Un Servicio web RESTful en Java con Apache Tomcat 8 y Jersey 2 Jax-RS. Retornando XML, JSON y HTML

Los servicios *RESTful* son una práctica forma de desarrollar *servicios web* que funcionan sobre el protocolo *HTTP*, facilitando la comunicación e invocación de estos. Los servicios RESTful reciben peticiones por medio de direcciones URL (o URIs) y retornan una respuesta particular en algún formato adecuado (XML, JSON, HTML, texto plano, etc.).

En esta sección desarrollaremos un servicio web RESTful en Java usando la librería *Jersey* la cual es una implementación de *JAX-RS*. Nuestro servicio será muy simple, recibirá una petición a una dirección específica y retornará un mensaje en texto en algún formato (según la petición), será como un Hola Mundo en RESTful, pero con más detalles y adiciones :D .

Nuestro servicio lo desplegaremos en un servidor *Apache Tomcat* en su versión 8 que podremos descargar del enlace (apache-tomcat), usaremos la librería Jersey en su versión 2.14 la cual implementa JAX-RS2.0 que podemos descargar en (Librería Jersey). Del .zip de Jersey, tomaremos todos los .jar que están en cada una de las carpetas (api, ext y lib).

La estructura de carpetas para nuestro proyecto será la siguiente (explicación paso a paso más abajo):

```
-> Disco local C: -> servidor -> webapps -> RESTful -> WEB-INF -> lib -> *.jar -> classes -> com -> app -> ws -> WebService.java -> web.xml
```

Estructura de carpetas para el servicio web RESTful con Java y Apache Tomcat

Una vez descargado *apache tomcat* lo podremos descomprimir en una carpeta llamada *"servidor"* en nuestro disco C:/ para acceder a él fácilmente, dentro de la carpeta servidor encontraremos otra carpeta llamada *webapps*, al interior de ésta crearemos una nueva carpeta llamada *RESTful* (fijarse en las mayúsculas y minúsculas) y dentro de ésta crearemos otra llamada *WEB-INF* (debe estar en mayúsculas), esta carpeta WEB-INF tendrá dentro de sí un archivo llamado *web.xml*, una carpeta llamada *lib* y otra llamada *classes*. Dentro de la carpeta lib pondremos todos los archivos .jar de .zip de jersey. Dentro de la carpeta classes pondremos la estructura del paquete de nuestro servicio web que será com.aap.ws, quiere decir que la carpeta ws contendrá el archivo .java del servicio web que llamaremos *WebService.java*.

Ahora veamos el contenido de los archivos WebService.java y web.xml

El archivo WebService.java

El archivo *WebService.java* será el código de funcionamiento del servicio web, será el que indique qué hacer al recibir peticiones por algún tipo de mensaje HTTP (*put*, *get*, *delete*, *post*, etc.).

```
package com.app.ws; //Esta es la estructura de paquete creada import javax.ws.rs.*; //Importamos la librería para manejar RESTful @Path("getMessage/{type}") //Especificamos una ruta que se debe usar para invocar este método y un parámetro (tipo) public class WebService { @GET //Indicamos que este método se ejecutará al recibir una petición por
get @Produces({"text/plain", "text/html","text/xml", "application/json"})//Indicamos que el tipo de salida es texto plano, XML, HTML o JSON public StringmostrarMensaje(@PathParam("type") String tipo)//Método que recibe como parametro el valor de type en la
URL {if(tipo.equalsIgnoreCase("texto")) { return "Éste es mi primer servicio RESTful con Java";
```

```
} else if(tipo.equalsIgnoreCase("html")) { return "<html lang='es'><head><meta charset='UTF-8'/>
<title>WS</title></head><body><h1>Éste es mi primer servicio RESTful con Java</h1></body></html>";
} elseif(tipo.equalsIgnoreCase("xml")) { return "<?xml version='1.0' encoding='UTF-8'?><root>
<value>Éste es mi primer servicio RESTful con Java</value></root>";
} else if(tipo.equalsIgnoreCase("json")) { return "{\"root\":{\"value\":\"Éste es mi primer servicio
RESTful con Java\"}}"; } else { return "Tipo no soportado"; } } }
```

Tenemos entonces en el código anterior un servicio web que funciona en la ruta *getMessage/type* donde type puede ser cualquier cosa, pero solo se aceptará texto, HTML, XML y JSON si es diferente a estos, se retorna un mensaje en texto plano diciendo "Tipo no soportado". Hay que notar que únicamente tenemos un método el cual puede ser llamado usando el método get de HTTP, es decir que lo podremos invocar incluso desde el navegador web. Este método al ser el único posee múltiples tipos *MIME* de respuestas (para XML, HTML, JSON y texto plano) que se especificaron en un pequeño arreglo en el marcador *@Produces*. Cabe resaltar que el valor del parámetro enviado por URL se obtiene con la anotación @PathParam("type"), nótese que debe coincidir con lo que se puso en @Path.

Muy bien ahora lo que haremos será compilar nuestro WebService.java. Debido a que no estamos usando ningún entorno de desarrollo usaremos el comando *javac* desde la consola

```
cd C:/servidor/webapps/RESTful/WEB-INF #Nos desplazamos a la carpeta WEB-INF de nuestro servicio
web javac -classpath lib/* classes/com/app/ws/*.java #Compilamos nuestro .java usando todos los .jar
de la carpeta lib
```

Al ejecutar esto desde la consola tendremos junto al archivo WebService.java un archivo llamado WebService.class.

Nota: Si obtienes algún problema con la compilación, verifica haber copiado bien el código de arriba y haberlo colocado en las carpetas que corresponden según el paquete.

El archivo web.xml

```
<?xml version="1.0" encoding="UTF-8"?> <web-app xmlns="http://java.sun.com/xml/ns/
javaee" version="3.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"> <servlet>
<servlet-name>RESTful</servlet-name> <servlet-class>org.glassfish.jersey.servlet.ServletContainer
</servlet-class> <init-param> <param-name>jersey.config.server.provider.packages</param-name> <param-
value>com.app.ws</param-value> </init-param> <load-on-startup>1</load-on-startup> </servlet>
<servlet-mapping> <servlet-name>RESTful</servlet-name> <url-pattern>/*</url-pattern> </servlet-
mapping> </web-app>
```

Este archivo aunque pequeño es totalmente indispensable para el correcto funcionamiento y despliegue de nuestro servicio web. No profundizaré mucho respecto al archivo *web.xml* como tal, solo mencionaré que el valor de <param-value> en <init-param> debe coincidir con la ruta del paquete donde se encuentra el archivo .java y que <url-pattern> especifica la ruta donde se "ubicará"" inicialmente el servicio web para ser invocado según el valor de la etiqueta *@Path*. Para este caso al poner /* indicamos que no es necesario añadir nada a la ruta y que después del slash (/) puede ir cualquier cosa; esto quiere decir que una ruta para invocar nuestro servicio web puede ser **http://localhost:8080/RESTful/getMessage/xml/**, si pusiéramos que el valor de <url-pattern> fuera /servicios/* entonces para invocar el servicio ya deberíamos usar la URL **http://localhost:8080/RESTful/servicios/getMessage/xml/**.

Una vez ya tenemos nuestro archivo WebService.java compilado y hemos creado nuestro archivo web.xml y todos los archivos, librerías, carpetas y demás se encuentran en la ruta correcta, solo nos queda arrancar el servidor Tomcat e invocar nuestro servicio web (usando el navegador por ejemplo).

Para correr el servidor, también lo podemos hacer desde consola o iniciando directamente el archivo startup.bat en windows o startup.sh en Linux que se encuentran en la carpeta bin de Tomcat, veámoslo desde consola:

```
cd C:/servidor/bin #Vamos a la carpeta bin del servidor startup
```

Si todo ha ido bien, podemos ingresar a la URL **http://localhost:8080/RESTful/getMessage/xml/** y obtendremos un mensaje que dice "Éste es mi primer servicio RESTful con Java", si miramos el código fuente de esa página podremos ver que el servicio escribió <?xml version='1.0' encoding='UTF-8'?><root><value>Éste es mi primer servicio RESTful con Java</value></root>

Si ingresamos a la dirección **http://localhost:8080/RESTful/getMessage/json/** obtendremos como respuesta lo siguiente: {"root":{"value":"Ã‰ste es mi primer servicio RESTful con Java"}} Notemos que la E tildada se ve mal puesto que el archivo JSON no posee formato a diferencia del XML y el HTML.

Si ingresamos por ejemplo a **http://localhost:8080/RESTful/getMessage/php/** obtendremos como respuesta: "Tipo no soportado" puesto que PHP no hace parte de nuestros tipos MIME considerados.