**SERVICIOS WEB BASADO EN REST (RESTfull)**

Quienes hayan usado SOAP para WebServices, sabrán que es bien fácil de diseñar, pero algo complicado de consumir. Lograr el concepto de "lenguaje único XML" es un dolor de cabeza. Y más aún si el cliente es tan simple como JavaScript, dónde manejar XML de SOAP provocaría suicidos masivos... o no usar WebServices.

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

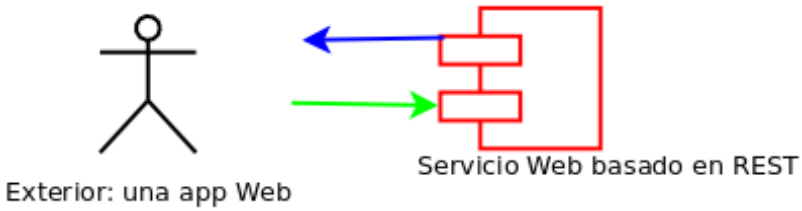
RESTfull es una propuesta muy interesante de Roy Fielding que permite manejar los servicios web con métodos definidos, manteniendo la simpleza del protocolo como XML, pero que cada servicio sea identificado únicamente con un solo URI.

# 1.Interfaz web.

Rest define cómo se establece una interacción entre sistemas basados en un navegador web y ***http,*** en otras palabras, nos va a permitir definir una ***interfaz web.***

**Interfaz:**

- Como exponernos al exterior para que puedan interactuar con nosotros. Para ello:
  a) Cómo recoger los datos para poderlos interpretar
  b) Como generar una salida para que nos la entiendas




**web:**

Red de páginas o recursos que se comunica por el protocolo ***http***

**http:**

Http es un protocolo sin estado. Cada mensaje http contiene en sí mismo toda la información necesaria para comprender la petición. El protocolo http establece en cada mensaje http un verbo o acción que encabeza la solicitud del mensaje Este puede ser:

1. GET Solicitar algo, es el tipo de mensaje por defecto.

2. POST Para facilitar datos al servidor. Lo manejamos principalmente en formularios.
3. DELETE Se facilita un dato al servidor para ser eliminado.
4. PUSH Se facilita datos al servidor para ser actualizados.

Estas acciones suelen ser comparadas con las operaciones asociadas a la tecnología de las bases de datos operaciones ***CRUD***: **CREATE**, **READ**, **UPDATE**, **DELETE**.

# 2 . URI y URL's amigables

Un URI es un ***identificador de recurso único.***

Una URL es el localizador de ese recurso.

Por ejemplo la URL http://localhost/tienda/producto.php, es también un URI que identifica un recurso (producto.php) y también proporciona los medios para localizar el recurso mediante la descripción de la forma de acceder a él (http://localhost/tienda/)

Una URL amigable es una URL que el cliente va a escribir en el navegador, aunque no existe tal cual en el servidor, sino que el servidor realiza una traducción.

Supongamos que en nuestro proyecto tenemos la URL:

```
http://localhost/reescritura/paginaAmigosConfianza.php?dato=pedro
```

Está claro, que a alguien le sería más fácil recordar y/o escribir:

```
http://localhost/reescritura/amigo/pedro
```

Para conseguir esto, lo que tendríamos que hacer es establecer una regla en el servidor de modo que traduzca una url en la otra. Esto se logra creando un fichero ***.htaccess*** con una serie de reglas, en el directorio a partir de dónde quiero URL amigables. Previamente en nuestro servidor web tengo que permitir sobreescritura de direcciones amigables:

**Pasos a seguir**

La sobreescritura de direcciones amigables la hace el módulo ***rewrite*** *de apache.*

- Primero miramos a ver si tenemos ese módulo instalado:

*a)* En XAMPP, ejecutamos el siguiente comando que nos listará los módulos instalados y buscamos rewrite.

```
/usr/sbin/apachectl -t -D DUMP_MODULES
```

*b)* Y en WAMPP miramos si tenemos en el fichero ***httpd.conf*** la siguiente línea sin comentar*:*

```
LoadModule rewrite_module modules/mod_rewrite.so
```

- En caso de no tenerlo instalado lo instalamos con:

```
sudo a2enmod rewrite
```

- Después reiniciamos el servicio

```
sudo service apache2 restart
```

- Hay que recordar que para que se apliquen las directivas del fichero ***.htaccess***, debemos tener habilitada para ese directorio (o para todos) la directiva ***AllowOverride All*** *en el fichero de configuración de Apache (****httpd.conf****).*

```
<Directory /var/www>
   AllowOverride All
</Directory>
```

- Ahora escribimos las siguientes reglas en el fichero .htaccess,

```
RewriteEngine on
RewriteBase /servicio_rest/
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule ^(.+)$ index.php?url=$1 [QSA,NC,L]
```

El significado de estas líneas es el siguiente:

1. Estamblecemos /servicio_rest/ como directorio base.

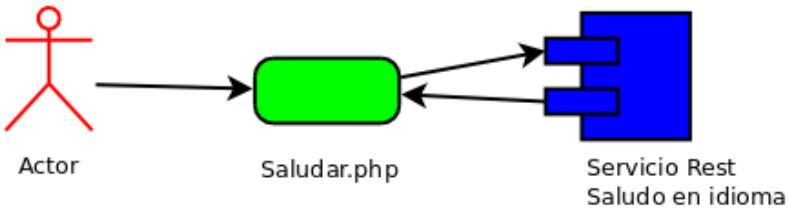
2. Las tres reglas siguiente establece que directorios (-d), ficheros (-f) y enlaces simbólicos (-l) que ya existen, sobre ellos no sea aplicada las reglas de reescritura.

3. El último punto, es la reescritura. En este caso tenemos la expresión regular ***^(.+)$***

    - Se puede leer como cualquier conjunto de caracteres al principio que contenga uno o más caracteres hasta el final.
    - Esta expresión regular será tomada seguida del dominio y el directorio establecido como base, es decir después de http://localhost/servicio_rest/ .....
    - Posteriormente el resto de dirección URL a partir del directorio base, se almacena en la variable $1
    - Y nos realiza la transformación siguiente: (1) Sería lo que escribe el cliente, y (2) lo que interpreta el servidor.
        - (1) http://localhost/servicio_rest/productos/1
        - (2) http://localhost/servicio_rest/index.php?url=productos/1

# 3 . Haciendo servicios REST

Vamos empezar haciendo un ejemplo en el que el servicio nos salude:




En el servidor lo que tenemos que hacer es recoger la solicitud que nos hacen y crear una respuesta. La solicitud puede ser GET POST PUT DELETE.

Para ello vamos a utilizar el Slim Framework que podemos descargar de la moodle.

**¿Qué es Slim Framework?**

Slim framwework es un micro framework para PHP que nos permite escribir rápidamente aplicaciones web y APIs. No tiene la potencia de otros frameworks de PHP como Laravel, Code Igniter, etc.. pero hace bastante bien su trabajo.

**Características:**

- Creador de rutas bastante potente
- Soporta métodos HTTP standard y personalizados.
- Parámetros de ruta con comodines y condiciones.
- Redirecciones de rutas, paros y saltos.
- Renderizado de plantillas y vistas personalizadas.
- Mensajes Flash.
- Encriptación segura de cookies con AES-256.
- Caché HTTP.
- Logging de accesos personalizado.
- Gestión de errores.
- Configuración sencilla.
- Requerimientos:
- Es necesario tener instalado PHP 5.3.0 o superior.

**Mi primer servicio web**

Una vez tenemos la carpeta “Slim” en nuestro alojamiento “amigable”, debemos crear un archivo “index.php” que utilice la librería Slim y en el que podamos ir creando nuestros servicios webs.
En nuestro primer caso el archivo index.php contendrá el siguiente código:

```
<?php
require 'Slim/Slim.php';

// El framework Slim tiene definido un namespace llamado Slim
// Por eso aparece \Slim\ antes del nombre de la clase.
\Slim\Slim::registerAutoloader();
// Creamos la aplicación
$app = new \Slim\Slim();
// Indicamos el tipo de contenido y condificación que devolveremos desde
el framework Slim
$app->contentType('application/json; charset=utf-8');

// Definimos las respuesta de la ruta base con un tipo de consulta GET
$app->get('/saludo', function () use ($app) {
        echo json_encode(array('msj' => 'Hola, esto es un saludo general'),
JSON_FORCE_OBJECT);
});
// Ejecutamos la aplicación creada
$app->run();
?>
```

También debemos poner en nuestro alojamiento “amigable” el ***.htaccess*** para que así lo sea:

```
RewriteEngine On
RewriteBase /Proyectos/Servicios_Web/REST/Ejercicio1/
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule ^(.+)$ index.php?url=$1 [QSA,NC,L]
```

Ahora mismo, ya tendríamos nuestro primer servicio REST por método GET creado.
Para consumir un servicio GET de REST desde php tenemos dos opciones:

a) Con la función *file_get_contents .* Nuestro ***saludar.php*** quedaría:

```
<?php
$response =
file_get_contents('http://localhost/Proyectos/Servicios_Web/REST/Ejercicio1/saludo');
// OJO:  Cuándo se crea JSON en el servidor, normalmente se meten carácteres
sucios !!
$decodedText=substr($response,3,strlen($response)-1);
$obj = json_decode($decodedText);

echo "Saludo que viene del servidor:". $obj->msj;
?>
```

b)  O con  **cURL.** Con cURL se podrían consumir cualquier tipo de servicio REST, en cambio con *file_get_contents,* sólo los de tipo GET*.* Para nuestro caso (saludar.php) quedaría:

```
<?php
//url contra la que atacamos

 $llamada = curl_init('http://localhost/Proyectos/Servicios_Web/REST/Ejercicio1/saludo');

 //a true, obtendremos una respuesta de la url, en otro caso,

 //true si es correcto, false si no lo es

 curl_setopt($llamada, CURLOPT_RETURNTRANSFER, true);

 //establecemos el verbo http que queremos utilizar para la petición

 curl_setopt($llamada, CURLOPT_CUSTOMREQUEST, "GET");

//obtenemos la respuesta

 $response = curl_exec($llamada);

 // Se cierra el recurso CURL y se liberan los recursos del sistema

 curl_close($llamada);

 if(!$response) {

    echo “Error al consumir el servicio Web”;

 }else{
```

```
    $decodedText=substr($response,3,strlen($response)-1);

    $obj = json_decode($decodedText);

    echo "<br/>Saludo que viene del servidor:". $obj->msj;

  }
```

Para crear los servicios  POST, DELETE y PUT, añadiríamos al index php:

```
$app->post('/saludo/nuevo', function () use ($app) {

        echo json_encode(array('msj' =>"Datos recibidos:" .$_POST["saludo1"]. " y ".
$_POST["saludo2"]), JSON_FORCE_OBJECT);

});

$app->delete('/elimino/:id', function ($id) use ($app) {

        echo json_encode(array('msj' =>"Eliminando el saludo con id:" .$id),
JSON_FORCE_OBJECT);

});

$app->put('/adapto/:id', function ($id) use ($app) {

          echo json_encode(array('msj' =>"Adaptando el saludo con id:" .$id),
JSON_FORCE_OBJECT);

});
```

Y con cURL lo consumiríamos (saludar.php):

```
function respuesta($res){

          if(!$res) {

                    echo "Error consumiendo el servicio Web";

           }else{

                    $decodedText=substr($response,3,strlen($res)-1);

                    $obj = json_decode($decodedText);

                    echo "<br/>Saludo que viene del servidor:". $obj->msj;

         }

}

// Creamos array para mandar los datos

$datosPost=['saludo1'=>'dwes',  'saludo2' =>'abc123.'];

 curl_setopt($llamada, CURLOPT_URL,
"http://localhost/Proyectos/Servicios_Web/REST/Ejercicio1/saludo/nuevo");

curl_setopt($llamada, CURLOPT_POST, true);

// enviamos el array data

 curl_setopt($llamada, CURLOPT_POSTFIELDS, http_build_query($datosPost));

$response=curl_exec($llamada);

respuesta($response);
```

```
curl_setopt($llamada, CURLOPT_URL,
"http://localhost/Proyectos/Servicios_Web/REST/Ejercicio1/elimino/3");

curl_setopt($llamada, CURLOPT_CUSTOMREQUEST, 'DELETE');

respuesta(curl_exec($llamada));


curl_setopt($llamada, CURLOPT_URL,
"http://localhost/Proyectos/Servicios_Web/REST/Ejercicio1/adapto/2");

curl_setopt($llamada, CURLOPT_CUSTOMREQUEST, 'PUT');

respuesta(curl_exec($llamada));
```