DOCUMENTACIÓN TECNICA WEBSOCKET

OMAR TORRES YUSTE 2º DAM

PROGRAMACION DE SERVICIOS Y PROCESOS

Índice

Introducción

El programa consiste en un servidor web hecho en java usando Java Sockets. El servidor es capaz de gestionar una biblioteca , más concretamente de libros usando una base de datos MySQL, procesa peticiones HTTP (GET ,POST, PUT, DELETE), devolviendo los datos en formato JSON

Arquitectura

El código se organizó siguiendo el modelo de diseño DAO y principios de la programación orientada a objetos, separando la lógica de la red, la del server y la del CRUDen diferentes clases explicadas a continuación.

Gestion de conexión (DbConector.java)

Esta clase es la encargada de conectarse a la base de datos y testear su correcto funcionamiento, tiene 3 variables url, user y password para conectarse posteriormente con sus métodos a la base de datos, consta de 3 métodos:

testConnection(): Se encarga de probar a conectarse a la base de datos y si no lo consigue tira un SQLException

```
   public void testConnection() throws SQLException {
        try (Connection con = connect()) {
            if (con == null || con.isClosed()) {
                throw new SQLException("No se pudo establecer la conexión
a  la base de datos.");
            }
        }
    }
```

connect(): Se encarga de abrir la conexión a la base de datos usando la url, user y password de la clase

```
  public Connection connect() throws SQLException {
        return DriverManager.getConnection(url, user, password);
    }
```

close(): Se encarga de cerrar la conexión de la base de datos, comprueba que la sesión esté abierta y si es así la cierra, y si ocurre algún error tira un SqlException

```
  public void close(Connection con) {
        try {
            if (con != null && !con.isClosed()) {
                con.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

Modelo de datos(Book.java)

Esta clase es la encargada de definir como es un libro, en ella estan las variables de un libro como id, title, author y year. Además tiene un constructor para a la hora de instanciar el libro nos pide rellenar las variables con datos, no tiene ningun metodo en especial simplemente tiene getter y setter para sus variables y un metodo para convertirlo en formato json

toJson():

```
   public String toJson() {
        return "{\"id\":" + id + ",\"titulo\":\"" + title +
"\",\"autor\":\"" + author + "\",\"anio\":" + year + "}";
    }
```

Si le pasamos un libro con los siguientes datos
id → 1 , title → Don quijote, author → Miguel de cervantes, year → 1605

Devolvería:


```
{
   "id": 1,
   "titulo": "El Quijote",
   "autor": "Miguel de Cervantes",
   "anio": 1605
}
```

Procesamiento de datos (JsonParser.java)

Esta clase se encarga de extraer información de un texto tipo jso, actualmente solo tiene un método

extraer(String json, String campo**):** Pasamos el json como un String y pasamos también el campo que queremos sacar del json, esto devolverá el campo pedido de un Json

```
 public static String extraer(String json, String campo) {
        try {
            String llave = "\"" + campo + "\"";
            int inicioLlave = json.indexOf(llave);
            if (inicioLlave == -1) return "";
            int inicioValor = json.indexOf(":", inicioLlave) + 1;
            int finValor;
            if (json.trim().substring(json.indexOf(":",
inicioLlave)).trim().startsWith("\"")) {
                inicioValor = json.indexOf("\"", inicioValor) + 1;
                finValor = json.indexOf("\"", inicioValor);
            } else {
                int finComa = json.indexOf(",", inicioValor);
                int finLlave = json.indexOf("}", inicioValor);
                finValor = (finComa != -1 && finComa < finLlave) ? finComa
: finLlave;
            }
            return json.substring(inicioValor,
finValor).trim().replace("\"", "");
        } catch (Exception e) { return ""; }
    }
```

Si le pasamos por ejemplo el json del quijote anterior, y pasamos como campo titulo, nos sacara únicamente El quijote y eso será lo que devolverá el método

Acceso a datos(BookDAO.java)

Esta clase es la encargada de acceder a los datos y realizar todas las operaciones CRUD necesarias para el sistema usando principalmente el conector a la base de datos y el ResultSet para ejecutar consultas Mysql, como variable tiene una variable de tipo DbConector que se le pasará al dao a la hora de instanciarlo en el main pasandole la conexión que nos interese

Anotación: Hice que en el DAO se le pasara la conexión de esa manera pensando en un "futuro" donde por lo que sea se tuviera que usar otra base de datos mas no solamente el de library_db, de esta manera si por ejemplo se añadiera algo que requiera guardarse en otra base de datos se podría pasar en el DAO del objeto nuevo a guardar en otra base de datos, no se me ocurre ningún ejemplo ahora mismo pero me pareció una buena idea integrarlo de esta manera pensando en un posible futuro

getAll(): Este método se encarga de conectarse a la base de datos pasada en la variable conector, y devuelve un arraylist de libros

```
   public List<Book> getAll() throws SQLException {
        List<Book> books = new ArrayList<>();
        try (Connection con = conector.connect()) {
            ResultSet rs = con.createStatement().executeQuery("SELECT *
FROM books");
            while (rs.next()) {
                books.add(new Book(
                    rs.getInt("id"),
                    rs.getString("title"),
                    rs.getString("author"),
                    rs.getInt("year")
                ));
            }
        }
        return books;
    }
```

getbyid(int id**)**: Este método se conecta a la base de datos de la variable conector y busca el libro cuyo id coincida con el id pasado como parámetro en el método

```
    public Book getById(int id) throws SQLException {
    try (Connection con = conector.connect()) {
        PreparedStatement ps = con.prepareStatement("SELECT * FROM books
WHERE id = ?");
        ps.setInt(1, id);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            return new Book(rs.getInt("id"), rs.getString("title"),
rs.getString("author"), rs.getInt("year"));
        }
    }
    return null;
}
```

insert(Book book**):** En este método insertamos el objeto libro pasado por parámetros en la base de datos

```
  public void insert(Book book) throws SQLException {
        try (Connection con = conector.connect()) {
            String sql = "INSERT INTO books (title, author, year) VALUES
(?, ?, ?)";
            PreparedStatement ps = con.prepareStatement(sql);
            ps.setString(1, book.getTitle());
            ps.setString(2, book.getAuthor());
            ps.setInt(3, book.getYear());
            ps.executeUpdate();
        }
    }
```

update(Book book**)**: El método update se encarga de actualizar los datos de un libro, se le pasa como parámetro el nuevo libro a insertar y hace el update del libro, esto se explica mejor en el main ya que allí se obtiene el libro anterior y se recoge los cambios nuevos para agregarlos finalmente al update y hacer el cambio.

```
public void update(Book book) throws SQLException {
    try (Connection con = conector.connect()) {
        String sql = "UPDATE books SET title = ?, author = ?, year = ?
WHERE id = ?";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setString(1, book.getTitle());
        ps.setString(2, book.getAuthor());
        ps.setInt(3, book.getYear());
        ps.setInt(4, book.getId());
        ps.executeUpdate();
    }
}
```

delete(int id**):** Este método se encarga de eliminar el libro del id indicado.

```
  public void delete(int id) throws SQLException {
        try (Connection con = conector.connect()) {
            PreparedStatement ps = con.prepareStatement("DELETE FROM books
WHERE id = ?");
            ps.setInt(1, id);
            ps.executeUpdate();
        }
    }
```

Servidor principal(Main.java)

La función de esta clase es gestionar todo el ciclo de vida del servidor, inicia la escucha peticiones y actúa como el controlador delegando las diferentes tareas a sus correspondientes clases. Aquí tenemos muchos tipos de variables pero las principales serían el conector que es un objeto de la clase Dbconector que se encarga de almacenar los datos para la conexión de la Db, como la url, usuario y contraseña y dao que es un objeto de BookDAO para la gestión del crud de libros.
Al ser esta la clase más larga de todas voy a dividir la explicación por partes

Inicialización del servidor y comprobación del estado

```
DbConector conector = new
DbConector("jdbc:mysql://localhost:3306/library_db", "root", "");
        BookDAO dao = new BookDAO(conector);


 try {
            System.out.println("Verificando la conexion a la base de
datos...");
            conector.testConnection();
            System.out.println("Base de datos conectada correctamente.");
```

Aqui le pasamos como comente antes la url, usuario y contraseña e instanciamos conector y dao pasandole el conector, acto seguido hacemos un try para probar el método testConecction para verificar que se pueda acceder a la base de datos, si puede acceder imprime por pantalla base de datos conectada correctamente y si no lanza un SQLException, esto lo implemente para prevenir situaciones donde el programa no funcione correctamente debido a que no se pueda conectar a la base de datos, ya que sin esta comprobación aunque no conectarse seguiría adelante.

Escucha de Sockets / peticiones

```
 try (ServerSocket servidor = new ServerSocket(8080)) {
                System.out.println("Servidor iniciado en el puerto " +
servidor.getLocalPort());

                while (true) {
                    try (Socket cliente = servidor.accept()) {
```

En esta parte del código usamos la clase ServerSocket para abrir el puerto 8080 para escuchar peticiones, abrimos un bucle infinito para mantener abierto la escucha de peticiones por parte del cliente

Análisis de la petición

```
String[] partes = linea.split(" ");
                        String metodo = partes[0];
                        String ruta = partes[1];
```

El servidor lee la primera línea de la petición y la divide mediante el método .split(" ").
Esto permite identificar qué quiere hacer el usuario (metodo) y sobre qué recurso (ruta).

Procesamiento de los headers

```
    int longitud = 0;
                        String cabecera;
                        while (!(cabecera = entrada.readLine()).isEmpty())
{
                            if
(cabecera.toLowerCase().startsWith("content-length:")) {
                                longitud =
Integer.parseInt(cabecera.split(":")[1].trim());
                            }
                        }
```

Leemos los metadatos de la petición para leer y saber la longitud del contenido recibido y lo almacenamos en la variable longitud , para usarlo posteriormente para leer correctamente la petición

Procesamiento del cuerpo

```
String body = "";
                        if (longitud > 0) {
                            char[] bodyChars = new char[longitud];
                            int charsRead = 0;
                            while (charsRead < longitud) {
                                int result = entrada.read(bodyChars,
charsRead, longitud - charsRead);
                                if (result == -1) {
                                    break;
                                }
                                charsRead += result;
                            }
                            body = new String(bodyChars);
                        }
```

Creamos la variable tipo string body para almacenar el cuerpo de la petición posteriormente, comparamos que la longitud no sea 0 y leemos caracter a caracter hasta que no queden mas y los metemos en body

Enrutamiento y manejo de estados HTTP

Rutas sin id

Get /books

```
String respuestaJson = "";
String httpStatus = "404 Not Found";

    try {
       if (ruta.equals("/books")) {
         switch (metodo) {
            case "GET":
             List<Book> lista = dao.getAll();
              StringBuilder sb = new
             stringBuilder("[");
               for (int i = 0; i < lista.size(); i++) {
s              sb.append(lista.get(i).toJson());
                if (i < lista.size() - 1) sb.append(",");
                }
           respuestaJson = sb.append("]").toString();
           httpStatus = "200 OK";
           break;
```

Creamos variable respuesta y httpStatus que por default la ponemos en 404 Not found, y a continuación comparamos si la ruta es /books, si es asi comparamos el método para saber si es un Get o un Post, e ne caso que sea un get pues creamos una lista de libros obtenidos a través del método getall de dao, para posteriormente construir la el json recorriendo la lista de cada libro y usando su método de toJson de cada libro para meterlo en el contenedor [ ] del formato json.

Post /books

```
case "POST":
          String titulo = JsonParser.extraer(body, "titulo");
          String autor = JsonParser.extraer(body, "autor");
          int annio = parseAnio(JsonParser.extraer(body, "anio"));
          Book newBook = new Book(0, titulo, autor, annio);
           dao.insert(newBook);
          respuestaJson = "{\"status\":\"creado\"}";
           httpStatus = "201 Created";
           break;
          default:
           httpStatus = "405 Method Not Allowed";
           respuestaJson = "{\"error\":\"Metodo no permitido\"}";
            break;
              }
```

En esta sección en el caso de que método sea Post pues extraemos los campos 1 a 1 del json mandado en el cuerpo, y le asignamos una variable para el título, autor, año como annio, y con estos datos creamos el nuevo objeto de libro y cambiamos el status a 201 Created, y en el caso de que no sea ninguno de esos 2 métodos pondremos el error 405 method not allowed y cambiamos la respuesta del json con un mensaje de error indicando que ese método no está permitido

Rutas con id

Para gestionar las rutas con id se compara que la ruta comience por books y el tamaño de la ruta sea superior a 7, ya que si es superior a 7 indica que hay más valores aparte de books, en este caso serían los id, además de guardar en una variable llamada id el valor después de books/ , siendo este el id, y parseando el valor de la String a int para que los metodos de busqueda funcionen correctamente

Get /books/{id}

```
else if (ruta.startsWith("/books/") && ruta.length() > 7) {
    int id = Integer.parseInt(ruta.substring(7));
      switch (metodo) {
        case "GET":
       Book b = dao.getById(id);
      if (b != null) {
       respuestaJson = b.toJson();
        httpStatus = "200 OK";
       } else {
       respuestaJson = "{\"error\":\"Libro no encontrado\"}";
            httpStatus = "404 Not Found";
  }
  break;
```

Comparamos el método pedido, en caso de ser el Get creamos un objeto libro y le asignamos el objeto que devuelve el método del dao getbyid(id) que es el libro cuyo id coincide con el indicado, y en caso de que el libro creado tenga contenido es decir sea diferente a null pues convertimos ese dato en json con el toJson del objeto y lo asignamos a respuestaJson y además cambiamos el estado de httpStatus a 200 OK, y en caso de que no lo encuentre pues mandamos como respuesta error libro no encontrado y el status como 404 not found

Put /books/{id}

```
 case "PUT":
  if (dao.getById(id) != null) {
   String titulo = JsonParser.extraer(body, "titulo");
   String autor = JsonParser.extraer(body, "autor");
   int annio = parseAnio(JsonParser.extraer(body, "anio"));
   dao.update(new Book(id, titulo, autor, annio));
   respuestaJson = "{\"status\":\"actualizado\"}";
   httpStatus = "200 OK";
} else {
 respuestaJson = "{\"error\":\"Libro no encontrado para actualizar\"}";
 httpStatus = "404 Not Found";
   }break;
```

Si el método es Put, comprobamos que id que nos han pasado es válido comprobando con getbyid que el objeto libro con ese id existe, y si existe pues usando la clase JsonParser extraemos los campos del body para asignarle cada campo a una variable, titulo autor y annio, y posteriormente hacemos el update con el nuevo libro usando la id que nos pasaron en la petición y creamos la respuesta json como Status:actualizado, tambien ponemos el status como 200 OK, y en caso de no encontrar el libro del id indicado pues mandamos como status un 404 not found y un mensaje de error en la respuesta json indicando que el libro no se encontro

Delete /books/{id}

```
case "DELETE":
     if (dao.getById(id) != null) {
      dao.delete(id);
      respuestaJson = "{\"status\":\"borrado\"}";
      httpStatus = "200 OK";
      } else {
       respuestaJson = "{\"error\":\"Libro no encontrado para borrar\"}";
      httpStatus = "404 Not Found";
     }

    }
```

Si el método es delete, comprobamos que el libro con el id pasado por la petición exista, y en caso de que exista usamos el método delete del dao y le pasamos como parámetro el id, y devolvemos la respuesta como borrado y el status como 200 ok , en caso de que no se encuentre pues la respuesta devolverá libro no encontrado y el status un 404 not found

Manejo de errores

```
} catch (Exception e) {
 httpStatus = "500 Internal Server Error";
 respuestaJson = "{\"error\":\"SQL Error: " + e.getMessage() + "}";
                        }
```

Manejamos los errores con el catch y devolviendo el status 500 con la respuestaJson con el error

```
  try {
                            sendResponse(cliente, httpStatus,
respuestaJson);
                        } catch (IOException e) {
                            System.err.println("Error al enviar la
respuesta: " + e.getMessage());
                        }
```

Y aquí probamos a mandar la respuesta al cliente, y lo gestionamos con try catch, si da error nos imprime en pantalla error al enviar la respuesta y el mensaje de error

Envio de respuestas al cliente

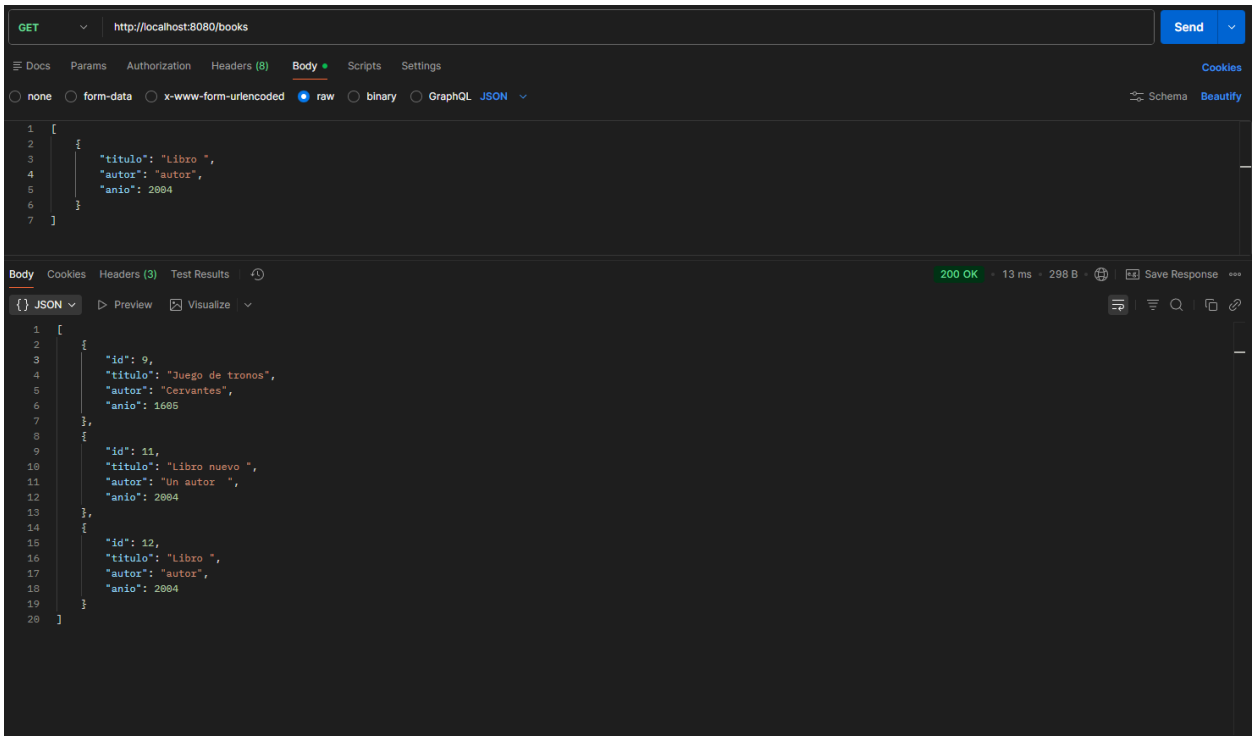
```
 private static void sendResponse(Socket cliente, String status, String
jsonBody) throws IOException {
        OutputStreamWriter osw = new
OutputStreamWriter(cliente.getOutputStream(), StandardCharsets.UTF_8);
        PrintWriter salida = new PrintWriter(osw, true);

        salida.print("HTTP/1.1 " + status + "\r\n");
        salida.print("Content-Type: application/json; charset=UTF-8\r\n");
        salida.print("Content-Length: " +
jsonBody.getBytes(StandardCharsets.UTF_8).length + "\r\n");
        salida.print("Connection: close\r\n");
        salida.print("\r\n");
        salida.print(jsonBody);
        salida.flush();
    }
```

Este método se encarga de transformar los datos procesados por Java en un mensaje que el navegador o Postman puedan entender.
Su función es escribir en el canal de salida del cliente siguiendo la estructura obligatoria de una respuesta HTTP.

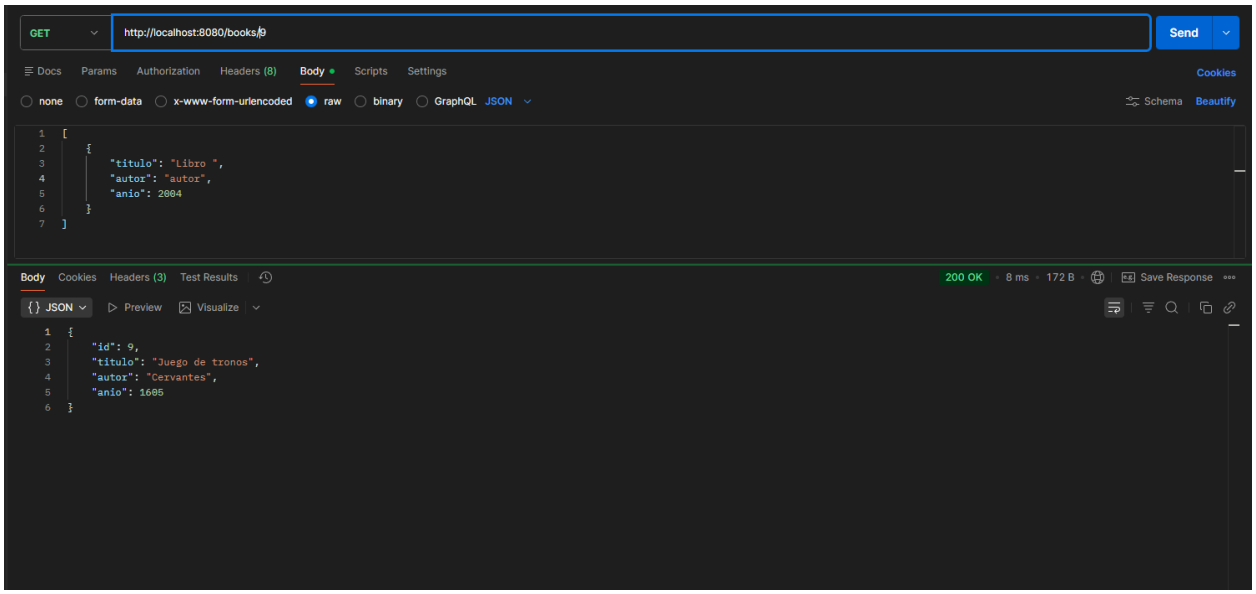
Prueba de peticiones

Get

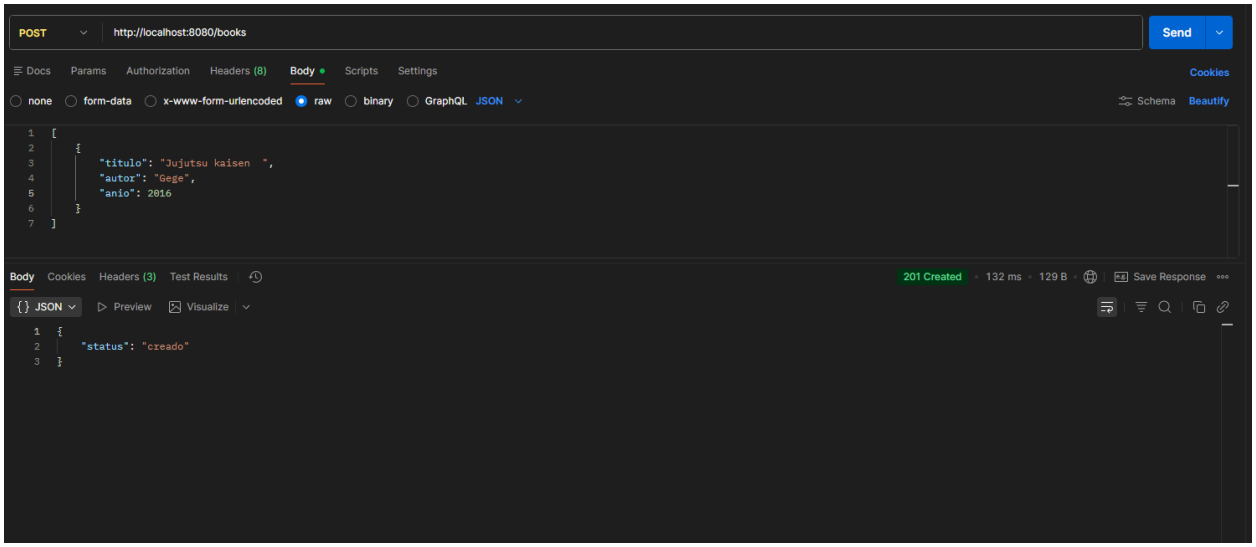
/books

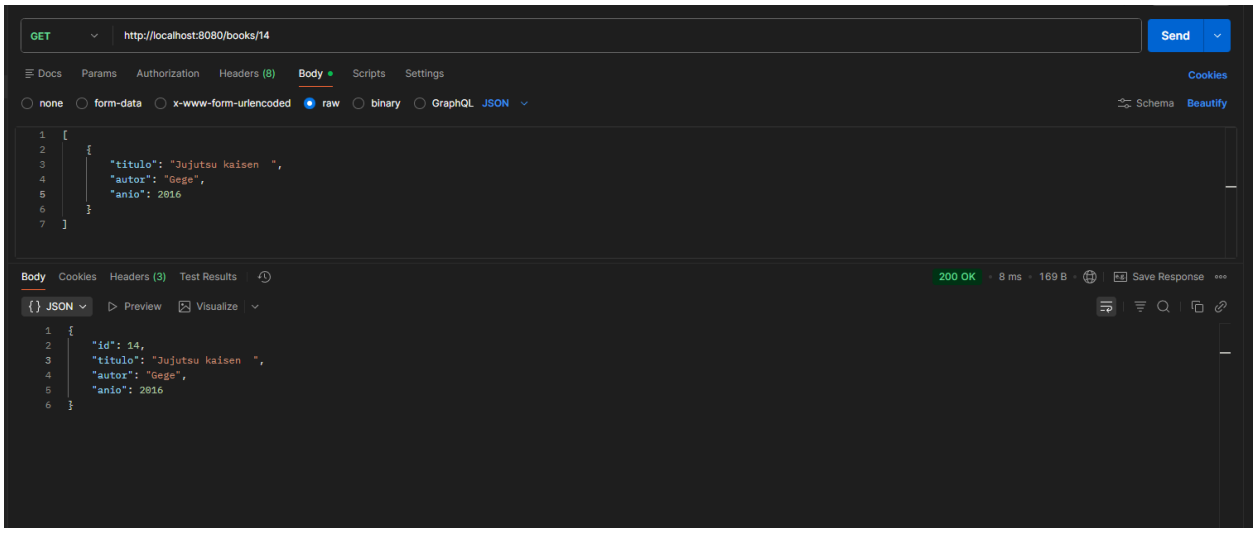


/books/{id}


GET
http://localhost:8080/books/9
Send
Docs
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Schema
Beautify
[
{
"titulo": "Libro ",
"autor": "autor",
"anio": 2004
}
]
Body
Cookies
Headers (3)
Test Results
200 OK
8 ms
172 B
Save Response
{ } JSON
Preview
Visualize
{
"id": 9,
"titulo": "Juego de tronos",
"autor": "Cervantes",
"anio": 1605
}

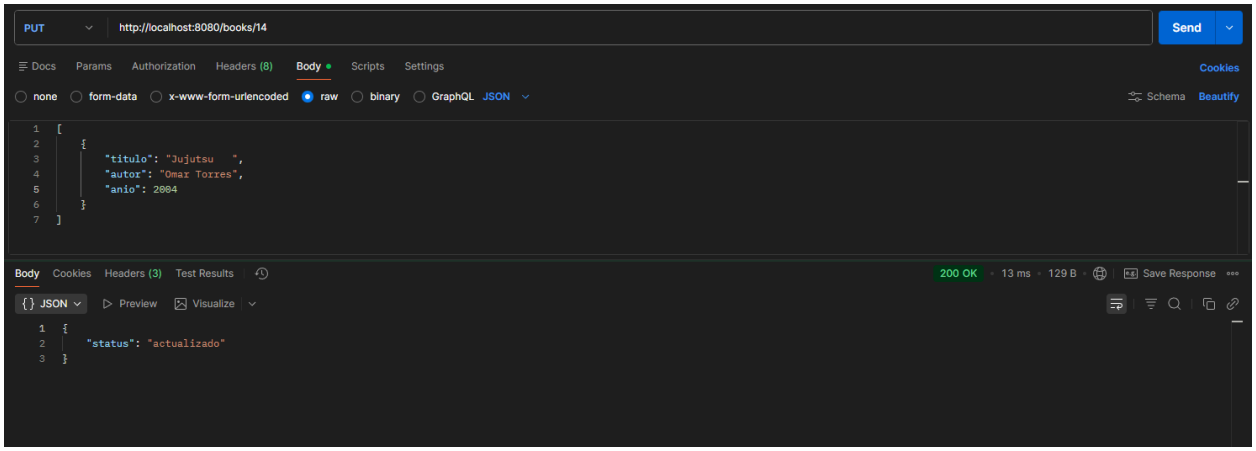
Post

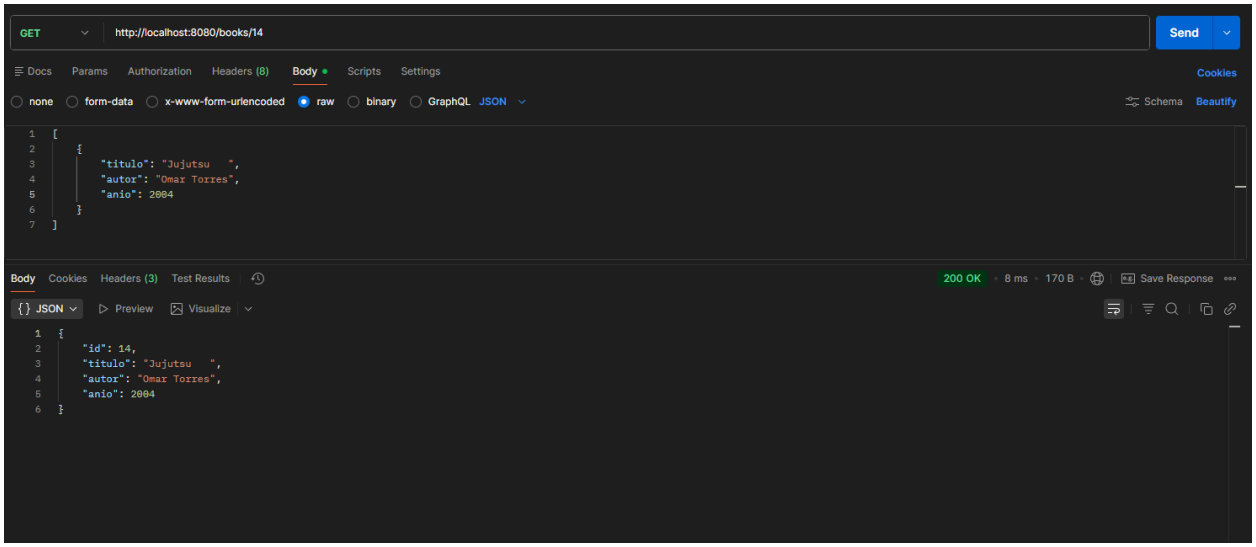

POST
http://localhost:8080/books
Send
Docs Params Authorization Headers (8) Body Scripts Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Schema Beautify
[
{
"titulo": "Jujutsu kaisen ",
"autor": "Gege",
"anio": 2016
}
]
Body Cookies Headers (3) Test Results
201 Created 132 ms 129 B Save Response
JSON Preview Visualize
{
"status": "creado"
}


GET
http://localhost:8080/books/14
Send
Docs Params Authorization Headers (8) Body Scripts Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Schema Beautify
[
{
"titulo": "Jujutsu kaisen ",
"autor": "Gege",
"anio": 2016
}
]
Body Cookies Headers (3) Test Results
200 OK 8 ms 169 B Save Response
JSON Preview Visualize
{
"id": 14,
"titulo": "Jujutsu kaisen ",
"autor": "Gege",
"anio": 2016
}

PUT

Voy a modificar el que cree en el post anterior, el libro jujutsu kaisen con id 14 y le pondre jujutsu a secas, le cambio el autor por omar torres y la fecha por 2004





Delete

Voy a borrar el libro de jujutsu