



Práctica 7

Servicios Web REST

(No evaluable)

Guadalupe Ortiz Bellot
Programación Paralela y Distribuida
Curso 2016-2017

SERVICIOS WEB REST

► Contenido

- Conceptos básicos de REST
- Restful Hello World
- Familiarízate con las etiquetas de Rest y pruébalas
- Invocar un servicio remoto con Jersey
- Instrucciones de entrega de la práctica 7
- Anexo 1. Cómo evitar el mensaje de error al desplegar
- Anexo 2. Libros y referencias de consulta

SERVICIOS WEB REST

► Contenido

- Conceptos básicos de REST
- Restful Hello World
- Familiarízate con las etiquetas de Rest y pruébalas
- Invocar un servicio remoto con Jersey
- Instrucciones de entrega de la práctica 7
- Anexo 1. Cómo evitar el mensaje de error al desplegar
- Anexo 2. Libros y referencias de consulta

SERVICIOS WEB REST

➤ Contenido

- **Conceptos básicos de REST**

- **JAX-RS**

- **Ejemplo**

- **Formatos de transferencia de datos**

- Familiarízate con las etiquetas de Rest y pruébalas
- Restful Hello World
- Invocar un servicio remoto con Jersey
- Instrucciones de entrega de la práctica 7
- Anexo 1. Cómo evitar el mensaje de error al desplegar
- Anexo 2. Libros y referencias de consulta

CONCEPTOS BÁSICOS DE REST

➡ JAX-RS (i)

- Mediante el uso de esta API: se puede marcar un POJO a través de anotaciones que permiten identificar:
 - ➡ un recurso como una URI
 - ➡ un conjunto de métodos bien definidos para acceder a los recursos (GET, POST, etc)
 - ➡ múltiples formatos de representación de un recurso

```
@GET
@Path("/saludo")
@Produces(MediaType.TEXT_PLAIN)
public String hola( ) { //... }
```

CONCEPTOS BÁSICOS DE REST

➡ JAX-RS (ii)

- En tiempo de ejecución, el entorno que implementa la especificación JAX - RS es responsable de la invocación de la aplicación Java correspondiente mediante el **mapeo de la solicitud HTTP** con uno de los **métodos Java** que satisface la petición.
- Hay que determinar la **clase y métodos** Java que representan el recurso, el **tipo de recurso** y el **método HTTP** de la invocación.

http://applicationName/saludo

GET
Texto Plano
public String hola()

CONCEPTOS BÁSICOS DE REST

➤ JAX-RS (iii)

- Ofrece soporte para el uso de los métodos estándar HTTP GET, POST, PUT, DELETE, HEAD y OPTIONS
 - **GET:** Recuperar un recurso
 - **POST:** Crear un recurso
 - **PUT:** Actualizar un recurso
 - **DELETE:** Borrar un recurso
 - **HEAD:** Como GET, pero no devuelve el cuerpo. Se utiliza para obtener metainformación acerca del recurso. Si no hay método marcado con HEAD, se hace un GET y se descarta el cuerpo.
 - **OPTIONS:** Pide a las opciones de comunicación disponibles. Si no hay ningún método marcado con @OPTIONS, se genera una respuesta automática.

CONCEPTOS BÁSICOS DE REST

► Ejemplo (i)

```
@Path("orders")
```

```
public class OrderResource {
```

```
@GET
```

```
public List<Order> getAll() {...}
```

```
@GET
```

```
@Path("{oid}")
```

```
public Order getOrder(@PathParam("oid") int id) {...}
```

```
@DELETE
```

```
@Path("{id}")
```

```
public void deleteXml(@PathParam("id") int id) {...}
```

```
@XmlRootElement
```

```
public class Order {...}
```

<http://localhost:8080/store/webresources/orders>
<http://localhost:8080/store/webresources/orders/1>

CONCEPTOS BÁSICOS DE REST

➡ Ejemplo (ii)

```
@GET
@Path("/{oid}")
@Produces({"application/xml", "application/json"})
public Order getOrder(@PathParam("oid") int id) { ...}
```

```
@POST
@Path("/create")
@Consumes("application/x-www-form-urlencoded")
public Order createOrder(@FormParam("id") int id,
    @FormParam("name")String name) { ...}
```

```
@PUT
@Path("/{id}")
@Consumes("*/xml")
public Order putXml(@PathParam("id") int id, String content) {...}
```

CONCEPTOS BÁSICOS DE REST

► Formatos de transferencia de datos (i)

- El cliente consulta o actualiza los recursos a través de la URI mediante el intercambio de representaciones del recurso.
- Dichas representaciones contienen información en formatos tales como HTML, XML o JavaScript Object Notation (JSON).
- El cliente debe conocer el tipo devuelto por el servicio.
- Por lo general, el cliente especifica la representación que quiere (Accept), y el servidor devuelve los recursos deseados en ese formato.
- Toda la información necesaria para procesar una petición de un recurso está contenida dentro de la propia solicitud, con lo que la interacción es sin estado.

CONCEPTOS BÁSICOS DE REST

► Formatos de transferencia de datos (ii)

Especificación en el servicio

- De forma predeterminada, un recurso REST se publica o se consume con el * / * tipo MIME.
- Un recurso REST puede restringir los tipos de medios admitidos por la petición y la respuesta con las anotaciones @Consumes and @ Produce, respectivamente.
- Estas anotaciones se pueden especificar en la clase o en los métodos. Si la anotación se especifica en el método anula la de la clase.

FORMATOS DE TRANSFERENCIA DE DATOS

➤ Formatos de transferencia de datos (iii)

Especificación en el cliente

- Content-Type: indica el tipo enviado (por ejemplo "text/plain", "text/xml", "text/html", "application/json", "application/atom+xml")
- Accept: indica el tipo de recurso que se espera recibir
- Con jquery:
 - contentType (envío)
 - dataType (espero recibir)

SERVICIOS WEB REST

► Contenido

- Conceptos básicos de REST
- **Restful Hello World**
 - **Ejercicio 1. Pasos a seguir**
 1. Creación de un proyecto web dinámico
 2. Inclusión de las librerías Jersey
 3. Creación del paquete y de la clase
 4. Creación del archivo web.xml
 5. Despliegue del servicio en el servidor
 6. Prueba desde un cliente de navegador
 - Familiarízate con las etiquetas de Rest y pruébalas
 - Invocar un servicio remoto con Jersey
 - Instrucciones de entrega de la práctica 7
 - Anexo 1. Cómo evitar el mensaje de error al desplegar
 - Anexo 2. Libros y referencias de consulta

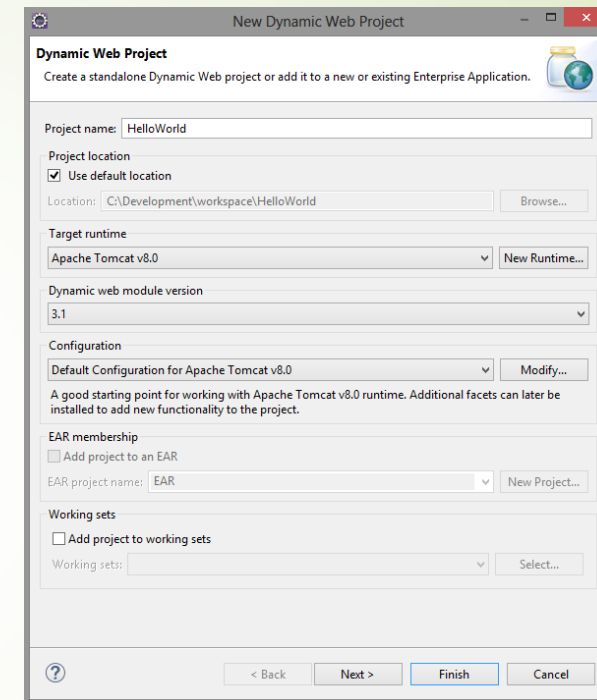
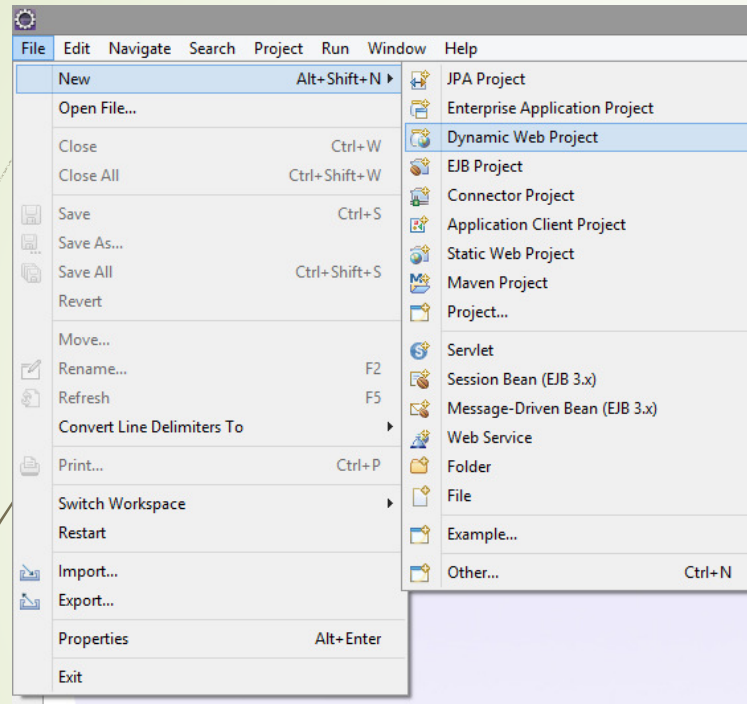
RESTFUL HELLO WORLD

► Ejercicio 1. Pasos a seguir:

1. Crear un Dynamic Web Project
2. Copiar las librerías de Jersey en WebContent/Web-Inf/Lib
3. Crear un paquete dentro de JavaResources/src y la clase Hello dentro del paquete.
4. Crear el archivo web.xml en WebContent/Web-Inf
5. Desplegar el servicio en el servidor (Run as->Run on Server)
6. Probar desde el cliente para Rest de Chrome (<https://chrome.google.com/webstore/detail/simple-restclient/fhjcajmcbbmldlhcmfajhfbgofnpcjmb>))

RESTFUL HELLO WORLD

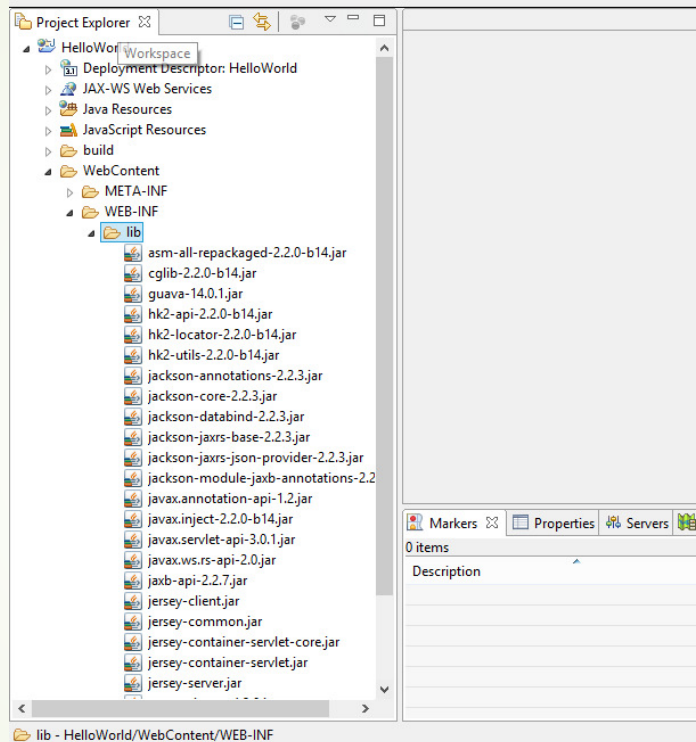
➤ 1. Creación de un proyecto web dinámico



- Creamos un Dynamic Web Project llamado HelloWorld:
 - New → Dynamic Web Project
 - Rellenar Project name y seleccionar el Tomcat que acabamos de crear en el campo Target runtime y → Finish
- **MUY IMPORTANTE: Respetar mayúsculas y minúsculas, especialmente las convenciones de Java para la inicial**

RESTFUL HELLO WORLD

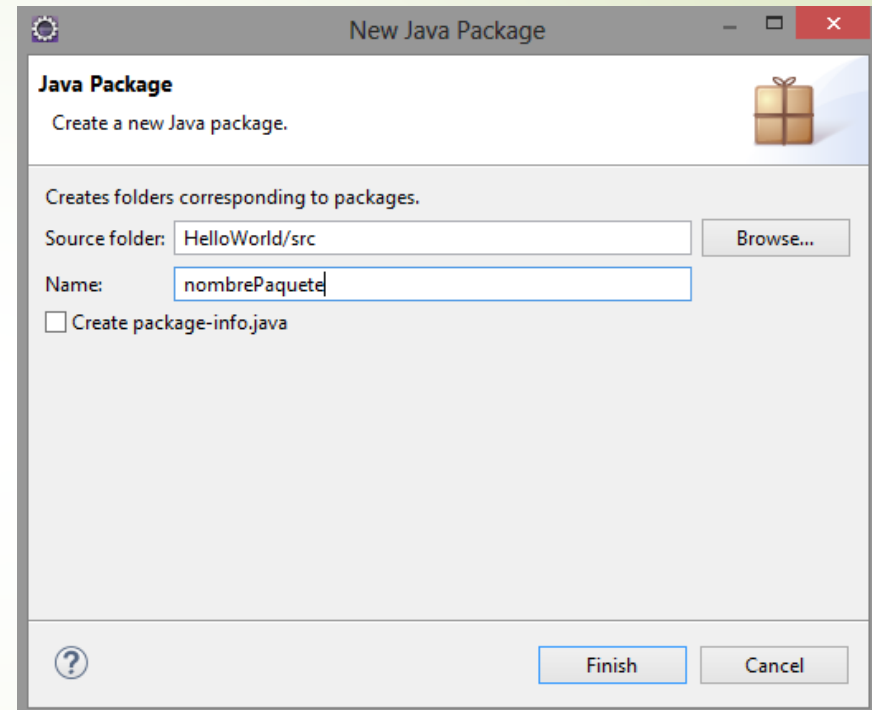
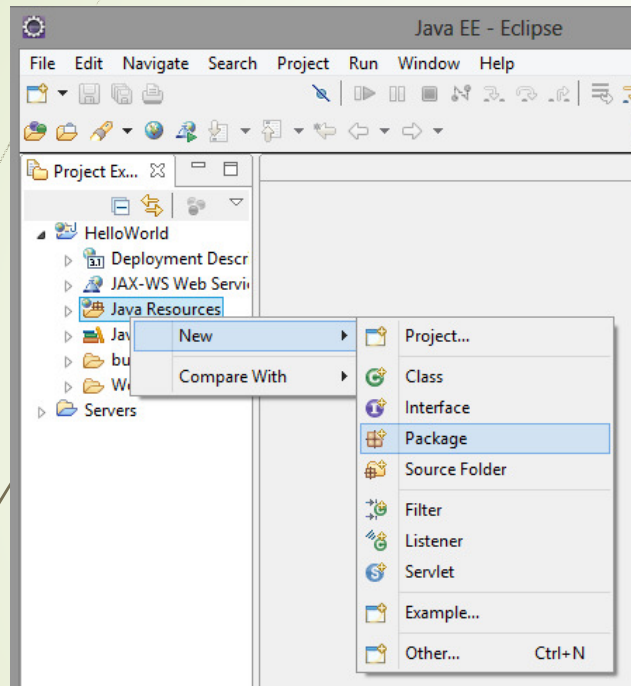
➡ 2. Inclusión de las librerías Jersey



- Copiamos las librerías de Jersey en WebContent/WEB-INF/lib (previamente las habremos descomprimido)
- Si no aparecen dentro de la carpeta después de pegarlas, refrescar el proyecto (Botón derecho sobre el proyecto → Refresh)

RESTFUL HELLO WORLD

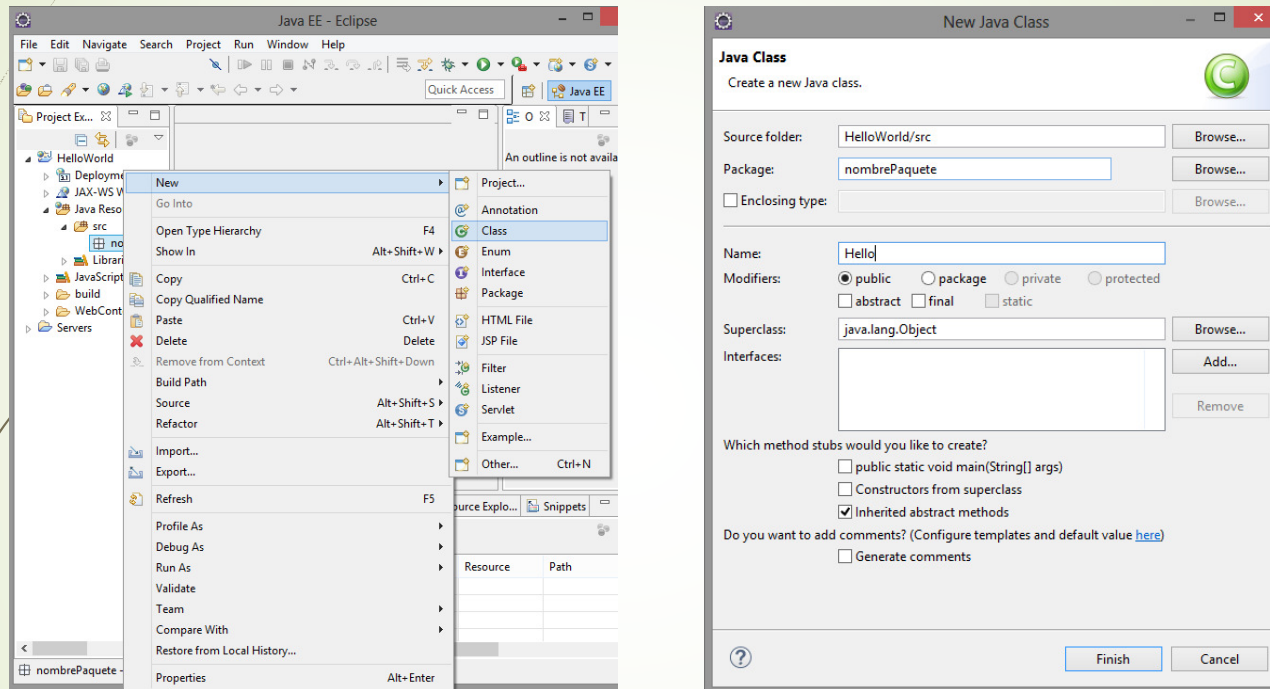
➡ 3. Creación del paquete y de la clase (i)



- Creamos un paquete **dentro de JavaResources/src** llamado **nombrePaquete**.
- **MUY IMPORTANTE:** Respetar mayúsculas y minúsculas, especialmente las convenciones de Java para la inicial

RESTFUL HELLO WORLD

➡ 3. Creación del paquete y de la clase (ii)



- Creamos la clase Hello e introducimos el código de la siguiente transparencia.
- **MUY IMPORTANTE: Respetar mayúsculas y minúsculas, especialmente las convenciones de Java para la inicial**

RESTFUL HELLO WORLD

➡ 3. Creación del paquete y de la clase (iii)

```
package nombrePaquete;
```

```
import javax.ws.rs.GET;
```

```
import javax.ws.rs.Path;
```

```
import javax.ws.rs.Produces;
```

```
import javax.ws.rs.core.MediaType;
```

```
@Path("/hello")
```

```
public class Hello {
```

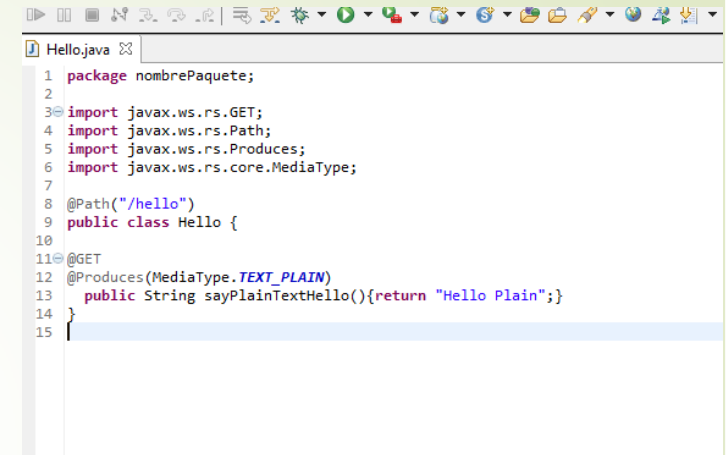
```
@GET
```

```
@Produces(MediaType.TEXT_PLAIN)
```

```
    public String sayPlainTextHello(){return "Hello Plain";}
```

```
}
```

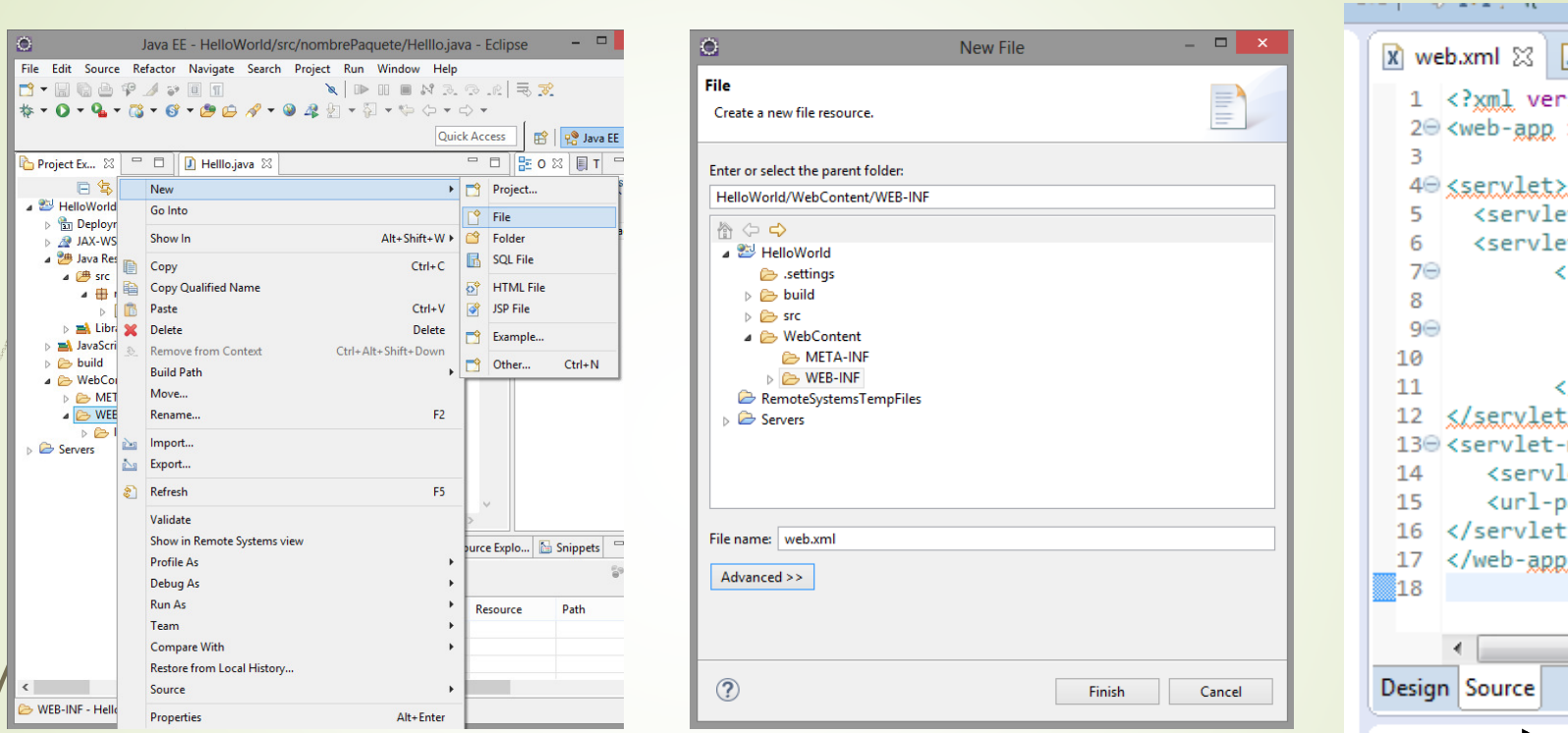
NOTA: No olvidéis salvar el archivo tras pegar o modificar el código.



```
1 package nombrePaquete;
2
3 import javax.ws.rs.GET;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.Produces;
6 import javax.ws.rs.core.MediaType;
7
8 @Path("/hello")
9 public class Hello {
10
11     @GET
12     @Produces(MediaType.TEXT_PLAIN)
13     public String sayPlainTextHello(){return "Hello Plain";}
14 }
15
```

RESTFUL HELLO WORLD

➡ 4. Creación del archivo web.xml (i)



- Creamos el archivo web.xml en WebContent/WEB-INF
- Se abre automáticamente con el editor XML. Pinchar en la pestaña inferior "Source", para verlo en modo texto
- Copiamos en él código de la siguiente transparencia y salvamos.

RESTFUL HELLO WORLD

► 4. Creación del archivo web.xml (ii)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
```

```
<servlet>
```

```
<servlet-name>Mi Servicio Hello REST</servlet-name>
```

```
<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
```

```
<init-param>
```

```
<param-name>jersey.config.server.provider.packages</param-name>
```

```
<param-value>
```

```
nombrePaquete, com.fasterxml.jackson.jaxrs.json</param-value>
```

```
</init-param>
```

```
</servlet>
```

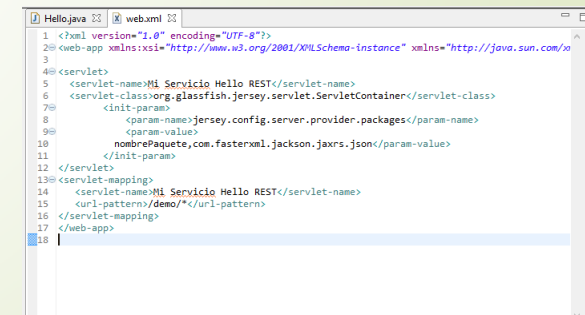
```
<servlet-mapping>
```

```
<servlet-name>Mi Servicio Hello REST</servlet-name>
```

```
<url-pattern>/demo/*</url-pattern>
```

```
</servlet-mapping>
```

```
</web-app>
```

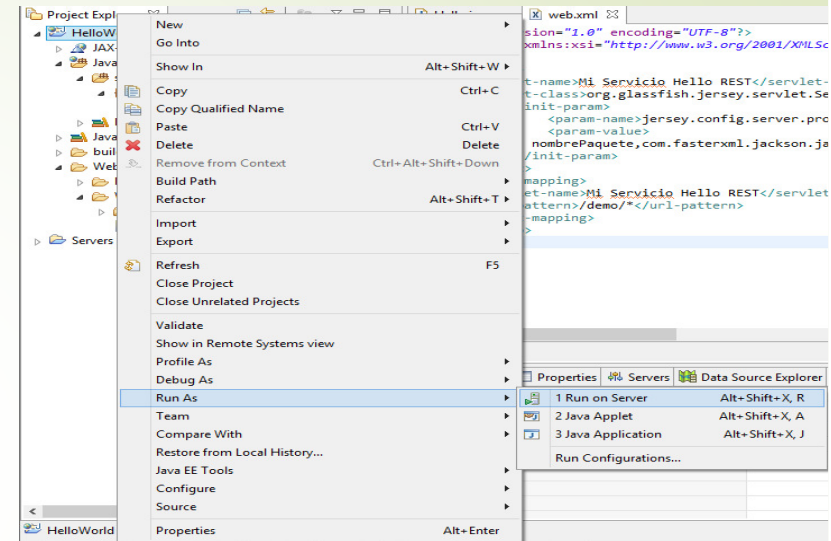


NOTA: dependiendo del sistema operativo las comillas puede que se copien mal. Si da error revisar las comillas.

RESTFUL HELLO WORLD

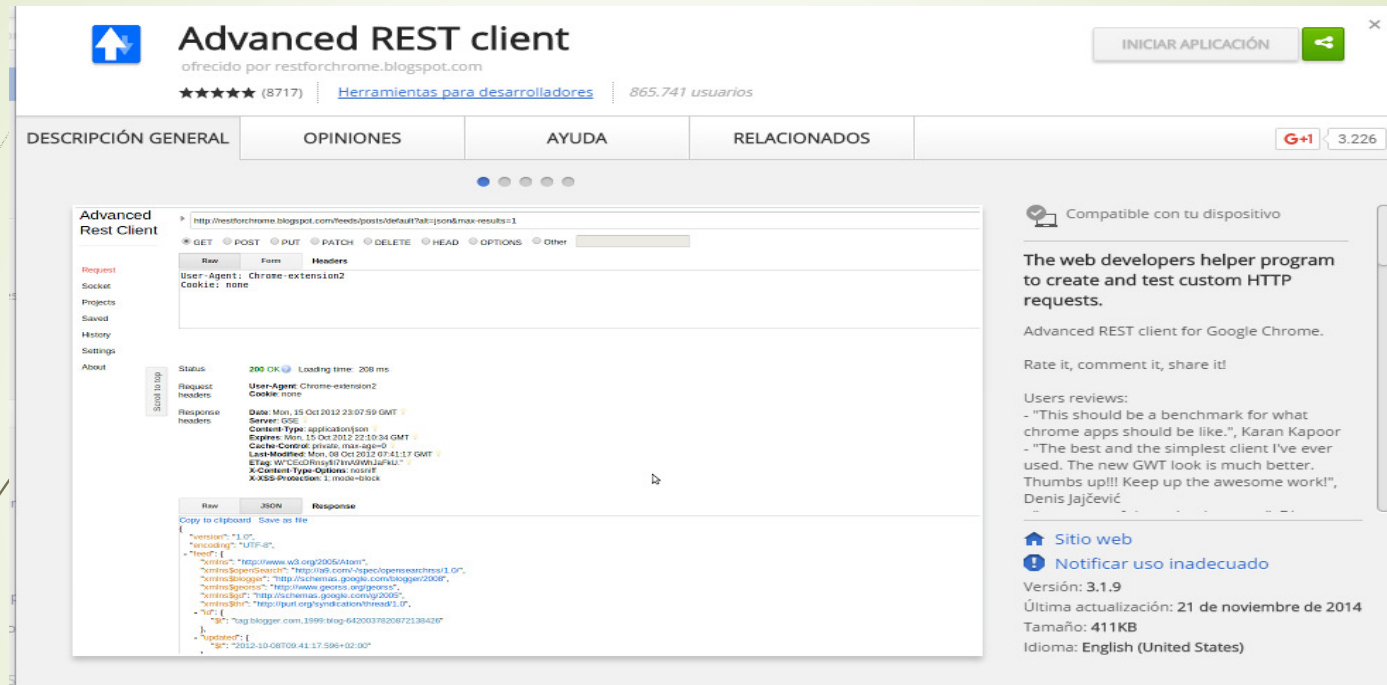
► 5. Despliegue del servicio en el servidor

- Desplegamos en el servidor Tomcat que hemos creado anteriormente: Botón derecho sobre el proyecto → Run as → Run on Server → Elegimos el servidor creado → Finish
- Preguntará si queremos reiniciar el servidor, le damos a OK
- Si todo ha ido bien al desplegar el contenido del servidor, saldrá debajo el servicio sincronizado.
- Si no habría que pinchar en la pestaña de la consola para ver el error.
- No hay que preocuparse si en el navegador de Eclipse sale Error 404.
- NOTA: Para ver la ventana de consola, problemas o log de errores: Window → Show View → General → la opción correspondiente



RESTFUL HELLO WORLD

➡ 6. Prueba desde el cliente de un navegador (i)

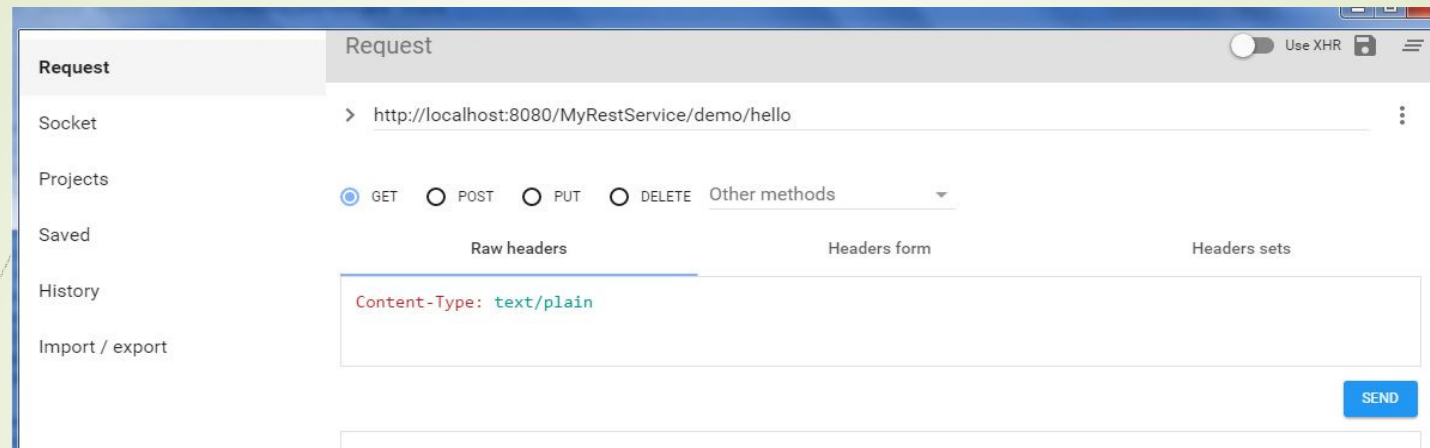


- Instalar un cliente Rest en el navegador (por ejemplo la extensión Advanced Rest Client en Chrome)
- Nota: Existen muchos clientes Rest para Firefox y Chrome
- Para abrir buscarla en el navegador y abrirla. Opcionalmente, se puede instalar el menú de aplicaciones de Chrome para acceder más rápidamente a la extensión o incluirla en la barra de marcadores.

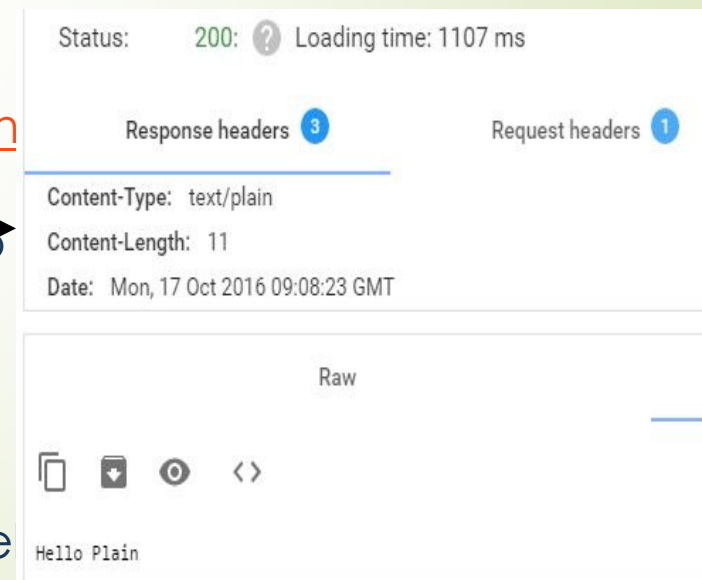


RESTFUL HELLO WORLD

➡ 6. Prueba desde el cliente de un navegador (ii)



- Pulsamos sobre el icono de REST dentro de Chrome, ponemos la siguiente URI: <http://localhost:8080/HelloWorld/demo/hello>, seleccionamos GET y pulsamos sobre Send. Nos debería dar el resultado siguiente:
- La URI se ha formado con la dirección del servidor Tomcat (localhost:8080), el nombre del proyecto (HelloWorld), el nombre de la raíz que establecimos en el archivo web.xml (demo) y el nombre del path indicado en la clase (hello).



SERVICIOS WEB REST

➤ Contenido

- Conceptos básicos de REST
- Restful Hello World
- **Familiarízate con las etiquetas de Rest**
 - **@Path**
 - **@PathParam**
 - **@QueryParam**
 - **@XmlRootElement**
 - **@Post**
 - **@FormParam**
 - **@PUT, @DELETE**
- Invocar un servicio remoto con Jersey
- Instrucciones de entrega de la práctica 7
- Anexo 1. Cómo evitar el mensaje de error al desplegar
- Anexo 2. Libros y referencias de consulta

Familiarízate con las etiquetas de Rest

➡ @Path

- Añade la siguiente función dentro de Hello.java:

```
@GET
@Path("/hello2")
@Produces(MediaType.TEXT_PLAIN)
public String sayPlainTextHello2(){return "Hello Plain 2";}
```

- Fíjate que le hemos añadido el Path a la función
- Fíjate que el servidor Tomcat se reinicia solo para añadir la nueva función (debes esperar a que termine para poder probarla)
- Ahora para poder probarla habría que añadir ese path a toda la ruta raíz: <http://localhost:8080/HelloWorld/demo/hello/hello2>
- Pruébalo
- NOTA: la URI contiene el path de la clase seguido del del método:
 - ➡ No es necesario que la clase tenga path, en tal caso se usa solo el del método.
 - ➡ Si la clase lleva path, pero el método no, busca un método que cumpla con la solicitud (en este caso GET que devuelve texto plano) que no tenga path; si hay más da una dará lugar a error.

Familiarízate con las etiquetas de Rest

➡ @PathParam

- Añade la siguiente función dentro de Hello.java:

```
@GET
@Path("/helloId/{oid}")
@Produces(MediaType.TEXT_PLAIN)
public String sayHelloWithId(@PathParam("oid") int id)
{return "Hello Plain " + id;}
```

- Nota: nuevas etiquetas requieren nuevos Import (import javax.ws.rs.PathParam;). **OJO, siempre con la raíz javax.ws**
- Le hemos añadido al Path un identificador de parámetro entre llaves
- En los parámetros de la función hemos identificado dicho parámetro poniendo @PathParam("oid") antes del tipo del parámetro correspondiente, siendo oid el identificador usado en el path
- Para poder probarla hay que reemplazar {oid} por el valor que queramos que tenga ese parámetro, i.e: <http://localhost:8080/HelloWorld/demo/hello/helloId/5>
- Pruébalo
- Si quieres añadir múltiples parámetros sepáralos por /, cada uno con sus llaves :@PathParam("/helloDate/{year}/{month}/{day}")

Familiarízate con las etiquetas de Rest

➡ @QueryParam

- Nos sirve para enviar directamente los parámetros de un formulario HTML por GET.
- Añade la siguiente función dentro de Hello.java:

```
@GET
@Path("/helloQuery")
@Produces(MediaType.TEXT_PLAIN)

public String sayHelloWithQuery(@QueryParam("name")String
name,@QueryParam("surname")String surname )
{return "Hello " + name + " "+ surname;}
```

- Para poder probarla hay que añadir los parámetros en el path como sigue:
- <http://localhost:8080/HelloWorld/demo/hello/helloQuery?name=Guadalupe&surname=Ortiz>
- Pruébalo

Familiarízate con las etiquetas de Rest

➡ @XmlRootElement (i)

- @XmlRootElement se añade a una clase que se quiere usar como parámetro de retorno. Nos va a permitir que se hagan transformaciones automáticas a tipo XML o tipo JSON
- Crea una nueva clase:

```
package nombrePaquete;  
  
import javax.xml.bind.annotation.XmlRootElement;  
  
@XmlRootElement  
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    public int getDay() {return day;}  
    public void setDay(int day) {this.day = day;}  
    public int getMonth() {return month;}  
    public void setMonth(int month) {this.month = month;}  
    public int getYear() {return year;}  
    public void setYear(int year) {this.year = year;} }
```


Familiarízate con las etiquetas de Rest

➡ @XmlRootElement (ii)

- Añade un nuevo método a tu clase Hello:

```
@GET
@Path("/dateJSON")
@Produces({"application/json"})
public MyDate getDate_JSON() {
    MyDate oneDate = new MyDate();
    oneDate.setDay(25);
    oneDate.setMonth(12);
    oneDate.setYear (2014);
    return oneDate;}

```

- Pruébala
- NOTA. Puedes usar tanto `@Produces({"application/json"})` como `@Produces({MediaType.APPLICATION_JSON})`
- Pruébalo
- Pruébalo ahora poniendo `@Produces({"application/xml"})` o `@Produces({MediaType.APPLICATION_XML})`

Familiarízate con las etiquetas de Rest

➡ @POST (i)

- Añade un nuevo método:

```
@POST
@Path("/name")
@Consumes(MediaType.TEXT_PLAIN)
@Produces(MediaType.TEXT_PLAIN)
public String HelloName(String myName){
    return "Hello "+myName;
}
```

- Hemos añadido la etiqueta que indica el tipo de dato que le vamos a enviar (@Consumes)
- Pruébalo:
 - ➡ Selecciona POST
 - ➡ En el desplegable elige text/plain
 - ➡ En el campo Payload escribe el texto a enviar

The screenshot shows a REST client interface with the following elements:

- Request** tab selected.
- URL: `http://localhost:8080/MyRestService/demo/hello/name`
- Method: **POST** (selected from a dropdown menu).
- Content-Type: **text/plain** (selected from a dropdown menu).
- Raw headers: `Content-Type: text/plain`
- Raw payload: `Lupe`
- SEND** button at the bottom right.

Arrows from the text instructions point to the **POST** method, the **text/plain** content type, and the **Lupe** payload.

Familiarízate con las etiquetas de Rest

➡ @POST (ii)

- Vamos a ver ahora un ejemplo con formato JSON
 1. Recuerda que el JSON debe ir entre llaves y está compuesto por una serie de claves, **separadas por comas, entre comillas** seguidas de **dos puntos** y del **valor** correspondiente.

Por ejemplo:

```
{  
  "Nombre":"Mariano",  
  "Edad":33  
}
```

1. Además para que se pueda hacer la conversion automática a la clase correspondiente, **las claves deben coincidir con los atributos privados de la clase.**

Familiarízate con las etiquetas de Rest

➡ @POST (iii)

- Añade un nuevo método:

```
@POST
@Path("/myDate2015")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public MyDate dateToString(MyDate myDate) {
    myDate.setYear(2015);
    return myDate;}
```

- Pruébalo
- Nota: En el **desplegable** elige **application/json**.
- En el **raw payload** asegúrate de enviar los datos correctamente en formato JSON. Recuerda que las **claves deben coincidir con los atributos privados de la clase** (en el ejemplo day, month, year). Visita la **transparencia anterior** para ver el formato del JSON.
- Prueba ahora la misma función enviando un JSON y recibiendo un XML

Familiarízate con las etiquetas de Rest

➡ @FormParam (i)

- Se utiliza para enviar directamente (mediante POST) parámetros de un formulario HTML
- Añade un nuevo método:

```
@POST
@Path("/myDateForm")
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
public MyDate dateToText(@FormParam("day") int myDay,
    @FormParam("month") int myMonth, @FormParam("year") int myYear){
    MyDate myDate = new MyDate();
    myDate.setDay(myDay);
    myDate.setMonth(myMonth);
    myDate.setYear(myYear);
    return myDate;
}
```

Familiarízate con las etiquetas de Rest

➡ @FormParam (ii)

- Pruébalo. Selecciona en el desplegable application/x-www-form-urlencoded
- Puedes añadir los parámetros uno a uno como Data Form o directamente en el raw payload separados por &.

Request

Use XHR

http://localhost:8080/MyRestService/demo/hello/myDateForm

GET POST PUT DELETE Other methods application/x-www-form-urlencoded

Raw headers Headers form Headers sets

Content-Type: application/x-www-form-urlencoded

Raw payload Data form Files

Form data for x-www-form-urlencoded parameters

day	3	×
month	5	×
year	2007	×

ADD ANOTHER PARAMETER

SEND

Raw payload

ENCODE PAYLOAD DECODE PAYLOAD

day=3&month=5&year=2007

Familiarízate con las etiquetas de Rest

➡ @PUT, @DELETE (i)

- Para poder modificar y borrar, y después comprobar que se ha efectuado correctamente vamos a crearnos una estructura estática dentro de la clase Hello:

```
private static Map<String, MyDate> myMap = new HashMap<>();  
static{  
    MyDate myDate = new MyDate();  
    myDate.setDay(25);  
    myDate.setMonth(12);  
    myDate.setYear(2014);  
    myMap.put("Navidad", myDate);  
    myMap.put("Nochebuena", myDate);  
    myMap.put("AnioNuevo", myDate);  
    myMap.put("Reyes", myDate);  
}
```


Familiarízate con las etiquetas de Rest

➡ @PUT, @DELETE (ii)

- Añade un nuevo método:

```
@PUT
@Path("/modifyDate/{date}")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.TEXT_PLAIN)

public String modifyImportantDate(@PathParam("date") String
key, MyDate myDate) {
myMap.put(key, myDate);
return "Date modified"; }
```

- Fíjate que para invocarlo tienes que añadir en el path el nombre de la fecha que quieres modificar (por ejemplo Navidad) y como raw payload el json con la nueva fecha.
- Después puedes comprobarlo haciendo un GET de la fecha.
- ¿Cómo sería el delete...?

SERVICIOS WEB REST

► Contenido

- Conceptos básicos de REST
- Restful Hello World
- Familiarízate con las etiquetas de Rest y pruébalas
- **Invocar un servicio remoto con Jersey**
- Instrucciones de entrega de la práctica 7
- Anexo 1. Cómo evitar el mensaje de error al desplegar
- Anexo 2. Libros y referencias de consulta

Invocar un servicio remote con Jersey

- Por ejemplo el servicio disponible en <https://api.ipify.org>
- Lo vamos a invocar y lo vamos a poner disponible también como método disponible de nuestro servicio para probarlo:

```
@GET
@Path("/ip")
@Produces(MediaType.TEXT_PLAIN)
public String IpObtain(){
    Client cliente = ClientBuilder.newClient();
    String t = cliente.target("https://api.ipify.org")
        .request(MediaType.TEXT_PLAIN).get(String.class);
    cliente.close();
    return "Mi ip es " +t;
}
```

SERVICIOS WEB REST

► Contenido

- Conceptos básicos de REST
- Restful Hello World
- Familiarízate con las etiquetas de Rest y pruébalas
- Invocar un servicio remoto con Jersey
- **Instrucciones de entrega de la práctica 7**
- Anexo 1. Cómo evitar el mensaje de error al desplegar
- Anexo 2. Libros y referencias de consulta

Instrucciones de entrega de la práctica 7

- Debe entregarse un único archivo comprimido .zip cuyo nombre será Practica7_NombreApellido1 (Practica5_ seguido del nombre y primer apellido del alumno).
- Dicho archivo contendrá:
 - El proyecto web dinámico creado con todos los ejemplos realizados en clase.
 - El war exportado del proyecto del servicio.

SERVICIOS WEB REST

► Contenido

- Conceptos básicos de REST
- Restful Hello World
- Familiarízate con las etiquetas de Rest y pruébalas
- Invocar un servicio remoto con Jersey
- **Anexo 1. Cómo evitar el mensaje de error al desplegar**
- **Anexo 2. Libros y referencias de consulta**

Anexo 1

► Cómo evitar el mensaje de error al desplegar

- Debes tener en cuenta que este error no es relevante ya que trata de cargar una web “en parte de cliente”, a pesar de tratarse de un servicio.
- En cualquier caso, si no quieres que aparezca este error:
 1. Dentro de la carpeta Webcontent del Proyecto añade un archivo Index.html con el texto que quieres que aparezca en la web.
 2. Dentro del web.xml antes de cerrar la etiqueta webapp añade lo siguiente:

```
<welcome-file-list>  
    <welcome-file>Index.html</welcome-file>  
</welcome-file-list>
```

Anexo 2

► Libros y referencias de consulta

Java EE 7 Essentials

By: Arun Gupta

Publisher: O'Reilly Media, Inc.

Pub. Date: August 23, 2013

Print ISBN-13: 978-1-4493-7017-6

Apache CXF Web Service Development

By: Naveen Balani ; Rajeev Hathi

Publisher: Packt Publishing

Pub. Date: December 17, 2009

Print ISBN-13: 978-1-847195-40-1

RESTful Java Web Services

By: Jose Sandoval

Publisher: Packt Publishing

Pub. Date: November 11, 2009

Print ISBN-13: 978-1-847196-46-0

Developing RESTful Services with JAX-RS 2.0, WebSockets, and JSON

By: Masoud Kalali; Bhakti Mehta

Publisher: Packt Publishing

Pub. Date: October 15, 2013

Print ISBN-13: 978-1-78217-812-5

REST with Java (JAX-RS) using Jersey - Tutorial

Lars Vogel

<http://www.vogella.com/articles/REST/article.html>

Creación de un servicio Web REST y su despliegue en Tomcat

Montes Cumbreira, Javier; Carmona Román, Salvador

<http://rodin.uca.es/xmlui/handle/10498/17553>

Tutorial de implementación y despliegue de un servicio web REST

Herrera Vaca, Alberto

<http://rodin.uca.es/xmlui/handle/10498/17551>