

CURSO JAVA EE

REST WEB SERVICES EN JAVA EE



Por el experto: Ing. Ubaldo Acosta



CURSO JAVA EE

www.globalmentoring.com.mx

Hola, te saluda nuevamente Ubaldo Acosta. Espero que estés listo para comenzar con esta lección..

Vamos a estudiar el tema de Rest Web Services utilizando Java EE.

¿Estás listo? ¡Vamos!

REST WEB SERVICES

REST: Representational State Transfer

Principios de una Arquitectura REST:

- ✓ **Recursos Direccionables:** Los recursos pueden ser solicitados por medio un URI.
- ✓ **Orientados a Representaciones:** Clientes y Servidores intercambian representaciones, la cual puede ser en XML, JSON, entre otros.
- ✓ **Interfaces Restringida:** Podemos utilizar las operaciones básicas HTTP: GET, POST, PUT y DELETE, esto es similar a SQL: SELECT, INSERT, UPDATE y DELETE respectivamente.

CURSO JAVA EE

www.globalmentoring.com.mx

Los SOAP Web Services utilizan HTTP como el mecanismo de transporte. Una petición GET o POST es ejecutada y un bloque de código XML es enviado al servidor. La URL que identifica al Servicio Web NO necesariamente indica qué tipo de operación debe realizarse del lado del servidor.

Los RESTful Web Services, por otro lado, se basan completamente en las operaciones soportadas por el protocolo HTTP para ejecutar la funcionalidad del lado del servidor. Cada llamada al Servicio Web, debe utilizar alguno de los siguientes métodos HTTP: GET, POST, PUT, DELETE, URL, HEAD u OPTIONS.

REST significa Representational State Transfer y nació por la necesidad de simplificar la creación de Web Services utilizando el protocolo HTTP como base. REST es una forma "ligera" y rápida de desarrollar y consumir Web Services. Debido a que el protocolo HTTP es utilizado prácticamente en todos lados donde utilicemos la Web, es posible reutilizar este mecanismo de comunicación como la base para la transmisión de Servicios Web.

La forma de crear un Web Services utilizando REST es a través de recursos (resources). Cada recurso tiene asociado una URI, y cada URI representa a su vez una operación del Web Service.

Ej: **/personas** - Es una URI que representa todas las entidades de tipo Persona de nuestro sistema.

Ej: **/personas/{id}** - Esta URI representa una orden en particular. A partir de esta URI, podremos leer (read), actualizar (update) y eliminar (delete) objetos de tipo Persona.

En esta lección estudiaremos a más detalle el tema de REST Web Services para exponer la lógica de negocio de nuestros EJB de Sesión de tipo Stateles.

MÉTODOS HTTP

Método HTTP	Significado en Restful Web Services
GET	Se utiliza para operaciones de sólo lectura. No generan ningún cambio en el servidor.
DELETE	Elimina un recurso en específico. Ejecutar esta operación múltiples veces no tiene ningún efecto.
POST	Cambia la información de un recurso en el servidor. Puede o no regresar información.
PUT	Almacena información de un recurso en particular. Ejecutar esta operación múltiples veces no tiene efecto, ya que se está almacenando la misma información sobre el recurso.
HEAD	Regresa solo el código de respuesta y cualquier cabecera HTTP asociado con la respuesta.
OPTIONS	Representa las opciones disponibles para establecer la comunicación en el proceso de petición/respuesta de una URI.

El protocolo HTTP está definido por el consorcio www.w3.org. Un conocimiento detallado de este protocolo para el estudio de Web Services no es necesario, sin embargo conforme se involucren cada vez más en la creación de Rest Web Services, más se deberá conocer acerca de este protocolo.

HTTP maneja 8 métodos, los cuales son: GET, DELETE, POST, PUT, HEAD, OPTIONS, TRACE y CONNECT. Cualquier petición enviada hacia un Servicio Web debe especificar cualquiera de los 6 métodos HTTP listados en la tabla. Se excluyen los métodos TRACE y CONNECT ya que no son relevantes para los Web Services de tipo RESTful.

- ✓ **GET:** Se utiliza para operaciones de sólo lectura. No generan ningún cambio en el servidor.
- ✓ **DELETE:** Elimina un recurso en específico. Ejecutar esta operación múltiples veces no tiene ningún efecto.
- ✓ **POST:** Cambia la información de un recurso en el servidor. Puede o no regresar información.
- ✓ **PUT:** Almacena información de un recurso en particular. Ejecutar esta operación múltiples veces no tiene efecto, ya que se está almacenando la misma información sobre el recurso.
- ✓ **HEAD:** Regresa solo el código de respuesta y cualquier cabecera HTTP asociado con la respuesta.
- ✓ **OPTIONS:** Representa las opciones disponibles para establecer la comunicación en el proceso de petición/respuesta de una URI.

Con estos métodos debemos tomar en cuenta dos características muy importantes. Los métodos *seguros* e *idempotentes*. Los métodos *seguros* (*no modifican el estado del sistema*) son aquellos que no hace otra tarea que recuperar información, por ejemplo los métodos GET y HEAD. Los métodos *idempotentes* son aquellos que siempre se obtiene el mismo resultado, sin importar cuantas veces se realice cierta operación. Métodos que son *idempotentes* son: GET, HEAD, PUT y DELETE. El resto de los métodos no son ni *seguros* ni *idempotentes*.

Aunque a nivel de programación es posible realizar más de una operación al momento de enviar una petición, por ejemplo, actualizar y eliminar, esto NO debería programarse de esta manera, ya que rompe con la idea básica de los REST Web Services.

Una gran ventaja de que las operaciones REST sean sobre el protocolo HTTP es que los administradores de servidores, redes y encargados de seguridad de las mismas, saben como configurar este tipo de tráfico, por lo que no es algo nuevo para ellos.

REQUEST HEADERS CABECEROS DE PETICIÓN

Los Cabeceros de Petición permiten obtener metadatos de la petición HTTP, como pueden ser:

- ✓ El Método HTTP utilizado en la petición (GET, POST, etc.)
- ✓ La IP del equipo que realizó la petición (192.168.1.1)
- ✓ El dominio del equipo que realizó la petición (www.midominio.com)
- ✓ El recurso solicitado (http://midominio.com/recurso)
- ✓ El navegador que se utilizó en la petición (Mozilla, MSIE, etc.)
- ✓ Entre varios cabeceros más...

CURSO JAVA EE

www.globalmentoring.com.mx

Al enviar y recibir mensajes con REST Web Services, es necesario tener conocimiento de códigos de estado. Algunos de los códigos de estado más utilizados son:

- 200 (Ok): Significa que la respuesta fue correcta, este el código de estado por default.
- 204 (Sin Contenido): El navegador continua desplegando el documento previo.
- 301 (Movido Permanentemente): El documento solicitado ha cambiado de ubicación, y posiblemente se indica la nueva ruta, en ese caso el navegador se redirecciona a la nueva página de manera automática.
- 302 (Encontrado): El documento se ha movido temporalmente, y el navegador se mueve al nuevo url de manera automática.
- 401 (Sin autorización): No se tiene permiso para visualizar el contenido solicitado, debido a que se trató de acceder a un recurso protegido con contraseña sin la autorización respectiva.
- 404 (No encontrado): El recurso solicitado no se encuentra alojado en el servidor Web.
- 500 (Error Interno del Servidor Web): El servidor web lanzó una excepción irrecuperable, y por lo tanto no se puede continuar procesando la petición.

Para una lista completa de los códigos de estado del protocolo HTTP se puede consultar el siguiente link:

http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

CABECEROS DE RESPUESTA Y TIPOS MIME

Los cabeceros de respuesta se utilizan para indicar al navegador Web cómo debe comportarse ante una respuesta de parte del servidor Web

Un ejemplo común es generar hojas de Excel, PDF's, Audio, Video, etc, en lugar de solamente responder con texto.

Para indicar el tipo de respuesta se utilizan los tipos MIME (Multipurpose Internet Mail Extensions)

Los tipos MIME son un conjunto de especificaciones con el objetivo de intercambiar archivos a través de Internet como puede ser texto, audio, vídeo, entre otros tipos.

CURSO JAVA EE

www.globalmentoring.com.mx

Los tipos MIME (**Multipurpose Internet Mail Extensions**) son el estándar de internet para definir el tipo de información que recibirá el cliente (navegador Web) al realizar una petición al servidor Web.

Los REST Web Services pueden devolver distintos tipos de contenido para un mismo recurso, por ejemplo:

- ✓ application/xml – Mensaje XML
- ✓ application/json - Mensaje JSON
- ✓ text/html – Salida HTML
- ✓ text/plain – Salida en texto plano
- ✓ application/octet-stream – Datos binarios

Lo anterior son los tipos de recursos que más se utilizan para el envío de información en REST. Lo más común es utilizar xml, sin embargo no está limitado a este tipo de información, ya que si estamos utilizando un cliente con Jquery y AJAX, es común que utilicemos la anotación de JSON en lugar de XML.

Un listado más completo de tipos MIME es el siguiente:

- **application/msword:** Microsoft Word document
- **application/pdf:** Acrobat (.pdf) file
- **application/vnd.ms-excel:** Excel spreadsheet
- **application/vnd.ms-powerpoint:** Powerpoint presentation
- **application/zip:** Zip archive
- **audio/x-wav:** Microsoft Windows sound file
- **text/css:** HTML cascading style sheet
- **text/html:** HTML document
- **text/xml:** XML document
- **image/gif:** GIF image
- **image/jpeg:** JPEG image
- **image/png:** PNG image
- **video/mpeg:** MPEG video clip
- **video/quicktime:** QuickTime video clip

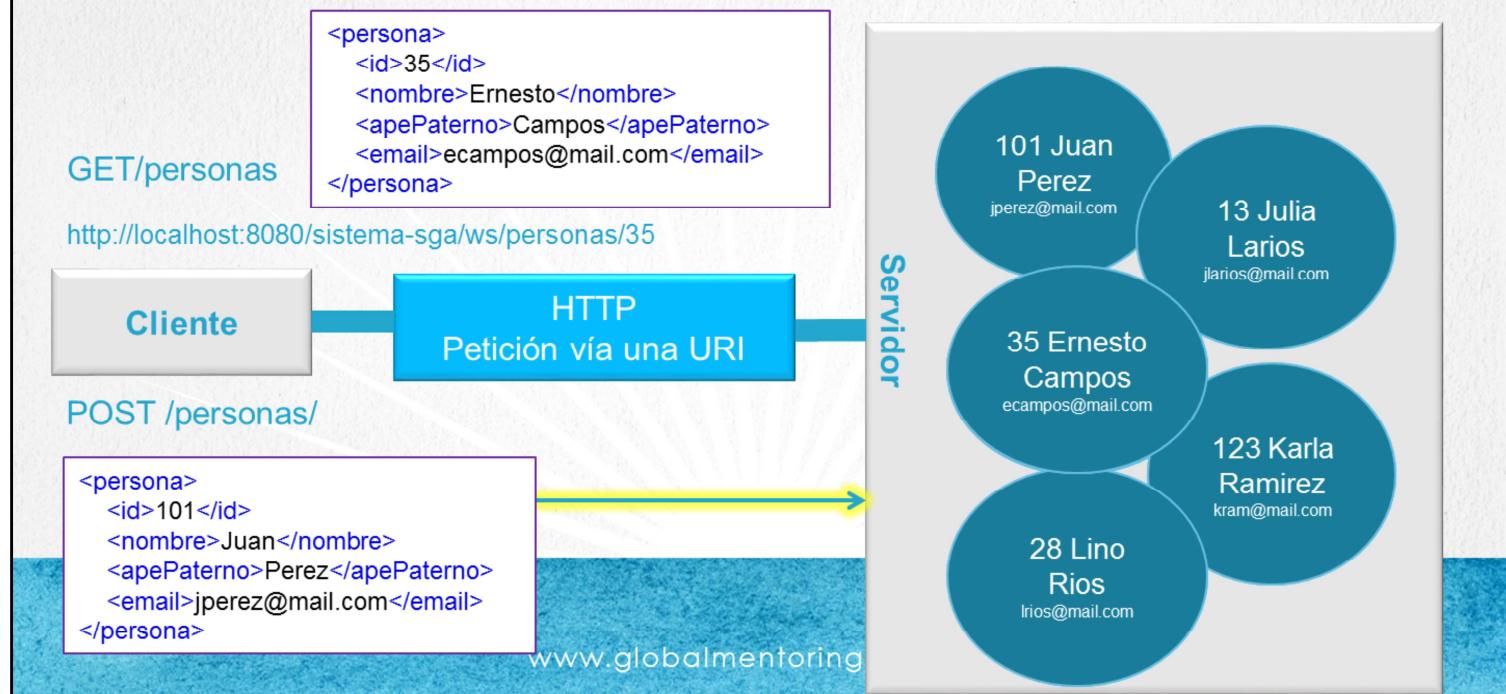
Para una lista más completa de tipos MIME, pueden consultar el siguiente link:

<http://www.freeformatter.com/mime-types-list.html>

REPRESENTACIONES CON REST

Representaciones REST

Recursos



Como podemos observar en la figura, REST utiliza el concepto de Representaciones, y cada representación apunta a un Recurso(s) del lado del Servidor Java.

Por ejemplo, para recuperar el objeto con id = 35, podemos utilizar el siguiente URI:

<http://localhost:8080/sistema-sga/ws/personas/35>

Esto regresará la representación del objeto en el tipo MIME solicitado. Por ejemplo, si se solicitó una representación en XML, la respuesta debería ser:

```
<persona>
<id>35</id>
<nombre>Ernesto</nombre>
<apePaterno>Campos</apePaterno>
<email>ecampos@mail.com</email>
</persona>
```

De manera similar, podemos tener las operaciones básicas para agregar, modificar y eliminar recursos del lado del servidor. Por ejemplo, las siguientes URL son ejemplos para cada una de las acciones mencionadas:

Agregar: POST /personas/ - Sigue la petición para agregar un nuevo recurso. Los datos se especifican en el documento XML a enviar. Ej.

```
<persona>
<id>101</id>
<nombre>Juan</nombre>
<apePaterno>Perez</apePaterno>
<email>jperez@mail.com</email>
</persona>
```

De manera similar, tenemos los ejemplos para modificar y eliminar:

Modificar: PUT /personas/123 – Sigue la petición para modificar el recurso 123 con un XML respectivo.

Eliminar: DELETE /personas/13 – Sigue la petición para eliminar el recurso 13

ANOTACIONES EN JAX RS

```

@Path("/personas")
public class PersonaServiceRS {

    @GET
    @Produces("application/xml , application/json")
    public List<Persona> listarPersonas(){..}

    @GET
    @Produces("application/xml")
    @Path("{id}") //hace referencia a /personas/{id}
    public Persona encontrarPersonaPorId(@PathParam("id") int id){..}

    @POST
    @Produces("application/xml")
    @Consumes("application/xml")
    public Response agregarPersona(Persona persona){..}

    @PUT
    @Produces("application/xml")
    @Consumes("application/xml")
    @Path("{id}")
    public Response modificarPersona(@PathParam("id") int id, Persona personaModificada){..}

    @DELETE
    @Path("{id}")
    public Response eliminarPersonaPorId(@PathParam("id") int id){..}
}

```

Como podemos observar en la figura, crear una clase que utilice JAX-RS para exponer un método como un servicio RESTfull Web Service en Java EE es muy simple. A continuación mencionaremos algunas de las anotaciones más utilizadas en JAX-RS, sin embargo, existen más anotaciones dependiendo del requerimiento a cubrir.

@Path: Esta anotación debe aparecer al inicio de la clase o en un método, e indica que esta clase/método se expondrá como un Servicio Web. Además, define la URI inicial del Servicio Web, la cual es relativa a la aplicación Web.

@GET, @POST, @PUT y @DELETE: Estas anotaciones se agregan a los métodos. Cada anotación representa el tipo de método HTTP que se va a utilizar. GET se utiliza para leer información, POST para agregar/modificar información. PUT se utilizar para agregar/modificar información y DELETE se utiliza para eliminar un elemento. En nuestro caso utilizaremos POST para agregar un nuevo recurso y PUT para modificar un recurso.

@PathParam: Para especificar parámetros se utilizan los signos { }. Los parámetros se adjuntan al método utilizando la anotación @PathParam. Puede haber múltiples parámetros. Ej. @Path("/personas/{tipo}/{id}")

@QueryParam: Permite procesar los parámetros del URL. Para especificar parámetros HTTP se agregan después de signo ?, y para agregar varios se utilizar el signo &. Ej.

<http://localhost:8080/webservice/personas?fechalinicio=01012012&fechaFin=31122012>

@Produces: Indica el tipo MIME que enviará al cliente y se debe especificar por cada método. Por ejemplo: @Produces({"application/json", "application/xml"})

@Consumes: Indica el tipo MIME que puede aceptar. Por ejemplo, en el caso de insertar una nueva Persona, podemos aceptar un mensaje XML indicando lo siguiente en el método a procesar la petición: @Consumes("application/xml"). JAX-RS utilizará JAXB para convertir el documento XML en una clase Java, para ello la clase Java de tipo Persona deberá tener la anotacion @XMLElement al inicio de la clase.

INTEGRACIÓN REST WS Y UNA APLICACIÓN WEB

Configuración del archivo web.xml para integrar JAX-RS con una aplicación Web:

```

<!-- Configuración de JAX-RS -->
<servlet>

    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>

    <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>mx.com.gm.sga.servicio.rest</param-value>
    </init-param>

    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/webservice/*</url-pattern>
</servlet-mapping>
```

CURSO JAVA EE

www.globalmentoring.com.mx

Si estamos utilizando las librerías del proyecto Jersey para desplegar REST Web Services, es necesario integrarlo con nuestra aplicación WEB, debido a que JAX-RS todavía no viene integrado de manera nativa con las aplicaciones Web.

En la figura podemos observar la configuración necesaria del API del proyecto de Jersey para poder producir y consumir RESTful Web Services.

Para que la aplicación web reconozca las URI de los Servicios Web respectivos, es necesario configurar el Servlet del API de Jersey. Además, se debe especificar el elemento <servlet-mapping> con el url-pattern a utilizar.

Por ejemplo, para solicitar el recurso persona id = 101, se debe utilizar el siguiente URL:

<http://localhost:8080/sistema-sga/webservices/personas/101>

Podemos observar que el URI mostrado incluye el url-pattern configurado anteriormente. Sin esta configuración no es posible ejecutar los Servicios Web.

INTEGRACIÓN EJB Y JAX RS

Código Servicio REST y EJB

```
@Path("/personas")
@Stateless
public class PersonaServiceRS {

    @EJB
    private PersonaService personaService;

    @GET
    @Produces("application/xml , application/json")
    public List<Persona> listarPersonas(){
        return personaService.listarPersonas();
    }
}
```

Código Clase Dominio Persona (API JAXB)

```
@XmlRootElement
public class Persona {

    private int idPersona;
    private String nombre;
    private String apePaterno;
    private String apeMaterno;
    private String email;
    private String telefono;

    public int getIdPersona() {
        return idPersona;
    }

    //Se omite el código restante
```

CURSO JAVA EE
www.globalmentoring.com.mx

Como hemos visto los EJB proveen una serie de servicios como seguridad, transaccionalidad, entre otras características. Este simple hecho tiene varias ventajas sobre clases puras de Java.

El API JAX-RS en combinación con los EJB hace muy simple la integración entre estas tecnologías y así exponer la funcionalidad de los EJB por medio de RESTful Web Services.

Existen varias formas de lograr esta integración. En la figura se observa una forma de realizar esta integración.

Sin embargo, lo que sí se debe planear con cuidado son los URI a utilizar, ya que de esto dependerá la interface que utilizará el cliente para poder consumir los RESTful Web Services.

Como podemos observar en la figura, para exponer la funcionalidad de un método EJB podemos crear una clase enfocada a exponer únicamente los métodos de los EJB que necesitemos. Sin embargo, esta clase a su vez debe ser un EJB de tipo Stateless para poder inyectar la funcionalidad de los EJB que se deseé.

Una vez realizada la inyección de dependencia del EJB, ya podemos utilizar los métodos del EJB, por ejemplo para desplegar el listado de personas.

Sin embargo, debido a que estamos regresando objetos de tipo Persona, esta clase se convierte de código Java a XML utilizando el API de JAXB. Para ello es necesario agregar la anotación @XmlRootElement a la clase Persona.

Con la configuración anterior ya tenemos listas nuestras clase Java para exponer funcionalidad de nuestros EJB como RESTful Web Services.

DESCRIPCIÓN DE REST WEB SERVICE

Documento WADL:

Ej. <http://localhost:8080/sistema-sga/webservice/application.wadl>



```

<application xmlns="http://wadl.dev.java.net/2009/02">
  <doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey: 1.11 12/09      10:27 AM"/>
  <grammars>
    <include href="application.wadl/xsd0.xsd">
      <doc title="Generated" xml:lang="en"/>
    </include>
  </grammars>
  <resources base="http://localhost:8080/sistema-sga/webservice/">
    <resource path="/personas">
      <method id="listarPersonas" name="GET">
        <response>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="persona" mediaType="application/xml"/>
        </response>
      </method>
      <method id="agregarPersona" name="POST">
        <request>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="persona" mediaType="application/xml"/>
        </request>
        <response>
          <representation mediaType="application/xml"/>
        </response>
      </method>
      <resource path="{id}">
        <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="id" style="template" type="xs:int"/>
        <method id="encontrarPersonaPorId" name="GET">
          <response>
            <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="persona" mediaType="application/xml"/>
          </response>
        </method>
      </resource>
    </resource>
  </resources>
</application>

```

Cuando creamos SOAP Web Services, el documento WSDL era el elemento central para describir el Web Service a utilizar.

De manera similar, con RESTful Web Services tenemos un documento conocido como WADL (Web Application Description Language). Este documento XML permite describir los RESTful Web Services de manera muy similar a un documento WSDL.

Este tipo de documento XML está en sus primeras etapas de desarrollo, y no ha sido adoptado como un estándar, a diferencia de WSDL, sin embargo permite autodocumentar el Servicio WEB así como el XSD asociado a dicho Web Services.

Esto tiene varias ventajas, por ejemplo, existe una herramienta para generar el código del lado del cliente.

`java -jar wadl2java.jar -o gen-src -p com.personas.rest http://localhost:8080/sistema-sga/webservice/application.wadl`

Sin embargo, este tipo de herramientas han recibido varias críticas debido a que la simplicidad de la creación de clientes REST hace innecesario este tipo de herramientas.

En nuestro caso crearemos el cliente de manera manual con ayuda de las clases del proyecto Jersey, el cual hace muy simple la tarea de crear clientes Java.

DESCRIPCIÓN DE REST WEB SERVICE

Documento XSD para validar el mensaje XML del Servicio Web:

Ej. <http://localhost:8080/sistema-sga/webservice/application.wadl/xsd0.xsd>



```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
  <xs:element name="persona" type="persona"/>
  <xs:complexType name="persona">
    <xs:sequence>
      <xs:element name="idPersona" type="xs:int"/>
      <xs:element name="apeMaterno" type="xs:string" minOccurs="0"/>
      <xs:element name="apePaterno" type="xs:string" minOccurs="0"/>
      <xs:element name="email" type="xs:string" minOccurs="0"/>
      <xs:element name="nombre" type="xs:string" minOccurs="0"/>
      <xs:element name="telefono" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

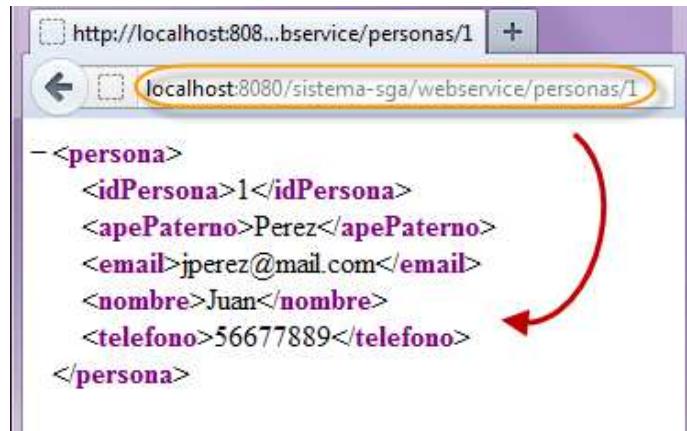
www.globalmentoring.com.mx

Como parte de la autodocumentación del Servicio Web, en el documento WADL se especifica el documento XSD que valida el documento XML a transmitir. Con el siguiente link es posible acceder a este documento.

<http://localhost:8080/sistema-sga/webservice/application.wadl/xsd0.xsd>

Como podemos observar en la figura, debido a que se está transmitiendo entidades de tipo Persona, es necesario validar que la información XML a transmitir sea válida conforme a este documento XSD.

Un ejemplo de un XML a transmitir es el siguiente:



Con el documento XSD mostrado en la lámina, es posible validar los mensajes XML a transmitir. En nuestro ejercicio mostraremos tanto el documento WADL como el documento XSD asociado a nuestro RESTful Web Service a desplegar.

CLIENTE REST WEB SERVICE

```

public class TestClienteRS {
    public static void main(String[] args) {
        Client client = Client.create();
        WebResource web = client.resource("http://localhost:8080/sistema-sga/webservice/personas/1");
        Persona persona = web.get(Persona.class);
        System.out.println("La persona es: " + persona.getNombre() + " " + persona.getApePaterno());
    }
}

@XmlRootElement
public class Persona {
    private int idPersona;
    private String nombre;
    private String apePaterno;
    private String apeMaterno;
    private String email;
    private String telefono;
    public int getIdPersona() {
        return idPersona;
    }
}
//Se omite el código restante

```

Código del Cliente Java

Clase de Dominio Persona del Cliente

La ventaja de un servicio REST es que las peticiones GET el navegador WEB las soporta por default, así que al colocar la URL del recurso REST a buscar, obtendremos en automático la respuesta respectiva. Por ejemplo al colocar la siguiente URI y estar levantado el servidor de GlassFish o el Servicio Web ya publicado, obtendremos la respuesta siguiente:

<http://localhost:8080/sistema-sga/webservice/personas/1>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<persona>
    <idPersona>1</idPersona>
    <apePaterno>Perez</apePaterno>
    <email>jperez@mail.com</email>
    <nombre>Juan</nombre>
    <telefono>56677889</telefono>
</persona>

```

Para crear un cliente Java, el proyecto <http://jersey.java.net/> proporciona varias clases para hacer muy simple el consumo de REST Web Services. En la figura se puede observar un ejemplo para consultar la persona con el idPersona = 1.

Al crear el ejercicio de SOAP Web Services, en automático nos creó las clases de dominio, por ejemplo, la clase de Persona. En este caso, debido a que no estamos utilizando herramientas de generación de código, es necesario crearlas manualmente. Para ello nos apoyaremos del documento WSDL, y del XSD asociado. Según hemos visto, para consultar estos documentos se debe utilizar la siguiente URL:

<http://localhost:8080/sistema-sga/webservice/application.wadl>

<http://localhost:8080/sistema-sga/webservice/application.wadl/xsd0.xsd>

Basado en el documento XSD podemos observar los atributos que necesitaremos para nuestros objetos de Entidad, necesarios para procesar la petición REST. Para ello, deberemos crear la clase Persona y agregar la anotación @XmlRootElement de JAXB en la definición de la clase Java. Estos temas los revisaremos a más detalle en el ejercicio que desarrollaremos a continuación.

EJERCICIOS CURSO JAVA EMPRESARIAL

- **ABRIR LOS ARCHIVOS DE EJERCICIOS EN PDF.**
- **EJERCICIO:** Creación Proyecto SGA con Rest Web Services.



CURSO JAVA EE
www.globalmentoring.com.mx

CURSO ONLINE

JAVA EMPRESARIAL

JAVA EE

Por: Ing. Ubaldo Acosta

**CURSO JAVA EE**www.globalmentoring.com.mx

En Global Mentoring promovemos la Pasión por la Tecnología Java. Te invitamos a visitar nuestro sitio Web donde encontrarás cursos Java Online desde Niveles Básicos, Intermedios y Avanzados, y así te conviertas en un experto programador Java.

A continuación te presentamos nuestro listado de cursos:

- | | |
|---|--|
| <ul style="list-style-type: none">✓ Lógica de Programación✓ Fundamentos de Java✓ Programación con Java✓ Java con JDBC✓ HTML, CSS y JavaScript✓ Servlets y JSP's✓ Struts Framework | <ul style="list-style-type: none">✓ Hibernate Framework & JPA✓ Spring Framework✓ JavaServer Faces✓ Java EE (EJB, JPA y Web Services)✓ JBoss Administration✓ Android con Java✓ HTML5 y CSS3 |
|---|--|

Datos de Contacto:Sitio Web: www.globalmentoring.com.mxEmail: informes@globalmentoring.com.mx