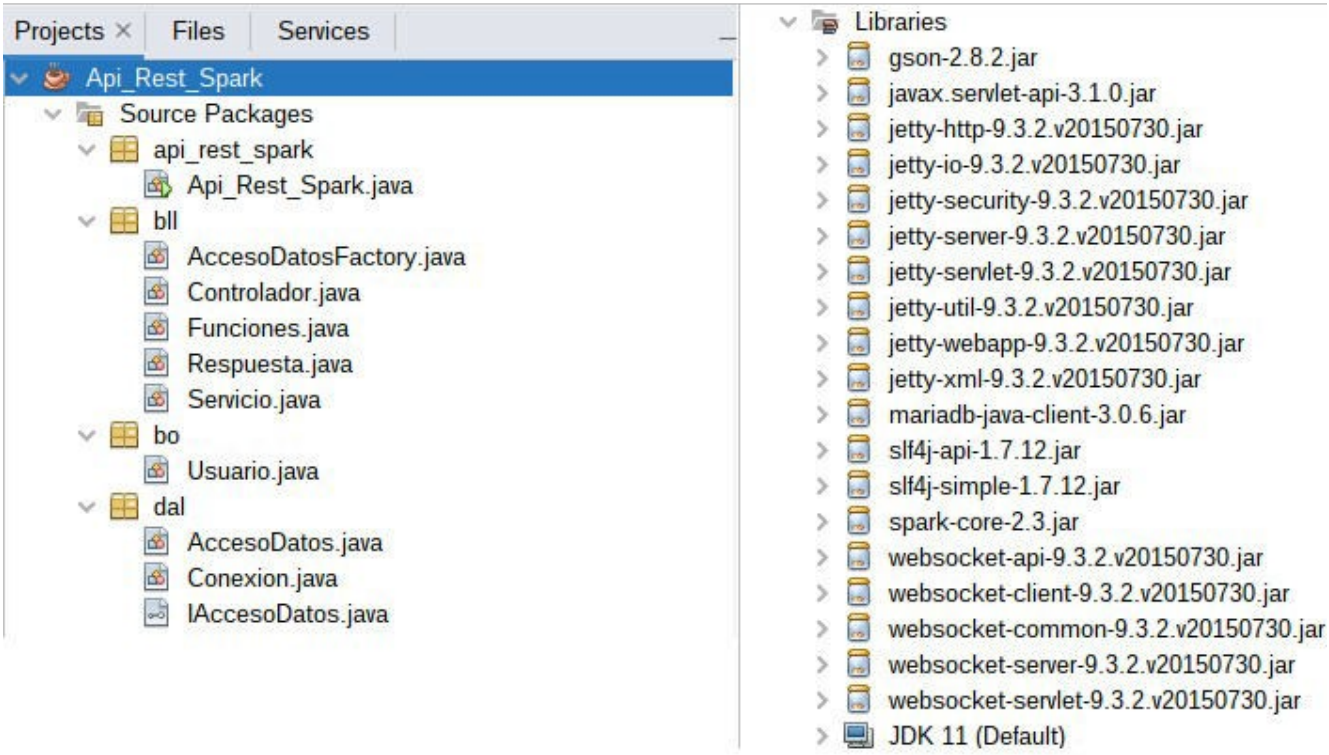
Ubicarás el directorio que descargaste de github, y dentro de la carpeta src, hay una llamada java, y a su vez, dentro de esta, encontrarás 4 carpetas, a saber: api_res_spark, bll, bo y dal.

Las seleccionarás todas y le das Ctrl + c y las copias, luego nos vamos al explorador de proyectos y nos ubicamos dentro de Source Packages





Una vez allí le damos Ctrl + v y pegamos los 4 paquetes, para evitar crear manualmente todas las clases requeridas; nuestro proyecto hasta ahora debería verse así:



Bien ahora te detallo como quedó configurado el proyecto, para que se entienda la arquitectura y distribución de carpetas en el proyecto, yo utilicé MVC donde quedaron 4 paquetes a saber:

Api_Rest_Spark: Es el main principal por donde comienza todo, y se define la operación a efectuar así:

- POST: Encargado de crear un nuevo registro (Usuario).
- PUT: Utilizado para actualizar un registro en concreto (El usuario según su ID)
- DELETE: Borra un registro específico (Un usuario según su ID).

BLL - Capa Lógica de Negocio: En esta alojamos toda la lógica propia del mundo que se pretende modelar, donde quedarán nuestros controladores. En realidad es una capa que sirve de enlace entre las vistas y los modelos (BO), respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de puente entre modelos y vistas, acá en este paquete quedan 5 clases a saber:

- AccesoDatosFactory: Aquí emplearemos el patrón de Creación: Factory Method al cual le pasaremos como argumento la Interfaz IAccesoDatos, e invocaremos el método: CrearControlador alojado en la clase Funciones que nos devolverá, ya la clase AccesoDatos instanciada.

- Controlador: Como su nombre lo indica es el controlador manager general de todo el sistema. Todo tiene que pasar por esta clase.

- Funciones: Esta es una clase transversal, desde donde se instancia el Factory Method.

- Respuesta: Es simplemente una clase desde donde controlamos los diferentes mensajes a visualizar.

- Servicio: es la clase que llama el controlador quien a la vez llama la interfaz y el DAO correspondiente para efectuar las operaciones solicitadas.

BO - Capa Objetos de Negocio: Acá queda alojado nuestro único objeto de negocio llamado: Usuario, que es la clase que representa esta estructura en BD, y que solo tiene sus atributos con sus respectivos get y set.

DAL - Capa Acceso a Datos: Es la capa que contiene todos los mecanismos de acceso a nuestra BD, acá en este paquete quedan tres clases a saber:

- Conexion: Clase que conecta con el proveedor de base de datos MariaDB.
- IAccesoDatos: Interfaz que solo expone los métodos que implementa el acceso a datos.
- AccesoDatos: Implementa la interfaz con todos sus métodos.

Funcionamiento

El funcionamiento es simple: la capa de presentación pregunta a la BLL por algún objeto, ésta a su vez puede opcionalmente desarrollar alguna validación, y después llamar al DAL, para que esta conecte con la base de datos y le pregunte por un registro específico, cuando ese registro es encontrado, éste es retornado de la base de datos al DAL, quien empaqueta los datos de la base de datos en un objeto personalizado y lo retorna al BLL, para finalmente retornarlo a la capa de presentación, donde podrá ser visualizado por el cliente en formato json.

Tanto la capa lógica de negocio como la capa de acceso de datos consiguen una referencia a los objetos en el BO. Además, la capa de negocio consigue una referencia a la capa de acceso de datos para toda la interacción de datos.

Los objetos del negocio se colocan en una capa diferente para evitar referencias circulares entre la capa del negocio y la de datos. Es importante destacar que en la DAL habrá una interfaz llamada IAccesoDatos, que será nuestra puerta de entrada, allí no hay implementación de ningún método, solo se exponen, quien use dicha interfaz es quien los debe implementar, para nuestro caso el DAL (AccesoDatos).

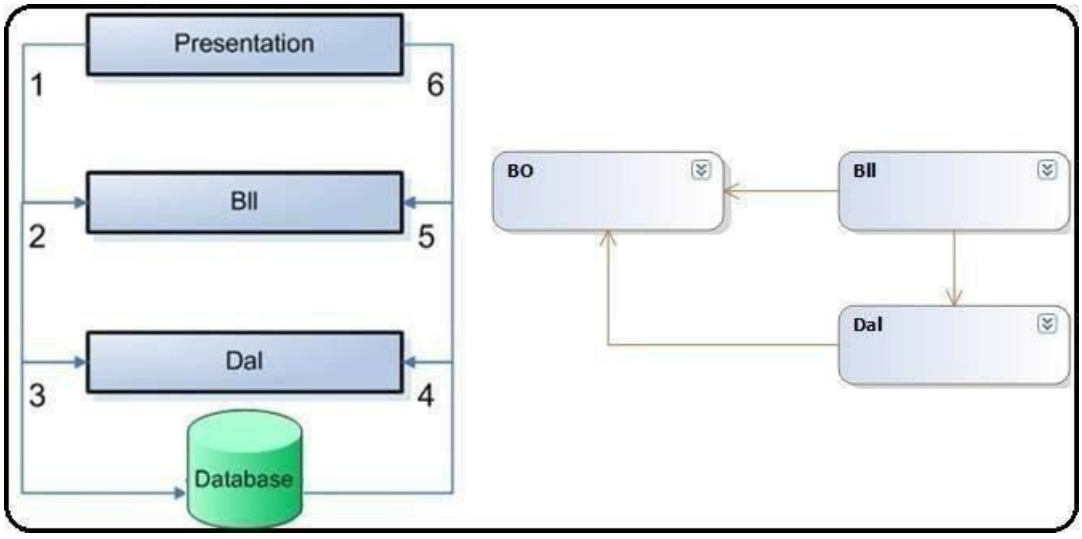
Nos apoyaremos en el patrón de creación: Factory Method que nos ayudará a la hora de instanciar la clase DAL, que debe implementar todos los métodos de la interface IAccesoDatos, lo que nos ahorrará trabajo ya que el conjunto de clases creadas pueden cambiar dinámicamente.

Nunca se llega directamente al DAL, siempre todo deberá pasar primero por el Controlador que será nuestro manager, encargado de orquestar, controlar y definir que método del DAL invocar, para que sea ésta última quien devuelva los resultados esperados, y dar así respuesta a las diferentes peticiones de la vista, requeridas por el usuario.

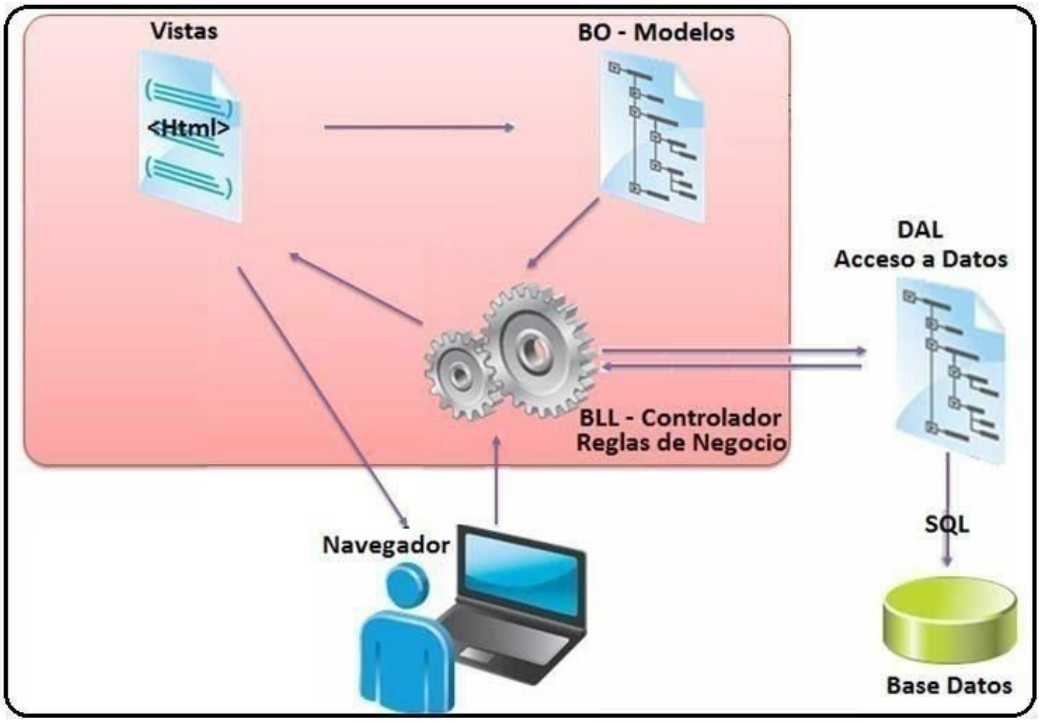
El DAL tiene por cada objeto del BO, métodos para obtener listados completos, o un solo ítem o registro, y por supuesto los demás métodos básicos del CRUD, para creación, actualización y eliminación de registros. La finalidad de separar los sistemas en capas, no es otra que facilitar su posterior mantenimiento, donde cada capa expone servicios que otras aplicaciones o capas pueden consumir, lo que se traduce en una simplicidad conceptual, de alta cohesión y bajo acoplamiento, facilitando así su reutilización, y MVC nos ayuda con esa tarea, este patrón de arquitectura es definitivamente una belleza, por la forma como trabaja.

Donde predomina una “organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.” Garlan & Shaw.

ESQUEMA DE FUNCIONAMIENTO



FUNCIONAMIENTO MVC



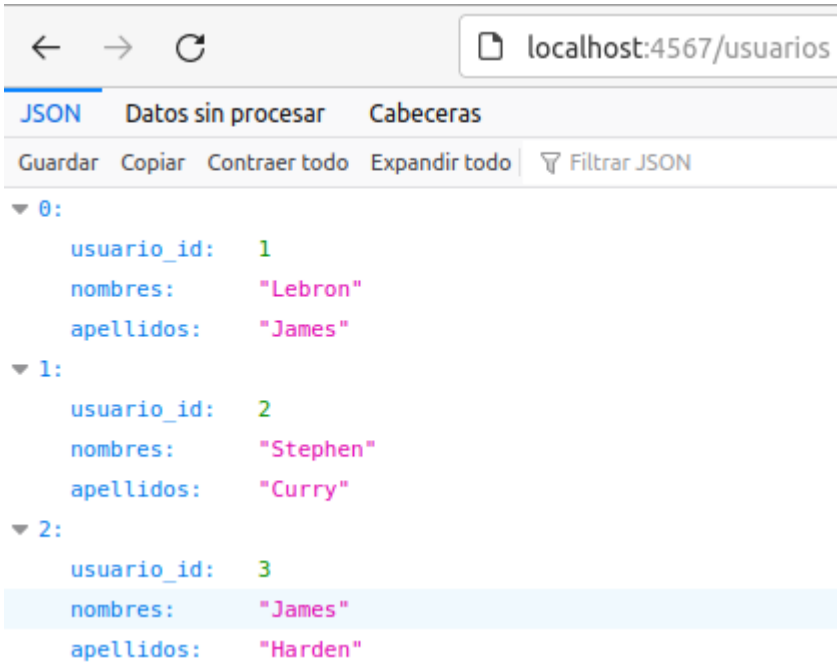
PRUEBAS API

Bien manos a la obra probemos en un Mozilla FireFox e ingresamos en la barra de direcciones: http://localhost:4567/usuarios

Por omisión, lo estará ejecutando en el puerto 4567

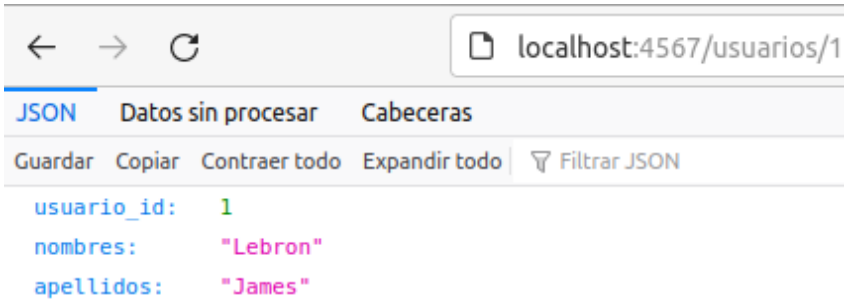
Obtener Todos los usuarios GET

-



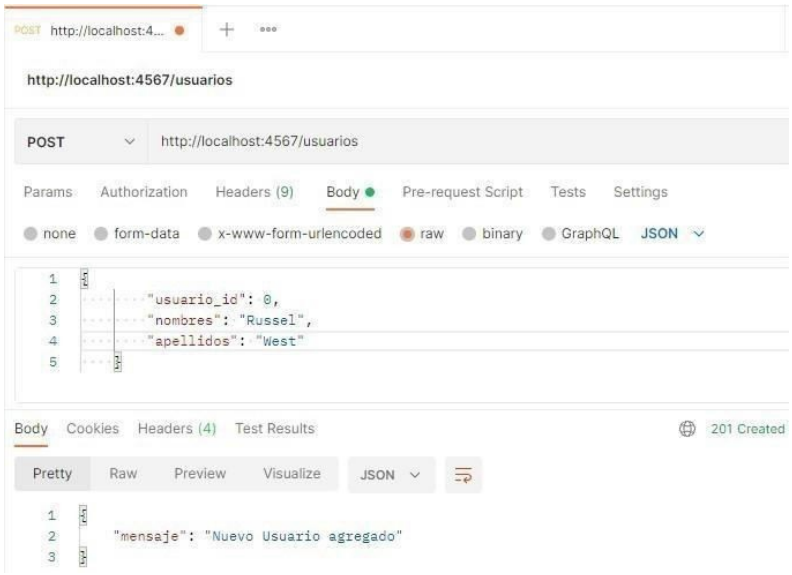
Obtener Un Usuario GET

http://localhost:4567/usuarios/1

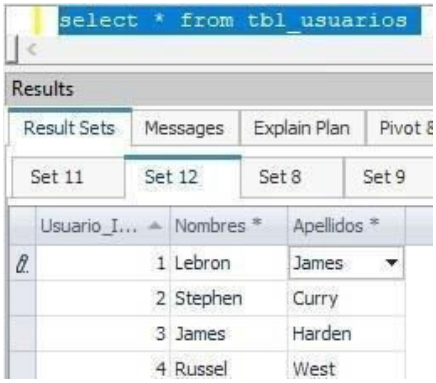


Agregar Un Usuario POST

http://localhost:4567/usuarios



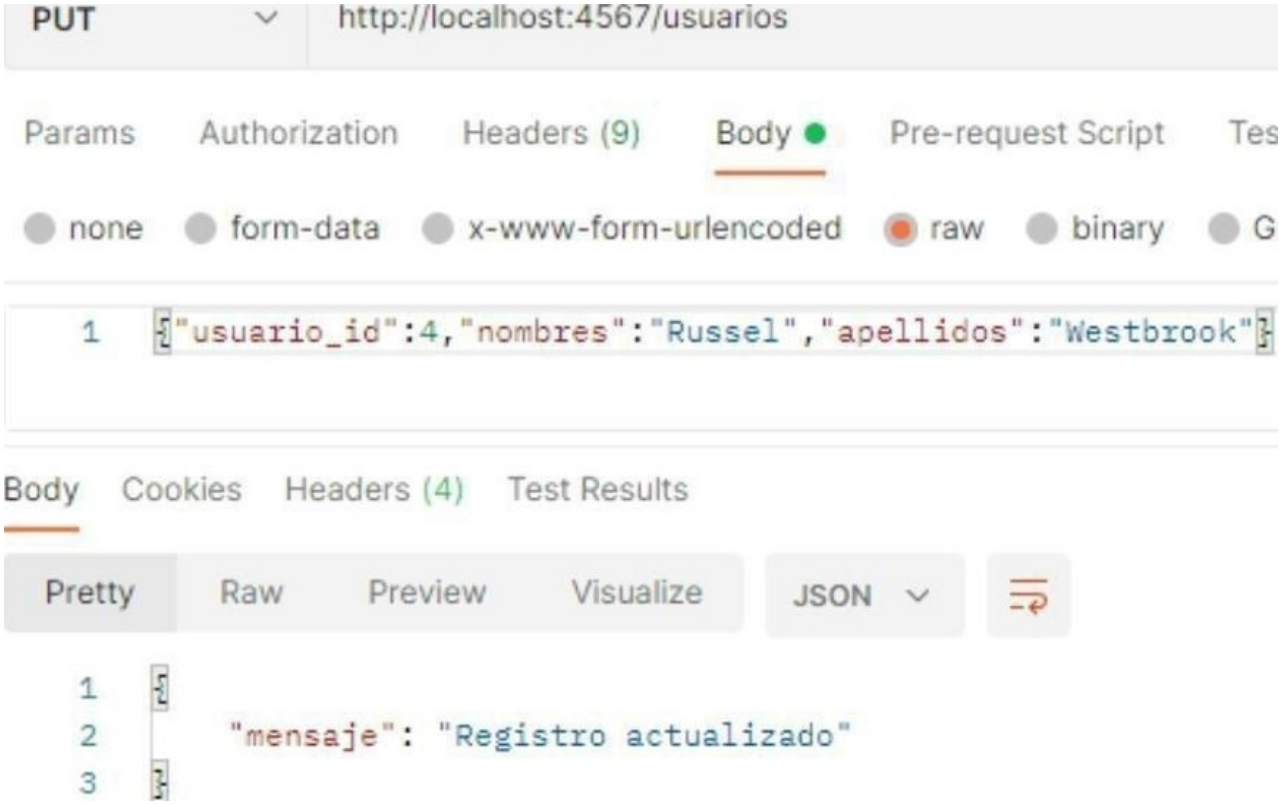
Verificando el Insert en BD



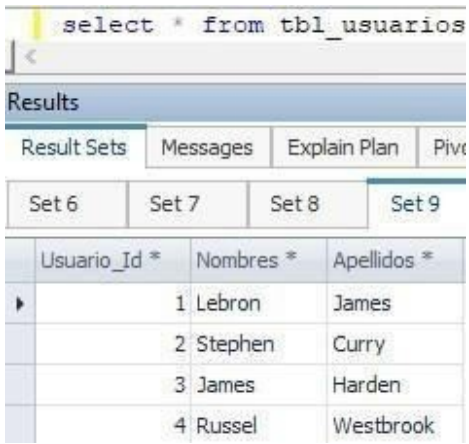
Actualizar Un Usuario PUT

http://localhost:4567/usuarios/4

Cambiaremos el apellido de este registro "West" por "Westbrook", que es el correcto apellido de este famoso jugador de baloncesto.



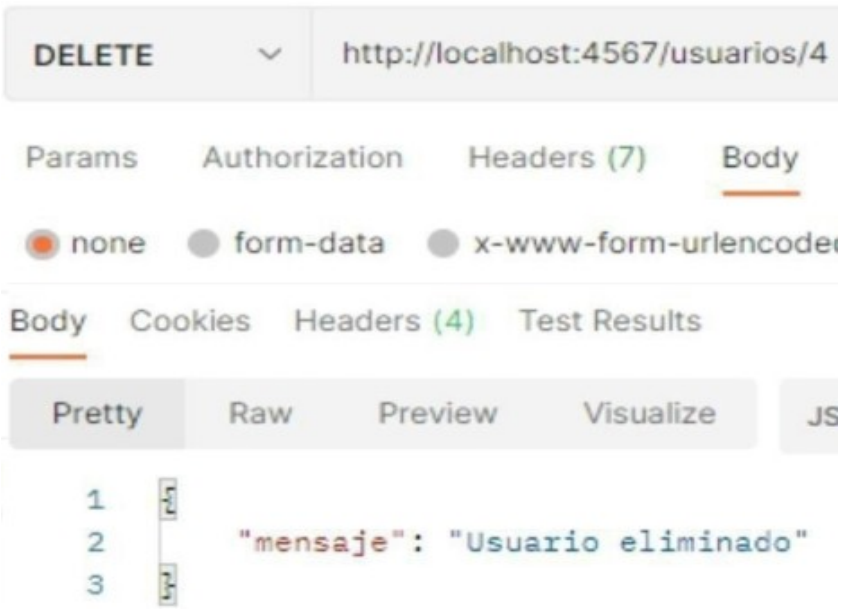
Verificando el Update en BD



Es importante indicar que tanto para crear y actualizar recursos, podemos enviar los datos en formato JSON o utilizando parámetros de un formulario, en este ejemplo se enviará un JSON, pero en las manos del programador queda la decisión de enviar parámetros o el JSON completo.

Eliminar Un Usuario DELETE

http://localhost:4567/usuarios/4



Verificando el Delete en BD

Es importante mencionar que el mensaje de respuesta está sujeto a lo que devuelva el procedimiento almacenado, para el caso de los procedimientos de inserción y borrado, siempre devolverá un 0, si la transacción fue exitosa, en caso contrario es que el registro que se intentó actualizar o borrar no existe, y devolverá un -2.

Finalmente, esta no es ni la mejor, ni la única forma de hacer las cosas, los microservicios en Java, se pueden construir apoyándose en varios frameworks, y la persistencia igual se puede realizar de muchas maneras, yo elegí los procedimientos almacenados para gestionar todo en el servidor, pero eres libre de hacerlo como convenga según las necesidades.