Web services © ADR Infor SL

Indice

Web services

Al finalizar esta unidad el alumno será capaz de crear un web service utilizando REST. Del mismo modo también será capaz de crear aplicaciones web que consuman web services de terceros.

Dentro del ecosistema de las aplicaciones web, es habitual que unas aplicaciones se comuniquen con otras con frecuencia. Como ya sabemos, una aplicación web no se encuentra aislada e incomunicada en su servidor, sino que **puede interactuar con otras aplicaciones y comunicarse con ellas**.

Con toda seguridad, durante el transcurso de la vida profesional de cualquier programador de aplicaciones web, se encontrará en algún momento con la **necesidad de solicitar información a una aplicación web** de un tercero o **requerir que ésta, realice cierta acción**.

Igualmente, con frecuencia necesitaremos que **aplicaciones externas se comuniquen con nuestra aplicación web** para realizar operaciones o suministrarles datos.

Tenemos una tienda online en la que vendemos nuestros productos.

Hemos llegado a un acuerdo con los propietarios de otra tienda online para que ésta, pueda vender nuestros productos en su página.

La tienda online con la que hemos llegado al acuerdo, tiene que ser capaz de comunicarse con la nuestra para **solicitar información de los productos disponibles, así como su stock**. También necesitará **crear ordenes de envío de productos** en nuestro sistema de manera directa.

Ya que las aplicaciones web no se encuentran en el mismo servidor y puede que ni siquiera utilicen la misma tecnología para su funcionamiento, **debemos proveerlas de mecanismos específicos para que éstas puedan comunicarse**. Éstos mecanismos son los **web services**.

¿Qué es un web service?

Un web service es una funcionalidad que desarrollamos en nuestra aplicación web para que, **mediante llamadas HTTP**, un tercero pueda solicitar información o realizar acciones en la misma.

En el funcionamiento del web services, siempre intervienen dos actores:

Proveedor del servicio
Es el encargado de suministrar un **endpoint** (también llamado directamente **web service**) para que terceros puedan **obtener información o realizar acciones en su aplicación web**

Consumidor del servicio
También llamado **cliente**. Mediante peticiones HTTP a un endpoint suministrado por el proveedor de servicio, **solicita información o la realización de acciones** en su aplicación web.

Como estás observando, **un web service es una arquitectura cliente-servidor** en la que el **consumidor del servicio actúa como cliente** y el **proveedor del servicio como servidor**.

Aplicaciones móviles y web services

Los web services no se utilizan únicamente en aplicaciones web. **Cualquier aplicación o tecnología capaz de realizar peticiones HTTP a través de internet, puede consumir web services**.

Un caso frecuente es el uso de aplicaciones móviles. Los dispositivos móviles suelen ser sistemas bastante limitados y, en la mayoría de sistemas, **no disponen de drivers para conectar a bases de datos externas**.

Sin embargo, estamos acostumbrados a que los datos de nuestras aplicaciones móviles se guarden en la nube (es decir, en una base de datos externa al dispositivo). Éstas aplicaciones **utilizan un web service para almacenar y recuperar información** de un servidor externo.

Siempre que veas una aplicación móvil que recupera o almacena datos en la nube, **un web service está actuando**.

Crear un web service básico

En este apartado, vamos a ver cómo desarrollar un web service **básico** en la que dos aplicaciones ser comunican entre sí.

Anotación: Sobre este ejemplo

No vamos a seguir una norma estándar concreta para hacerlo, de ésto nos encargaremos más adelante.

El objetivo en este momento es comprender la mecánica básica y el potencial del web service, por lo que utilizaremos una programación y una arquitectura **mínima** para poder entender su funcionamiento con sencillez.

Generar un endpoint

La misión del proveedor del servicio es **suministrar un endpoint** al cual llegarán las peticiones HTTP.

A este endpoint lo llamaremos **web service**.

El web service mostrará cierta información o realizará ciertas acciones cuando le lleguen las peticiones correspondientes. Para este primer ejemplo básico, **utilizaremos parámetros GET** para especificar al web service la acción que queremos que realice.

Vamos a ver las especificaciones que debe cumplir nuestro web service para este ejemplo:

En nuestra aplicación web, tenemos una **tabla de productos** que deseamos permitir consultar y manipular a través de un web service.

El endpoint de nuestro web service estará localizado en:

http://localhost/web_service/productos.php

El web service podrá realizar las siguientes acciones:

Consulta

Para solicitar al web service que nos muestre todos los productos utilizaremos la siguiente URL:

http://localhost/web_service/productos.php?accion=consulta

El web service devolverá los datos en formatoCSV de forma similar a la siguiente:

1,Eau de toilette vaporizador
2,Shampoo 250 ml.
3,Kit de manicura

Inserción

Para solicitar al web service que inserte un nuevo producto utilizaremos una URL similar a la siguiente:

http://localhost/web_service/productos.php?accion=insercion&valor=Cortauñas

Eliminación

Para solicitar al web service que elimine un producto utilizaremos una URL similar a la siguiente:

http://localhost/web_service/productos.php?accion=eliminacion&id=2

Desarrollar un web service básico - Parte 1

Desarrollar un web service básico - Parte 2

Llamar al endpoint

Hemos visto cómo lanzando peticiones HTTP al web service, podemos consultar y manipular los datos de la aplicación web. Hasta ahora hemos lanzado estas peticiones HTTP de forma manual utilizando nuestro navegador, pero lo realmente interesante es que **una aplicación web, ajena al servicio, sea capaz de consultar y manipular los datos**.

El web service, atenderá todas las peticiones HTTP, con independencia de si es un humano o una máquina quien realiza la solicitud. Por lo tanto, si somos el consumidor del servicio, nuestro papel es generar un **script que se comunique con el web service utilizando peticiones HTTP**.

Proceso: Realizar peticiones HTTP desde PHP

Existen varias formas de enviar una petición HTTP desde nuestros scripts de PHP. La más sencilla de ellas es utilizar la función file_get_contents, que devuelve el contenido de una URL determinada.

Para este ejemplo básico, utilizaremos esta opción debido a su sencillez.

Consumir un web service - Parte 1

Consumir un web service - Parte 2

Descarga: Ejemplo de web service básico

Puedes descargar el ejemplo de web service que acabamos de desarrollar completo, haciendo clic aquí.

Estandarización de web services

En el ejemplo anterior, hemos creado un web service, de la forma que nos ha parecido más cómoda y funcional para nuestro caso concreto. No obstante, al habernos "inventado" la forma de gestionar el web service, **estamos obligando a los programadores a aprender cómo funciona nuestro endpoint**.

En muchas ocasiones, los web services pueden ser muy extensos y complejos, teniendo la capacidad de realizar cientos de acciones diferentes.

Esto implica un **gasto de tiempo enorme en aprender el funcionamiento** del web service por parte de los programadores y un esfuerzo extra del proveedor del servicio que **deberá elaborar los manuales y la documentación** correspondiente para el uso del web service.

Por este motivo, existen **protocolos estándar** que especifican cómo deben funcionar los web service.

Utilizando estos protocolos, los programadores **no necesitan aprender cómo funciona el web service** ya que, si conocen el protocolo, sabrán cómo comunicarse con él.

Por su parte, los proveedores del servicio **no necesitan elaborar una documentación tan exhaustiva** ya que la mecánica del funcionamiento del web service es estándar y puede ser consultada externamente.

Por estos motivos, siempre que desarrollemos un web service, **debemos utilizar un protocolo estándar**.

En la actualidad, existen dos protocolos estándar para la elaboración de web services:

SOAP

Se trata de un protocolo estándar que define cómo dos objetos pueden comunicarse por medio de intercambio de datos XML.

Desde su desarrollo en 1998, ha gozado de gran popularidad siendo durante muchos años el estándar de facto de comunicación entre máquinas.

En la actualidad, su uso esta muy extendido, pero comienza a ser desplazado a favor de REST.

REST

Es una arquitectura basada **exclusivamente** en los métodos y características del **protocolo HTTP**.

Utilizando únicamente las especificaciones del protocolo HTTP, es capaz de proporcionar una API para poder realizar diferentes operaciones entre la aplicación que ofrece el servicio web y el cliente.

Está reemplazando paulatinamente a SOAP debido a su simplicidad y a día de hoy conviven ambos protocolos para el desarrollo de web services.

Por su sencillez y su extensa utilización en la actualidad, en este curso, vamos a estudiar **REST** como protocolo estándar para la generación de web services.

REST

La arquitectura REST utiliza **exclusivamente las características del protocolo HTTP** para definir cómo deben comunicarse dos sistemas entre sí. Esto significa, que podemos construir servicios web, que pueden ser consumidos por **cualquier dispositivo que utilice HTTP** sin necesidad de ninguna característica extra.

Ya que el protocolo de comunicación HTTP es el encargado de transferir la información de la web, apoyarse exclusivamente en él es una idea fantástica para desarrollar servicios web, que simplificará el desarrollo de los mismos.

Gracias a esta simplicidad, **REST se está convirtiendo en el nuevo estándar actual de comunicación**, frente a otras alternativas como SOAP que han gozado de una amplia utilización durante los últimos diez años.

Anotación: Uso del protocolo HTTP

El protocolo HTTP es muy amplio. Habitualmente, en las páginas o aplicaciones web,**se realiza un uso parcial** de las funcionalidades que pone a nuestra disposición.

No obstante, para implementar REST, **necesitaremos un uso más exhaustivo de estas funcionalidades**, lo que nos llevará a necesitar un conocimiento más amplio de su funcionamiento.

EJEMPLO: Habitualmente en la web utilizamos dos métodos de comunicación: GET y POST. Sin embargo, el protocolo HTTP **define muchos métodos más** que necesitaremos utilizar para implementar REST, como son: **PUT**, **DELETE** o **PATCH**.

Estándar HTTP

Si deseas conocer en profundidad todas las especificaciones del protocolo HTTP, puedes consultar sus estándares oficiales:

HTTP 1.1: https://tools.ietf.org/html/rfc2616
HTTP 2.0: https://tools.ietf.org/html/rfc7540

Aunque puede resultar interesante, esta documentación es muy densa y compleja de asimilar.

Para realizar este curso, no necesitas leerla ya que explicaremos todo lo que necesitas saber para implementar REST paso a paso.

La arquitectura REST se basa en una serie de axiomas que debemos cumplir para realizar la comunicación entre el cliente y el proveedor del servicio:

No guardar el estado en el servidor

Nunca debemos utilizar cookies, sesiones o cualquier otro mecanismo para guardar información entre peticiones HTTP.

Cada petición HTTP **debe contener toda la información necesaria** para ejecutarla, por lo que nunca necesitaremos tener en cuenta peticiones anteriores para interpretar la solicitud.

EJEMPLO: Si para suministrar la información, requerimos un usuario y contraseña, dicho usuario y contraseña debe ser enviado en todas y cada una de las peticiones HTTP que realicemos y nunca almacenarlos en el servidor en cookies o sesiones.

Utilizar URLs para identificar cada recurso

Las siglas **U R L** significan Uniform Resource Locator (**Localizador uniforme de recursos**). Aunque normalmente asociamos este concepto a la *"dirección de la página"* el propósito para el cual fue diseñado es la **identificación única de un recurso**.

REST se basa en un uso correcto del protocolo HTTP y de todos los elementos que intervienen en la transferencia de información. **La arquitectura de URLs es vital en REST** y debemos utilizarla y organizarla de forma correcta.

Existen una serie de normas que debemos cumplir a la hora de conformar una URL de forma adecuada:

- Cada URL **debe identificar a un único elemento**.
- Un elemento **sólo debe tener una URL**
- **Nunca** debemos **indicar una acción** en una URL.
- **Nunca** debemos **especificar el formato** en el que deseamos obtener los datos en una URL.
- **Nunca** debemos **especificar un filtrado de información** en una URL.
- Todas las URLs deben tener una **jerarquía lógica**.

Algunos ejemplos de URLs correctas son:

http://localhost/web_service/productos/1478
http://localhost/web_service/productos/7845
http://localhost/web_service/clientes/577
http://localhost/web_service/clientes/577/factura/14

Vamos a ver también ejemplos de URLs incorrectas para comprender lo que no debemos hacer:

URL incorrecta	Motivo
http://localhost/web_service/productos/1478/editar	Nunca debemos indicar una acción en una URL. Como veremos a continuación utilizaremos los métodos HTTP para indicar la operación que deseamos realizar sobre el recurso.
http://localhost/web_service/productos/1478.pdf	Nunca debemos especificar el formato en el que deseamos obtener los datos en una URL. Al indicar la extensión .pdf, estamos indicando que deseamos el contenido en formato PDF, lo cual es incorrecto. Más adelante veremos que debemos utilizar una cabecera HTTP específica para indicar el formato con el que queremos que se nos devuelva el recurso solicitado.
http://localhost/web_service/productos/noticias/2017	Nunca debemos especificar un filtrado de información en una URL. Debemos realizar el filtrado mediante una query string de forma similar a la siguiente: http://localhost/web_service/productos/noticias?year=2017
http://localhost/web_service/factura/14/cliente/577	Todas las URLs deben tener una jerarquía lógica. Las facturas sólo tienen un cliente y los clientes tienen varias facturas. Por ese motivo, la factura no debe contener clientes, sino que los clientes deben contener facturas. Cuanto más lógica sea nuestra arquitectura de URLs, más fácil será de usar nuestro web service.

Utilizar métodos HTTP para indicar la acción a realizar sobre el recurso

Normalmente, en nuestras comunicaciones HTTP utilizamos el método GET o el método POST. Sin embargo **el protocolo HTTP define métodos adicionales** a estos, que pretenden cubrir el espectro completo de acciones que se pueden definir para un recurso.

Debido a que REST se basa por completo en el protocolo HTTP, **utiliza los métodos definidos en el mismo para indicar qué acción se va a realizar sobre el recurso solicitado**.

Los métodos HTTP que vamos a utilizar en nuestro web service REST son los siguientes:

Método	Acción
GET	Recupera los datos del recurso especificado por la URL. Si deseamos filtrar estos datos, utilizaremos parámetros enviados mediante una query string.
POST	Inserta un nuevo recurso. Especificaremos los datos del nuevo recurso mediante el envío de parámetros HTTP.
PUT	Modifica todos los datos del recurso especificado por la URL. Especificaremos los nuevos valores de todos los datos del recurso mediante el envío de parámetros HTTP.
PATCH	Modifica algunos datos del recurso especificado por la URL. Especificaremos los nuevos valores de los datos que deseamos modificar mediante el envío de parámetros HTTP.
DELETE	Elimina el recurso especificado por la URL.

Si recibimos en nuestro web service una llamada HTTP a la siguiente URL:

http://localhost/web_service/productos/1478

Mediante el método **DELETE**, nos están solicitando que **eliminemos el producto 1478**.

Indicar el formato en el que deseamos obtener la información

Cuando solicitamos los datos de un recurso (utilizando el método GET), la información nos podría ser devuelta en muchos formatos diferentes: PDF, HTML, CSV...

Para que el web service, sea lo más versatil posible y encaje perfectamente con cualquier aplicación web externa, suele dar la posibilidad de **proporcionar los datos en varios formatos distintos**. De modo que **es el cliente** el que indicará en la llamada HTTP **en qué formato desea recibir los datos**.

HTTP dispone de una cabecera específica para indicar el formato en el cual deseamos recibir el resultado de la solicitud. Se trata de la **cabecera Accept**.

Dentro de la cabecera Accept, indicaremos el **tipo MIME** del formato que esperamos recibir. Podemos especificar varios tipos MIME, separados por comas por orden de preferencia, por si nuestra primera opción no está disponible.

Si recibimos en nuestro web service una llamada HTTP a la siguiente URL:

http://localhost/web_service/productos/1478

Mediante el método **GET**, y la cabecera HTTP **Accept** contiene el siguiente valor:

application/pdf, text/html

Nos están solicitando que proporcionemos los datos del producto 1478**en formato PDF** o, si esta opción no existe, **en formato HTML**.

Aunque existen infinidad de formatos en los que solicitar información, el estándar de facto es proporcionar los datos en **XML** o **JSON**. Por ello, los principales tipos MIME que debemos solicitar a la hora de consumir un servicio REST o que debemos proporcionar a la hora de desarrollarlo son los siguientes:

Tipo MIME	Formato
text/xml	Formato XML
application/hal+xml	Formato XML con HATEOAS (Lo estudiaremos a continuación)
application/json	Formato JSON
application/hal+json	Formato JSON con HATEOAS (Lo estudiaremos a continuación)

Utilizar códigos de estado HTTP para indicar el resultado de la operación

HTTP dispone de una lista de códigos de estado para **indicar el resultado de una petición HTTP**.

Aunque muchos de ellos se utilizan con frecuencia, como el 404 que indica que el recurso no existe, o el 200 que indica que los datos se enviaron con normalidad, muchos otros, no son tan conocidos.

Todo servicio REST **debe utilizar los códigos de estado HTTP de forma adecuada**, indicando siempre el resultado de la operación solicitada. A continuación puedes ver la lista de códigos de estado que utilizaremos para nuestros web service en REST:

Código	Significado	Utilización
200	OK	Respuesta estándar para peticiones correctas. Lo utilizaremos en peticiones GET en la que devolvemos los datos del recurso solicitado.
201	Created	La petición ha sido completada y, como resultado, se ha creado un nuevo recurso. Lo utilizaremos en peticiones POST en la que creamos el recurso inidicado.
202	Accepted	La petición ha sido aceptada. Lo utilizaremos en peticiones PUT, PATCH o DELETE en la que realizamos la acción solicitada sobre el recurso.
400	Bad Request	La solicitud contiene sintaxis errónea.
401	Unauthorized	El cliente no ha suministrado una autentificación (usuario y contraseña) correcta.
403	Forbidden	El cliente se ha autentificado, pero no dispone de permisos para realizar la acción solicitada.
404	Not Found	El recurso no existe.
405	Method Not Allowed	El recurso no permite el método indicado en la solicitud.
406	Not Acceptable	El servidor no es capaz de devolver los datos en ninguno de los formatos aceptados por el cliente, indicados por éste en la cabecera "Accept" de la petición.

Hipermedia - HATEOAS

Como hemos comentado, los formatos más habituales a la hora de devolver la información de un recurso son **XML** o **JSON**.

Ejemplo: Datos devueltos de un cliente en formato XML

```
<?xml version="1.0"?>
<cliente>
    <id>1274</id>
    <nombre>Alberto Ruiz Jalón</nombre>
    <nif>16587482F</nif>
    <direccion>Paseo marítimo 34, 1ºD</direccion>
    <ciudad>Estepona</ciudad>
    <provincia>Málaga</provincia>
    <codigo_postal>29680</codigo_postal>
</cliente>
```

Al utilizar estos formatos, aparte de obtener los datos del recurso,**obtenemos también la estructura del mismo**. Conociendo esta estructura, no es necesario ningún tipo de documentación o explicación sobre qué datos se almacenan en cada recurso ya que podemos recuperarlos con una simple llamada GET.

Debido a la versatilidad de los formatos XML o JSON, podríamos incluso**introducir recursos relacionados** con el recurso solicitado dentro de la estructura, de forma que la aplicación web que consuma el servicio, **pudiese descubrir nuevos elementos** y navegar por ellos. A este concepto se le conoce como**hipermedia**.

Ejemplo: Hipermedia

- Solicitamos los datos de un cliente mediante una llamada GET.
- El web service nos devuelve un XML con los **datos del cliente**.
- Adicionalmente, dentro del XML se nos proporciona **información sobre las facturas del cliente y sus albaranes**.

Para desarrollar un servicio REST **es obligatorio implementar hipermedia**. Para ello, existe un formato estándar llamado **HATEOAS** que define cómo hacerlo.

La sintaxis del formato HATEOAS es la siguiente:

```
<links>
    <link rel="TIPO_RECURSO" title="NOMBRE_RECURSO" href="URL_RECURSO" />
    <link rel="TIPO_RECURSO" title="NOMBRE_RECURSO" href="URL_RECURSO" />
    <link rel="TIPO_RECURSO" title="NOMBRE_RECURSO" href="URL_RECURSO" />
</links>
```

Ejemplo: HATEOAS

```
<?xml version="1.0"?>
    <cliente>
    <id>1274</id>
    <nombre>Alberto Ruiz Jalón</nombre>
    <nif>16587482F</nif>
    <direccion>Paseo marítimo 34, 1ºD</direccion>
    <ciudad>Estepona</ciudad>
    <provincia>Málaga</provincia >
    <codigo_postal>29680</codigo_postal >
    <links>
        <link rel="factura" title="Factura A175"
href="http://localhost/web_service/facturas/A175" />
        <link rel="factura" title="Factura A247"
href="http://localhost/web_service/facturas/A247" />
        <link rel="albaran" title="Albarán 154" href="
http://localhost/web_service/albaranes/154" />
    </links>
    </cliente>
```

Implementando **hipermedia** con el estándar **HATEOAS**, dotamos a las aplicaciones web que consuman un servicio REST de la **capacidad de descubrir nuevos elementos y relacionarlos entre sí**

De esta forma, se podrían automatizar por completo las acciones de explotación de recursos de servicios web.

Generar la petición HTTP que llama al servicio REST

Como has podido comprobar, las peticiones HTTP que consumen servicios REST conllevan una cierta complejidad. No podemos generarlas directamente desde el navegador, ni utilizar la función file_get_contents ya que ambos métodos carecen de las opciones necesarias para describir por completo una solicitud HTTP a un servicio REST.

Para generar solicitudes HTTP que consuman servicios REST desde PHP, utilizaremos las funciones de la biblioteca **cURL**.

La biblioteca cURL

cURL es una biblioteca de funciones que permite **generar solicitudes HTTP complejas** y recuperar tanto el **contenido** como las **cabeceras HTTP** devueltas por el servidor.

Esta biblioteca se distribuye mediante una extensión de PHP que, debido a su amplia utilización, suele estar **instalada y activada por defecto** en todos los servidores.

No obstante, si tu servidor no tiene activada la extensión, deberás configurarla adecuadamente antes de utilizar las funciones que describiremos a continuación.

Puedes encontrar la documentación completa de la biblioteca cURL en la página web oficial de PHP:

http://php.net/manual/es/book.curl.php

La sintaxis para realizar una petición HTTP utilizando cURL es la siguiente:

```
$peticion = curl_init();
curl_setopt($peticion,CURLOPT_URL,$url);
curl_setopt($peticion,CURLOPT_RETURNTRANSFER,true);
$resultado = curl_exec($peticion);
```

$url es la URL a la cual queremos llamar.

$resultado contendrá el contenido de la respuesta del servidor.

De esta forma, podemos realizar una solicitud HTTP básica, pero para consumir servicios REST necesitamos indicar datos adicionales en la petición. A continuación veremos cómo indicar en cURL cada uno de estos datos para generar una llamada a un servicio REST completa:

Método
Por defecto, las peticiones HTTP se realizan mediante el método **GET**. No obstante, podemos personalizar la petición indicando los métodos **POST**, **PUT**, **PATCH** o **DELETE** añadiendo la siguiente línea de código a la solicitud: curl_setopt($peticion,CURLOPT_CUSTOMREQUEST,$metodo);

$metodo es el nombre del método que se empleará en la solicitud HTTP.

El siguiente fragmento de código solicitará la eliminación del producto 458 al web service:

```
$peticion = curl_init();
curl_setopt($peticion,CURLOPT_URL,'http://localhost/web_service/productos/458');
curl_setopt($peticion,CURLOPT_RETURNTRANSFER,true);
curl_setopt($peticion,CURLOPT_CUSTOMREQUEST,'DELETE');
$resultado = curl_exec($peticion);
```

Parámetros

Para crear o modificar recursos, necesitamos enviar los datos del recurso mediante **parámetros HTTP**.

Para indicar parámetros HTTP en la solicitud, añadiremos la siguiente línea de código:

```
curl_setopt($peticion,CURLOPT_POSTFIELDS,$parametros);
```

$parametros indica los parámetros a enviar en la solicitud. Debemos indicar los mismos en formato **query string** (aunque no se enviarán como una query string sino como parámetros HTTP).

Truco: Función http_build_query

Existe una función muy útil para formatear los parámetros: http_build_query.

Esta función **transforma un array en una query string** de forma automática.

En el siguiente ejemplo podrás ver cómo utilizarla.

El siguiente fragmento de código modificará los datos del producto 458:

```
$datos['stock'] = 45;
$datos['nombre'] = 'DKNY energizing vaporizador 100ml.';
$parametros = http_build_query($datos);
// $parametros contiene 'stock=45&nombre=DKNY+energizing+vaporizador+100ml.'

$peticion = curl_init();
curl_setopt($peticion,CURLOPT_URL,'http://localhost/web_service/productos/458');
curl_setopt($peticion,CURLOPT_RETURNTRANSFER,true);
curl_setopt($peticion,CURLOPT_CUSTOMREQUEST,'PATCH');
curl_setopt($peticion,CURLOPT_POSTFIELDS,$parametros);
$resultado = curl_exec($peticion);
```

Formato

En los servicios REST, es el cliente el que indica en qué formato desea recibir los datos mediante la **cabecera HTTP Accept**.

Para establecer la cabecera HTTP Accept, añadiremos la siguiente línea de código a la solicitud:

```
curl_setopt($peticion,CURLOPT_HTTPHEADER,['Accept: ' . $formato]);
```

$formato contendrá una cadena de texto con el tipo MIME o la lista de tipos MIME, separados por comas y ordenados por preferencia, que deseamos obtener.

El siguiente fragmento de código solicitará los datos del producto 458,**preferiblemente en formato XML**, o en **JSON si éste no está disponible**:

```
$formato = 'text/xml, application/json';
$peticion = curl_init();
curl_setopt($peticion,CURLOPT_URL,'http://localhost/web_service/productos/458');
curl_setopt($peticion,CURLOPT_RETURNTRANSFER,true);
curl_setopt($peticion,CURLOPT_HTTPHEADER,['Accept: ' . $formato]);
$resultado = curl_exec($peticion);
```

Usuario y contraseña

Como hemos comentado, en los servicios REST **está prohibido almacenar información de solicitudes o sucesos anteriores**. Por esa razón, si el web service requiere de una autenticación, mediante usuario y contraseña para poder ser utilizado, dicha autenticación **deberá ser enviada en todas y cada una de las peticiones HTTP**.

El estándar de HTTP, provee un mecanismo para el envío de credenciales, que utilizaremos para identificarnos en nuestros servicios REST.

Para indicar el usuario y la contraseña dentro de la petición HTTP, utilizaremos la siguiente sintaxis:

curl_setopt($peticion,CURLOPT_USERPWD,$usuario . ':' . $contrasena);

$usuario especifica el nombre de usuario.

$contrasena especifica la contraseña del usuario.

El siguiente fragmento de código solicitará los datos del producto 458, indicando el usuario y la contraseña que accederán al mismo. El servidor puede aceptar o rechazar la petición en función de si las credenciales son correctas o no.

```
$usuario = 'jmperez';
$contrasena = 'sdUy785';
$peticion = curl_init();
curl_setopt($peticion,CURLOPT_URL,'http://localhost/web_service/productos/458');
curl_setopt($peticion,CURLOPT_RETURNTRANSFER,true);
curl_setopt($peticion,CURLOPT_USERPWD,$usuario . ':' . $contrasena);
$resultado = curl_exec($peticion);
```

Ejemplo: Llamada HTTP completa a servicio REST mediante cURL

```
$url = 'http://localhost/web_service/productos/458';
$datos['stock'] = 45;
$datos['nombre'] = 'DKNY energizing vaporizador 100ml.';
$parametros = http_build_query($datos);
$formato = 'text/xml, application/json';
$usuario = 'jmperez';
$contrasena = 'sdUy785';

$peticion = curl_init();
curl_setopt($peticion,CURLOPT_URL,$url);
curl_setopt($peticion,CURLOPT_RETURNTRANSFER,true);
curl_setopt($peticion,CURLOPT_CUSTOMREQUEST,'PATCH');
curl_setopt($peticion,CURLOPT_POSTFIELDS,$parametros);
curl_setopt($peticion,CURLOPT_HTTPHEADER,['Accept: ' . $formato]);
curl_setopt($peticion,CURLOPT_USERPWD,$usuario . ':' . $contrasena);
$resultado = curl_exec($peticion);
```

Generar la petición HTTP - Parte 1

Generar la petición HTTP - Parte 2

Configurar el endpoint del web service

Como en todo web service, el primer paso que debemos dar para comenzar el desarrollo es **definir el endpoint** al cual llegarán todas las peticiones HTTP de los clientes.

En el caso de servicios REST, ya hemos visto que la URL definirá el recurso accedido. Esto significa que, en este caso, **la URL no especifica la estructura de carpetas y archivos del servidor**

Todas las URLs que accedan al endpoint, **deben ser procesadas por el mismo script PHP**, que se encargará de atender las peticiones.

El endpoint de nuestro servicio REST es el siguiente:

http://localhost/web_service/

Al cual le pueden llegar peticiones de distintas URLs:

- **http://localhost/web_service/productos/458**
- **http://localhost/web_service/facturas/A175**
- **http://localhost/web_service/albaranes/154**

Todas ellas, deben ser atendidas por el mismo script que se encuentra en:

http://localhost/web_service/index.php

Para desasociar la estructura de la URL de la estructura de carpetas del servidor, debemos utilizar un proceso denominado **reescritura de URLs**.

La reescritura de URLs consiste en establecer una serie de reglas que **transforman la URL solicitada en otra URL diferente**. Esta última URL será la que se ejecute en el servidor.

Mecanismo de reescritura de URLs

Todos los servidores web proveen mecanismos para reescribir URLs. Apache, utiliza el módulo **mod_rewrite** para este objetivo.

La reescritura de URLs es un proceso muy habitual para organizar adecuadamente la estructura de URLs de un sitio o aplicación web. Es por ello que el módulo mod_rewrite, suele estar **instalado y activado por defecto** en todos los servidores web Apache. No obstante, en el caso de que fuese así, deberíamos configurar el servidor web para activar el módulo.
en el caso de que fuese así, deberíamos configurar el servidor web para activar el módulo.

Si quieres aprender en profundidad a reescribir URLs consulta la documentación oficial de **mod_rewrite** en la página de Apache:

http://httpd.apache.org/docs/current/mod/mod_rewrite.html

Para configurar la reescritura de URLs para nuestro servicio REST, **utilizaremos un archivo .htaccess** para indicar al servidor las directivas de Apache correspondientes. El contenido de este archivo será el siguiente:

```
Options +FollowSymLinks
RewriteEngine on
RewriteRule ^web_service/(.*) web_service/index.php?recurso=$1 [QSA]
```

En este caso web_service es la ubicación de nuestro endpoint, cuando desarrolles tu propio servicio REST, deberás cambiar la ubicación por la correspondiente a tu endpoint.

Ejemplo: URL reescrita

Cuando llegue al servidor la siguiente URL:

http://localhost/web_service/productos/458

Apache la transformará en la siguiente:

http://localhost/web_service/index.php?recurso=productos/458

De esta forma, siempre se ejecutará el mismo script en todas las ocasiones y podremos obtener el recurso de la llamada contenido en la URL original.

Configurar el endpoint

Recuperar datos de la petición desde el web service

Una vez que todas las URLs que llaman al web service, son procesadas por el mismo script PHP, debemos recuperar desde dicho script **toda la información enviada por el cliente**.

A continuación veremos cómo recuperar cada uno de los datos suministrados por el cliente desde el script del endpoint:

Recurso

Tal y como hemos definido en nuestra reescritura de URLs, la porción de la URL que representa el recurso, se añadirá al **parámetro GET llamado recurso**.

Esto quiere decir, que para conocer el recurso que nos están solicitando solo debemos consultar:

```
$_GET['recurso']
```

Un cliente llama a nuestro web service con la siguiente URL:

http://localhost/web_service/productos/458

En este caso `$_GET['recurso']` contendrá `'productos/458'`.

Método

Para conocer la acción que debemos realizar sobre el recurso, podemos consultar el método utilizado en la solicitud HTTP. Para ello utilizaremos la siguiente sintaxis:

```
$_SERVER['REQUEST_METHOD']
```

Un cliente llama a nuestro web service mediante el método **DELETE**.

En este caso `$_SERVER['REQUEST_METHOD']` contendrá `'DELETE'`.

Parámetros

Por defecto, PHP está preparado para recuperar parámetros enviados mediante los métodos GET o POST, utilizando los arrays **$_GET** y **$_POST** respectivamente. No obstante, **no nos provee ninguna estructura de este tipo para recuperar parámetros enviados por PUT, PATCH** o cualquier otro método.

Para obtener parámetros enviados mediante métodos diferentes a GET o POST, **debemos consultar directamente el flujo de entrada a PHP**. Para hacer esto utilizaremos la siguiente sintaxis:

```
file_get_contents('php://input')
```

Esta instrucción recuperará los **parámetros HTTP** en formato **query string**.

Un cliente llama a nuestro web service y nos envía los siguientes parámetros:

parametro1 = valor1
parametro2 = valor2

Si ejecutamos la siguiente instrucción:

```
$parametros = file_get_contents('php://input');
```

La variable $parametros tendrá el siguiente valor:

```
'parametro1=valor1&parametro2=valor2'
```

Truco: Obtener parámetros POST del flujo de entrada

También podemos recuperar los parámetros enviados por **POST** mediante el flujo de entrada a PHP, por lo que lo más sencillo es seguir la siguiente norma:

- Los parámetros enviados por **GET** los obtendremos de **$_GET**.
- Los parámetros enviados por **cualquier otro método**, los obtendremos del **flujo de entrada a PHP**.

Formato

El cliente, debe suministrarnos en la cabecera **HTTP Accept**, el tipo MIME o lista de tipos MIME de los formatos que acepta. Podemos obtener la información suministrada por el cliente en esta cabecera mediante la siguiente sintaxis:

```
$_SERVER['HTTP_ACCEPT']
```


Esta componente del valor $_SERVER **contiene exactamente el mismo texto** que el cliente ha introducido en la **cabecera Accept**. El web service, deberá interpretarlo y **respetar el orden de preferencia** indicado por el cliente a la hora de suministrar los datos en un formato u otro.

Un cliente llama a nuestro web service y rellena la cabecera Accept con la siguiente texto:

application/hal+xml, text/xml

En este caso, $_SERVER['HTTP_ACCEPT'] tendrá el siguiente valor:

```
'application/hal+xml, text/xml'
```

Usuario y contraseña

Ya hemos visto, que el cliente debe suministrar sus credenciales **mediante el uso del protocolo HTTP**. Para recuperar el usuario y contraseña enviados, debemos usar la siguiente sintaxis:

```
$_SERVER['PHP_AUTH_USER'] // Contiene el nombre de usuario

$_SERVER['PHP_AUTH_PW'] // Contiene la contraseña
```

Un cliente llama a nuestro web service y proporciona las siguientes creedenciales:

Usuario: jmperez
Contraseña: sdUy785

En este caso, `$_SERVER['PHP_AUTH_USER']` tendrá el siguiente valor:

'jmperez'

Y `$_SERVER['PHP_AUTH_PW']` tendrá el siguiente valor:

'sdUy785'

Recuperar los datos de la petición

Generar la respuesta

Una vez recuperada toda la información que nos ha suministrado el cliente, el siguiente paso será **realizar las acciones solicitadas** y **generar la respuesta acorde**. La respuesta será procesada por el cliente para obtener los datos suministrados, o para conocer si la acción solicitada se ha realizado correctamente.

Vamos a ver las diferentes partes de las que se compone esta respuesta y cómo establecerlas:

Código de estado

En respuesta a todas las peticiones que lleguen a nuestro web service, **debemos indicar el código de estado HTTP** correspondiente. De esta forma, el cliente conocerá con exactitud si su petición ha tenido éxito o no y porqué.

Para establecer el código de estado HTTP con el que responderemos a la petición del cliente, utilizaremos la siguiente sintaxis:

```
http_response_code($codigo);
```

$codigo debe contener el código de estado con el que responderemos.

Un cliente llama a nuestro web service y nosotros ejecutamos la siguiente línea de código a la hora de generar la respuesta.

```
http_response_code(404);
```

El cliente recibirá el código **404** y entenderá que el recurso que ha solicitado, **no existe**.

A continuación puedes ver de nuevo la lista de códigos de estado que utilizaremos para nuestros web service en REST:

Código	Significado	Utilización
200	OK	Respuesta estándar para peticiones correctas. Lo utilizaremos en peticiones GET en la que devolvemos los datos del recurso solicitado.
201	Created	La petición ha sido completada y, como resultado, se ha creado un nuevo recurso. Lo utilizaremos en peticiones POST en la que creamos el recurso inidicado.
202	Accepted	La petición ha sido aceptada. Lo utilizaremos en peticiones PUT, PATCH o DELETE en la que realizamos la acción solicitada sobre el recurso.
400	Bad Request	La solicitud contiene sintaxis errónea.
401	Unauthorized	El cliente no ha suministrado una autentificación (usuario y contraseña) correcta.
403	Forbidden	El cliente se ha autentificado, pero no dispone de permisos para realizar la acción solicitada.
404	Not Found	El recurso no existe.
405	Method Not Allowed	El recurso no permite el método indicado en la solicitud.
406	Not Acceptable	El servidor no es capaz de devolver los datos en ninguno de los formatos aceptados por el cliente, indicados por éste en la cabecera "Accept" de la petición.

Formato

Ya que el cliente puede indicarnos una lista de tipos MIME para indicar los formatos que acepta, el web service debe indicar **en cuál de los formatos está enviando los datos** Para esto utilizaremos la cabecera HTTP **Content-Type**.

Para establecer la cabecera Content-Type utilizaremos la siguiente sintaxis:

```
header('Content-Type: ' . $formato . '; charset=' . $codificacion);
```

$formato debe contener el tipo MIME del formato de los datos devueltos.

$codificacion debe contener el nombre de la codificación de caracteres que estamos utilizando para devolver los datos.

Un cliente llama a nuestro web service y rellena la cabecera Accept con la siguiente texto:

application/hal+xml, text/xml

A la hora de generar la respuesta, el web service ejecuta la siguiente línea de código:

```
header('Content-Type: application/hal+xml; charset=UTF-8');
```

De esta forma, indicamos al cliente que estamos suministrándole la información en **XML con HATEOAS** utilizando una codificación de caracteres **UTF-8**.

Anotación: Formato de contenidos no textuales

Si el formato de la respuesta **no es en modo texto** (por ejemplo, vamos a devolver un **PDF** o una **imagen**), no es necesario indicar la codificación de caracteres.

En este caso, omitiremos esta parte en la cabecera quedándose la instrucción de la siguiente forma:

```
header('Content-Type: ' . $formato);
```

Contenido

Debemos preparar nuestro servicio REST para devolver la información solicitada en la **mayor cantidad de formatos posible**. De esta forma, ofreceremos un servicio versátil y de calidad a los clientes que consuman nuestro web service.

Es imprescindible implementar al menos **XML** o **JSON** (o ambos) para poder añadir **hipermedia** mediante HATEOAS a los datos. Recuerda que este es un **requisito obligatorio** para crear un servicio REST completo. También puedes suministrar datos en muchos otros formatos: CSV, XLS, PDF... Esto enriquecerá tu servicio y las posibilidades de consumo del cliente.

En cuanto al contenido de los datos, se nos pueden plantear dos situaciones:

ES UN RECURSO

Si la URL solicitada apunta a un recurso concreto, debemos enviar los datos de dicho recurso.

Ejemplo: Respuesta XML de un recurso

Un cliente llama a nuestro web service con la siguiente URL:

http://localhost/web_service/productos/458

El web service devolverá el siguiente contenido:

```
<?xml version="1.0"?>
<producto>
    <id>458</id>
    <nombre>Kit de manicura</nombre>
    <precio>24.99</precio>
    <stock>785</stock>
    <links>
        <link rel="factura" title="Factura A175"
href="http://localhost/web_service/facturas/A175" />
        <link rel="factura" title="Factura A247"
href="http://localhost/web_service/facturas/A247" />
    </links>
</producto>
```

ES UN LISTADO

Si la URL solicitada no apunta a un recurso, sino a la categoría del recurso, debemos suministrar los enlaces de todos los recursos disponibles en esa categoría mediante HATEOAS.

Ejemplo: Respuesta XML de un listado

Un cliente llama a nuestro web service con la siguiente URL:

http://localhost/web_service/productos/

El web service devolverá el siguiente contenido:

```
<?xml version="1.0"?>
<productos>
<links>
    <link rel="producto" title="Kit de manicura"
href="http://localhost/web_service/productos/458" />
    <link rel="producto" title="Shampoo 250 ml."
href="http://localhost/web_service/productos/157" />
    <link rel="producto" title="Eau de toilette vaporizador"
href="http://localhost/web_service/productos/745" />
</links>
</productos>
```

Anotación: Respuesta de peticiones HTTP de solicitud de acciones.

En el caso de que la petición HTTP no sea una solicitud de información (es decir, no sea una petición GET), **no es necesario enviar ningún contenido, ni establecer la cabecera Content-Type** para establecer su formato.

Para peticiones, POST, PUT, PATCH o DELETE, **enviaremos únicamente el código de estado** para informar del resultado del proceso.

Generar la respuesta

Recuperar datos de la respuesta

El último paso que nos queda por dar, ahora que nuestro web service está desarrollado, es **recuperar desde el cliente la respuesta** que éste ha suministrado a nuestra petición HTTP.

El cliente debe ser capaz de interpretar los datos devueltos y el código de estado para realizar, en consecuencia, las acciones que necesite. Recordemos que estábamos utilizando **cURL**, para realizar la solicitud, luego continuaremos utilizando las características de ésta biblioteca para obtener la información de respuesta.

Todas las acciones que realizaremos a continuación serán ejecutadas después de la petición HTTP, es decir, después de la siguiente línea:

```
$resultado = curl_exec($peticion);
```

Vamos a ver cómo recuperar cada uno de los datos de la respuesta del web service:

Código de estado

Para obtener el código de estado devuelto por el servidor, utilizaremos la siguiente sintaxis:

```
curl_getinfo($peticion,CURLINFO_HTTP_CODE)
```

Hemos solicitado la información del siguiente recurso al web service:

http://localhost/web_service/productos/989

Al consultar `curl_getinfo($peticion,CURLINFO_HTTP_CODE)`, nos devuelve el valor 404.

Esto significa que el recurso, **no existe**.

Formato

Para obtener el formato del contenido devuelto, utilizaremos la siguiente sintaxis:

```
curl_getinfo($peticion,CURLINFO_CONTENT_TYPE)
```

Esta instrucción nos devolverá el texto exacto que el web service ha incluido en la cabecera **Content-Type**.

Hemos solicitado la información del siguiente recurso al web service:

http://localhost/web_service/productos/458

Al consultar `curl_getinfo($peticion,CURLINFO_CONTENT_TYPE)`, nos devuelve el siguiente valor:

```
'application/hal+xml; charset=UTF-8'
```

Esto significa que el formato del contenido es **XML con HATEOAS** y utiliza la codificación de caracteres **UTF-8**.

Contenido

El contenido de la respuesta nos es devuelto directamente en la ejecución de curl_exec:

```
$resultado = curl_exec($peticion);
```

$resultado contendrá el contenido de la respuesta.

La interpretación de la respuesta, **depende evidentemente del formato del contenido**. Es nuestra labor procesar el formato en cuestión y extraer los datos.

Sin embargo, PHP nos proporciona **métodos para interpretar fácilmente los formatos más habituales**. Podemos procesar XML con la biblioteca **SimpleXML** y JSON con la función **json_decode**. Vamos a ver un ejemplo de cada uno de estos formatos:

Ejemplo: Procesar XML con SimpleXML

El web service nos devuelve el siguiente XML:

```
<?xml version="1.0"?>
<cliente>
    <id>1274</id>
    <nombre>Alberto Ruiz Jalón</nombre>
    <nif>16587482F</nif>
    <direccion>Paseo marítimo 34, 1ºD</direccion>
    <ciudad>Estepona</ciudad>
    <provincia>Málaga</provincia>
    <codigo_postal>29680</codigo_postal>
</cliente>
```

Ejecutaremos las siguientes instrucciones:

```
libxml_use_internal_errors(true);
$xml = simplexml_load_string($resultado);
```

$xml será un **objeto con la estructura del XML** indicado.

Si quisiéramos mostrar por pantalla el nombre del cliente, utilizaríamos la siguiente instrucción:

```
echo $xml -> nombre;
```

Y el navegador mostraría por pantalla:

Alberto Ruiz Jalón

Ejemplo: Procesar JSON con json_decode

El web service nos devuelve el siguiente JSON:

```
{
    "cliente":
    {
        "id": "1274",
        "nombre": "Alberto Ruiz Jalón",
        "nif": "16587482F",
        "direccion": "Paseo marítimo 34, 1ºD",
        "ciudad": "Estepona",
        "provincia": "Málaga",
        "codigo_postal": "29680"
    }
}
```

Ejecutaremos la siguiente instrucción:

```
$json = json_decode($resultado,true);
```

$json será un **array con la estructura del XML** indicado.

Si realizamos un print_r de $json, obtendremos el siguiente resultado:

```
Array
(
   [cliente] => Array
      (
         [id] => 1274
         [nombre] => Alberto Ruiz Jalón
         [nif] => 16587482F
         [direccion] => Paseo marítimo 34, 1ºD
         [ciudad] => Estepona
         [provincia] => Málaga
         [codigo_postal] => 29680
      )
)
```


Recuperar datos de la respuesta

Descarga: Ejemplo de servicio REST

Puedes descargar el ejemplo de servicio REST que hemos desarrollado durante los vídeos, haciendo clic aquí.

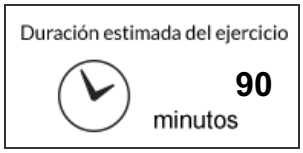
Hemos aprendido

En esta unidad hemos aprendido:

- Mediante un **web service**, una aplicación web externa a nuestro servidor, puede **recuperar información y realizar acciones mediante llamadas HTTP**.
- Para desarrollar un web service, es recomendable seguir un **estándar** para la comunicación entre el cliente y el proveedor de servicio. Los estándares más utilizados hoy en día son:
  - **SOAP**
  - **REST**
- El estándar **REST** se basa por completo en el **protocolo HTTP** para comunicar cliente y proveedor de servicio.
- La arquitectura REST se basa en una serie de principios:
  - **No guardar el estado** en el servidor (sesiones, cookies...)
  - Utilizar **URLs** para identificar cada recurso.
  - Utilizar **métodos HTTP** para indicar la acción a realizar sobre el recurso.
  - Indicar el **formato** en el que deseamos obtener la información.
  - Utilizar **códigos de estado HTTP** para indicar el resultado de la operación.
  - Implementar **hipermedia** mediante **HATEOAS**.
- Para consumir un servicio REST, utilizaremos **cURL** para realizar las solicitudes HTTP.
- Para centralizar todas las URLs que llegan al endpoint del servicio REST utilizaremos la **reescritura de URLs** del servidor web.
- Los formatos más utilizados para devolver los datos de un recurso son **XML** y **JSON**.

Ejercicios

Ejercicio: Implementar un servicio REST y un cliente que lo consuma

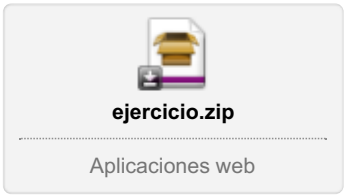


Para finalizar este ejercicio, deberás desarrollar una aplicación web que se comunique, mediante un servicio REST, con un servidor ajeno a ella. Será necesario desarrollar tanto el **servicio REST** como la **aplicación que lo consuma**.

- El **servicio REST** proveerá datos de productos y permitirá insertar nuevos productos o eliminar los existentes.
- La **aplicación cliente** dotará de funcionalidad a una interfaz que, sirviéndose del servicio REST, permitirá gestionar los productos.

Lo necesario para comenzar

1. Descarga el archivo **ejercicio.zip** y descomprímelo en el directorio público de tu servidor.
2. Contiene dos carpetas **webservice**, donde deberás implementar el servicio REST y **cliente**, donde deberás implementar el cliente.
3. Ambas aplicaciones, ya disponen de lo necesario para comunicarse entre sí mediante REST.
4. La aplicación cliente tiene la interfaz ya maquetada y sólo debes dotarla de funcionalidad.
5. Adicionalmente, encontrarás el archivo **bbdd.sql** con la estructura y datos de la base de datos a las que se conectará el web service.



Pasos a seguir

1. Realiza una petición al servicio REST desde el cliente, **solicitando el listado de productos**. Puedes utilizar la función **llamarREST** para comunicarte.
2. Programa el web service para que **devuelva la información del listado de productos**. Recuerda que **tendrás que validar tanto el usuario y la contraseña, como el resto de información de la solicitud** Debes devolver la respuesta correspondiente y el **código HTTP** adecuado en cada situación.
3. Con los datos devueltos por el web service, **integra los productos en la interfaz**
4. Dota de funcionalidad al formulario de **añadir producto**. Debe realizar la petición al servicio REST adecuada para realizar esta acción.

5. Desarrolla la funcionalidad del web service de añadir productos.
6. Dota de funcionalidad a los formularios de **eliminar productos**. Deben realizar la petición al servicio REST adecuada para realizar esta acción.
7. Desarrolla la funcionalidad del web service de eliminar productos.

Solución

Solución al ejercicio

Recursos

Enlaces de Interés

https://tools.ietf.org/html/rfc2616
https://tools.ietf.org/html/rfc2616
Estándar oficial del protocolo HTTP 1.1

https://tools.ietf.org/html/rfc7540
https://tools.ietf.org/html/rfc7540
Estándar oficial del protocolo HTTP 2.0

http://php.net/manual/es/book.curl.php
http://php.net/manual/es/book.curl.php
Documentación de la biblioteca cURL en la página oficial de PHP.

http://httpd.apache.org/docs/current/mod/mod_rewrite.html
http://httpd.apache.org/docs/current/mod/mod_rewrite.html
Documentación oficial de mod_rewrite en la página de Apache.

Preguntas Frecuentes

1. **¿Puedo crear un web service sin seguir ningún estándar?**

   Puedes, pero no es recomendable. Seguramente tengas problemas en el futuro si no proporcionas un estándar conocido ya que, los clientes te lo solicitarán.

   Te recomiendo utilizar siempre REST o SOAP para no tener que desarrollar de nuevo tu web service por una necesidad futura.

2. **En mi servicio REST, necesito que me envíen por GET un parámetro llamadorecurso. ¿Puedo hacerlo tal y como hemos configurado mod_rewrite?**

   En este caso, deberás reconfigurar tu .htaccess para utilizar un parámetro diferente que indique el recurso. En caso contrario, la reescritura generaría dos parámetros con el mismo nombre, lo cual sería incorrecto.

   Es muy importante asegurarse de que el parámetro al cual introducimos la parte de la URL que contiene el recurso, nunca sea utilizado en otra situación.

3. **¿Puedo utilizar HATEOAS en JSON?**

   Claro, sólo tienes que utilizar la misma estructura que hemos indicado para utilizar HATEOAS en XML. Puedes transcribirla directamente a formato JSON y utilizarla.

Glosario.

- **API**: Application Programming Interface (interfaz de programación de aplicaciones) es un conjunto de funciones y procedimientos que se ofrecen para que un tercero pueda interactuar con ellos. Por ejemplo: YouTube pone a nuestra disposición una API para obtener datos de vídeos e integrarlos en nuestros desarrollos.

- **CSV**: Valores separados por comas (comma-separated values) es un formato sencillo de representar datos en el que una coma, o punto y coma separaría las columnas y un salto de línea, las filas. Suele utilizarse con mucha frecuencia como formato de exportación o importación de datos, especialmente si trabajamos con hojas de cálculo.

- **CURL**: Biblioteca de funciones que nos permite generar peticiones HTTP detalladas.

- **HATEOAS**: Estándar que define cómo debemos implantar hipermedia en los recursos de nuestros servicios REST.

- **Hipermedia**: En la arquitectura REST, se denomina hipermedia a la definición de recursos relacionados con el recurso consultado. Implementando hipermedia, permitimos al cliente el autodescubrimiento de nuevos recursos.

- **REST**: Protocolo estándar para la creacion de web services basado exclusivamente en los métodos y características del protocolo HTTP.

- **SOAP**: Protocolo estándar, utilizado para el desarrollo de web services, que define cómo dos objetos pueden comunicarse por medio de intercambio de datos XML.

- **Web service**: Funcionalidad que desarrollamos en nuestra aplicación web para que, mediante llamadas HTTP, un tercero pueda solicitar información o realizar acciones en la misma.