Contenido

20/04/2020: Unidad 8: Servicios web.

En esta unidad se hace una introducción a los servicios web, muy utilizados en la actualidad.

Servicios Web

Los servicios web nos ofrecen una forma de intercambiar información entre dos equipos a través de una red informática.

Cuando usamos servicios web, el servidor puede ofrecer un punto de acceso a la información que quiere compartir, controlando de esta manera el acceso a la misma desde otras aplicaciones sin tener que dar, por ejemplo acceso a una base de datos.

Normalmente nos referimos con el término Servicio Web a un conjunto de métodos a los que podemos llamar desde cualquier sitio de internet. La forma de llamar al servicio es independiente de la plataforma que se use y del lenguaje de programación en el que se haya programado el servicio. Una de sus características es que utilizan el protocolo HTTP para el intercambio de información.

SOAP (Simple Object Access Protocol)

SOAP es un protocolo que usa el lenguaje XML para intercambiar información entre aplicaciones. Está especialmente diseñado para utilizar HTTP como protocolo de comunicación, aunque podría utilizar otro, pero en ese caso, ya no estaríamos trabajando con servicios web.

Usaremos SOAP para conectarnos a un servicio e invocar métodos remotos.

PHP nos ofrece diferentes formas de programar servicios SOAP que **nos evitan tener que tratar con las interioridades del protocolo SOAP.**

De las implementaciones de SOAP que podemos usar con PHP, cabe destacar tres: NuSOAP, PEAR::SOAP y PHP5 SOAP. Las tres nos permiten crear tanto un cliente como un servidor SOAP, pero existen algunas características que las diferencias:

- **PHP5 SOAP** es la implementación de SOAP que se incluye con PHP a partir de la versión 5 del lenguaje. Es una extensión nativa (escrita en lenguaje C) y por tanto más rápida que las otras posibilidades. Es la que utilizaremos en los ejemplos.
- **NuSOAP** es un conjunto de clases programadas en PHP que ofrecen muchas funcionalidades para utilizar SOAP. Funcionan también con PHP4, y permite generar automáticamente el documento WSDL correspondiente a un servicio web.
- **PEAR::SOAP** es un paquete PEAR (PHP Extension and Application Repository) que permite utilizar SOAP con PHP a partir de su versión 4. Al igual que NuSOAP, también está programado en PHP.

DESCRIPCIÓN DEL SERVICIO WSDL

Si el servicio lo hemos creado nosotros, podemos comenzar a utilizarlo **programando el correspondiente cliente**. Al haberlo creado nosotros, sabemos cómo acceder a él, es

decir, en qué URL está accesible, qué parámetros recibe, cuál es su funcionalidad, y qué valores devuelve.

Si queremos que nuestro servicio web sea accesible a aplicaciones desarrolladas por otros programadores, deberemos crear un documento WSDL que les indique cómo usar el servicio.

WSDL 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).

El objetivo de cada una de las secciones del documento WSDL es el siguiente:

- **types.** Incluye las definiciones de los tipos de datos que se usan en el servicio.
- **message.** Describe los datos que se intercambian en la petición y respuesta del servicio. Cada servicio web tiene normalmente dos mensajes, uno de entrada y otro de salida.
- **portType.** se definen las operaciones que se pueden realizar y los mensajes que están involucrados en ellas. Cada función se define dentro de su portType como una operación (operation).
- **binding.** Define cómo va a transmitirse la información de cada portType y el formato del mensaje.
- **service.** Contiene una lista de elementos de tipo port. Cada port indica dónde (en qué URL) se puede acceder al servicio web.

Vamos a verlos un poco más:

Etiqueta <types>

Existen servicios web sencillos que trabajan con tipos de datos simples (por ejemplo, strings, enteros etc.. Existen también servicios web más elaborados, que pueden requerir o devolver un array de elementos, o incluso objetos. **Para crear y utilizar estos servicios, hay que definir los tipos de elementos que se transmiten, ya que no son datos propios del XML Schema:** es decir, hay que indicar de qué tipo son los valores del array, o qué miembros poseen los objetos que maneja. La definición de tipos en WSDL se realiza utilizando la etiqueta types. Por ejemplo:

```
<types>
  <xsd:schema targetNamespace="http://localhost/dwes/ut6">
   <xsd:complexType name="direccion">
     <xsd:all>
       <xsd:element name="ciudad" type="xsd:string"/>
       <xsd:element name="calle" type="xsd:string"/>
       <xsd:element name="numero" type="xsd:string"/>
       <xsd:element name="piso" type="xsd:string"/>
```

```
      <xsd:element name="CP" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
  <xsd:complexType name="usuario">
   <xsd:all>
     <xsd:element name="id" type="xsd:int"/>
     <xsd:element name="nombre" type="xsd:string"/>
     <xsd:element name="direccion" type="tns:direccion"/>
     <xsd:element name="email" type="xsd:string"/>
   </xsd:all>
  </xsd:complexType>
 </xsd:schema>
</types>
```

En el código anterior**, se definen dos tipos de datos usando XML Schema: dirección y usuario.** Esta es la forma en que se definen en WSDL las clases para transmitir la información de sus objetos.

En WSDL, las clases se definen utilizando los tipos complejos de XML Schema. Al utilizar all dentro del tipo complejo, estamos indicando que la clase contiene esos miembros, aunque no necesariamente en el orden que se indica (si en lugar de all hubiésemos utilizado sequence, el orden de los miembros de la clase debería ser el mismo que figura en el documento).

Los métodos de la clase forman parte de la lógica de la aplicación y no se definen en el documento WSDL.

Aunque en WSDL se puede usar cualquier lenguaje para definir los tipos de datos, es aconsejable usar XML Schema, indicándolo dentro de la etiqueta types e incluyendo el espacio de nombres correspondiente en el elemento definitions.

```
<definitions
  name="WSDLusuario"
  targetNamespace="http://localhost/dwes/ut6"
  xmlns:tns="http://localhost/dwes/ut6"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  …
>
```

El otro tipo de datos que puede que necesitemos definir en los documentos WSDL, son los arrays. Para definir un array, no existe en el XML Schema un tipo base adecuado que podamos usar. En su lugar, se utiliza el tipo Array definido en el esquema enconding de SOAP. Por ejemplo, podríamos añadir un tipo array de usuarios al documento anterior haciendo:

```
<xsd:complexType name="ArrayOfusuario">
 <xsd:complexContent>
   <xsd:restriction base="soapenc:Array">
     <xsd:attribute
         ref="soapenc:arrayType" arrayType="tns:usuario[]"
     />
   </xsd:restriction>
 </xsd:complexContent>
</xsd:complexType>
```

Al definir un array en WSDL, se debe tener en cuenta que:

- El atributo arrayType se utiliza para indicar qué elementos contendrá el array.
- Se debe añadir al documento el espacio de nombres SOAP encoding:

```
<definitions
  name="WSDLusuario"
  targetNamespace="http://localhost/dwes/ut6"
  xmlns:tns="http://localhost/dwes/ut6"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  …
>
```

El nombre del array debería ser ArrayOfXXX, dónde XXX es el nombre del tipo de elementos que contiene el array.

En muchas ocasiones no será necesario definir tipos propios, y por tanto en el documento WSDL no habrá sección types; será suficiente con utilizar alguno de los tipos propios de XML Schema, como xsd:string, xsd:float o xsd:boolean.

Etiqueta <message>:

En la etiqueta <message> se definen los datos que se intercambian en la petición y respuesta del servicio. Cada servicio suele tener dos mensajes, uno de entrada y otro de salida. En la entrada se definen los parámetros que va a recibir el servicio, y en la salida los datos que el servicio va a devolver.

Cada parámetro se define dentro de una etiqueta <part>. Cada elemento <part> se asocia con un tipo de dato concreto..

Cada mensaje tiene además un atributo name que será único, normalmente el nombre de los mensajes terminará en Request en los mensajes de entrada y en Response en los de salida.

```
<message name="getUsuarioRequest">
  <part name="id" type="xsd:int"/>
</message>
<message name="getUsuarioResponse">
  <part name="getUsuarioReturn" type="tns:usuario"/>

 </message>
```

En un documento WSDL podemos especificar dos estilos de enlazado: document (contiene cualquier tipo de contenido que queramos enviar entre aplicaciones) o **RPC** (se usa para invocar métodos de forma remota, es el que usaremos en los ejemplos). La selección que hagamos influirá en cómo se transmitan los mensajes dentro de las peticiones y respuestas SOAP.

El estilo de enlazado RPC está más orientado a sistemas de petición y respuesta que el document (más orientado a la transmisión de documentos en formato XML). En este estilo de enlazado, cada elemento message de WSDL debe contener un elemento part por cada parámetro (de entrada o de salida), y dentro de éste indicar el tipo de datos del parámetro mediante un atributo type, como se muestra en el ejemplo anterior.

Además, cada estilo de enlazado puede ser de tipo encoded o literal (aunque en realidad la combinación document/encoded no se utiliza). Al indicar encoded, estamos diciendo que vamos a usar un conjunto de reglas de codificación, como las que se incluyen en el propio protocolo SOAP (espacio de nombres http://schemas.xmlsoap.org/soap/encoding/), para convertir en XML los parámetros de las peticiones y respuestas.

Nosotros trabajaremos con estilo de enlazado RPC/encoded.

Etiqueta <portype>

Las funciones que creas en un servicio web, en un documento WSDL se conocen con el nombre de **operaciones**.

En lugar de definirlas una a una, es necesario agruparlas en lo que en WSDL se llama portType. Un portType, por tanto, contiene una lista de funciones (operaciones), indicando para cada una la lista de parámetros de entrada y de salida que le corresponden.

Por ejemplo:

```
<portType name="usuarioPortType">
  <operation name="getUsuario">
    <input message="tns:getUsuarioRequest"/>
    <output message="tns:getUsuarioResponse"/>
  </operation>
</portType>
```

A no ser que el servicio web sea bastante complejo, el documento WSDL contendrá un único portType.

Cada <portType> debe tener un atributo name único. A su vez, cada elemento <operation> debe tener un atributo name que ha de corresponder con el nombre de la función que realiza. En el ejemplo, la operación "getUsuario" contiene un elemento input para indicar las funciones que hace una petición enviando un parámetro y un elemento output para la respuesta. Estos elementos contienen el atributo message que hace referencia al correspondiente mensaje.

- 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 el caso más habitual: funciones que reciben algún parámetro, se ejecutan, y devuelven un resultado.

Es posible (pero muy extraño) encontrar funciones a la inversa: sólo con un parámetro output (el servidor envía una notificación al cliente) o con los parámetros output e input por ese orden (el servidor le pide al cliente alguna información). Por tanto, al definir una función (un elemento operation) se debe tener cuidado con el orden de los elementos input y output.

Etiqueta <binding>

En el elemento binding se dan detalles de cómo una operación <portype> se va a transmitir.

Para el protocolo SOAP, el enlace es **<soap:binding>** y el transporte es mensajes SOAP sobre HTTP.

Para el portype anterior crearemos por ejemplo, un binding:

WDSL incluye extensiones para SOAP 1.1 que permiten definir detalles específicos para este protocolo:

- **SOAP:binding:** indica que el enlace va a ser SOAP.
  - En el atributo style se especifica el formato o codificación del mensaje, normalmente RPC,
  - En el atributo transport se indica el protocolo de transporte para indicar http indicamos el valor http://schemas.xmlsoap.org/soap/http.

- **SOAP:operation:** indica el enlace de una operación (operation) específica (URL de la función)

- **SOAP:body:** especifica los detalles de los mensajes de entrada y salida especificando el estilo de codificación SOAP y el namespace asociado al servicio especificado.

Fíjate que el atributo type hace referencia al portType creado anteriormente

```
<binding name="usuarioBinding" type="tns:usuarioPortType">
  <soap:binding
      style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"
  />
  <operation name="getUsuario">
    <soap:operation
     soapAction="http://localhost/dwes/ut6/getUsuario.php?getUsuario"
    />
    <input>
     <soap:body
        use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost/dwes/ut6"
     />
    </input>
    <output>
     <soap:body
        use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost/dwes/ut6"
     />
    </output>
  </operation>
</binding>
```

Etiqueta <service>

Por último, falta definir el elemento service. Normalmente sólo encontraremos un elemento service en cada documento WSDL. En él, se hará referencia al binding anterior.

Contiene una lista de elementos de tipo port. Cada port indica dónde (en qué URL) se puede acceder al servicio web.

Por ejemplo:

```
<service name="usuario">
  <port name="usuarioPort" binding="tns:usuarioBinding">
    <soap:address
        location="http://localhost/dwes/ut6/getUsuario.php"
    />
  </port>
</service>
```

Ejemplos de servicios web con PHP SOAP

IMPORTANTE: Para poder utilizar la extensión SOAP hay que editar el fichero php.ini, buscar la directiva **extensión=soap,** y quitar el punto y como que la precede, si lo tiene (es decir, descomentarlo).

Las dos clases principales que usaremos para crear y consumir servicios son SoapServer y SoapClient. La primera, permite comunicarnos con un servicio web, y la segunda la usaremos para crear nuestros propios servicios.

Vamos a ver algunos ejemplos de creación y utilización de servicios web utilizando PHP SOAP:

NOTA: debéis cambiar las URI y las URL de los ejemplos por las que tengáis en vuestros proyectos.

Crear servicio web (I)

Carpeta 00_servicioWeb

En este ejemplo creamos un servicio en el que obtendremos una frase aleatoria de un conjunto de frases escritas en inglés y después accederemos a él. Para ello tenemos:

- **Fichero Library.php:** es la clase en la que se encuentra la función (método eightBall) 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, es decir, creando un objeto de la misma y accediendo al método eightBall.

Como el ejemplo trata de crear y usar un servicio necesitamos otros ficheros:

- **Fichero serverSOAP.php:** script que crea el servidor del servicio. En él:
  - 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, y el segundo parámetro el array de opciones definido anteriormente.
  - Indicamos el nombre de la clase donde se ofrece el servicio, en nuestro ejemplo "Library", mediante el método setClass. Esto nos da acceso a todos los métodos definidos en la clase.
  - Por último activamos el servicio mediante el método handle() de la clase soapServer.

- **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 del servicio con su ruta absoluta. Después, se hace una llamada a la función ofrecida y se comprueba que ha funcionado mostrando el resultado. **Observa que en este script no se incluye en ningún momento ningún archivo externo, ni el que contiene el método que se ofrece ni el que crea el servicio**

Ejemplo Crear servicio web (II)

En este ejemplo vamos a crear un servicio SOAP que sume o reste dos números y usarlo. Tampoco utilizaremos WSDL

Vamos a trabajar con dos carpetas diferentes en este ejercicio, una será en la que creemos el servicio y la otra hará de cliente que lo solicita (de esta forma simularemos localizaciones diferentes).

Servidor del servicio: carpeta 01_servicioWeb1

Tenemos en esta carpeta dos ficheros:

- **Server.php** es la clase en la que están definidos los métodos que vamos a ofrecer como servicio. En este ejemplo simplemente son dos métodos que suman y restan dos números que les llegan como parámetros.

- **serverSOAP.php:** es el servidor del servicio. En este script debemos:
  - Incluir la clase donde se encuentran los métodos que vamos a ofrecer como servicio.
  - Debemos indicar la URI del servicio. En el ejemplo la guardamos en una variable.
  - Creamos el objeto de tipo SoapServer en el que en el primer parámetro le pasamos null, ya que correspondería con el WSD (que nosotros no estamos utilizando), y en el segundo parámetro le pasamos un array, en este caso solo con la URI.
  - Utilizamos el método setClass para hacer referencia a la clase que ofrece nuestro servicio y tener acceso a los métodos definidos en ella.
  - Por último activamos el servicio con el método handle().

Cliente del servicio: carpeta 01_servicioWeb1Cliente

En esta carpeta tenemos un fichero llamado cliente.php desde el que se solicitará el servicio.

- Como no especificamos WSDL debemos indicar la URL (archivo del servidor del servicio con su ruta) y la URI (directorio donde está el servicio) donde vamos a consumir el servicio. Lo guardamos en dos variables.
- Creamos un objeto de tipo SOAP cliente al que le pasamos como primer parámetro null (correspondería con el WSDL), y como segundo parámetro un array con la URL y la URI del servicio.
- Después invocamos a los métodos sumar y restar.
- En este ejemplo además utilizamos la excepción que nos proporciona la extensión SOAP para el control de errores.

Si ejecutamos este script vemos el resultado de sumar y restar dos números.

Carpeta: 01_servicioWeb1_funciones

Este ejemplo es muy similar al anterior, aunque con algunos cambios:

- Las funciones que se ofrecen como servicio ,sumar y restar, no se definen dentro de una clase, sino como funciones independientes que se encuentran definidas dentro del fichero “funciones.php”.
- Por tanto, en el servidor ya no podemos indicar las funciones que se ofrecen como servicio mediante el método “setClass”. En este caso, hay que utilizar el método “addFunction()” de la clase SoapServer. Este método (addFunction) no funciona para indicar funciones que están definidas en una clase. En ese caso hay que usar setClass.
- En el fichero cliente no hay que hacer ningún cambio.

Crear servicio web (III)

Este es un ejemplo de creación de un servicio web con fichero WSDL.

IMPORTANTE: hay que comprobar en el archivo php.ini que la directiva **soap.WSDL_cache_enabled** del archivo php.ini tiene el valor **0**, de lo contrario, (con el valor predeterminado 1) los cambios que se realizan en los archivos WSDL no tendrán un efecto inmediato.

- Esta situación se presenta cuando los métodos que se ofrecen como servicio no son conocidos por las personas que los quieran usar (nombre, parámetros y valores de retorno).
- Para describir los servicios, cómo acceder a ellos y los parámetros que debemos pasar, publicaremos un documento WSDL.
- Hemos visto en la teoría, que elaborar manualmente un documento WSDL puede resultar complicado y tedioso, por lo que usaremos herramientas que nos ayudan a generarlo.
- En primer lugar, debemos comentar los métodos que ofrecemos como servicio con las directivas o marcas usados por lenguajes que generen documentación como Javadoc o phpDocumentator. Las etiquetas @param y @return son las más importantes y las que usaremos. Con @param especificamos el tipo de dato que esperamos como parámetro, y con @return, el tipo de datos que devolveremos en la respuesta.
- Vamos a utilizar el mismo ejemplo que antes, una clase con dos métodos, sumar y restar.

Carpeta 02_servicioWebWSDL

- En esta carpeta se crearán los archivos necesarios para crear el servicio y ponerlo en marcha. También se creará el archivo WSDL que vamos a utilizar y que contiene la descripción del servicio.
- Recordemos que la directiva **soap.WSDL_cache_enabled** del archivo php.ini debe tener el valor **0** .
- Después comentamos con un formato específico, como el que se genera con JavaDoc, la clase donde se encuentran las funciones que vamos a ofrecer como servicio. Utilizaremos las etiquetas @param y @return.
- Para crear el archivo WSDL, vamos a descargar la librería WSDLDocument del siguiente enlace https://code.google.com/archive/p/WSDLdocument/. Una vez descargada, copiamos el archivo WSDLDocument.php a la carpeta en la que tenemos nuestro proyecto.
- Creamos un archivo php que nos servirá para crear el fichero WSDL. Este fichero lo he llamado en el ejemplo “generaWSDL2.php” y en él:
    - Lo primero que hacemos es desactivar las notificaciones, ya que de no hacerlo se muestran este tipo de avisos, que no son errores como tales, pero hacen que el fichero WSDL no se genere correctamente. Para ello usamos la instrucción **error_reporting(E_ALL ^ E_NOTICE);**

    - Incluimos el archivo en el que se encuentra la clase que ofrece el servicio (Server.php) y el archivo WSDLDocument.php que contiene la clase la WSDLDocument.
    - Indicamos mediante la instrucción header("Content-type: text/xml"); que el contenido va a ser un archivo XML, de esta forma, podemos ver directamente en el navegador el fichero WSDL. Si esta instrucción no se incluye, el fichero no se ve directamente, sino que hay que ver el código fuente de la página generada para poder verlo.
    - Creamos un objeto de tipo WSDLDocument pasándole como parámetros el nombre de la clase en la que están los servicios que se ofrecen, y las rutas donde se ofrece el servicio y el espacio de nombres destino.
    - Por último hacemos que genere el fichero WSDL en formato XML .
- Si ejecutamos este fichero, generaWSDL2.php y vemos, bien directamente, o bien mirando el código fuente de la salida (dependiendo de si hemos incluido la instrucción header indicada anteriormente o no), el documento WSDL generado.
- Copiamos el código del fichero WSDL un archivo en blanco y revisamos que la codificación sea UTF-8. Lo guardamos con el nombre que queramos, por ejemplo, “miWSDL.WSDL”
- A continuación creamos el servicio. Este fichero llamado serverSOAP.php es muy parecido al anterior. En él:
    - Al crear el objeto servidor le pasamos como único parámetro la ruta al archivo WSDL que hemos creado.
    - Si ponemos en el navegador la ruta al servicio añadiéndole al final ?WSDL, podemos ver en el navegador el fichero wsdl. En el ejemplo sería **http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWSDL/serverSOAP.php?wsdl**
    - Indicamos mediante setClass la clase donde se ofrecen los servicios.
    - Por último activamos el servicio mediante el método handle().

Antes de ver cómo se consume el servicio veamos con un poco más de detalle el archivo WSDL creado:

DOCUMENTO WSDL GENERADO:

Hay que observar el documento WSDL generado para poder saber el nombre de los métodos que ofrece el servicio, además de los parámetros (y tipo de datos) que espera recibir y cómo acceder (URL de acceso). En él podemos encontrar:

- **definitions:** Es el elemento raíz, permite especificar el espacio de nombres del documento *target namespace*, el nombre, y otros prefijos utilizados en el documento WSDL. Un ejemplo de definición de prefijo es: **xmlns:WSDL=http://schemas.xmlsoap.org/WSDL/** que especifica que todos los elementos dentro del documento de esquemas con el target namespace "http://schemas.xmlsoap.org/WSDL/" tendrán el prefijo WSDL.

```
<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/">
```

- **types:** En él se definen los tipos de datos que se intercambian en el mensaje, en el ejemplo se utilizan datos simples que están incluidos en el schema por tanto, no se definen datos complejos.

```
<wsdl:types>
    <xsd:schema targetNamespace="http://localhost/DWES_P_U1/27_ServiciosWeb/02_servicioWebWSDL/"/>
</wsdl:types>
```

- **message:** por cada operación que ofrece el servicio, se definen dos mensajes, uno de entrada y otro de salida. Cada mensaje puede tener una o más partes, siendo cada una de ellas el equivalente a los parámetros de la llamada a la función. En el ejemplo, **para el método sumar se generan dos mensajes sumarRequest y sumarResponse. Dentro de cada uno de ellos están definidas las partes correspondientes a los parámetros que utiliza cada mensaje**.

```
<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>
```

- **portType:** es el elemento más importante del WSDL. En él se definen los servicios web (las operaciones que puede realizar: sumar y restar), y los mensajes involucrados (definidos en el apartado anterior). En el ejemplo están definidas las operaciones sumar y restar. En el ejemplo, para la suma el mensaje de input es sumarRequest y el de salida es restarRequest.

```
<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>
```

- **binding:** Los bindings son definiciones concretas de los portTypes. En el ejemplo podemos ver que se utilizan las extensiones de WSDL para SOAP:
    - En el elemento **<soap-env:binding>** se indica el estilo del mensaje (RPC, invocar procedimientos de forma remota), y el protocolo de transporte, HTTP.
    - En el elemento **<soap-env:operation>** se indica el enlace de la operación específica, hay un elemento de este tipo para cada operación (sumar o restar) que se puede realizar.
    - En el elemento **<soap-env:body>** se especifican detalles de los mensajes de entrada y salida (estilo de codificación, y namespace asociado al servicio)

```
<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_servi
        <wsdl:input>
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/enc
        </wsdl:input>
            <wsdl:output>
        <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encodir
        </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_servi
        <wsdl:input>
            <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/enc
        </wsdl:input>
        <wsdl:output>
        <soap-env:body xmlns="http://schemas.xmlsoap.org/wsdl/soap/" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encodir
    </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

- **service:** Define el servicio como una colección de elementos port a los que se puede acceder. Un port se define asociando una dirección de red con un binding, de los definidos en el documento. Dicha dirección de red es la dirección (URL) donde el servicio actúa, y por lo tanto, será la dirección a la que las aplicaciones deberán conectarse para acceder al servicio.

```
<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>
```

Carpeta servicioWSDLCliente

Ahora crearemos un cliente que consuma el servicio que hemos ofrecido. En primer lugar, debemos observar el fichero WSDL que se ha generado para saber cómo se llaman las funciones que ofrece el servicio y los tipos de datos que debemos pasar como parámetro.

Para ello creamos el fichero **cliente.php** en el que:

- Creamos un objeto de tipo SOAP cliente pasándole como parámetro el archivo WSDL que hemos creado con su localización.
- Después hacemos referencia a los métodos que ofrece el servicio y mostramos el resultado por pantalla (conocemos los nombres de los métodos porque los hemos visto en el archivo WSDL).

Consumo de un servicio externo

Vamos a ver cómo consumir un servicio externo del que conocemos su descripción mediante un archivo WSDL.

Carpeta 03_servicioConsumo1

Hemos visto en el ejemplo anterior que leyendo el archivo WSDL que contiene la descripción de un servicio obtenemos la información necesaria para acceder a él.

Si embargo, esta información también se puede obtener usando los métodos **__getTypes ()** y **__getFunctions ()** de la clase **SoapClient**, de esta forma obtenemos la información que necesitamos de forma más sencilla, sobre todo si el archivo WSDL es muy extenso **.** En este ejemplo veremos cómo.

Fichero averiguarServicio.php

En este fichero se crea un objeto de la clase SoapClient al que se le pasa como único parámetro el fichero WSDL del servicio.

A continuación, utilizando los métodos __getTypes() y __getFunctions averiguamos información sobre ese servicio.

- El método **__getFunctions()** devuelve un array con todas las funciones descritas en el archivo WSDL.
- El método **__getTypes()** devuelve un array con los tipos descritos en el archivo WSDL.

Si lo probamos con el ejemplo anterior en el que se ofrecían los métodos sumar y restar, la salida por pantalla, sería como la de la imagen, en primer lugar los métodos y en segundo lugar los tipos (en este caso un array vacío puesto que no se han descrito tipos especiales):

```
C:\xampp\htdocs\DWES_P_U1\27_ServiciosWeb\03_servicioConsumo1\averiguarServicios.php:16:
array (size=2)
  0 => string 'float sumar(float $n1, float $n2)' (length=33)
  1 => string 'float restar(float $n1, float $n2)' (length=34)

C:\xampp\htdocs\DWES_P_U1\27_ServiciosWeb\03_servicioConsumo1\averiguarServicios.php:17:
array (size=0)
  empty
```

Si probamos con un servicio externo, por ejemplo, en la página W3Schools se ofrece un servicio de conversor de temperatura tanto de grados Celsius a Fahrenheit como a la inversa. https://www.w3schools.com/xml/tempconvert.asmx?WSDL

(LAS FUNCIONES SALEN REPETIDAS porque están definidas tanto para protocolo soap como post)

Si observamos la salida por pantalla, en este caso vemos que hay cuatro métodos definidos, que parecen repetidos, esto es porque están definidos en el archivo WSDL para SOAP y para POST (etiqueta binding), además los parámetros que espera recibir son de tipo FahrenheitToCelsius o CelsiusToFahrenheit y el tipo de dato que devuelve.

```
C:\xampp\htdocs\DWES_P_U1\27_ServiciosWeb\03_servicioConsumo1\averiguarServicios.php:16:
array (size=4)
  0 => string 'FahrenheitToCelsiusResponse FahrenheitToCelsius(FahrenheitToCelsius $parameters)' (length=80)
  1 => string 'CelsiusToFahrenheitResponse CelsiusToFahrenheit(CelsiusToFahrenheit $parameters)' (length=80)
  2 => string 'FahrenheitToCelsiusResponse FahrenheitToCelsius(FahrenheitToCelsius $parameters)' (length=80)
  3 => string 'CelsiusToFahrenheitResponse CelsiusToFahrenheit(CelsiusToFahrenheit $parameters)' (length=80)
```

La salida del método __getTypes() nos muestra que los tipos de datos que devuelve son estructuras, equivalente a una clase (son tipos de datos definidos). Por ejemplo. la clase FahrenheitToCelsius tiene una propiedad de tipo string llamada Fahrenheit.

```
C:\xampp\htdocs\DWES_P_U1\27_ServiciosWeb\03_servicioConsumo1\averiguarServicios.php:17:
array (size=4)
  0 => string 'struct FahrenheitToCelsius {
 string Fahrenheit;
}' (length=50)
  1 => string 'struct FahrenheitToCelsiusResponse {
 string FahrenheitToCelsiusResult;
}' (length=73)
  2 => string 'struct CelsiusToFahrenheit {
 string Celsius;
}' (length=47)
  3 => string 'struct CelsiusToFahrenheitResponse {
 string CelsiusToFahrenheitResult;
}' (length=73)
```

Fichero consumirServicio.php

Ejecutando el fichero “averiguarServicio.php” hemos averiguado la información que necesitamos para acceder al servicio que convierte una temperatura de Celsius a Fahrenheit y viceversa.

En este script vamos a utilizarlo. Para ello, como siempre creamos un servicio de la clase SoapClient pasándole como único parámetro el URL de acceso al fichero WSDL.

Una vez creado, se invocan los métodos que ofrecen el servicio. En el script anterior pudimos ver que los parámetros que esperan son objetos (tipos complejos), por ejemplo, en el caso de querer pasar de grados Celsius a Fahrenheit el dato que espera es una propiedad que se llama Celsius que es una string y el método a invocar se llama "CelsiusToFahrenheit.

En nuestro código llamamos a ese método. Para pasar un objeto como parámetro tenemos que utilizar un array asociativo en el que cada clave ha de ser el nombre de la propiedad del objeto, si no dará error.

Guardamos el resultado de invocar a ese método en una variable. Si mostramos ese resultado con la instrucción var_dump veremos que nos ha devuelto un objeto de tipo "CelsiusToFahrenheitResponse" con una propiedad con visibilidad pública por lo que podemos acceder directamente a ella para obtener su valor..

Finalmente, mostramos el resultado por pantalla accediendo a la propiedad "CelsiusToFahrenheitResponse" del objeto que tiene la respuesta del método.

Hacemos lo mismo utilizando el método que convierte de Fahrenheit a Celsius.

Carpeta 03_servicioConsumo2

Este ejemplo obtiene a partir del código de un país su capital utilizando un servicio externo.

Se trata de averiguar la capital de un país a partir de su código de país ISO http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL , (por ejemplo, España = 'ES')

http://utils.mucattu.com/iso_3166-1.html

Fichero averiguarServicios.php

En este script, igual que en el ejemplo anterior, utilizamos los métodos __getFunctions() y __getTypes() para conocer al forma de acceder al servicio.

Podemos observar que hay muchos métodos, el que vamos a utilizar es "CapitalCity" que devuelve un objeto de tipo CapitaCityResponse y que recibe como parámetro un objeto de tipo CapitalCity.

Si vemos que contienen estos objetos vemos que el objeto CapitalCity tiene una propiedad de tipo string que se llama "CountryISOCode" que corresponde al código ISO del país. A su vez el objeto CapitalCityResponse tiene una propiedad llamada "CapitalCityResult", también de tipo string.

Ahora que ya sabemos cómo se llama el método que queremos usar y el parámetro que recibe y el valor que devuelve construimos nuestro cliente del servicio.

Fichero clientePais.php

Creamos el cliente Soap pasando como parámetro la URL al fichero WSL del servicio.

Creamos el array de parámetros, ya que al ser un objeto ha de pasarse como array asociativo cuyas claves han de coincidir con el nombre de las propiedades del objeto.

Por último llamamos al método y mostramos el resultado que se encuentra en la propiedad CapitalCityResult del objeto de tipo CapitalCityResponse que devuelve el método.

Capeta 03_servicioConsumo3

En este ejemplo vamos a utilizar alguna herramienta que nos genere partiendo del fichero WSDL de un servicio (que debe estar activo) el código PHP que necesitamos para utilizarlo como si estuviéramos trabajando con clases PHP.

Podemos usar alguna herramienta para consumir un servicio de forma sencilla como WSDL2php https://sourceforge.net/projects/WSDL2php/ o easyWSDL2php https://sourceforge.net/projects/easyWSDL2php/.

En este ejemplo vamos a usar easyWSDL2php con el ejemplo anterior que convertía de grados Celsius a Fahrenheit o viceversa.

En primer lugar debemos descargarla y copiamos los archivos index.php, eWSDL2php.php y WSDL2php.php a la carpeta donde lo vayamos a utilizar.

NOTA IMPORTANTE: daba error, he tenido que poner <?php en lugar de <? En los archivos index.php y WSDL2php.php

Al ejecutar el archivo index.php nos pide la URL del servicio que queremos convertir a clase y el nombre que le queremos dar a la clase. En nuestro ejemplo la URL es https://www.w3schools.com/xml/tempconvert.asmx?WSDL y le damos el nombre tempConvert a la clase y pulsamos el botón “Generate code”.

En el campo textarea nos aparece el código generado. Lo copiamos a un archivo de tipo clase PHP con el nombre que le dimos, “tempConvert”.

So observamos ese archivo, veremos que se han creado las clases correspondientes a los métodos que ofrece el servicio y además ha creado la clase tempConvert que contiene el código que se encarga de crear el objeto SoapClient y los métodos que realizan las llamadas a los métodos del servicio usando el objeto SoapClient. Hay que fijarse en el tipo de parámetro y valor que devuelven esos métodos.

Ahora vamos a utilizar el servicio usando las clases PHP que se han creado pero en lugar de utilizar la clase soapClient, crearemos un objeto de la clase tempConvert creado con easyWSDL2php y accederemos a los métodos de esa clase que hacen la llamada a los métodos del servicio.

Fichero cliente.php.

En él:

- Incluimos la clase que se ha creado con easyWSDL2php.

- Creamos un objeto de la clase tempConvert (nombre que dimos a la clase al generarla).
- Creamos un objeto de la clase CelsiustoFahrenheit que es el que el método espera recibir.
- Asignamos al atributo Celsuis de ese objeto creado, el valor en grados Celsius que queremos convertir.
- Guardamos en una variable el resultado (que es un objeto) de ejecutar el método CelsiusToFahrenheit pasándole el objeto CelsiusToFahrenheit.
- Mostramos el resultado que está en el Atributo FahrenheitToCelsiusResult del objeto de la clase CelsiusToFahrenheitResponse.

Hacemos lo mismo para obtener el valor en grados Fahrenheit de una temperatura en Celsius.

Carpeta 03_servicioConsumo4

La herramienta utilizada en el ejemplo anterior solo sirve si tenemos la URL de un archivo WSDL que ofrece un servicio, pero no sirve si tenemos un archivo WSDL, por ejemplo, descargado en nuestro ordenador (aunque el servicio debe seguir activo, en caso contrario, no funcionará porque no podrá realizar las operaciones definidas en él), por ejemplo en el caso del archivo WSDL de la teoría, con la herramienta que vamos a utilizar en el ejemplo se generarían las clases correspondientes, pero el servicio ya no está activo por lo que no funcionará).

Para este tipo de casos, podemos utilizar WSDL2php. Veamos un ejemplo con el archivo WSDL del ejemplo anterior.

En primer lugar hay que descargarse la herramienta, aunque la que viene en la teoría no sirve para versiones de PHP superiores a la 6. Descargaremos una similar que sí sirve para PHP7 que se encuentra en https://gitlab.com/tomas131315/wsdl2php, aunque hay más que podéis investigar y utilizar.

Descargamos el código y lo descomprimimos en la carpeta donde tengamos nuestro proyecto.

En este caso, el código descargado contiene un archivo llamado “generate.php” que es el que ejecutaremos para crear las clases PHP del servicio.

Desde la consola de Windows y situados en la carpeta donde lo hayamos copiado, teclearemos **“php generate.php <nombre del archivo.wsdl>”**

Por ejemplo, “ php generate.php w3schoolsService.wsdl” (he guardado el archivo WSDL en mi ordenador con ese nombre)

El código PHP que genera, lo guarda en un directorio que, si no existe, se crea dentro de esa misma carpeta llamado “out”. Si lo observamos vemos que, a diferencia del ejemplo anterior se han creado varios archivos, cada uno con una clase:

- El archivo TemConvert,php es una clase que hereda de la clase SoapClient y contiene el código que se encarga de crear el objeto SoapClient y los métodos que

hacen la llamada al servicio como está definido en el fichero WSDL (en el elemento operation está definida la URL del servicio).

- Los archivos CelsiusToFahrenheit.php, CelsiusToFahrenheitResponse.php, FahrenheitToCelsius.php y FahrenheitToCelsiusResponse.php contienen las clases correspondientes a los métodos que se usan en el servicio y las propiedades que se corresponden con los parámetros y valores que devuelven. Observa que las propiedades son de tipo "protected" por lo que no se pueden acceder directamente a ellas, por tanto hay que utilizar los métodos get y/o set correspondientes.
- El archivo autoload.php que ha generado la herramienta utiliza la capacidad que nos ofrece PHP para hacer autoloading (carga automática de clases). Básicamente lo hace el fichero autoload es ejecutar automáticamente la función registrada mediante la función de PHP spl_autoload_register y comprobar si existe una clase de la que se está intentando crear una instancia. Si es así, incluye el fichero que la contiene pudiendo así crear el objeto. Para usarlo, simplemente incluimos el archivo autoload.php en nuestro código (sería equivalente a incluir todos los archivos de las clases).
- Ahora crearemos el fichero que consumirá el servicio "**cliente.php"**. En él:
    - Creamos un objeto de la clase TempConvert.
    - Llamamos al método CelsiusToFahrenheit pasando como parámetro un objeto de la clase CelsiusToFarhenheit que a su vez en su constructor espera recibir el valor de la propiedad Celsius. Si no se hace así dará error.
    - Guardamos el resultado en un objeto que será de tipo CelsiusToFahrenheitResponse. Para poder acceder a la propiedad que contiene el valor de la respuesta necesitamos utilizar el método getCelsiusFahrenheitResult, ya que la propiedad tiene visibilidad protegida.
    - Por último mostramos el resultado.
    - Hacemos lo mismo pero convirtiendo a Fahrenheit.

Servicios Rest

Otro tipos de servicios que se utilizan mucho actualmente son los llamados servicios Rest o Restul, también se les conoce como Restful API

- 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 distinguen por que se basan en el protocolo de aplicación HTTP.
- Permite la comunicación entre cliente y servidor en diferentes lenguajes y plataformas.

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.
- Usa los métodos HTTP definidos en el protocolo nativo: GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE, CONNECT (los cuatro primeros son los más importantes), cada uno utilizado para una función específica. Por ejemplo:
    - 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
- 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, …). Esto permite que el servicio sea utilizado por distintos clientes escritos en diferentes lenguajes, corriendo en diversas plataformas y dispositivos.

Códigos de respuesta del servidor

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.

Ejemplo creación y consumo de un Servicio 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 utilizando la extensión "Insomnia".
- Podéis encontrar el ejemplo y el código en: https://www.codigonaranja.com/2018/crear-restful-web-service-php
- También hay un vídeo en Youtube que lo explica: https://www.youtube.com/watch?v=CgpwuY9nud4

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.

Carpeta 04_crearAPI_Rest

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.

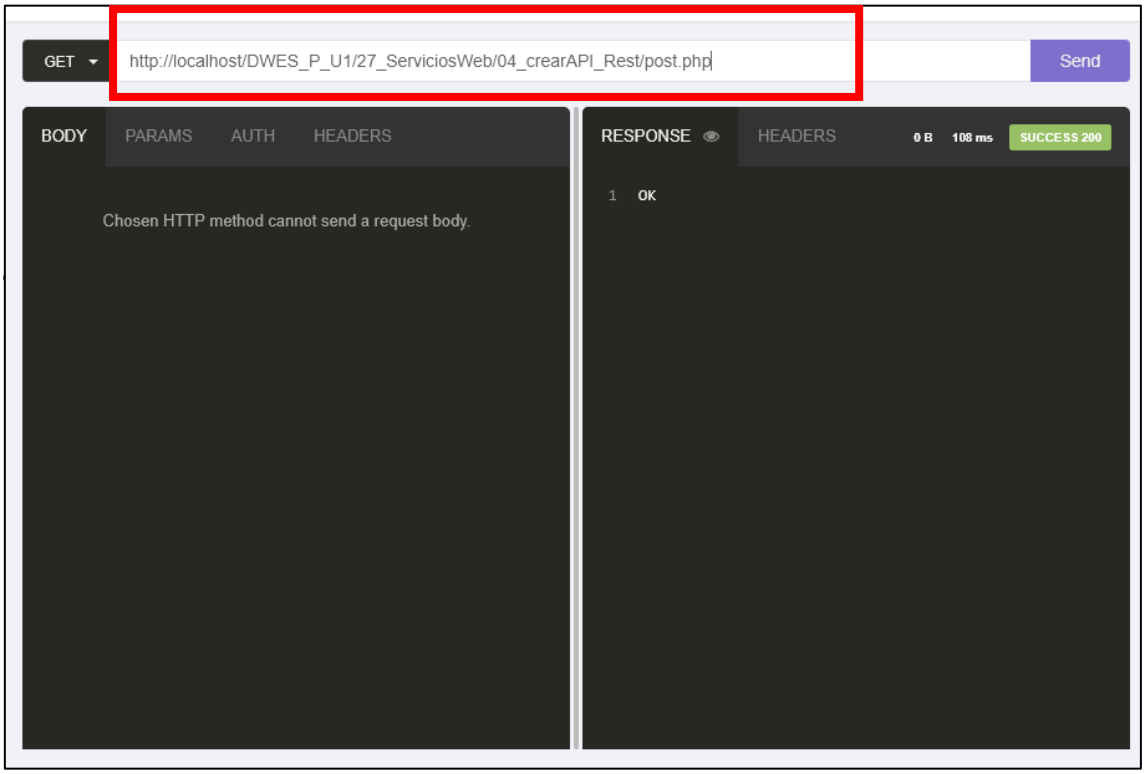
Además Creamos una base de datos llamada "blog" e importamos el archivo .sql que viene incluido en el ejemplo.

El código del 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:
    - **La función getParams()** va a formar una cadena con los parámetros que hay que pasar a una sentencia preparada de tipo update separados por comas.
    - **La función bindAllValues()** asocia cada parámetro y su valor al statement de la consulta que se va a hacer a la BD.
- **Post.php:** contiene el código en el que se comprueba el tipo de petición recibida y se realizan las acciones pertinentes. En él:
    - En primer lugar se obtiene una conexión a la base de datos.
    - Después va comprobando el tipo de petición recibida, usando para ello la variable superglobal $_SERVER['REQUEST_METHOD'].

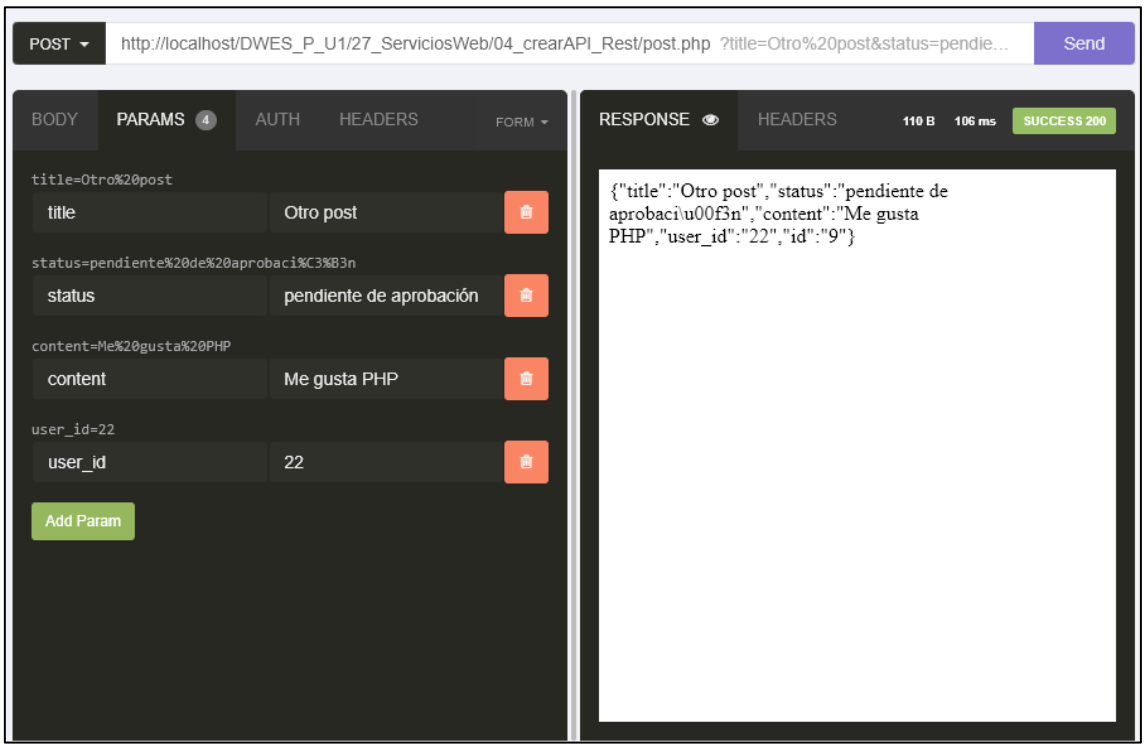
- Si recibe una petición de tipo GET (obtener datos de la base de datos), pueden darse dos casos, que se quieran recuperar todos los post publicados en el blog o uno determinado, identificado por su id que llegará como parámetro.
- Si, en cambio, recibe una petición de tipo POST (insertar datos en la 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.
- 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.
- 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.

- Para probar el código desde la extensión **insomnia** de Chrome, hay que instalarla en el navegador, y abrirla desde las aplicaciones de Google.
- En la pantalla que aparece En la pantalla que aparece hay que introducir la URL de la aplicación

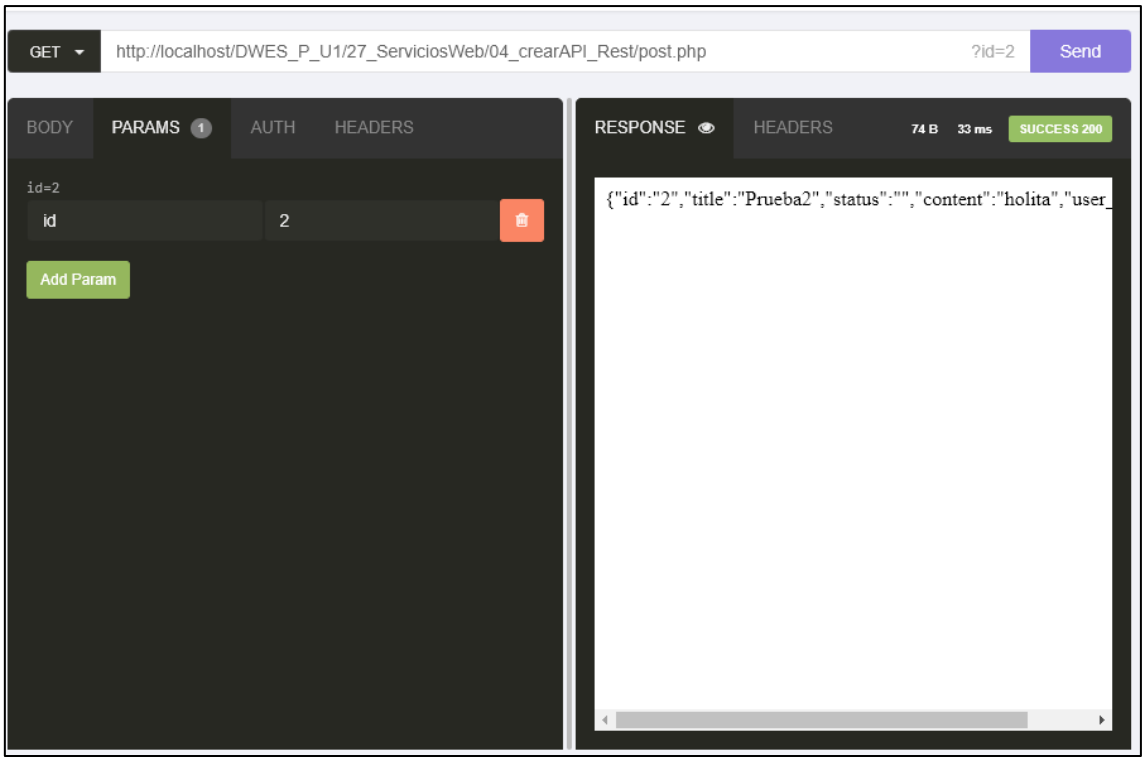


- Después seleccionaremos el tipo de petición que queremos hacer, por ejemplo POST para insertar datos en la BD.

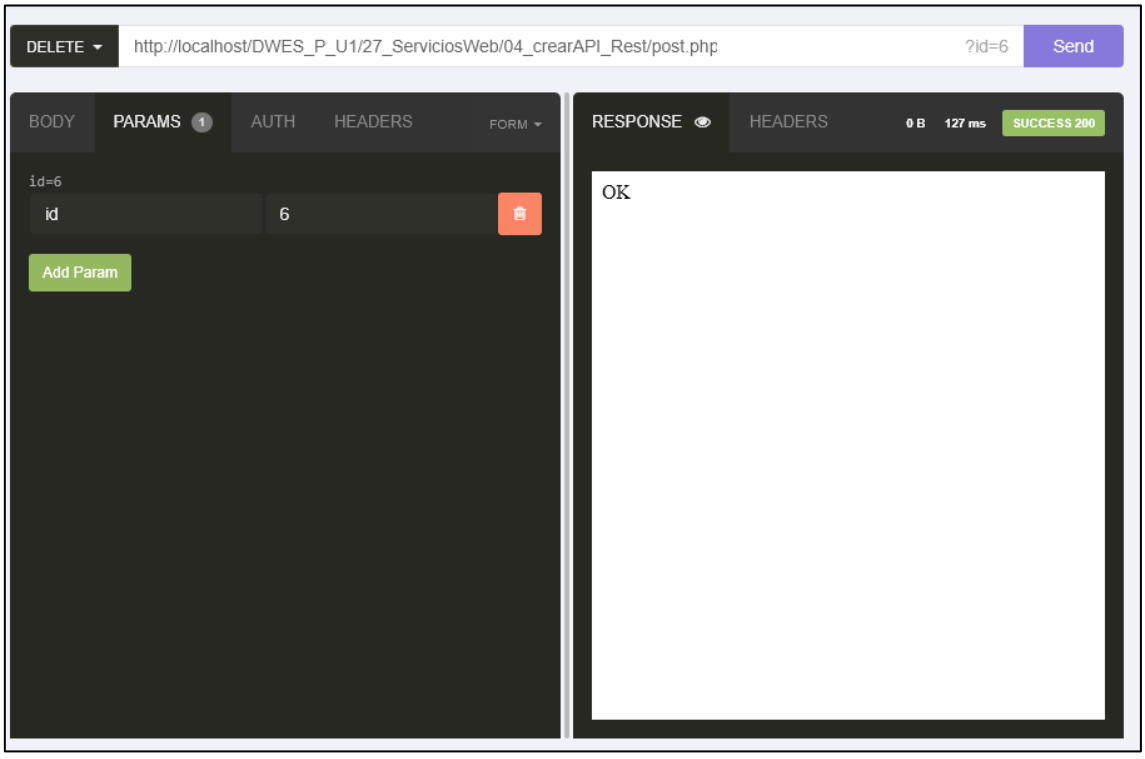
- En la pestaña PARAMS insertaremos los parámetros necesarios: title, status, content y user_id y sus respectivos valores.
- Para enviar la petición se pulsa el botón “send”.



- Ahora queremos hacer, por ejemplo, GET del post que tiene id=2.
- Cambiamos el tipo de petición a GET.
- En la pestaña PARAMS insertaremos los parámetros necesarios: id y el valor 2.
- Pulsando el botón “send” se envía la petición. Observa la respuesta y como se forma la URL de la petición

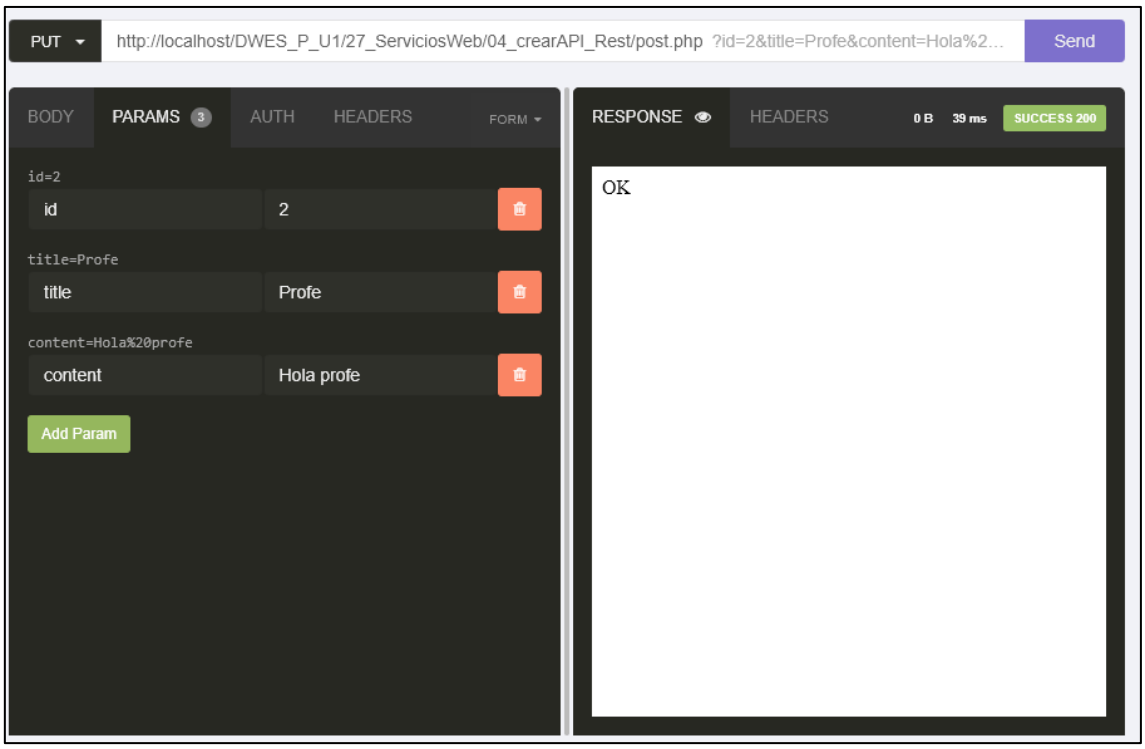


- Ahora queremos borrar el post que tiene id=6
- Cambiamos el tipo de petición a DELETE.
- En la pestaña PARAMS insertaremos los parámetros necesarios: id y el valor 6.



- Ahora queremos modificar el post que tiene id=2
- Cambiamos el tipo de petición a PUT.

- En la pestaña PARAMS insertaremos los parámetros que queremos cambiar, el id del post a cambiar, y en este caso el título y el contenido.



Esto ha sido una breve introducción a los servicios Rest con un sencillo ejemplo en el que podéis ver cómo se prueban los servicios de este tipo.

Muchas aplicaciones como Google, Facebook, etc. ofrecen este tipo de servicios, y para poder utilizarlos normalmente es necesario registrarse y utilizar una API Key que te proporcionan, necesaria para poder consumir el servicio.

También duelen ofrecer documentación, tutoriales y/o ejemplos de uso en función del tipo de servicio al que quieras acceder.

Fin Resumen tutoría