Servicios web

Servicios web

- Puede ocurrir que **las aplicaciones que desarrollemos necesiten compartir información con otras.**
- Por ejemplo, imaginemos una tienda online. Manejará datos sobre productos, precios, descripción, etc. Los proveedores de la tienda trabajan con información muy parecida. Nos facilitaría mucho el trabajo poder acceder directamente a la información del proveedor.
- El proveedor, lógicamente no nos puede dar acceso a su base de datos, ya que tendría que dar acceso a todos sus clientes y eso supone un alto riesgo de que se produzcan errores (debes facilitarles la estructura de la BD).
- Por otro lado, es probable que la información que necesitemos esté disponible en la página del proveedor, pero, para que la pueda obtener un programa, éste tendría que contemplar un procedimiento para buscar el producto concreto dentro de las etiquetas HTML de la página y extraer su contenido.

Servicios web

- Los servicios web se crearon para facilitar este intercambio de información.
- Normalmente nos referimos con **Servicio web a una colección de procedimientos (métodos) a los que podemos llamar desde cualquier lugar de Internet** o de nuestra intranet, siendo este mecanismo de invocación totalmente independiente de la plataforma que utilicemos y del lenguaje de programación en el que se haya implementado internamente el servicio.
- Los clientes del servicio, no necesitan conocer la estructura interna de almacenamiento, no necesitan programar un mecanismo de acceso a la información. **El servicio web le proporciona un punto de acceso directo a la información que les interesa.**
- De esta forma, en el ejemplo de la tienda y los proveedores, estos podrían ofrecer a sus clientes un servicio en el que, por ejemplo, enviando el código de un producto pudiéramos obtener el nombre, la descripción, el precio, etc.

Servicios web

Características de los servicios web:

- Deben ser accesibles desde la web. (usan el protocolo HTTP/HTTPS)
- Debe contener una descripción de sí mismo.
- Debe poder ser localizado.
- La respuesta que obtenemos no es una página web sino la información que se pidió.
- Al usar el protocolo HTTP funcionan sobre cualquier servidor web.

Servicios web

- **El cliente y el servidor de un servicio web, deberán ponerse de acuerdo en la forma en que se van a enviar las peticiones y las respuestas.** Por ejemplo, ¿cómo vamos a indicar que queremos el precio de un determinado producto? Y a su vez, ¿cómo envía el servidor la respuesta?
- Para responder a esta pregunta **existen diferentes protocolos para intercambiar información**. Por ejemplo **SOAP usa XML** para ese intercambio.
- El cliente puede conocer el nombre del método del servidor al que va a pedir la información (getPrecio), y que debe enviarle un parámetro que sea el código del artículo, pero si no lo sabe resulta de gran utilidad que hubiera una forma donde poder consultar los métodos que ofrece el servidor y cómo se deben usar.
- Para resolver esta cuestión, se desarrolló un lenguaje basado en XML llamado **WSDL que sirve para describir los servicios web, indicando cómo se debe acceder a un servicio y cómo usarlo.**

Servicios web: SOAP

- SOAP = Simple Access Protocol
- Es un protocolo que **utiliza XML para intercambiar información entre aplicaciones.**
- Normalmente utilizaremos SOAP para conectarnos a un servicio e invocar métodos remotos, aunque puede ser utilizado de forma más genérica para enviar cualquier tipo de contenido.
- Está especialmente diseñado para trabajar sobre HTTP.
- Tenemos varias formas de usar Soap en PHP: Soap PHP 5, NuSoap y Pear::Soap. En los ejemplos vamos a usar Soap nativo.

Servicios web: SOAP

Podemos distinguir dos tipos de mensajes según su contenido:

- **Mensajes orientados al documento:** Contienen cualquier tipo de contenido que queramos enviar entre aplicaciones.
- **Mensajes orientados a RPC:** Este tipo de mensajes servirá para invocar procedimientos de forma remota (Remote Procedure Calls). Es un tipo más concreto del tipo anterior, ya que en este caso como contenido del mensaje especificaremos el método que queremos invocar junto a los parámetros que le pasamos, y el servidor nos deberá devolver como respuesta un mensaje SOAP con el resultado de invocar el método.

Servicios web: SOAP

WSDL

- Una vez que hayamos creado un servicio web, podemos programar el correspondiente cliente y comenzar a utilizarlo.
- Si el servicio lo hemos creado nosotros, sabemos cómo acceder a él: en qué URL está accesible, qué parámetros recibe, cuál es la funcionalidad que aporta, y qué valores devuelve.
- Sin embargo, si queremos es que el servicio web sea accesible a aplicaciones desarrolladas por otros programadores, deberemos indicarles cómo usarlo creando un documento WSDL que describa el servicio.
- **WSDL (Web Services Description Language)** es un lenguaje basado en XML que utiliza unas reglas determinadas para generar el documento de descripción de un servicio web. Una vez generado ese documento, se suele poner a disposición de los posibles usuarios del servicio (normalmente se accede al documento WSDL añadiendo **?wsdl** a la URL del servicio).

Existen herramientas que generan el documento wsdl de forma automática

Servicios web: **Ejemplo**

- Crearemos un servicio en el que obtendremos una frase aleatoria en inglés de un conjunto de frases, y después accederemos a él. Para ello tenemos:
- **Fichero Library.php:** es la clase en la que se encuentra la función que vamos a ofrecer como servicio.
- **Fichero usoDirectoClase.php:** este script es una prueba de como accederíamos de forma normal al método de la clase, creando un objeto de la misma y accediendo a la función. Pero el ejemplo se trata de crear y usar un servicio por tanto necesitamos otros ficheros:
- **Fichero serverSOAP.php:** script que crea el servidor del servicio. Para ello:
  - Incluimos la clase en la que se ha definido el servicio (Library.php).
  - Guardamos en una variable el array de opciones que pasamos al servidor, en este caso la uri donde está el servicio web.
  - Creamos el servicio mediante la clase SoapServer, siendo el primer parámetro null ya que no estamos utilizando fichero WSDL.
  - Activamos el servicio mediante el método handle().
- **Fichero cliente.php:** crea un cliente del servicio mediante la clase clientSoap, pasando como parámetros la URI del servicio y el nombre del archivo que crea el servidor con su ruta absoluta. Después se hace una llamada a la función ofrecida y se comprueba que ha funcionado mostrando el resultado.

IMPORTANTE: Para poder utilizar la clase SOAP en PHP, hay que editar el fichero php.ini y quitar el punto y coma (si lo tiene) en la directiva **extension=soap**

Servicios web: **Ejemplo**

En el fichero “Library.php” se encuentra definida una clase con un método que devuelve un elemento de un array obtenido de forma aleatoria.
Este es el método que vamos a ofrecer como servicio.

```php
<?php
/*
 * clase en la que está definida una función que vamos a ofrecer como servicio.
 * La función array_rand devuelve (en este caso) una opción aleatoria del array
 * https://www.php.net/manual/es/function.array-rand
 */
class Library {
    //eightBall: evuelve un mensaje aleatorio de un array de mensajes
    public function eightBall() {
     $options = array(
      "Without a doubt",
      "As I see it, yes",
      "Most likely",
      "Reply hazy, try again",
      "Better not tell you now",
      "Concentrate and ask again",
      "Don't count on it",
      "Very doubtful");
     return $options[array_rand($options)];
    }
}
```

```php
<?php

include('Library.php');
$lib = new Library();
$response = $lib->eightBall();
echo $response;
```

En el fichero “usoDirectoClase.php” creamos un objeto de la clase Library e invocamos al método de la forma habitual.

Servicios web: **Ejemplo**

En el fichero “serverSOAP.php” creamos el servidor del servicio (objeto de la clase SoapServer).
Si lo ejecutamos directamente no hace nada, ya que solo crea el servicio y lo activa.

```php
<?php
    //incluimos la clase donde está definido el servicio
    include('Library.php');
    /* guardamos en la variable $options el array de opciones que pasaremos al
     * servidor, en este caso la uri donde está el servicio
     * 'uri' =>'directorio donde está el servicio'
     */
    $options = array('uri' => 'http://localhost/DWES_P_U1/27_ServiciosWeb/00_servicioWeb')
    /* creamos el servicio mediante la clase SoapServer, siendo el primer
     * parámetro null ya que no estamos usando un fichero WSDL
     */
    $server = new SoapServer(NULL, $options);

    /* Establecemos la clase donde está el servicio (nombre de la clase, no del
     * fichero php
     */
    $server->setClass('Library');
    // ponemos en marcha el servicio
    $server->handle();
```

Servicios web: **Ejemplo**

En el fichero “cliente.php” creamos el cliente o consumidor del servicio (objeto de la clase SoapClient) y llamamos al método que nos ofrece. Mostrando el resultado.
Observa en el código que no incluimos ni la clase ni el archivo servidor del servicio.

```php
<?php
/* vamos a crear un cliente del servicio mediante la clase ClientSoap a la que
 * le pasamos como parámetros la uri del servicio y el nombre del archivo que
 * tiene el servidor del servicio con su ruta.
 * 'uri' => 'directorio donde está el servicio'  => identifica al recurso
 * 'location' => 'ruta absoluta del fichero que tiene el objeto SoapServer'
 *
 * Después se hace una llamada al método ofrecido y mostramos el resultado.
 */

$options = array('uri' => 'http://localhost/DWES_P_U1/27_ServiciosWeb/00_servicioWeb',
                 'location' => 'http://localhost/DWES_P_U1/27_ServiciosWeb/00_servicioWeb/serverSOAP.php');

try{
    $cliente = new SoapClient(null, $options);
    $response = $cliente->eightBall();
    echo $response;
} catch (SoapFault $e) {
    echo "ERROR: " . $e->getMessage();
}
```

Servicios web: SOAP

WSDL: ¿cómo funciona?

- El servidor entrega al cliente el archivo WSDL que nos indica los métodos y propiedades que en ese servicio que están disponibles.
- El cliente hace la petición al servidor en el formato que espera el servidor según las especificaciones del fichero WSDL en el que se indican los parámetros y tipo de parámetros que acepta.
- El servidor entrega el resultado de la consulta.

Puedes ver un ejemplo online aquí:
https://svn.apache.org/repos/asf/airavata/sandbox/xbaya-web/test/Calculator.wsdl

Servicios web: SOAP

Estructura del documento WSDL:

- **<definitions>:** comienzo del documento. Esta etiqueta incluye a todas las demás.
- **<types>:** se definen los tipos de datos que se usan en el servicio web.

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:soap-env="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:tns="http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWSDL/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  targetNamespace="http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWSDL/">
    <wsdl:types>
        <xsd:schema targetNamespace="http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWSDL/"/>
    </wsdl:types>
```

Servicios web:

```
<wsdl:portType name="ServerPortType">
    <wsdl:operation name="sumar">
        <wsdl:documentation>Suma dos números y devuelve el resultado</wsdl:documentation>
        <wsdl:input message="tns:sumarRequest"/>
        <wsdl:output message="tns:sumarResponse"/>
    </wsdl:operation>
    <wsdl:operation name="restar">
        <wsdl:documentation>Resta dos números y devuelve el resultado.</wsdl:documentation>
        <wsdl:input message="tns:restarRequest"/>
        <wsdl:output message="tns:restarResponse"/>
    </wsdl:operation>
</wsdl:portType>
```

Estructura de un documento WSDL:

- Las etiquetas **“<portype>”** contienen una lista de operaciones. Cada etiqueta <portype> contiene un atributo name.
- En los documentos WDSL las funciones que se crean en un servicio web se llaman **“operaciones” (<operations>).**
- En cada función (operación) definida se indica su nombre (name), que corresponde con el nombre de la función que ofrece y la lista de parámetros de entrada y salida correspondientes (definidos en los mensajes).
- Además, en función del tipo de operación de que se trate, contendrá:
  - Un elemento input para indicar funciones que no devuelven valor (su objetivo es sólo enviar un mensaje al servidor).
  - Un elemento input y otro output **(en este orden)** para funciones que reciben algún parámetro, se ejecutan, y devuelven un resultado.
- Los elementos input y oputput contienen un atributo **“message”** para hacer referencia al mensaje correspondiente.

Servicios web: SOAP

```
<wsdl:message name="sumarRequest">
    <wsdl:part name="n1" type="xsd:float"/>
    <wsdl:part name="n2" type="xsd:float"/>
</wsdl:message>
<wsdl:message name="sumarResponse">
    <wsdl:part name="sumarReturn" type="xsd:float"/>
</wsdl:message>
<wsdl:message name="restarRequest">
    <wsdl:part name="n1" type="xsd:float"/>
    <wsdl:part name="n2" type="xsd:float"/>
</wsdl:message>
<wsdl:message name="restarResponse">
    <wsdl:part name="restarReturn" type="xsd:float"/>
</wsdl:message>
```

Estructura de un documento WSDL:

- **<message>:** se definen los métodos y parámetros para realizar la operación, o los valores que devuelve.
- Cada servicio web tiene dos mensajes: entrada (request) y salida (response).
- En la entrada se definen los parámetros que va a recibir el servicio web en una petición, y en la salida se definen los datos que el servicio va a devolver.
- Cada mensaje contiene cero o más parámetros **“<part>”** uno para cada parámetro. Describe los datos que se intercambian en la petición y respuesta del servicio.
- A su vez, cada parámetro se asocia con un tipo de dato concreto.

Servicios web: SOAP

- En el elemento **<binding>** es dónde se dan detalles de como una operación <portype> se va a transmitir.
- Para el protocolo SOAP, el enlace es **<soap: binding>** , y el transporte es mensajes SOAP sobre el protocolo HTTP.
- WDSL incluye extensiones para SOAP 1.1 que permiten definir detalles específicos para este protocolo:
  - **SOAP:binding**
  - **SOAP:operation**
  - **SOAP:body**

Servicios web: SOAP

- **SOAP:binding:** este elemento indica que el enlace va a ser a través de SOAP.
  - El atributo **style** indica el estilo general del formato del mensaje. Normalmente se usa RPC.
  - El atributo **transport** indica el protocolo de transporte de los mensajes SOAP. El valor http://schemas.xmlsoap.org/soap/http indica transporte HTTP

```
<wsdl:binding name="ServerBinding" type="tns:ServerPortType">
    <soap-env:binding xmlns="http://schemas.xmlsoap.org/wsdl/soap/" style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="sumar">
        <soap-env:operation xmlns="http://schemas.xmlsoap.org/wsdl/soap/" soapAction="http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWS
        <wsdl:input>
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/
        </wsdl:input>
            <wsdl:output>
        <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="restar">
        <soap-env:operation xmlns="http://schemas.xmlsoap.org/wsdl/soap/" soapAction="http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWS
        <wsdl:input>
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/
        </wsdl:input>
        <wsdl:output>
        <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

Servicios web: SOAP

- **SOAP:operation:** indica el enlace de una operación específica. Debe tener el atributo soapAction con la URL de la función.

```
<wsdl:binding name="ServerBinding" type="tns:ServerPortType">
    <soap-env:binding xmlns="http://schemas.xmlsoap.org/wsdl/soap/" style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="sumar">
        <soap-env:operation xmlns="http://schemas.xmlsoap.org/wsdl/soap/" soapAction="http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWS
        <wsdl:input>
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/
        </wsdl:input>
            <wsdl:output>
        <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="restar">
        <soap-env:operation xmlns="http://schemas.xmlsoap.org/wsdl/soap/" soapAction="http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWS
        <wsdl:input>
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/
        </wsdl:input>
        <wsdl:output>
        <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

Servicios web: SOAP

- **SOAP:body:** especifica los detalles de los mensajes de entrada y salida, el estilo de codificación SOAP y el namespace asociado al servicio especificado de los mensajes de entrada y salida (input y output).

```
<wsdl:binding name="ServerBinding" type="tns:ServerPortType">
    <soap-env:binding xmlns="http://schemas.xmlsoap.org/wsdl/soap/" style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="sumar">
        <soap-env:operation xmlns="http://schemas.xmlsoap.org/wsdl/soap/" soapAction="http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWS
        <wsdl:input>
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/
        </wsdl:input>
            <wsdl:output>
        <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="restar">
        <soap-env:operation xmlns="http://schemas.xmlsoap.org/wsdl/soap/" soapAction="http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWS
        <wsdl:input>
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/
        </wsdl:input>
        <wsdl:output>
        <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

Servicios web: SOAP

Estructura de un documento WSDL:

- **<service>:** contiene una lista de elementos de tipo “port” (en nuestro ejemplo solo tenemos uno).
- Cada port indica la URL donde se puede acceder al servicio web.
- De esta forma se asocia la dirección del servicio con un elemento binding (como se va a transmitir la información) definido en el servicio web.

```
  <wsdl:service name="Server">
      <wsdl:documentation/>
      <wsdl:port name="ServerPort" binding="tns:ServerBinding">
          <soap-env:address location="http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWSDL/serverSOAP.php"/>
      </wsdl:port>
  </wsdl:service>
/wsdl:definitions>
```

Servicios web

- Crear un **servicio web que sume o reste dos números** y usarlo.
- En primer lugar no utilizaremos tampoco archivos WDSL. Teniendo en carpetas separadas el servidor del servicio y los clientes para que estén en localizaciones diferentes.
- En la carpeta en la que se crea el servicio y se ofrece tendremos dos ficheros:
  - Server.php: contiene la clase con los métodos (servicios) que se van a ofrecer.
  - serverSOAP.php: en él se crea el servidor del servicio y se activa.
- Una vez completo y con el servicio funcionando crearemos el WDSL y lo utilizaremos en nuestro servicio.

Rest

Servicios web: Rest

- Rest (**RE**presentational **S**tate **T**ransfer) es un modelo de arquitectura web basado en el protocolo HTTP para mejorar la comunicación entre el cliente y el servidor.
- RestFul Web Service o RestFul API: servicio web basado en Rest.
- Es una alternativa a SOAP, con mejor rendimiento y más sencillo de utilizar.
- Las API Rest se basan en el protocolo de aplicación HTTP.
- Permite la comunicación entre cliente y servidor en diferentes lenguajes y plataformas.

Servicios web: Rest

Principios básicos de Rest:

- Las comunicaciones entre el cliente y el servidor son STATELESS (sin estado).
- Usa los métodos HTTP definidos en el protocolo nativo: GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE, CONNECT (los cuatro primeros son los más importantes).
- Utiliza una **sintaxis universal** para identificar los recursos (elementos de información). En un sistema REST, cada recurso es direccionable únicamente a través de su URI (Uniform Resource Identifier).
- La información se puede enviar en distintos formatos (XML, JSON, …).

Servicios web: Rest

Principios básicos de Rest: Las comunicaciones entre el cliente y el servidor son STATELESS (sin estado).

- El protocolo HTTP no conserva el estado de la comunicación, y por tanto, cada comunicación se trata de forma independiente.
- Los servicios web Rest incluyen, dentro de las cabeceras y cuerpos HTTP de una petición, toda la información que el servidor necesita para generar la respuesta.
- Al no guardar el estado y tratar cada comunicación de forma independiente:
  - El rendimiento del servicio web es mejor.
  - El diseño y la implementación de los componentes por el lado del servidor es más sencillo.

Servicios web: Rest

Principios básicos de Rest: usa los métodos HTTP definidos en el protocolo nativo.

- Cada uno usado para una función específica:
  - POST: para crear un recurso en el servidor.
  - GET: para recuperar un recurso del servidor.
  - PUT: para cambiar el estado de un recurso, o para actualizarlo.
  - DELETE: para eliminar o borrar un recurso.

Servicios web: Rest

Principios básicos de Rest: Utiliza una sintaxis universal para identificar los recursos (elementos de información).

- Cada URI identifica un recurso.
- Las URI de los servicios web deben ser intuitivas.
- Una buena práctica es definirlas como una estructura al estilo de los directorios.

Servicios web: Rest

Principios básicos de Rest: La información se puede enviar en distintos formatos (sobre todo XML y JSON).

- Esto permite que el servicio sea utilizado por distintos clientes escritos en diferentes lenguajes, corriendo en diversas plataformas y dispositivos.

Servicios web: Rest

- El servidor nos devolverá un código de respuesta a la petición en el encabezado. Los más utilizados son:
  - **200 OK**. Satisfactoria.
  - **201 Created**. Un *resource* se ha creado. Respuesta satisfactoria a un *request* POST o PUT.
  - **400 Bad Request**. El *request* tiene algún error, por ejemplo cuando los datos proporcionados en POST o PUT no pasan la validación.
  - **401 Unauthorized**. Es necesario identificarse primero.
  - **404 Not Found**. Esta respuesta indica que el ***resource*** requerido no se puede encontrar (La URL no se corresponde con un *resource*).
  - **405 Method Not Allowed**. El **método HTTP** utilizado no es soportado por este *resource*.
  - **409 Conflict**. Conflicto, por ejemplo cuando se usa un PUT request para crear el mismo resource dos veces.
  - **500 Internal Server Error**. Un error 500 suele ser un error inesperado en el servidor.

Servicios web: Rest

- Los servicios Restful que creemos no se pueden probar directamente en el navegador, ya que directamente no podemos enviar peticiones PUT o DELETE.
- Se pueden utilizar programas como Postman, la extensión de Chrome Insomnia, o la librería curl en línea de comandos https://curl.haxx.se/
- Vamos a ver un ejemplo muy sencillo de creación de un servicio Restful en PHP y consumirlo desde la extensión insomnia.
- Podéis encontrar el ejemplo y el código en: https://www.codigonaranja.com/2018/crear-restful-web-service-php
- Posteriormente veremos como consumir un servicio REST externo.

Servicios web

Ejemplo:

- Este ejemplo simula la creación de un servicio REST en PHP para una aplicación de tipo blog.
- Descargamos el código del ejemplo y lo copiamos a un proyecto en el directorio htdocs.
- Creamos una base de datos llamada “blog” e importamos el archivo .sql del ejemplo.
- El ejemplo consta de tres ficheros:
  - **Config.php:** contiene los datos de configuración para la conexión a la base de datos.
  - **Utilis.php:** contiene funciones de usuario que necesitaremos al recibir las peticiones de servicio.
  - **Post.php:** contiene el código en el que se comprueba el tipo de petición recibida y se realizan las acciones pertinentes.

Servicios web

Ejemplo: archivo config.php

```php
/*
 * Contiene un array con los datos para la conexión a la BD.
 */
$db = [
    'host' => 'localhost',
    'username' => 'root',
    'password' => '',
    'db' => 'blog' //Cambiar al nombre de tu base de datos
];
```

Servicios web

Ejemplo: utils.php

- Contiene una función que devuelve la conexión a la BD usando PDO.
- Además hay otras dos funciones definidas:
- **getParams:** va a formar la cadena de parámetros que hay que pasar en una sentencia UPDATE en función de los que lleguen en la petición.
- **bindAllValues:** asocia los parámetros y su valor al statement de la consulta que se va a hacer a la BD.

```php
//Obtener parametros para updates
//convierte a string separado por comas los elementos del array
function getParams($input)
{
    $filterParams = [];
    foreach($input as $param => $value)
    {
        $filterParams[] = "$param=:$param";
    }
    return implode(", ", $filterParams);
}
```

```php
//Asociar todos los parametros a un sql
function bindAllValues($statement, $params)
{
    foreach($params as $param => $value)
    {
        $statement->bindValue(':'.$param, $value);
    }
    return $statement;
}
```

NOTA: *bindValue() es muy parecido a bindParam(). La diferencia está en que al usar bindParam() la variable se vincula al parámetro como una referencia y se evalúa en el momento en que se llama al método execute(). En cambio, con bindValue() la variable se vincula antes del método execute(), por lo que si se le asigna otro valor a la variable antes de llamar al método execute() no lo coge, mantiene el primero que se le asignó.*

Servicios web

Ejemplo: post.php

- Este script es el realiza las acciones necesarias en función de las peticiones que recibe.
- En primer lugar obtiene una conexión a la BD.
- Después va comprobando el tipo de petición recibida, usando para ello la variable superglobal $_SERVER['REQUEST_METHOD'].
- Si recibe una petición de tipo GET (obtener datos de la base de datos), pueden darse dos casos, que se quieran recuperar todos los post publicados en el blog o uno determinado, identificado por su id que llegará como parámetro.

```php
if ($_SERVER['REQUEST_METHOD'] == 'GET')
{
    //comprobamos si estamos solicitando un id específico de un post
    if (isset($_GET['id']))
    {
        //Mostrar un post
        $sql = $dbConn->prepare("SELECT * FROM posts where id=:id");
        $sql->bindValue(':id', $_GET['id']);
        $sql->execute();
        header("HTTP/1.1 200 OK");
        echo json_encode(  $sql->fetch(PDO::FETCH_ASSOC)  );
        exit();
    }
    else {
        //Mostrar lista de post
        $sql = $dbConn->prepare("SELECT * FROM posts");
        $sql->execute();
        $sql->setFetchMode(PDO::FETCH_ASSOC);
        header("HTTP/1.1 200 OK");
        //lo trasformamos a notación json
        echo json_encode( $sql->fetchAll()  );
        exit();
    }
}
```

Servicios web

Ejemplo: post.php

- Si, en cambio, recibe una petición de tipo POST (insertar datos en la base de datos), recoge los parámetros que le han llegado, forma la consulta y la ejecuta.
- Si se ha realizado correctamente, muestra en notación JSON los datos que han llegado como parámetros.

```php
if ($_SERVER['REQUEST_METHOD'] == 'POST')
{
    //$input = $_POST;
    //IMPORTANTE he usado $_REQUEST porque si lo pruebo desde insomnia
    //no le llega por post como estaba en el ejemplo original.
    $input = $_REQUEST;
    //print_r($input);
    $sql = "INSERT INTO posts
          (title, status, content, user_id)
          VALUES
          (:title, :status, :content, :user_id)";
    $statement = $dbConn->prepare($sql);
    bindAllValues($statement, $input);
    $statement->execute();
    $postId = $dbConn->lastInsertId();
    if($postId)
    {
        $input['id'] = $postId;
        header("HTTP/1.1 200 OK");
        //lo trasformamos a notación json
        echo json_encode($input);
        exit();
    }
}
```

Servicios web

Ejemplo: post.php

- Si, lo que recibimos es una petición de tipo DELETE (borrar un post concreto de la base de datos), recoge la clave del post a borrar que le ha llegado, forma la consulta y la ejecuta.
- Si se ha realizado correctamente, devuelve el encabezado con código 200 que indica que la petición se ha resuelto de forma correcta.

```php
if ($_SERVER['REQUEST_METHOD'] == 'DELETE')
{
    $id = $_GET['id'];
    $statement = $dbConn->prepare("DELETE FROM posts where id=:id");
    $statement->bindValue(':id', $id);
    $statement->execute();
    header("HTTP/1.1 200 OK");
    exit();
}
```

Servicios web

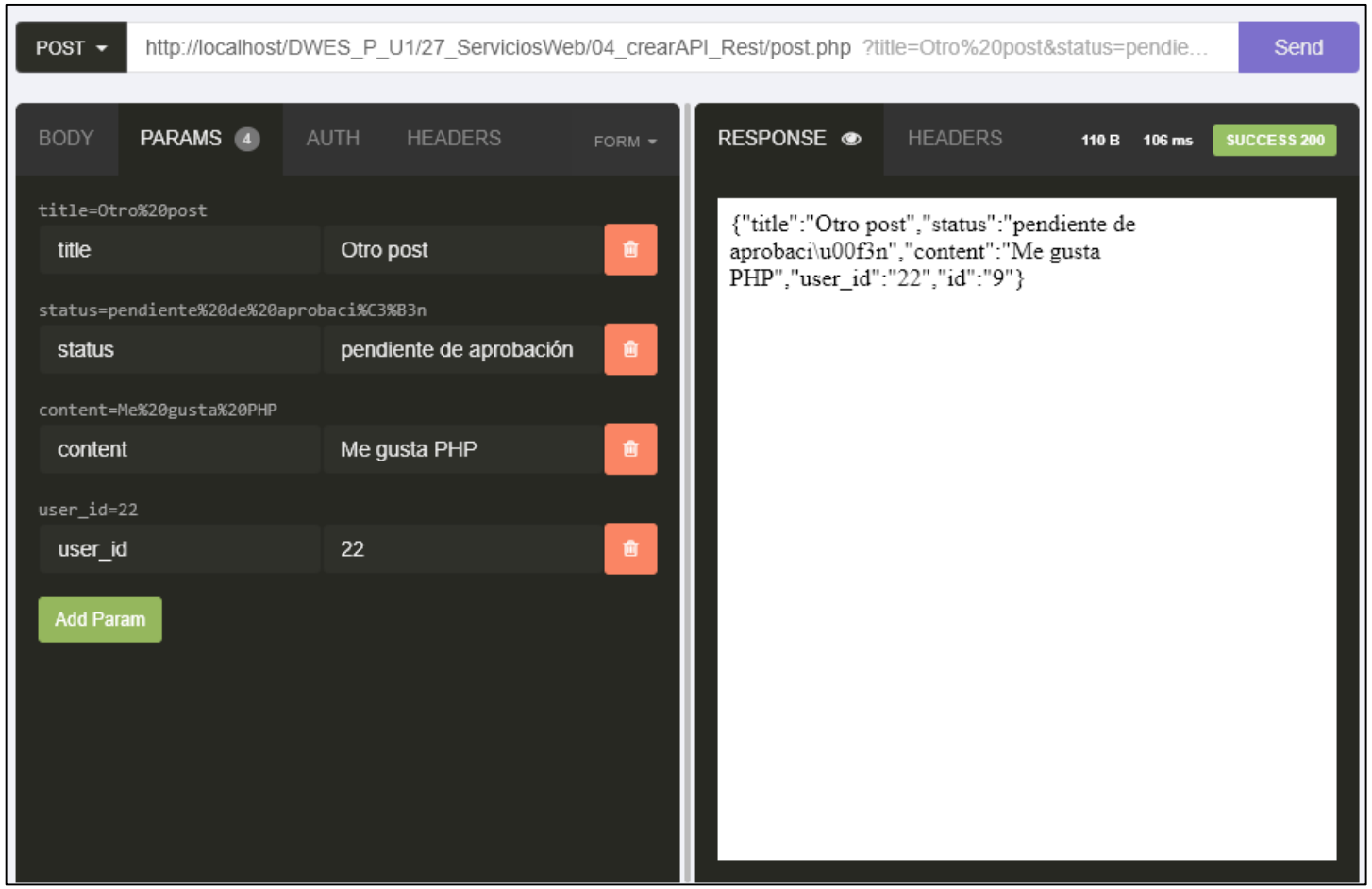
Ejemplo: post.php

- Por último, si lo que recibimos es una petición de tipo PUT (cambiar datos en la base de datos), recoge los parámetros que le han llegado, forma la consulta y la ejecuta.

```php
if ($_SERVER['REQUEST_METHOD'] == 'PUT')
{
    $input = $_GET;
    $postId = $input['id'];
    $fields = getParams($input);
    $sql = "
          UPDATE posts
          SET $fields
          WHERE id='$postId'
           ";

    $statement = $dbConn->prepare($sql);
    bindAllValues($statement, $input);
      echo $sql;
    $statement->execute();
    header("HTTP/1.1 200 OK");
    exit();
}
```

Servicios web



Servicios web

En el código del ejemplo faltan algunas validaciones genéricas (tal y como indican en el artículo que lo explica) que serían obligatorias en un servicio en funcionamiento real:

- Al hacer una petición sobre un id que no exista debería devolverse un error 404.
- Debería existir un mecanismo de autenticación para que solo usuarios autorizados puedan eliminar, actualizar o insertar información.

Autenticación en Rest

https://www.youtube.com/watch?v=r7tZehA2tJY

https://www.itdo.com/blog/cual-es-el-mejor-metodo-de-autentificacion-en-un-api-rest/

https://www.incibe-cert.es/blog/seguridad-oauth-2-0

Autenticación en Rest

- Debemos proteger el acceso a nuestra API para evitar que puedan borrar o manipular los recursos sin autorización.
- Cada petición al servicio debe llevar algún tipo de credencial de autenticación que verifique la identidad del usuario que quiere utilizar el servicio.
- Aunque podríamos crear nuestro propio protocolo de acceso, siempre es mejor utilizar los estándares que ya existen ya que están mejor preparados frente a posibles ataques, se actualizan y ofrecen soporte. Además herramientas como Postman incluyen opciones de autenticación.

Autenticación en Rest

Algunas opciones de autenticación:

- Autentificación básica.
- Autentificación basada en Id de usuario y token de acceso.
- Autentificación basada en API key.
- OAuth 2.0.

Autenticación en Rest

Algunas opciones de autenticación: Autentificación básica.

- Es la forma más sencilla (usuario y contraseña).
- Las credenciales van en la cabecera de la petición, por lo que son vulnerables a ataques informáticos aunque la información vaya codificada.
- No se debe utilizar sobre HTTP (HTTPS cifra la comunicación, es más seguro)

Autenticación en Rest

Algunas opciones de autenticación: Autentificación basada en Id de usuario y token de acceso.

- En este caso, el usuario se identifica la primera vez que accede al servicio con sus credenciales usuario y contraseña.
- El servidor genera un token único basado en esas credenciales que almacena en la base de datos y se lo envía al usuario para que desde ese momento se identifique con él y no con las credenciales.
- Los tokens suelen estar codificados con la fecha y la hora y pueden tener tiempo limitado de uso, tras el cual los usuarios deben volver a identificarse.

Autenticación en Rest

Algunas opciones de autenticación: Autentificación basada en API key.

- Este método implica configurar el acceso al servicio antes de poder empezar a utilizarlo.
- Tu servicio generará una clave (key) y una clave secreta (secret key) para cada usuario que quiera acceder a tu servicio.
- El cliente o usuario deberá enviar la clave y la clave secreta cada vez que necesite consumir el servicio.
- Este sistema es más seguro que los anteriores pero necesitas crear claves seguras y almacenarlas.

Autenticación en Rest

Algunas opciones de autenticación: OAuth 2.0

- OAuth = Open Authorization.
- Es un método de autenticación estándar seguro para aplicaciones de escritorio, móviles y web.
- Permite a aplicaciones de terceros acceder de forma limitada a los recursos de un servicio HTTP sin tener que desvelar sus credenciales.
- Dicho de otra forma define cómo un usuario puede compartir información del proveedor de un servicio con un cliente o consumidor, sin que este tenga acceso a las credenciales del usuario en el proveedor del servicio.
- Lo utilizan Google, Twitter, Facebook, etc.

OAuth

- Por ejemplo, si una aplicación "X" solicita a Google acceso a los calendarios del usuario “Pepe", Google pedirá a “Pepe" permiso indicándole qué aplicación es la que solicita el acceso y a qué información.
- Si el usuario “Pepe" concede permiso, la aplicación "X" podrá acceder a los datos que solicitó a través del servicio de Google.
- El funcionamiento sería:
  - La aplicación web se comunica con el servidor de autorización OAuth2, indicando la información a la que quiere acceder y el tipo de acceso a la misma.
  - El servidor de autorización OAuth2 requiere al usuario legítimo de la aplicación web a que inicie sesión con su cuenta de Google(si aún no lo ha hecho), y le redirige a una página en la que le pide su consentimiento para otorgar acceso a su información privada.
  - Si el usuario da su consentimiento, el servidor de autorización OAuth2 devuelve a la aplicación web que había solicitado el acceso un código de autorización.
  - Antes de poder acceder a la información privada del usuario, la aplicación web debe intercambiar ese código de autorización por otro código de acceso.
  - Utilizando el código de acceso, la aplicación puede utilizar el servicio de Google para gestionar la información privada del usuario, dentro de los límites de acceso que se han otorgado.
  - Los códigos de acceso tienen un tiempo de vida limitado. Cuando caducan, la aplicación ha de comunicarse de nuevo con el servidor de autorización OAuth2 para obtener un código de refresco.