**Sesión 9: “Preparando el backend - creando servicios” - Java**

* ¿Para que los servicios?
* Haciendo servicios
* Estructuras de datos

**Objetivos de la sesión:**

* Comprender como los servicios REST ayudan a la comunicación ente componentes.
* Manejar tipos de datos JSON

**Desarrollo de la sesión**

* Comprender como los servicios REST ayudan a la comunicación ente componentes.

Transferencia de estado representacional (en inglés representational state transfer) o REST es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.

Si bien el término REST se refería originalmente a un conjunto de principios de arquitectura —descritos más abajo—, en la actualidad se usa en el sentido más amplio para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc) sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP.

REST es una forma sencilla de organizar interacciones entre sistemas independientes. En teoría, REST no está vinculado a la web, pero casi siempre se implementa como tal, y se inspiró en HTTP. Como resultado, REST puede usarse dondequiera que HTTP pueda.

Debido a que REST se ha inspirado en HTTP y juega a sus fortalezas, es la mejor manera de aprender cómo funciona HTTP.

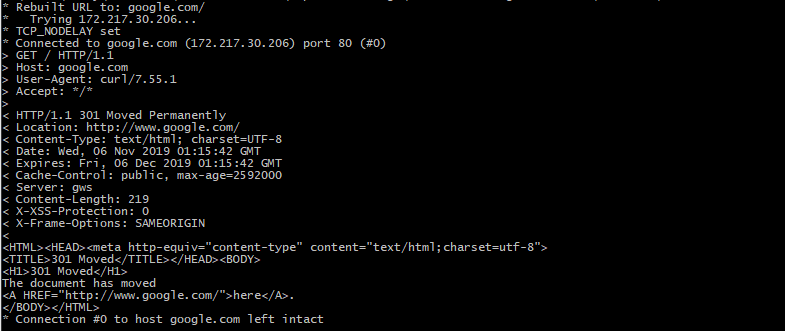
El Protocolo de transferencia de hipertexto (en inglés: Hypertext Transfer Protocol o HTTP) es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.

HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse. HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores. HTTP es el protocolo que permite enviar documentos de un lado a otro en la web. Un protocolo es un conjunto de reglas que determina qué mensajes se pueden intercambiar y qué mensajes son respuestas apropiadas a otros.

En HTTP, hay dos funciones diferentes: servidor y cliente. En general, el cliente siempre inicia la conversación; El servidor responde. HTTP está basado en texto; Es decir, los mensajes son esencialmente bits de texto, aunque el cuerpo del mensaje también puede contener otros medios. El uso del texto facilita el monitoreo de un intercambio HTTP.

Los mensajes HTTP se hacen de un encabezado y un cuerpo. El cuerpo a menudo puede permanecer vacío; Contiene los datos que desea transmitir a través de la red, con el fin de utilizarlo de acuerdo con las instrucciones en el encabezado. El encabezado contiene metadatos, como la información de codificación; Pero, en el caso de una solicitud, también contiene los métodos HTTP importantes. En el estilo REST, encontrará que los datos de cabecera suelen ser más significativos que el cuerpo.

Ejecutar en la consolad de comandos “curl -v google.com”



URL Un localizador de recursos uniforme (más conocido por las siglas URL, del inglés Uniform Resource Locator)1​ es un Identificador de recursos uniforme (Uniform Resource Identifier, URI) cuyos recursos referidos pueden cambiar, esto es, la dirección puede apuntar a recursos variables en el tiempo. Están formados por una secuencia de caracteres de acuerdo a un formato modélico y estándar que designa recursos en una red como, por ejemplo, Internet.

El término host o anfitrión se usa en informática para referirse a las computadoras u otros dispositivos (tablets, móviles, portátiles) conectados a una red que proveen y utilizan servicios de ella.

Métodos de petición HTTP

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque estos también pueden ser sustantivos, estos métodos de solicitud a veces son llamados HTTP verbs. Cada uno de ellos implementan una semántica diferente, pero algunas características similares son compartidas por un grupo de ellos: ej. un request method puede ser safe, idempotent, o cacheable.

GET

El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.

HEAD

El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.

POST

El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

PUT

El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

DELETE

El método DELETE borra un recurso en específico.

CONNECT

El método CONNECT establece un túnel hacia el servidor identificado por el recurso.

OPTIONS

El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.

TRACE

El método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.

PATCH

El método PATCH es utilizado para aplicar modificaciones parciales a un recurso.

Cada solicitud especifica un cierto verbo o método HTTP, en el encabezado de la solicitud. Los verbos HTTP le indican al servidor qué hacer con los datos identificados por la URL.

Códigos de respuesta

Estos son algunos códigos de respuesta HTTP, que a menudo se utilizan con REST:

200 OK

Este código de respuesta indica que la solicitud se ha realizado correctamente.

201 Created

Esto indica que la solicitud tuvo éxito y se creó un recurso. Se utiliza para confirmar el éxito de una solicitud PUT o POST.

400 Bad Request

La solicitud fue malformada. Esto sucede especialmente con las solicitudes POST y PUT, cuando los datos no pasan la validación o están en el formato incorrecto.

404 Not Found

Esta respuesta indica que no se pudo encontrar el recurso necesario. Esto generalmente se devuelve a todas las solicitudes que apuntan a una URL sin recurso correspondiente.

401 Unauthorized

Este error indica que debe realizar la autenticación antes de acceder al recurso.

405 Method Not Allowed

El método HTTP utilizado no es compatible con el de este recurso.

409 Conflict

Esto indica un conflicto. Por ejemplo, está utilizando una solicitud PUT para crear el mismo recurso dos veces.

500 Internal Server Error

Cuando todo lo demás falla; En general, se utiliza una respuesta 500 cuando el procesamiento falla debido a circunstancias imprevistas en el lado del servidor, lo que provoca el error del servidor.

JSON (acrónimo de JavaScript Object Notation, «notación de objeto de JavaScript») es un formato de texto sencillo para el [intercambio de datos](https://es.wikipedia.org/w/index.php?title=Intercambio_de_datos&action=edit&redlink=1). Se trata de un subconjunto de la notación literal de [objetos](https://es.wikipedia.org/wiki/Objeto_(programaci%C3%B3n)) de [JavaScript](https://es.wikipedia.org/wiki/JavaScript), aunque, debido a su amplia adopción como alternativa a [XML](https://es.wikipedia.org/wiki/XML), se considera (año [2019](https://es.wikipedia.org/wiki/2019)) un formato independiente del lenguaje.

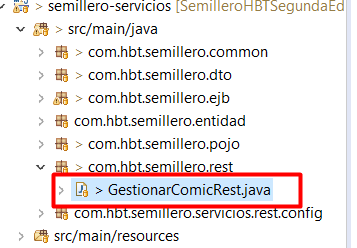
Creando los servicios rest

El proyecto previamente ya esta configurado para usar javax.ws.rs que es la librería JAX-RS o Java API for RESTful Web Services es una API del lenguaje de programación Java que proporciona soporte en la creación de servicios web de acuerdo con el estilo arquitectónico Representational State Transfer (REST).1​ JAX-RS usa anotaciones, introducidas en Java SE 5, para simplificar el desarrollo y despliegue de los clientes y puntos finales de los servicios web.

La clase JaxRSActivator contiene la anotación que determina la ruta de la aplicación a la cual se puede acceder esto identifica la ruta de la aplicación que sirve como URI base para todos los URI de recursos proporcionados por javax.ws.rs.Path. Solo se puede aplicar a una subclase de javax.ws.rs.core.Application.

Identificador uniforme de recursos (URI): una secuencia de caracteres que permite la identificación completa de cualquier recurso abstracto o físico. Localizador uniforme de recursos (URL): un subconjunto de URI que, además de identificar dónde está disponible un recurso, describe el mecanismo principal para acceder a él.

Crear el paquete “com.hbt.semillero.rest” y posteriormente crear la clase GestionarComicRest.java,



¿Qué se requieren para que la clase se pueda determinar cómo REST?

Se requieren una serie de anotaciones de la librería javax.ws.rs los cuales le dan mas significado a la misma JAX-RS proporciona algunas anotaciones para ayudar a mapear una clase recurso (un POJO) como un recurso web. Entre estas anotaciones se incluyen:

@Path especifica la ruta de acceso relativa para una clase recurso o método.

@GET, @PUT, @POST, @DELETE y @HEAD especifican el tipo de petición HTTP de un recurso.

@Produces especifica los tipos de medios MIME de respuesta.

@Consumes especifica los tipos de medios de petición aceptados.

Además, proporciona anotaciones adicionales para los parámetros de método para extraer información de la solicitud. Todas las anotaciones @\*Param toman una clave de alguna forma que se utiliza para buscar el valor requerido.

@PathParam enlaza el parámetro a un segmento de ruta.

@QueryParam enlaza el parámetro al valor de un parámetro de consulta HTTP.

@MatrixParam enlaza el parámetro al valor de un parámetro de matriz de HTTP.

@HeaderParam enlaza el parámetro a un valor de cabecera HTTP.

@CookieParam enlaza el parámetro a un valor de cookie.

@FormParam enlaza el parámetro a un valor de formulario.

@DefaultValue especifica un valor por defecto para los enlaces anteriores cuando la clave no es encontrada.

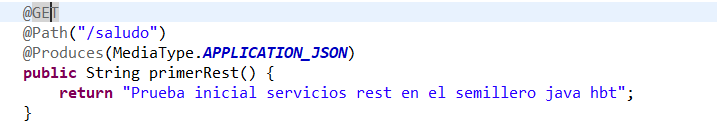
@Context devuelve todo el contexto del objeto. (Por ejemplo: @Context HttpServletRequest request)

Teniendo en claro lo anterior inicialmente el POJO debe definir la ruta de acceso definida y esto se realiza anotando la clase con

@Path("/GestionarComic")

public class GestionarComicRest {}

¿Como se puede acceder el primer método de prueba?

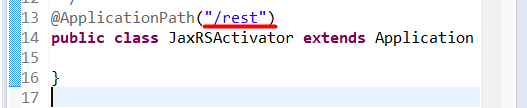


¿Cómo se puede acceder al nuevo servicio?

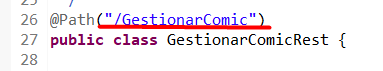
Primero se debe saber el contexto del modulo de servicios, esto se define en el pom.xml del ear



Luego se toma de la clase JaxRSActivator el @ApplicationPath



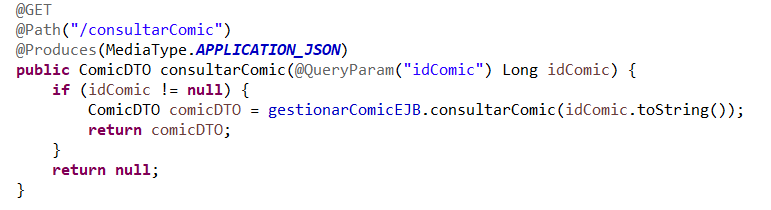
Después se toma el path del pojo



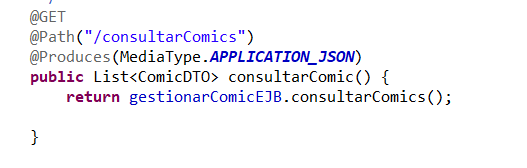
Y por ultimo el path del método, quedando la ruta de la siguiente manera:

http://localhost:8085/semillero-servicios/rest/GestionarComic/saludo

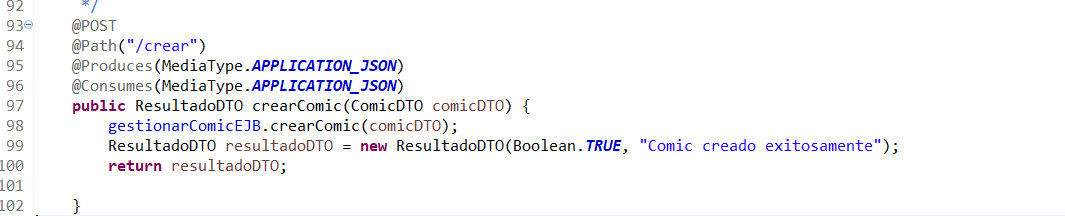
¿Cómo se puede crear un servicio que dado un idComic retorne toda la información del comic?



¿Cómo se puede crear un servicio que retorne toda la información de los comics?



¿Cómo se puede crear un servicio que permita persistir un comic?



¿Cómo se puede crear un servicio que permita modificar un comic?

Subiendo el servidor

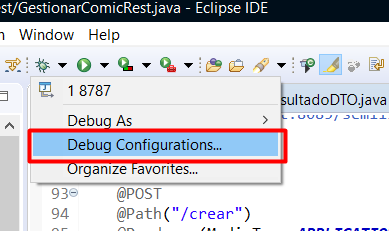
Como activar el debug en el archivo wildfly-15.0.1.Final\bin\standalone.conf.bat

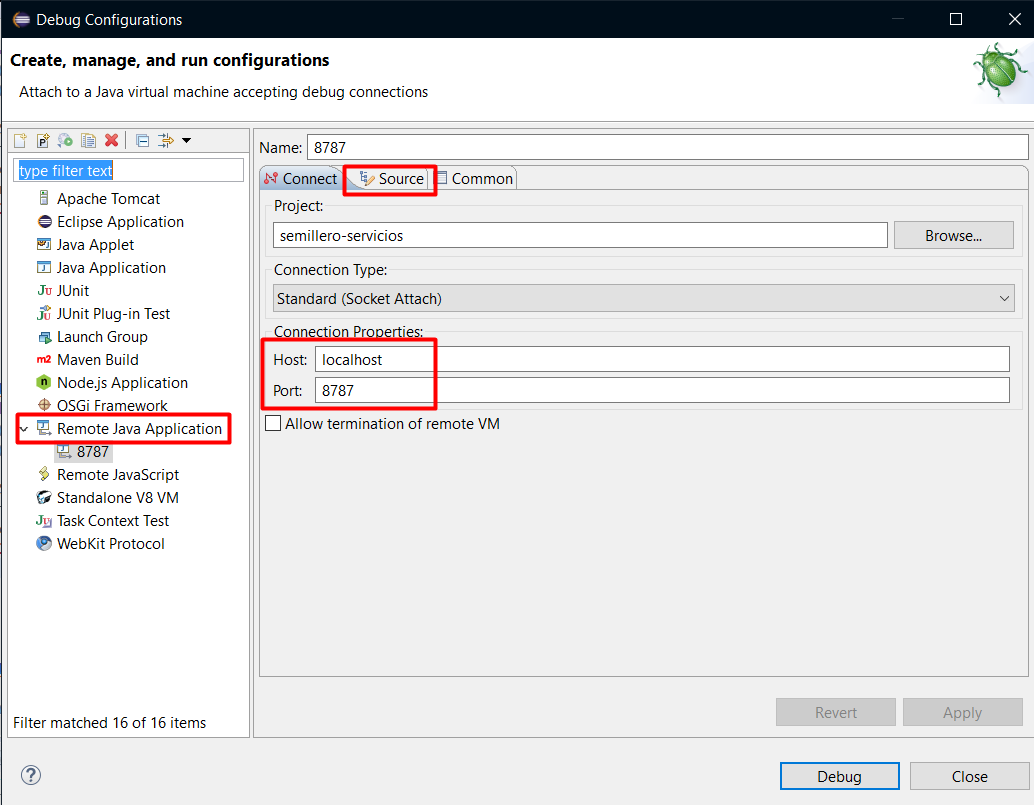
Se debe descomentar la línea (quitar la palabra “rem”)

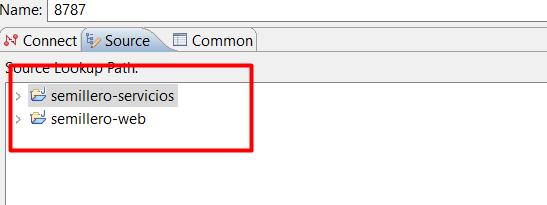
rem set "JAVA\_OPTS=%JAVA\_OPTS% -agentlib:jdwp=transport=dt\_socket,address=8787,server=y,suspend=n"



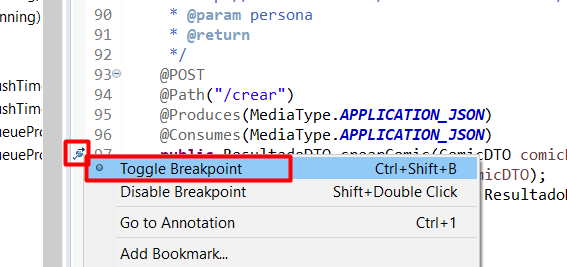
Después de que arranque el servidor se debe configurar el debug en el IDE en este caso eclipse



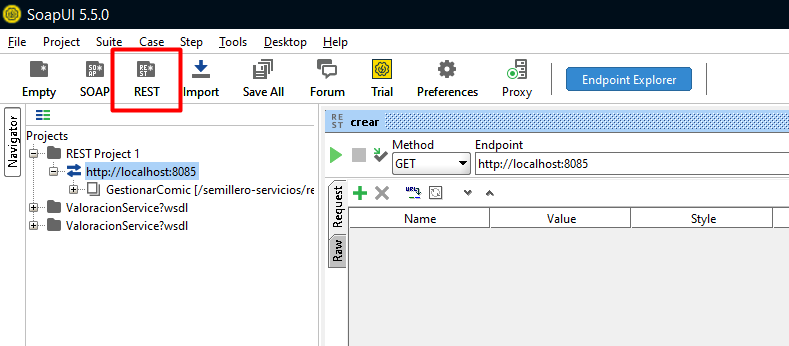


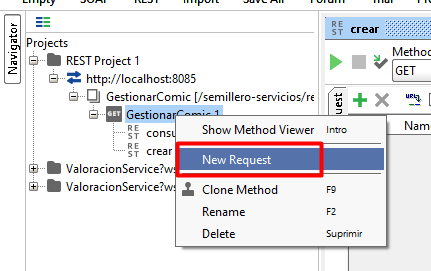


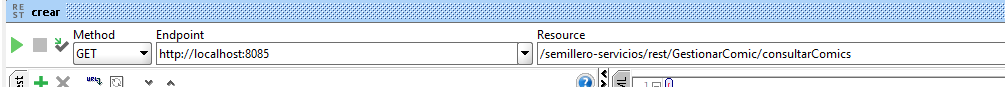
Luego se debe agregar los puntos en los cuales queremos parar la ejecución de la aplicacion



Como se puede crear la conexión entre el soap y la aplicación







Depurando errores….

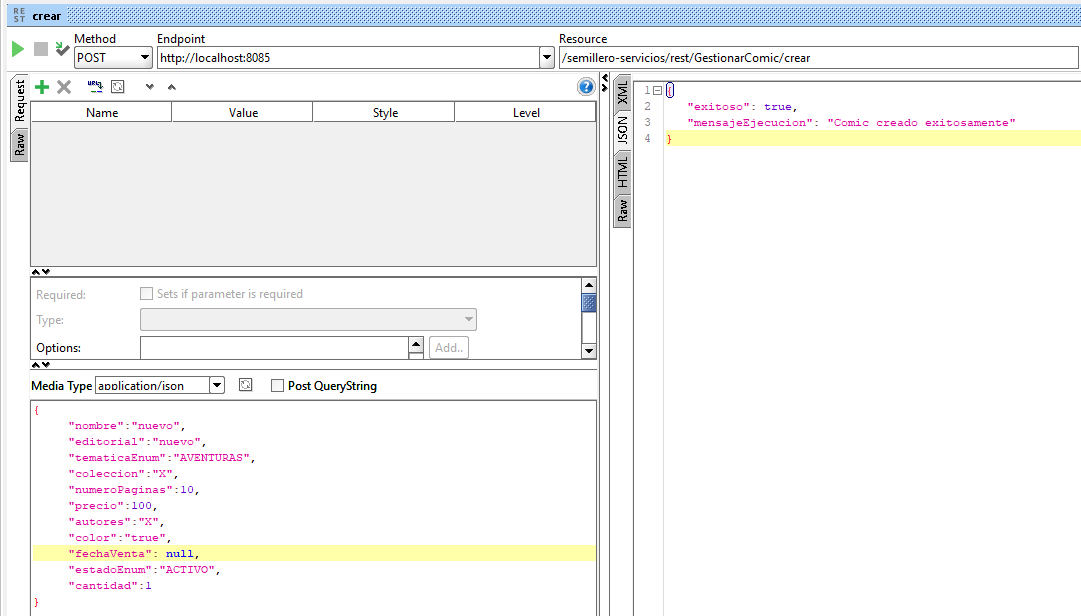
GestionarComicBean.java

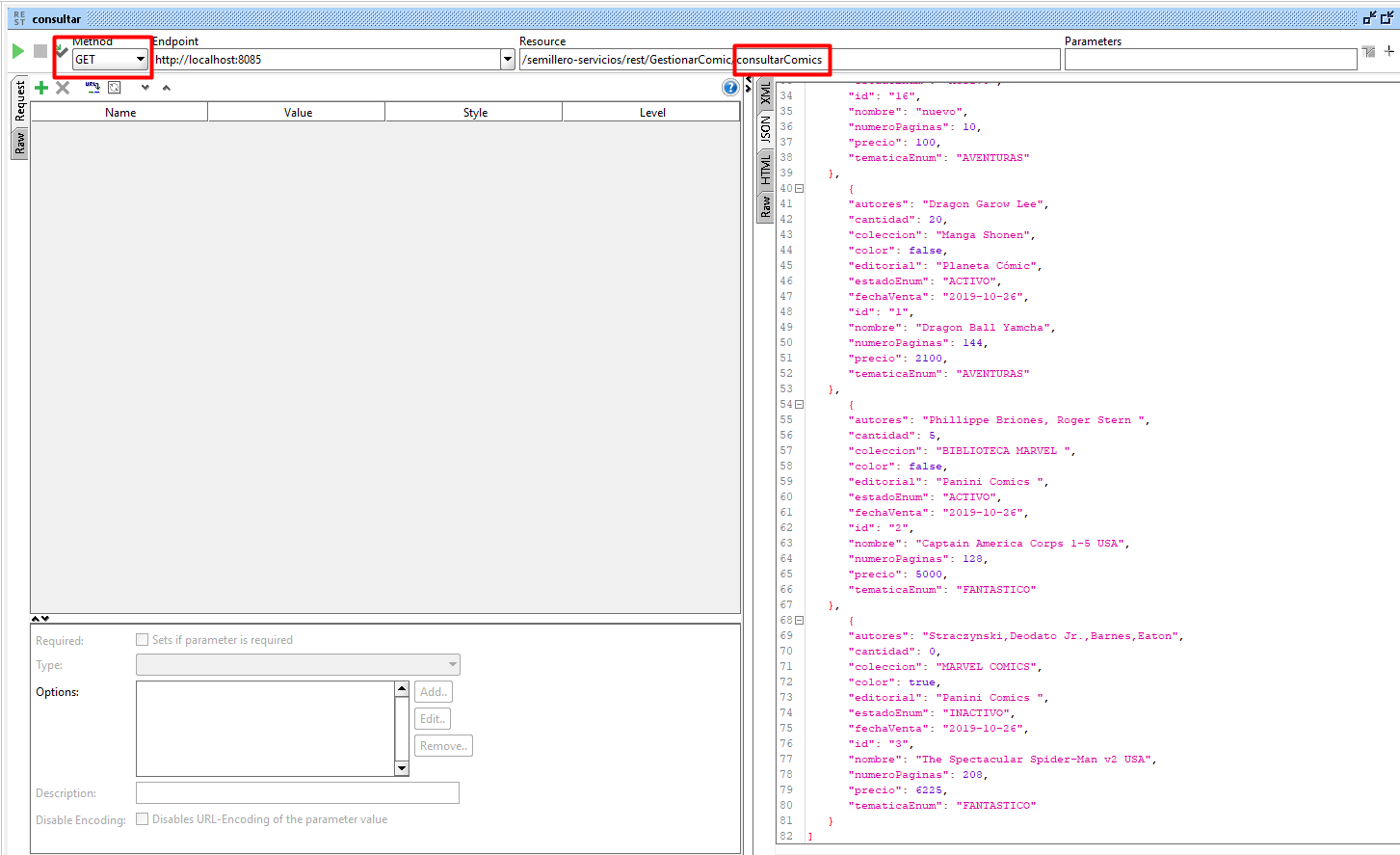
Aspectos para tener en cuenta id en comic.java debe ser Long

Crear dos métodos para convertir información de ComicDTO.java a Comic.java

Convertir de Long a String el id

Resultado esperado





**Ejercicio sesión 6**

* Realizar el servicio que dado un idComic permite llamar la lógica de negocio que permite quitar un registro comic de la BD, hacer la comunicación con el soapUI

**Conceptos adicionales**

**Enlaces de interés**