educación IT
CODE YOUR FUTURE

# Bootcamp Java Developer

## Fase 3 - Java Architect

Módulo 20

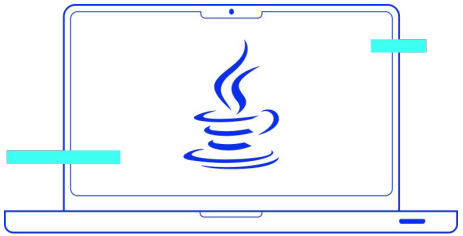
# REST

# REST

La **Transferencia de Estado Representacional** o **REST** por sus siglas en Inglés (*Representational State Transfer*) es una técnica de arquitectura de software para sistemas hipermedia distribuidos, como la *World Wide Web*.

Si bien el término REST se refiere a un conjunto de principios de arquitectura, en la actualidad se utiliza en un sentido más amplio para **describir cualquier interfaz web simple con XML o JSON y HTTP**, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios Web SOAP *(Simple Object Access Protocol).*

A partir de la versión *1.1* en adelante, JAX-RS es una parte oficial de **Java EE 6**.



# Recursos

Un concepto clave en REST es la existencia de **recursos** (elementos de información), que pueden ser accedidos utilizando un **identificador global** (un Identificador Uniforme de Recurso - **URI**). Para manipular estos recursos, los componentes de la red (clientes y servidores) se comunican a través de una interfaz estándar (HTTP) e intercambian representaciones de estos recursos.

La **petición** puede ser transmitida por cualquier número de conectores (por ejemplo: clientes, servidores, cachés, túneles, y otros) pero cada uno lo hace sin "ver más allá" de su propia petición (lo que se conoce como *stateless* -sin estado-, otra restricción de REST, que es un principio común con muchas otras partes de la arquitectura de redes y de la información).

Así, **una aplicación puede interactuar con un recurso conociendo su identificador y la acción requerida**, no necesita conocer si existen cachés, proxys, cortafuegos, túneles o cualquier otra cosa entre ella y el servidor que guarda la información.

La aplicación, sin embargo, debe comprender el formato de la información devuelta (la representación), que es por lo general, un documento HTML o XML, aunque también puede ser una imagen o cualquier otro contenido.

# JAX-RS

Como mencionamos en el módulo anterior, ***Java API for RESTful Web Services*** **(JAX-RS)** es una especificación para la creación de servicios web basados en el estilo arquitectónico *Representational State Transfer* (REST). JAX-RS utiliza anotaciones, introducidas en **Java SE 5**, para simplificar el desarrollo y despliegue de los clientes y puntos finales de los servicios web.

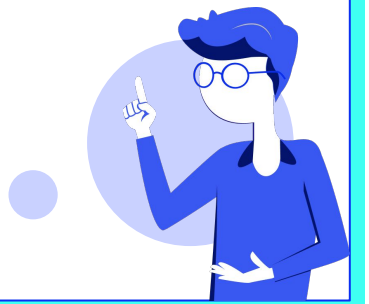
A partir de la versión 1.1 en adelante, JAX-RS es parte oficial de Java EE 6.

**JAX-RS** cuenta con una implementación de referencia llamada ***Jersey***, se pueden observar detalles en su página oficial:

Jersey.java.net

Se puede obtener la especificación completa en:

JCP.org



# RESTful

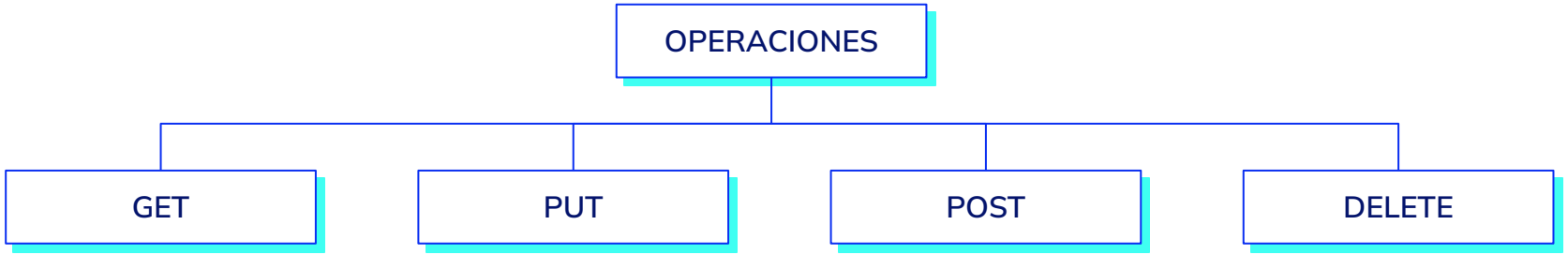
Los sistemas que siguen los principios **REST** se denominan a menudo como ***RESTful***.

Los **principios fundamentales** de REST son:

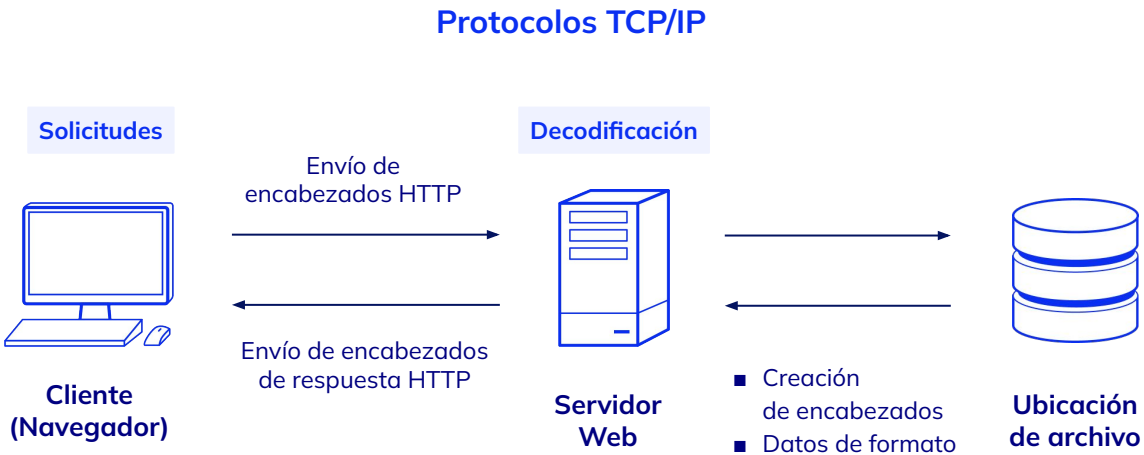
- **Un protocolo cliente/servidor sin estado *(Stateless)***: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes.

- **Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información**: HTTP define un conjunto de operaciones, las más importantes son `POST`, `GET`, `PUT` y `DELETE`. Con frecuencia estas operaciones se equiparan a las operaciones CRUD en bases de datos (ABMC en castellano: Alta, Baja, Modificación y Consulta) que se requieren para la persistencia de datos. Los métodos `GET` y `POST` son conocidos por los desarrolladores por los formularios web (`<form method="get"/>`).

Cada **operación** uno tiene una tarea específica:

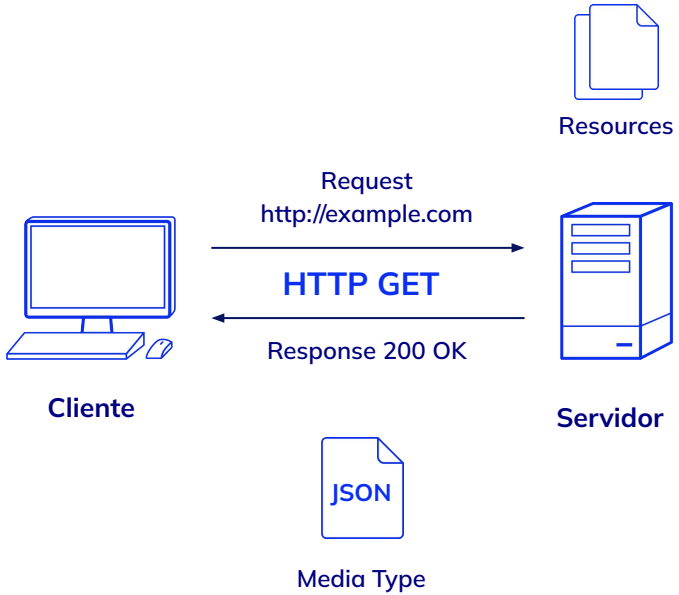


Una **sintaxis universal** para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su **URI**.



| | | |
|---|---|---|
| **HTTP Response** | **Status Line** | HTTP/1.1 200 OK Status Line |
| | **General Headers** | **Date**: Thu, 20 May 2004 21:12:58 GMT<br>**Connection**: close General Headers |
| | **Response Headers** | **Server**: Apache/1.3.27<br>**Accept-Ranges**: bytes Response Headers |
| | **Entity Headers** | **Content-Type**: text/html<br>**Content-Length**: 170 Entity Headers<br>**Last-Modified**: True, 18 May 2004 10:14:49 GMT |
| | **Message Body** | `<html>`<br>`<head>`<br>`<title>Welcome to the Amazing Site!</title>`<br>`</head>`<br>`<body>`<br>`<p>This site is under construction. Please come back later. Sorry!</p>`<br>`</body>`<br>`</html>`<br>Message Body |

El uso de **hipermedios** tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

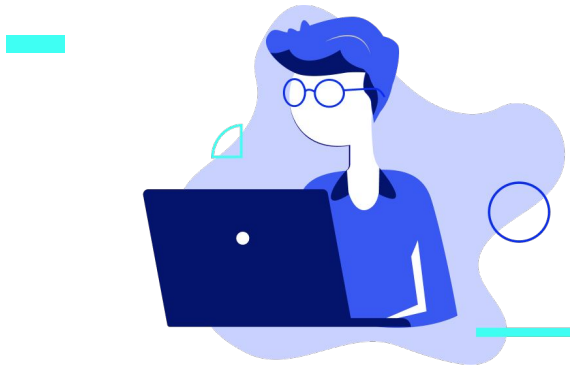


# Jersey Java Client

La implementación **Jersey** de **Jax-RS** brinda una API de clases para generar clientes de un web service REST desde clases Java.

En conjunto con las clases generadas por la API de Jersey, se puede utilizar la herramienta **XJC** para generar las clases a partir del **XSD** que se observa en el **WADL**.

Puedes obtener la documentación completa de la API en: Jersey.java.net



# JSON

JSON (acrónimo de *JavaScript Object Notation*, «notación de objeto de JavaScript») **es un formato de texto sencillo para el intercambio de datos.** Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera (año 2019) un formato independiente del lenguaje.

**Fuente:** JSON - Wikipedia

Los tipos de **datos disponibles con JSON** son:

- **Números**: se permite que sean negativos y, opcionalmente, pueden contener parte fraccional separada por puntos.

  **Ejemplo**: 123.456

- **Cadenas**: representan secuencias de cero o más caracteres. Se ponen entre doble comilla y se permiten cadenas de escape.

  **Ejemplo**: "Hola".

- **Booleanos**: representan valores booleanos y pueden tener dos valores: `true` y `false`.

- **`null`**: representan el valor nulo.

- **Array**: representa una lista ordenada de cero o más valores que pueden ser de cualquier tipo. Los valores se separan por comas y el vector se mete entre corchetes.

  **Ejemplo**: `["juan","pedro","jacinto"]`

- **Objetos**: son colecciones no ordenadas de pares de la forma `nombre:valor`. El nombre debe ser una cadena. El valor puede ser de cualquier tipo. El nombre y el valor están separados por dos puntos `:` y las parejas están separadas por comas `,` . Los objetos se escriben entre llaves `{}` y los nombres de las parejas se escriben entre comillas dobles `“”`.

## Ejemplo

```
{"departamento":8,"nombredepto":"Ventas","director": "juan rodriguez",
"empleados":[{"nombre":"Pedro","apellido":"Fernandez"},{"nombre":"Jacinto","apellido":"Benavente"}]}
```

| XML | JSON |
| --- | --- |
| **Ventajas**<br>• Tiene un formato muy estructurado y fácil de comprender.<br>• Puede ser validado fácilmente mediante *Schemas* (XSD).<br>• Se pueden definir estructuras complejas y reutilizables. | **Ventajas**<br>• Formato sumamente simple.<br>• Velocidad de procesamiento alta.<br>• Archivos de menor tamaño. |
| **Desventajas**<br>• Es más complicado de entender.<br>• El formato es sumamente estricto.<br>• Lleva más tiempo procesarlo.<br>• Un error con los namespace puede hacer que todo el documento sea invalido. | **Desventajas**<br>• Tiene una estructura enredosa y difícil de interpretar a simple vista. |

# ¡Sigamos trabajando!