**UNIVERSIDAD CIENTIFICA DEL SUR FACULTAD DE INGENIERIA DE SISTEMAS EMPRESARIALES**



**TALLER DE PROGRAMACIÓN**

**Servicios REST**

Presentado por:

CÉSAR BALAGUER

LIMA – PERU

2016

Contenido

[Concepto 3](#_Toc468789353)

[Servicio Web 3](#_Toc468789354)

[Arquitecturas más comunes 3](#_Toc468789355)

[RPC (Remote Procedure Calls) 3](#_Toc468789356)

[SOA (Arquitectura Orientada a Servicios) 3](#_Toc468789357)

[Niveles 4](#_Toc468789358)

[Nivel 1: Uso correcto de URIs 4](#_Toc468789359)

[Nivel 2: HTTP 5](#_Toc468789360)

[Nivel 3: Hypermedia 7](#_Toc468789361)

[Principios de REST 7](#_Toc468789362)

[Escalabilidad 7](#_Toc468789363)

[Generalidad de interfaces 8](#_Toc468789364)

[Puesta en funcionamiento independiente 8](#_Toc468789365)

[Compatibilidad 8](#_Toc468789366)

[Comparativo REST vs SOAP 8](#_Toc468789367)

[Bibliografía 10](#_Toc468789368)

# Concepto

REST (Representational State Transfer) es un tipo de arquitectura de desarrollo Web el cual se apoya íntegramente en HTTP. Permite crear servicios y aplicaciones, los cuales van a poder ser utilizados por cualquier dispositivo o cliente que entienda HTTP, esto lo hace más simple y convencional que otras opciones como SOAP o XML-RPC.

# Servicio Web

"W3C define los servicios Web como sistemas software diseñados para soportar una interacción interoperable máquina a máquina sobre una red. Los Servicios Web suelen ser APIs Web que pueden ser accedidas dentro de una red (principalmente Internet) y son ejecutados en el sistema que los aloja"[[1]](#footnote-2)

# Arquitecturas más comunes

## RPC (Remote Procedure Calls)

Esta arquitectura presenta una interfaz para las llamadas a procedimientos y funciones distribuidas, la cual es familiar y conocida por los desarrolladores.

Las primeras herramientas se centraron en esta visión, algunos lo denominaron "Primera Generación de Servicios Web" razón por la cual este estilo es muy extendido o utilizado, sin embargo últimamente ha sido criticado al extremo que muchos especialistas creen que debe desaparecer.

## SOA (Arquitectura Orientada a Servicios)

En esta arquitectura la unidad básica para la comunicación es el mensaje más que la operación. También es conocido como Servicios Orientados a Mensajes.

Estos servicios son soportados por la mayor parte de los desarrolladores de software y analistas. Al contrario que los servicios RPC, este estilo es débilmente acoplado, esto en cierta forma es preferible dado a que se centra en WSDL mas que en los detalles de la implementación

# Niveles

Existen tren niveles de calidad al momento de aplicar REST cuando se desarrolla una aplicación Web utilizando una API. Estos niveles son:

* Uso correcto de URIs
* Uso correcto de HTTP
* Implementar Hypermedia

Adicionalmente de estas reglas, se debe considera que nunca se debe guardar el estado en el servidor, toda la información requerida que se solicita debe estar en la consulta por parte cliente.

Al no guardar el estado, REST da mucho juego, se puede escalar mejor sin tener que preocuparse por el almacenamiento de variables de sesión u otros temas, se puede jugar con varias tecnologías para servir diversas partes o recursos incluso de una misma API.

## Nivel 1: Uso correcto de URIs

Durante el desarrollo de una web o de una aplicación web, las URL permite acceder a las páginas, secciones o documentos del sitio web. Cada página, información de una sección, archivos son llamados **recursos** cuando se habla de REST.

Estos recursos son la información a la que se desea acceder y que se desea modificar o borrar, independientemente de su formato.

Las URL (Uniform Resource Locator) son un tipo de URI (Uniform Resource Identifier) que además de identificar el recurso, también nos permite localizarlo.

Una URL se estructura de la siguiente manera:

[protocolo]://[hostname][:puerto (opcional)]/[ruta del recurso]?[consulta de filtrado]

por ejemplo:

https://www.youtube.com/watch?v=m46Z0-HXySo

Existen varias reglas básicas para ponerle nombre a una URI:

* Los nombres de URI no deben implicar una acción, por lo tanto deben evitarse usar verbos
* Deben ser únicos, no se debe tener más de una URI para identificar un mismo recurso
* Deben ser independientes de formato
* Deben mantener una jerarquía lógica
* Los filtrados de información de un recurso no se hacen en la URI

Por ejemplo:

https://compras.indraweb.net/cgi/facturas/0988110.pdf

Esta URI sería incorrecta, esto debido a que se está mencionando la extensión pdf en esta, una URI debe ser independiente del formato; la URI correcta sería:

https://compras.indraweb.net/cgi/facturas/0988110

De la misma manera para el ejemplo:

https://compras.indraweb.net/cgi/facturas/0988110/editar

Se está incluyendo un verbo por lo que esta URI tampoco sería correcta, una URI debe ser independiente al tipo de acción que se va a realizar, puede editarse, consultarse o eliminarse, esta debería ser siempre la misma URI:

https://compras.indraweb.net/cgi/facturas/0988110

## Nivel 2: HTTP

El conocimiento de HTTP es básico para desarrollar una web, para desarrollar con APIs REST existen aspectos claves que se deben dominar:

* Métodos HTTP

Al momento de crear URIs no se deben poner verbos ni verboides, esto se vio en el punto anterior, para manipular los recursos, HTTP entrega los siguientes métodos con los cuales se deben operar:

* + GET: para consultar y leer recursos
  + POST: para crear recursos
  + PUT: para editar recursos
  + DELETE: para eliminar recursos
  + PATCH: para editar partes concretas de un recurso

Por ejemplo:

Para una consulta de clientes podemos tener el siguiente recurso:

GET/cliente

Para crear un nuevo cliente se tendría el siguiente recurso:

POST/cliente

Para editar la información de un cliente en su totalidad:

PUT/cliente/C10307

Para eliminar un cliente:

DELETE/cliente/C10307

Para modificar parte de la información de un cliente:

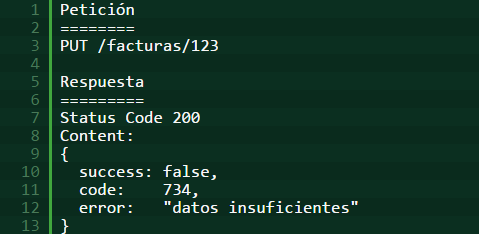
PATCH/cliente/C10307

La mayoría de los desarrolladores Web utilizan únicamente los métodos GET y POST para realizar todas las acciones mencionadas, si se trabaja con REST sería un error y podría ocasionar problemas obligando al desarrollador a incluir verbos en las URLs

* Códigos de estado

Uno de los errores más comunes que presentan los desarrolladores Web al momento de construir un API suele ser crear sus propias herramientas en lugar de utilizar las ya existentes como por ejemplo los códigos de error y códigos de estado.

Por ejemplo:

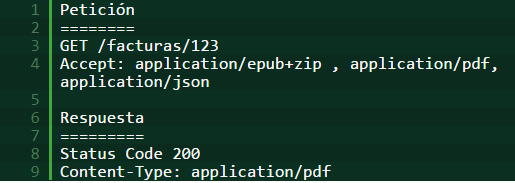


En el ejemplo mostrado se devuelve un código de estado 200 que significa que la petición se ha realizado correctamente, sin embargo, se devuelve un error y no el recurso solicitado en la URL

Este tipo de errores presenta varios inconvenientes:

* + No es REST
  + No es estándar
  + El cliente que accede a la API debe conocer el funcionamiento y como tratar los errores por lo que requiere mayor esfuerzo
  + Se debe mantener sus propios códigos y mensajes de error, con todo lo que eso conlleva.
* Aceptación de tipos de contenido

HTTP permite especificar el formato en que se desea recibir el recurso, se pueden indicar varios en orden de preferencia, para ello se utiliza el header **Accept**; La api programada devolverá el recurso en el primer formato de la lista que se encuentre disponible, en caso no se pueda mostrar en ninguno de los formatos indicados, devolverá el código HTTP 406



En el ejemplo se solicita una factura en formato comprimido con ZIP y de no existir entonces en pdf o json por orden de preferencia. El servidor retorna la factura en formato pdf

## Nivel 3: Hypermedia

Permite conectar mediante vínculos las aplicaciones clientes con las APIs esto permite a los clientes no preocuparse por conocer previamente como acceder a los recursos. Mediante la Hypermedia se añade información extra al recurso sobre otros recursos relacionados con él.

En el siguiente ejemplo se verá como se indica en un xml el enlace al recurso factura que se encuentra relacionada con el mismo:

<pedido>

<id>666</id>

<estado>Procesado</estado>

<links>

<link rel="factura">

http://example.com/api/pedido/666/factura

</link>

</links>

</pedido>

Hypermedia es útil para que el cliente no requiera conocer las URLs de los recursos, de tal manera que puede evitar hacer mantenimientos en estos, en caso estas cambien, es útil para automatizar los procesos entre Apis sin que se requiera interacción humana.

# Principios de REST

## Escalabilidad

De la misma interacción con los componentes. Las Web han crecido exponencialmente y a pesar de ello no han degradado su rendimiento.

## Generalidad de interfaces

Es gracias al protocolo HTTP que cualquier cliente puede interactuar con cualquier servidor HTTP sin necesidad de utilizar alguna configuración especial, sin embargo esto no es así para alternativas como SOAP

## Puesta en funcionamiento independiente

Los clientes y servidores pueden ser puestas en funcionamiento durante muchos años, por tanto, los servidores antiguos deben ser capaces de entenderse con clientes actuales y viceversa. Diseñar un protocolo que permita este tipo de características puede resultar muy complicado. HTTP permite la extensibilidad mediante el uso de cabeceras.

## Compatibilidad

Es compatible con componentes intermedios como proxys, caches que permiten mejorar el rendimiento, firewalls, gateways que permiten encapsular sistemas no propiamente Webs.

# Comparativo REST vs SOAP

|  |  |  |
| --- | --- | --- |
|  | REST | SOAP |
| Tecnología | Interacción dirigida por el usuario por medio de formularios | Flujo de eventos orquestados |
| Pocas operaciones con muchos recursos | Muchas operaciones con pocos recursos |
| Mecanismo consistente de nombrado de recursos | Falta de mecanismo de nombrado |
| Se centra en la escalabilidad y rendimiento a gran escala para sistemas distribuidos hipermedia | Se centra en el diseño de aplicaciones distribuidas |
| Protocolo |  |  |
| XML autodescriptivo | Tipado fuente, XML, Schema |
| HTTP | Independiente del transporte |
| HTTP como protocolo de aplicación | HTTP como protocolo de transporte |
| Síncrono | Síncrono y Asíncrono |

|  |  |  |
| --- | --- | --- |
| Gestión del Estado | El servidor no tiene estado | Elservidor puedemantener el estado de la conversación |
| Los recursos contienen datos y enlaces | Los mensajes solo contienen datos |
| Los clientes mantienen los estados siguiendo los enlaces | Los clientes mantienen los estados suponiendo el estado del servicio |
| Técnicas para añadir sesiones (cookies) | Técnicas para añadir sesiones (cabeceras de sesion) |
| Seguridad | Https | Ws-Security |
| Implementado desde hace muchos años | Las implementaciones son recientes |
| Comunicación punto a punto segura | Comunicación origen - destino segura |

# Bibliografía

[1] http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf

Rest vs Web Services

Rafael Navarro Marset

Modelado, Diseño e Implementación de Servicios Web 2006, 2007

[2] http://asiermarques.com/2013/conceptos-sobre-apis-rest/

Conceptos sobre APIs REST

Asier Marqués

Apuntes personales, desarrollo de software y negocios en Internet

1. Rafael Navarro Marset. Modelado, Diseño e Implementación de Servicios Web (2006 - 2007) [↑](#footnote-ref-2)