

TRABAJO FINAL TALLER DE PROGRAMACIÓN

**UNIVERSIDAD CIENTÍFICA DEL SUR - CPE**

**Tema**: Servicios Rest con Java EE

**Nombre**: Zany Vílchez

Luis Ávila

Fernando Parodi

**Profesor**: Gustavo Coronel

2016

**INDICE**

Contenido

1. [RESÚMEN 2](#_Toc458701693)
2. [ANTECEDENTES 3](#_Toc458701694)
3. [FUNDAMENTO TEÓRICO 4](#_Toc458701695)
4. [CASO DEMOSTRATIVO 8](#_Toc458701696)
5. [DEBATE 12](#_Toc458701697)
6. [CONCLUSIONES 13](#_Toc458701698)
7. [BIBLIOGRAFÍA 14](#_Toc458701699)

# RESÚMEN

Vivimos en un mundo tan globalizado, en donde Internet juega un papel importante, ya que es un sistema que permite la comunicación constante entre Clientes y Servidores.

Hacer que ésta comunicación sea lo más eficaz posible es el principal objetivo de todas las tecnologías WEB.

Desde hace unos años, los servicios Web no han parado de aumentar y mejorar, por lo que es ahí donde suman un papel importante los servicios Web Rest con apoyo de la plataforma Java Enterprise Edition (JEE).

Java Enterprise Edition es una plataforma conocida de programación, la cual maneja varias especificaciones de API tales como JDBC, RMI, E-mail, Servicios Web, XML, etc y define como coordinarlos.

Los servicios Web Rest son un estilo de Arquitectura de Software, muy vinculado desde sus orígenes al protocolo HTTP.

# 

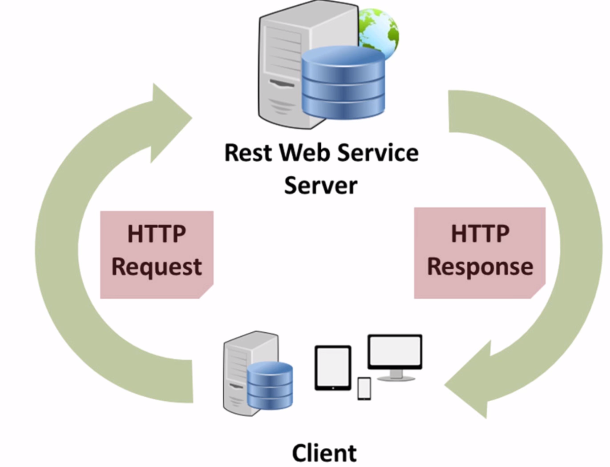
# ANTECEDENTES

Los servicios Web Rest así como la plataforma Java tienen más de 15 años en el medio.

Todo empieza en el año 1999, cuando el Ingeniero Roy Fielding estaba intentando estandarizar el modo en el que debía hacerse la comunicación entre Clientes y servidores para lograr que sean lo más eficaces posibles.

Éste personaje fue uno de los desarrolladores del protocolo HTTP y será quien defina un conjunto de principios en torno a los estándares HTTP y URI.

La arquitectura REST se hizo popular debido al crecimiento de la importancia de los Web Services así como a los problemas que giraban en torno al uso del SOAP.



# FUNDAMENTO TEÓRICO

El SOAP para Web Service es fácil de diseñar pero algo complicado de consumir, ya que se necesita toda una API para construir los clientes utilizando el WSDL y más aún si el cliente es tan simple como JAVA SCRIPT.

Además con SOAP se permite crear un solo servicio y ponerle varios métodos. Esto puede llevar un mal diseño del servicio ya que podría tener un servicio que haga todo: por ejemplo, un servicio de manejo de Clientes que permita también manejar Proveedores.

Los servicios Web Rest y Rest Full son una propuesta muy interesante ya que permite manejar los servicios web con métodos definidos, manteniendo la simpleza del protocolo como XML, pero que cada servicio sea identificado únicamente como URI.

Es importante destacar que los servicios de las redes sociales como Facebook, Twitter son basados en RestFull.

REST

Es una Arquitectura de Software muy vinculado, desde sus orígenes, al protocolo http.

Consiste en una serie de directrices y mejores prácticas para crear Web Services escalables.

Principios que definen la Arquitectura REST:

1. **Todo es un recurso:** Todo lo que se mueve a través de internet es un recurso los cuales se representan por el formato específico que tienen: Image, JPG, Video, text/HTML, text/XML.
2. **Con identificador único:** Cada recurso debe ser identificable por un código único. Accesible a través de URIS.
3. **Usa métodos HTTP estándar:** Los 8 métodos son GET, POST, PUT, DELETE, HEAD, OPTION, TRACE y CONNECT. Todos son conocidos como verbos.
4. **Pueden tener múltiples representaciones:** Puede ser representado en múltiples formatos distintos a los almacenados. Por ejemplo un recurso XML puede ser representado en formato JSON.
5. **Comunicaciones sin estado:** No se puede tener persistencia en las transacciones. El servidor trata cada transacción de forma independiente y la comunicación siempre devuelve un estado final (Recurso completo). Protege al cliente de los cambios que se puedan llevar a cabo en el servidor.

Las ventajas de utilizar los servicios REST son:

1. Separación del recurso y la representación
2. Visibilidad
3. Seguridad
4. Escalabilidad
5. Rendimiento

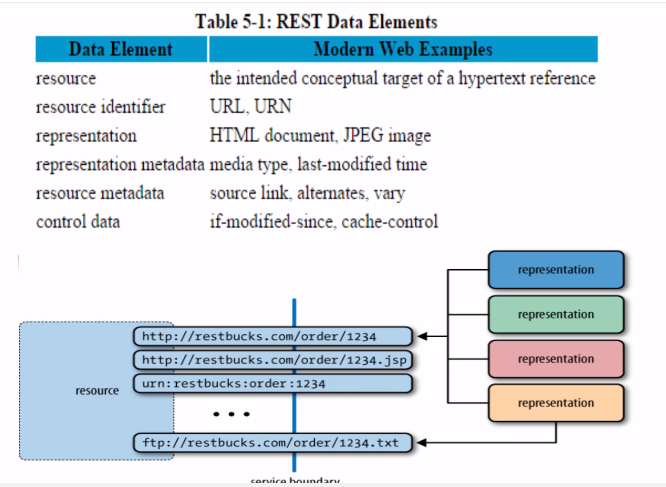
**Separación del Recurso:**

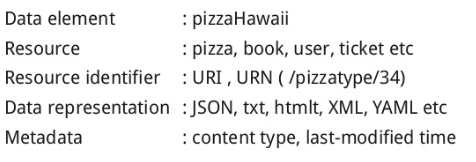
Un recurso simplemente es un conjunto de información, puede tener múltiples representaciones y no tiene estados, corresponde al que lo llama especificar en el encabezado de la petición HTTP el tipo de contenido. Una vez recibida esa información será la aplicación del servidor la encargada de manejar la representación y devolver el estado http apropiado.

**Http 200:** Ok en el caso de éxito

**Http 400:** Petición incorrecta, representación no soportada por el servidor

**Http500:** Error interno del servidor. Cuando pasa algo inesperado durante el proceso de petición

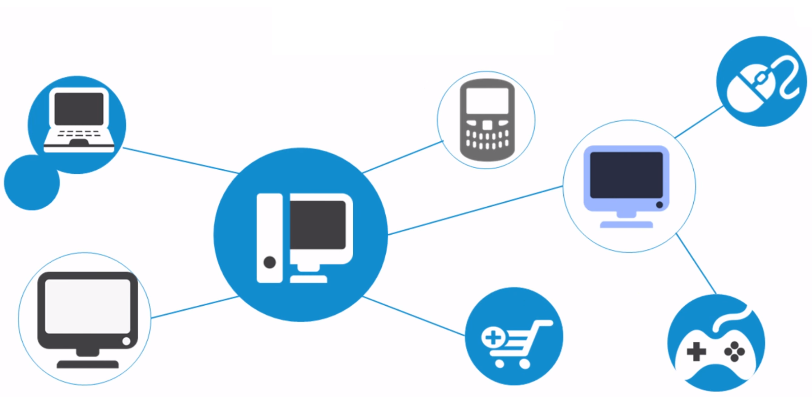




**Visibilidad:**

Cada aspecto del servicio debe ser auto descriptivo siguiendo el lenguaje natural HTTP. Eso garantiza las aplicaciones de monitorización y una comprensión clara de todas las comunicaciones http entre servicio Rest y el cliente que lo llama.

Esta visibilidad y lenguaje directo/claro son imprescindibles al estar trabajando con peticiones sin estado.

****

**Fiabilidad:**





**Escalabilidad y Rendimiento:**

Las aplicaciones deben escalar fácilmente a medida que su carga se incrementa.

Esto es lo que explica que no tener estado es crucial para cualquier aplicación que tenga que escalar, en el caso del mejor escenario, escalar la aplicación puede requerir más servidores lo que permite que haya un equilibrio en la carga.

Por lo tanto la escalabilidad se reduce a servir a los clientes para lo que necesiten en un tiempo aceptable. La idea es mantener la aplicación ejecutándose y evitar que se produzca una denegación en el servicio.

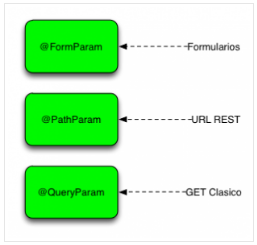
**Escalabilidad**: Número total de peticiones que la aplicación puede manejar

**Rendimiento**: Es medido por el tiempo necesario para que la petición sea procesada



# CASO DEMOSTRATIVO

Para comenzar, debemos entender en JEE el conjunto de anotaciones para REST es amplio. Se comentará a continuación varias de las más utilizadas cuando se crean este tipo de servicios.



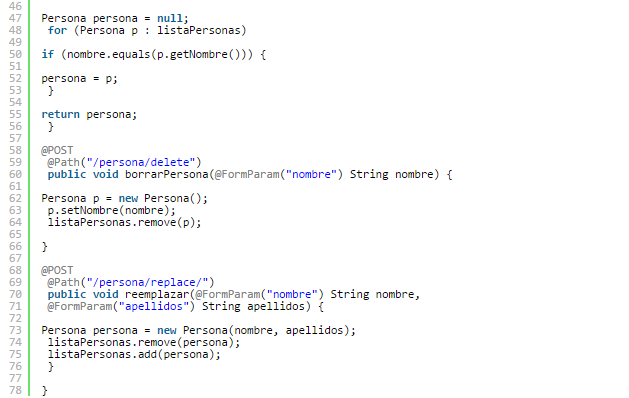
**@FormParam:** Esta anotación es una anotación a nivel de parámetro y sirve para ligar parámetros de un formulario HTML con variables del servicio REST.

**@PathParam:**Esta anotación es una anotación a nivel de parámetro y sirve para ligar parámetros de una petición clásica REST con las variables del servicio como por ejemplo /persona/nombre/Pedro. En donde la variable nombre tiene asignado el valor de Pedro.

**@QueryParam:**Esta anotación es una anotación útil ya que nos permite leer parámetros que vengan a través de una petición clásica GET como por ejemplo http://url?nombre=pedro

A continuación podemos ver un servicio REST similar a los anteriores que se apoya en este tipo de anotaciones para clarificar el código. En el se han usado anotaciones @POST para substituir @PUT y @DELETE y se han añadido las anotaciones comentadas antes.

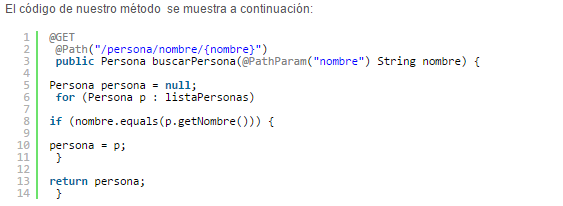




Para clarificar un poco el contenido del código vamos a revisar el método buscarPersona, el cual será ejecutado cuando la petición sea similar a la siguiente.

|  |  |
| --- | --- |
| **1** | **/Persona/nombre/pedro** |
|  |  |

El código de nuestro método  se muestra a continuación:



La anotación @PathParam que precede al parámetro “nombre” del método será la encargada de mapear el valor “pedro” a la variable nombre

|  |  |
| --- | --- |
| 1 | @PathParam("nombre") String nombre |

.De esta forma todo será mucho más sencillo. El resto de anotaciones realiza asignaciones similares pero apoyándose en los parámetros enviados por POST o por QueryString.

# DEBATE

**REST Vs SOAP**

En el momento que decidimos desarrollar nuestros servicios web, tenemos que tomar la decisión de que arquitectura será la más apropiada para nuestro sistema y el uso que vayamos a darle. A continuación se presentan las características de SOAP y REST, dos técnicas de arquitectura software orientadas a Webservices.

**SOAP** (siglas de  Simple Object Access Protocol)

* Es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambios de datos XML.
* El punto identificativo de SOAP es que las operaciones son definidas como puertos WSDL (Web Services Description Language).
* Es por esto que será aconsejable utilizar este protocolo en entornos donde se establecerá un contrato formal y donde se describirán todas las funciones de la interfaz así como el tipo de datos utilizados tanto de entrada como de salida. el lenguaje WSDL nos permitirá definir claramente cualquier detalle de las funciones de nuestro WS.

**REST (**Representational State Transfer**)**

* Es  un estilo de arquitectura de software para sistemas distribuidos tales como la web.
* A diferencia de SOAP, se centra en el uso de los estándares HTTP y XML para la transmisión de datos sin la necesidad de contar con una capa adicional.
* Las operaciones (ó funciones)  se solicitarán mediante GET, POST, PUT y DELETE, por lo que no requiere de implementaciones especiales para consumir estos servicios.
* Además se podrá utilizar JSON en vez de XML como contenedor de la información, por lo que será aconsejable utilizar este protocolo cuando busquemos mejorar el rendimiento, o cuando disponemos de escasos recursos, como sería el caso de los dispositivos móviles.

# CONCLUSIONES

No siempre REST es la mejor opción. Está surgiendo como una alternativa para diseñar servicios web con menos dependencia que su contraparte SOAP y los servicios basados en WSDL. De algún modo, REST es la vuelta a la Web antes de la aparición de los grandes servidores de aplicaciones, ya que hace énfasis en los primeros estándares de Internet, URI y HTTP. XML sobre HTTP es una interfaz muy poderosa que permite que aplicaciones internas, como interfaces basadas en Java puedan conectarse, ubicar y consumir recursos. De hecho, es justamente esta gran combinación que generó esta gran atención que tiene REST hoy en día.

Resulta muy flexible el poder exponer los recursos del sistema con un API REST, de manera de brindar datos a distintas aplicaciones, formateados en distintas maneras. REST ayuda a cumplir con los requerimientos de integración que son críticos para construir sistemas en donde los datos tienen que poder combinarse fácilmente (mashups) y extenderse. Desde este punto de vista, los servicios REST se convierten en algo mucho más grande.

# Bibliografía

<http://www.arquitecturajava.com/rest-json-y-java/>

<http://www.arquitecturajava.com/servicios-rest/>

<http://www.wetcom.com/content/como-desarrollar-un-servicio-web-rest-con-java-y-spring/>

<https://www.youtube.com/watch?v=8L-xo3cxCYo>

<http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html>

<https://www.adictosaltrabajo.com/tutoriales/creando-servicios-restful-con-netbeans-8/>

<https://www.ibm.com/developerworks/ssa/library/wa-aj-multitier2/>

<http://elblogdepicodev.blogspot.pe/2013/02/ejemplo-sencillo-de-web-service-con-resteasy.html>

<https://thanksnetwork.wordpress.com/2012/10/16/rest-vs-soap/>

<https://www.adictosaltrabajo.com/tutoriales/soavs-soap-rest/>