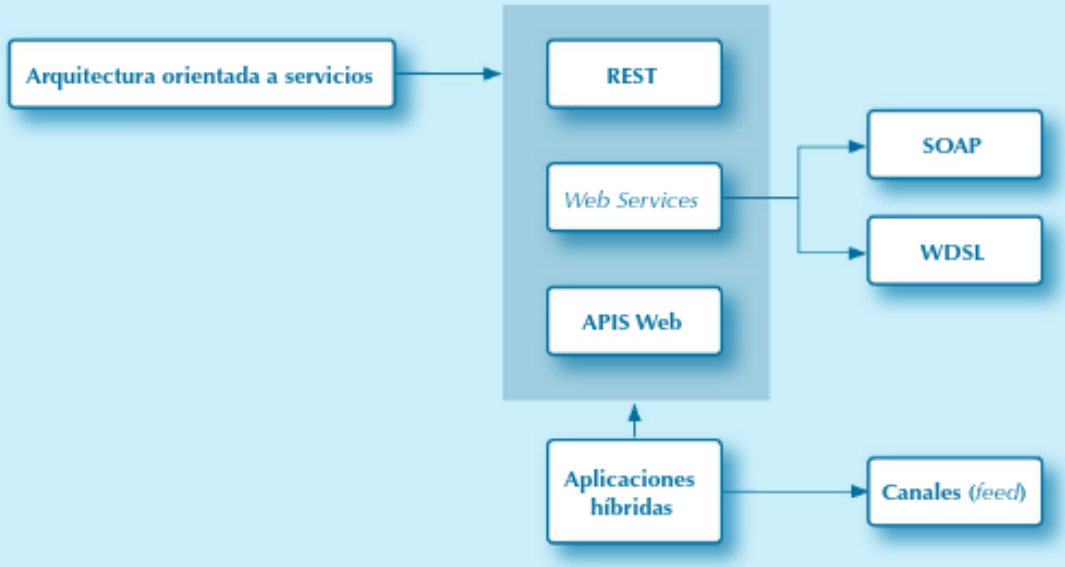
10

Servicios web y aplicaciones híbridas

Objetivos

- ✓ Presentar la arquitectura orientada a servicios.
- ✓ Conocer los elementos del protocolo SOAP.
- ✓ Entender y crear documentos WDSL.
- ✓ Crear servicios web.
- ✓ Utilizar servicios web.
- ✓ Comprender el funcionamiento de las aplicaciones híbridas.
- ✓ Desarrollar una aplicación híbrida.

Mapa conceptual



Glosario

Aplicación híbrida o *mashup*. Las aplicaciones híbridas o *mashups* integran datos y funcionalidad de diferente origen en una misma interfaz gráfica.

***Endpoint*.** Los servicios web exponen uno o más *endpoints* a los que se pueden enviar solicitudes.

Fuente o canal web. Las fuentes o canales web (*feed*) son un medio de sindicación de contenido al que los usuarios pueden suscribirse.

Servicio. Los servicios son componentes independientes que proveen alguna funcionalidad a otros componentes o sistemas.

Servicios web. Puede referirse a un servicio genérico accesible por la web o a arquitectura *Web Services*, servicios web, un estándar del W3C. Para diferenciar, en este capítulo el segundo caso se escribirá con mayúscula.

Sindicación. Consiste en publicar contenidos para que puedan ser utilizados en otras webs.

SOA. *Service Oriented Architecture*. En la arquitectura orientada a servicios se utilizan varios servicios en conjunto para obtener la funcionalidad de la aplicación

SOAP. *Simple Object Access Protocol*. Es un protocolo para intercambio de mensajes entre clientes y proveedores de servicios. Es un estándar del W3C.

Web API. Conjunto de *endpoints* accesible a través de la web. Proveen funcionalidad que se puede integrar en otras aplicaciones.

WSDL. *Web Service Description Language*. Lenguaje basado en XML para describir la funcionalidad ofrecida por un Servicio Web. Es un estándar del W3C.

10.1. Arquitecturas de programación orientadas a servicios

Un servicio es un componente que ofrece alguna funcionalidad. Por ejemplo, la previsión meteorológica o el tipo de cambio de una moneda. Los servicios son elementos independientes y accesibles de forma remota mediante una interfaz. Para cumplir su función, un servicio puede utilizar otros servicios.

Los servicios reciben peticiones y envían una respuesta. Un servicio puede exponer varios *endpoints*, cada uno con una funcionalidad. Por ejemplo, un servicio puede exponer un *endpoint* para obtener el tipo de cambio del euro respecto al dólar y otro *endpoint* para el tipo inverso.

En las arquitecturas orientadas a servicios, las aplicaciones se diseñan utilizando estos componentes. Con esta arquitectura modular las aplicaciones son más flexibles y se favorece la reutilización del código. Además, permite integrar componentes de diferente procedencia, de diferentes sistemas y en diferentes lenguajes.

Dentro del desarrollo web, el modelo SOA se puede implementar de varias maneras. Entre ellas:

- Arquitectura *Web Services*. Es un estándar del W3C. Los servicios tienen una interfaz en WSDL y otros sistemas realizan solicitudes al servicio utilizando SOAP.
- REST. *Representational State Transfer*. La arquitectura REST utiliza una interfaz uniforme que se apoya en los métodos HTTP.
- API Web. Conjunto de *endpoints* accesible a través de la web. Proveen funcionalidad que se puede integrar en otras aplicaciones. Un ejemplo típico es el API de Google Maps, para integrar los mapas de Google en otras páginas. Pueden ser del lado del cliente o del servidor.

10.2. Protocolos y lenguajes implicados. SOAP

El estándar *Web Services* se basa en una serie de protocolos:

- Para los mensajes se utiliza el SOAP, basado en XML.
- Para la descripción de los servicios, WSDL, también basado en XML.
- Para el descubrimiento de servicios, es decir, para que los clientes puedan encontrarlos, UDDI, también basado en XML. Su uso no está muy extendido.
- La comunicación entre los servicios y sus clientes se puede establecer por medio de varios protocolos. Algunos de los más habituales son HTTP, SMTP y FTP.

Cuadro 10.1
Pila de protocolos para *Web Services*

Protocolos	Función
UDDI	Descubrimiento de servicios
WSDL	Descripción de servicios
SOAP	Mensajes
HTTP, FTP, SMTP…	Comunicaciones

10.2.1. SOAP

SOAP define el formato de los mensajes que se intercambian entre el cliente y el servicio web. Un mensaje SOAP es un fichero XML con la siguiente estructura:

```
<Envelope>
   <Header>
      ...
   </Header>
   <Body>
      ...
   </Body>
   <Fault>
      ...
   </Fault>
</Envelope>
```

Los elementos `header` y `fault` son opcionales.
Por ejemplo, un mensaje de petición podría ser así:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:ns1="http://localhost/cap10/servidorsoapmate.php">
   <env:Header/>
   <env:Body>
      <ns1:potencia>
         <base>2</base>
         <exponente>3</exponente>
      </ns1:potencia>
   </env:Body>
</env:Envelope>
```

Dentro del elemento `Body` hay un elemento potencia, que contiene elementos `base` y `exponente`. Con este mensaje se solicita el *endpoint* `potencia`, y se pasan los parámetros `base` y `exponente` con los valores indicados.

10.2.2. Descripción de servicios web. WSDL

EL WDSL es un lenguaje basado en XML para describir la funcionalidad ofrecida por un servicio web. Aunque el formato es bastante descriptivo, también es complejo. Afortunadamente, los programadores, en la mayoría de los casos, no tienen que ocuparse de los detalles.

En WSDL un servicio es un conjunto de *endpoints*. En WSDL 1 para los *endpoints* se usa el elemento `port`. En WSDL 2, se sustituyó por `endpoint`.

Los elementos de un fichero WSDL son:

a) `types`: define los tipos de datos que se utilizan en los mensajes utilizando XSD.
b) `message`: define el formato de los mensajes de solicitud y respuesta.

c) `binding`: especifica los detalles del protocolo SOAP.
d) `service`: define los *endpoints* expuestos. Contiene los elementos `port`, cada uno es un *endpoint* de la aplicación. De hecho, en WSDL 2.0, se llaman *endpoint*.
e) `portType`: Detalla los mensajes de solicitud y respuesta de cada `port` (*endpoint*).

Es mejor verlo con un ejemplo. La siguiente imagen muestra las secciones principales de un fichero WSDL. Este documento describe un servicio que expone un *endpoint* para calcular potencias. Recibe dos argumentos, la base y el exponente, y devuelve el resultado de elevar la base al exponente.

```
<definitions name="Mate" targetNamespace="http://localhost/cap10/servidorsoapmate.php">
 +<types></types>
 +<portType name="MatePort"></portType>
 +<binding name="MateBinding" type="tns:MatePort"></binding>
 +<service name="MateService"></service>
 +<message name="potenciaIn"></message>
 +<message name="potenciaOut"></message>
</definitions>
```

Figura 10.1
Secciones
de un fichero WSDL.

Con `types` se definen los tipos de datos que se utilizan en los mensajes utilizando XSD. En este ejemplo se definen el elemento `potencia`, formado por dos elementos de tipo `float`, `base` y `exponente`, y el elemento `potenciaResponse`, que solo tiene un elemento `float`. El primero es el tipo de dato de los mensajes de petición; el segundo, el de los mensajes de respuesta.

```
<types>
   <xsd:schema targetNamespace="http://localhost/cap10/matewsdl.php">
      <xsd:element name="potencia">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="base" type="xsd:float"/>
               <xsd:element name="exponente" type="xsd:float"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="potenciaResponse">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="potenciaResult" type="xsd:float"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
   </xsd:schema>
</types>
```

Los elementos `message` especifican el formato de los mensajes de solicitud y respuesta. El mensaje `PotenciaIn`, el que se usa para solicitar el servicio, incluye un elemento potencia.

```
<message name="potenciaIn">
   <part name="parameters" element="tns:potencia"/>
</message>
```

Con portType se describen los mensajes de solicitud y respuesta de cada *endpoint*. El portType `MatePort` recibe un mensaje de tipo `potenciaIn` y devuelve un mensaje de tipo `potenciaOut` (definidos como elementos `message`).

```
<portType name="MatePort">
    <operation name="potencia">
        <documentation>/**</documentation>
        <input message="tns:potenciaIn"/>
        <output message="tns:potenciaOut"/>
    </operation>
</portType>
```

El elemento `binding` especifica los detalles del protocolo SOAP.

```
<binding name="MateBinding" type="tns:MatePort">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="potencia">
        <soap:operation soapAction="http://localhost/cap10/matewsdl.
        php#potencia"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
```

En el elemento `service` se definen los *endpoints* expuestos a través de elementos `port`. Cada elemento `port` es un *endpoint* de la aplicación. De hecho, en WSDL 2.0 se llaman *endpoint*. En el ejemplo, el elemento `port` utiliza `name="MatePort"`. Esto quiere decir que es el de tipo `MatePort`, definido con el elemento `portType`.

```
<service name="MateService">
    <port name="MatePort" binding="tns:MateBinding">
        <soap:address location="http://localhost/cap10/matewsdl.php"/>
    </port>
</service>
```

Recurso web

Puedes ver el documento completo accediendo a http://localhost/cap10/matewsdl.php.

10.3. Librerías de PHP para servicios web

La extensión SOAP de PHP permite crear clientes y servidores SOAP. La librería Zend\SOAP está desarrollada a partir de esta extensión y añade clases útiles para el desarrollo. Entre otras cosas, para generar automáticamente ficheros WDSL.

Para activar la extensión SOAP hay que "descomentar" la línea

```
extension=soap
```

del fichero **php.ini** y reiniciar el servidor para que la cargue.
La librería Zend\Soap se instala utilizando `composer:`

```
composer require zendframework/zend-soap
```

10.3.1. Generación de servicios web

Para crear un servicio web se utiliza un objeto de la clase Zend\Soap\Server. Este objeto se asocia con una clase cuyos métodos serán los métodos expuestos por el servicio. Por ejemplo, para un servicio con operaciones matemáticas se podría usar la siguiente clase:

```
class Mate{
    /**
    * @param float $base
    * @param float $exponente
    * @return float $resul
    */
    public function potencia($base, $exponente)
    {
        return pow($base, $exponente);
    }
}
```

El primer paso es generar un fichero WSDL con la descripción del servicio. Con la clase Zend\Soap\AutoDiscover se puede generar automáticamente mediante introspección del código, lo que ahorra la tarea al programador. Para que funcione bien es importante anotar los métodos con bloques DocBlock indicando al menos nombre y tipo de datos de los argumentos y tipo de dato devuelto, como se puede ver en la clase `Mate`.

El siguiente ejemplo genera el WSDL necesario para exponer el método de la clase `Mate`:

```
<?php
require_once __DIR__. '/vendor/autoload.php';
require "Mate.php";
$serverUrl = "http://localhost/cap10/matewsdl.php";
$soapAutoDiscover = new \Zend\Soap\AutoDiscover(new \Zend\Soap\Wsdl\
ComplexTypeStrategy\ArrayOfTypeSequence());
$soapAutoDiscover->setBindingStyle(array('style' => 'document'));
$soapAutoDiscover->setOperationBodyStyle(array('use' => 'literal'));
$soapAutoDiscover->setClass('Mate');
```

```
$soapAutoDiscover->setUri($serverUrl);
header("Content-Type: text/xml");
echo $soapAutoDiscover->generate()->toXml();
```

Si se accede desde el navegador, se obtiene:

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://localhost/cap10/matewsdl.php"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    name="Mate"
    targetNamespace="http://localhost/cap10/matewsdl.php">
    <types>
        <xsd:schema targetNamespace="http://localhost/cap10/matewsdl.php">
            <xsd:element name="potencia">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="base" type="xsd:float"/>
                        <xsd:element name="exponente" type="xsd:float"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="potenciaResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="potenciaResult" type="xsd:float"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    </types>
    <portType name="MatePort">
        <operation name="potencia">
            <documentation>/**</documentation>
            <input message="tns:potenciaIn"/>
            <output message="tns:potenciaOut"/>
        </operation>
    </portType>
    <binding name="MateBinding" type="tns:MatePort">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
            <operation name="potencia">
                <soap:operation
                    soapAction="http://localhost/cap10/matewsdl.php#poten-
                    cia"/>
```

```
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="MateService">
        <port name="MatePort" binding="tns:MateBinding">
        <soap:address location="http://localhost/cap10/matewsdl.php"/>
    </port>
    </service>
    <message name="potenciaIn">
        <part name="parameters" element="tns:potencia"/>
    </message>
    <message name="potenciaOut">
        <part name="parameters" element="tns:potenciaResponse"/>
</message>
</definitions>
```

Como se puede observar, utiliza los métodos y anotaciones de la clase para crear los tipos de datos, mensajes y *bindings*.

Partiendo de esta base, el siguiente fichero se encarga tanto de generar el fichero WSDL como de crear el servidor y pasarle las peticiones. Si se accede al ejemplo añadiendo el parámetro `wsdl` a la ruta (líneas 5-14), se devuelve el WSDL, como en el ejemplo anterior. Si no, crea el servidor SOAP y le pasa la petición.

```
1  <?php
2  require_once __DIR__. '/vendor/autoload.php';
3  require "Mate.php";
4  $serverUrl = "http://localhost/cap10/servidorsoapmate.php";
5  if (isset($_GET['wsdl'])) {
6      $soapAutoDiscover = new \Zend\Soap\AutoDiscover
7      (new \Zend\Soap\Wsdl\ComplexTypeStrategy\ArrayOfTypeSequence());
8      $soapAutoDiscover->setBindingStyle(array('style' => 'document'));
9      $soapAutoDiscover->setOperationBodyStyle(array('use' => 'literal'));
10     $soapAutoDiscover->setClass('Mate');
11     $soapAutoDiscover->setUri($serverUrl);
12     header("Content-Type: text/xml");
13     echo $soapAutoDiscover->generate()->toXml();
14 } else {
15     $soap = new \Zend\Soap\Server($serverUrl. '?wsdl');
16     $soap->setObject(new \Zend\Soap\Server\DocumentLiteralWrapper
           (new Mate()));
17     $soap->handle();
18 }
```

Para crear el objeto servidor, hay que llamar al constructor con la ruta del fichero WSDL (línea 15). La ruta en este caso es la del propio fichero, pero con el parámetro `wdsl`.

Una vez creado, se asocia con la clase `Mate` (línea 16) y se le pasa la petición con `handle()` (línea 17). Aquí se produce el envío de la solicitud.

Actividad propuesta 10.1

Añade un método para calcular el factorial de un número a la clase Mate. Anótalo correctamente y comprueba el nuevo fichero WSDL generado.

10.3.2. Utilización de servicios web

Para utilizar un servicio SOAP se emplea la clase `Client`. Se inicializa también a partir del fichero WSDL. El objeto que se obtiene tendrá como métodos los *endpoints* expuestos.

```
$cliente = new Zend\Soap\Client(ruta);
```

Si se crea un cliente para el fichero WSDL anterior, tendrá un método potencia, correspondiente al *endpoint* potencia. Para pasarle argumentos se usa un *array*. Las claves son los nombres que figuran en el fichero WSDL.

```
<?php
require_once __DIR__. '../vendor/autoload.php';
   $cliente = new Zend\Soap\Client('http://localhost/cap10/servidor-
   soapmate.php?wsdl');
$result = $cliente->potencia(['base' => 2, 'exponente' => 3]);
echo $result->potenciaResult;
```

Si se accede desde el navegador, mostrará un 8 (el resultado de elevar 2 a 3).

Como se puede ver en el ejemplo, el objeto también tiene un atributo `potenciaResult`, que guarda el resultado para el *endpoint* `potencia`. Si hubiera más *endpoints*, habría más atributos `<método>Result`.

Al llamar al método potencia, el cliente monta un mensaje SOAP utilizando las definiciones del fichero WSDL. Accede al *endpoint* potencia.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:ns1="http://localhost/cap10/servidorsoapmate.php">
<env:Header/>
<env:Body>
   <ns1:potencia>
      <base>2</base>
      <exponente>3</exponente>
   </ns1:potencia>
</env:Body>
</env:Envelope>
```

En el servidor, el *endpoint* potencia está asociado con el método potencia de la clase `Mate`. El servidor SOAP la llama utilizando los datos del mensaje. Con el valor devuelto, monta el mensaje de respuesta.

Actividad propuesta 10.2

Escribe un programa para probar el *endpoint* creado para el factorial de la actividad propuesta 10.1.

10.4. Aplicaciones híbridas

Las aplicaciones híbridas o *mashups* integran datos y funcionalidad de diferente origen en una misma interfaz gráfica. Por ejemplo, un agrupador de noticias que incluya las de varios periódicos o que use la API de Google Maps para mostrar donde han ocurrido. Se suelen clasificar según su utilidad:

1. *De consumidor*. Combina datos de fuentes públicas y las muestra en una interfaz gráfica sencilla.
2. *De empresa*. Aplicaciones que combinan sus propios recursos con otros servicios externos.
3. *De datos*. En este caso, se combinan datos de diferentes orígenes, pero el resultado no es necesariamente gráfico. Constituyen un nuevo servicio que puede ser utilizado desde otros sistemas.

10.4.1. Interfaces de programación y repositorios

En general, el contenido integrado proviene de:

- Servicios web.
- Fuentes o canales *(feeds)*.
- APIs web.

A) Fuentes o canales web

Las fuentes o canales web *(feed)* son un medio de sindicación de contenido muy extendido. En los canales, las páginas web comparten el nuevo contenido que van incorporando. Los usuarios se pueden suscribir a estos canales a través de un programa apropiado (puede ser el propio navegador). Los *podcasts*, por ejemplo, funcionan así. Los formatos más utilizados son RSS y ATOM.

El Instituto Geográfico Nacional tiene varios canales RSS. Uno de ellos muestra información sobre los últimos terremotos detectados en España: http://www.ign.es/ign/RssTools/sismologia.xml.

Si se accede desde el navegador, se verá:

Figura 10.2
Canal RSS en Firefox.

El navegador aplica sus propias reglas de estilo al documento. Se puede observar que ofrece la posibilidad de suscribirse al canal. De esta manera, se notificarían los cambios en el canal. Para ver el fichero propiamente dicho hay que acceder al código fuente.

El fichero tiene la estructura habitual del formato RSS. El elemento raíz es `rss` y contiene un elemento `channel`, con los datos del canal. Cada terremoto es un elemento `item`, dentro del cual hay varios campos, entre ellos `title`, con la descripción del evento, y `geo:lat` y `geo:long`, con la latitud y longitud del epicentro.

```
<?xml version="1.0" encoding="UTF-8"?>
<rss …>
  <channel>
    <title>GeoRSS Sismología IGN</title>
    <link>http://www.ign.es/ign/rss</link>
    <description>Este canal te permite …</description>
    <language>es</language>
    <dc:language>es</dc:language>
    <item>
      <title>-Info.terremoto: 01/12/2018 19:36:18</title>
      <link>…</link>
      <description>…</description>
      <guid>… </guid>
      <geo:lat>38.4962</geo:lat>
      <geo:long>-0.6367</geo:long>
    </item>
    <item>
      …
    </item>
  </channel>
</rss>
```

El siguiente ejemplo accede al canal y muestra una tabla con la descripción y las coordenadas de cada terremoto. En el bloque inicial de PHP carga los datos y los pasa a un *array*. En el segundo bloque, muestra el *array* en una tabla.

```
<?php
$fichero = new DOMDocument();
$fichero->load( "http://www.ign.es/ign/RssTools/sismologia.xml");
$salida = array();
$terremotos = $fichero->getElementsByTagName("item");
foreach($terremotos as $entry) {
    $nuevo = array();
    $nuevo["title"] = $entry->getElementsByTagName("title")
        [0]->nodeValue;
    $nuevo["lat"] = $entry->getElementsByTagName("lat")[0]->nodeValue;
    $nuevo["lng"] = $entry->getElementsByTagName("long")[0]->nodeValue;
    $salida[] = $nuevo;
}
?>
<html>
    <head>
        <meta charset = "UTF-8">
        <title>Últimos terremotos</title>
        <style> table, td, th { border-style: solid} </style>
    </head>
    <body>
        <table>
            <tr><td>Título</td><td>Longitud</td><td>Latitud</td></tr>
            <?php
            foreach($salida as $elemento) {
                $titulo = $elemento["title"];
                $lat = $elemento["lat"];
                $lon = $elemento["lng"];
                echo "<tr><td>$titulo</td><td>$lon</td><td>$lat</td></tr>";
            }
            ?>
        </table>
    </body>
</html>
```

El resultado es el siguiente:

Título	Longitud	Latitud
–Info.terremoto: 01/12/2018 19:36:18	-0,6367	38,4962
–Info.terremoto: 01/12/2018 4:46:43	-9,7053	36,7433
–Info.terremoto: 01/12/2018 4:21:16	-11,4621	36,7835
–Info.terremoto: 30/11/2018 15:18:25	-7,437	36,6502
–Info.terremoto: 30/11/2018 6:48:00	-16,1625	30,7435
–Info.terremoto: 29/11/2018 19:25:02	-10,3245	39,2279
–Info.terremoto: 29/11/2018 9:09:36	-0,0978	43,024

Figura 10.3
Tabla de terremotos.

Ejemplo

La mayoría de los periódicos tienen uno o más canales RSS para sus noticias. Un ejemplo es el canal de últimas noticias de *El País*:

http://ep00.epimg.net/rss/tags/ultimas_noticias.xml.

Escribe un programa que cargue la fuente y muestre una lista con los titulares. Los elementos de la lista tienen que ser los vínculos a la noticia en el periódico.

```
<?php
$fichero = new DOMDocument();
$fichero->load("http://ep00.epimg.net/rss/tags/ultimas_noticias.
xml");
$salida = array();
$noticias = $fichero->getElementsByTagName("item");
foreach($noticias as $noticia) {
    $nuevo = array();
    $nuevo["titular"] = $noticia
        ->getElementsByTagName("title")[0]
        ->nodeValue;
    $nuevo["url"]=$noticia->getElementsByTagName("link")[0]->nodeVa-
    lue;
    $salida[] = $nuevo;
}
?>
<html>
    <head>
        <meta charset = "UTF-8">
        <title>Últimas noticias</title>
    </head>
    <body>
        <ul>
            <?php
            foreach($salida as $noticia) {
                $url = $noticia["url"];
                $titular = $noticia["titular"];
                echo "<li><a href = '$url'>$titular</a>";
            }
            ?>
        </ul>
    </body>
</html>
```

Para integrar estos en una aplicación híbrida, una opción es utilizar AJAX. Así se podría tener una barra lateral con los datos de los últimos terremotos o noticias que se actualice periódicamente. En este caso es mejor devolver los datos en JSON, como hace el ejemplo **terremotos_json.php**.

```php
<?php
$fichero = new DOMDocument();
$fichero->load( "http://www.ign.es/ign/RssTools/sismologia.xml");
$salida = array();
$terremotos = $fichero->getElementsByTagName("item");
foreach($terremotos as $entry) {
    $nuevo = array();
    $nuevo["title"] = $entry->getElementsByTagName("title")
    [0]->nodeValue;
    $nuevo["lat"] = $entry->getElementsByTagName("lat")[0]->nodeValue;
    $nuevo["lng"] = $entry->getElementsByTagName("long")[0]->nodeValue;
    $salida[] = $nuevo;
}
echo json_encode($salida);
```

Es como el anterior, pero devuelve el *array* en JSON en lugar de mostrar la tabla.

[{"title":"-Info.terremoto: 01\/12\/2018 19:36:18","lat":"38.4962","lng":"-0.6367"}, {"title":"-Info.terremoto: 01\/12\/2018 :46:43","lat":"36.7433","lng":"-9.7053"},...]

Actividades propuestas

10.3. Busca en internet canales RSS de alguna temática que te interese.

10.4. A partir del ejemplo anterior, crea una página que utilice JavaScript para mostrar la información de los terremotos que se actualice cada cinco minutos.

B) Google Maps

La API de Google Maps es una de las más utilizadas en aplicaciones híbridas. Permite integrar un mapa en una aplicación web. Se puede escoger el nivel de *zoom*, donde está centrado y poner marcadores. Se utiliza desde JavaScript.

Actividad propuesta 10.5

Para utilizar la API de Google Maps hace falta una clave (API *key*) que proporciona Google. Consigue una para poder probar el ejemplo:

https://developers.google.com/maps/documentation/javascript/get-api-key

Para incluirlo en una página, se utiliza un elemento *script* como este:

```
<script async defer src="https://maps.googleapis.com/maps/api/
js?key=<CLAVE>&callback=<FUNCION>">
</script>
```

Hay que sustituir <CLAVE> por la clave de la API, y <FUNCION> por el nombre de la función JavaScript a la que se llama al cargar el mapa.

La página tiene que tener un `div` con `id = 'map'`, que es donde se mostrará el mapa y estas opciones de estilo.

```
<style>
    #map {
        height: 100%;
    }
    html, body {
        height: 100%;
        margin: 0;
        padding: 0;
    }
</style>
```

El ejemplo **mapa_terremotos.php** utiliza **terremotos_json.php** para obtener los datos de los últimos terremotos y los visualiza en un mapa.

- La función `initMap()` (líneas 18-46) se encarga de inicializar el mapa.
- Es la función que se especifica al incluir el *script* (líneas 48-50). Para que el ejemplo funcione, hay que introducir una clave válida.
- El mapa se crea en las líneas 21-26 indicando el nivel de *zoom*, donde está centrado y el tipo de mapa.
- En la línea 44 se hace una petición AJAX a **terremotos_json.php**. La respuesta será un *array* de objetos con campos `title`, `long` y `lat`, los nombres que aparecen en el documento JSON.
- Cuando se recibe la respuesta (líneas 28-42), con cada uno de los elementos se añade un nuevo marcador al mapa (líneas 30-37). Se utilizan los campos `lat` y `long` para situarlo, y `title` como título. Se muestra al pasar el ratón por encima.

```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <style>
5           #map {
6               height: 100%;
7           }
8               html, body {
9               height: 100%;
10              margin: 0;
11              padding: 0;
12          }
13          </style>
14      </head>
15      <body>
16          <div id="map"></div>
17          <script>
```

```
18        function initMap() {
19              var map;
20              var results;
21                map = new google.maps.Map(
22                  document.getElementById('map'),
23                  {zoom: 5.5,
24                  center: new google.maps.LatLng(40,-4),
25                  mapTypeId: 'terrain'
26                  });
27              var xhttp = new XMLHttpRequest();
28              xhttp.onreadystatechange = function() {
29                  if (this.readyState == 4 && this.status == 200) {
30                      results = JSON.parse(this.responseText);
31                     for (var i = 0; i < results.length; i++) {
32                          var title = results[i].title;
33                          var latLng = new google.maps.LatLng(
34                              results[i].lat, results[i].lng
35                          );
36                         var marker = new google.maps.Marker({
37                         position: latLng,
38                         title:title,
39                         map: map
40                         });
41                      }
42                    }
43            }
44            xhttp.open("GET", "terremotos_json.php", true);
45            xhttp.send();
46            }
47        </script>
48        <script async defer
49            src="https://maps.googleapis.com/maps/api/js?key= &callback=
                initMap">
50        </script>
51    </body>
52 </html>
```

El resultado en el navegador es el que se muestra en la figura 10.4.

Figura 10.4
Mapa de terremotos.

Resumen

- Los servicios son componentes independientes que se pueden utilizar desde diferentes aplicaciones o sistemas.
- El W3C mantiene una serie de estándares relacionados con los servicios web, entre ellos SOAP y WDSL.
- PHP tiene una extensión, SOAP, con la funcionalidad básica. Otras librerías, como Zend\Soap, ofrecen más facilidades de uso.
- El servidor de Zend\Soap se asocia con una clase, cuyos métodos serán los *endpoints* del servicio.
- Con la librería Zend\Soap los ficheros WSDL se pueden generan a partir de la clase asociada.
- El cliente Zend\Soap se inicializa con la ruta del fichero WSDL del servicio.
- Las aplicaciones híbridas integran datos y funcionalidad a través de servicios, canales y APIs web.
- Un ejemplo muy extendido es la API de Google Maps, para integrar mapas de Google en cualquier página web.
- Las fuentes o canales web son un medio de sindicación de contenidos. Los formatos más extendidos son ATOM y RSS.
- Muchos organismos públicos y páginas web utilizan RSS para publicar información.

Ejercicios propuestos

1. Escribe una página que utilice el servicio del apartado 10.4. El usuario introducirá base y exponente en un formulario.
2. Modifica el ejercicio anterior para que el formulario se envíe con AJAX.
3. Realiza las siguientes tareas:

   a) Escribe un servicio web con un *endpoint* que devuelva los datos de la tabla de categorías de la base de datos de pedidos en formato JSON. Puedes reutilizar la función del capítulo 6.

   b) Escribe un cliente para probarlo.

4. El Ministerio de Ciencia tiene un canal RSS para publicar noticias:

   http://www.ciencia.gob.es/portal/site/MICINN/rss

   Escribe un programa que muestre una tabla a partir de los datos del canal.
5. Elabora una página que integre los ejercicios 2, 3 y 4 utilizando AJAX.
6. Busca alguna API web con funcionalidad interesante e intenta ponerla en práctica. Por ejemplo:
   - Wikipedia: https://www.mediawiki.org/wiki/API:Main_page
   - Flicker: https://www.flickr.com/services/api/

Actividades de autoevaluación

1. La descripción de los servicios web se realiza mediante:
   - ☐ *a)* REST.
   - ☐ *b)* SOAP.
   - ☐ *c)* WSDL.
2. ¿Cuál de los siguientes términos no es un estándar del W3C?
   - ☐ *a)* SOAP.
   - ☐ *b)* WSDL.
   - ☐ *c)* SOA.
3. Las aplicaciones híbridas:
   - ☐ *a)* Integran datos de diversas fuentes.
   - ☐ *b)* Integran funcionalidad de diversas fuentes.
   - ☐ *c)* Integran funcionalidad y datos de diversas fuentes.
4. La extensión SOAP de PHP y la librería Zend\Soap utilizan:
   - ☐ *a)* WSDL 1.
   - ☐ *b)* WSDL 2.
   - ☐ *c)* Según se configure al instalar.
5. La clase Zend\Soap\Server:
   - ☐ *a)* Se asocia con una clase cuyos métodos serán los *endpoints* del servicio.
   - ☐ *b)* Contiene una serie de métodos que serán los *endpoints* del servicio.
   - ☐ *c)* Define una serie de *endpoints* por defecto que se pueden sobrescribir.
6. ¿Cuál de las siguientes afirmaciones es correcta?
   - ☐ *a)* Un servicio se compone de varios *endpoints*.
   - ☐ *b)* Un *endpoint* se compone de varios servicios.
   - ☐ *c)* Un servicio se compone de un *endpoint* y un *frontpoint*.
7. El constructor de la clase Zend\Soap\Client recibe:
   - ☐ *a)* Un fichero WSDL.
   - ☐ *b)* Una clase cuyos métodos serán los *endpoints* del servicio.
   - ☐ *c)* Un fichero RSS.
8. Los canales o fuentes RSS sirven para:
   - ☐ *a)* Compartir contenido.
   - ☐ *b)* Publicar servicios web.
   - ☐ *c)* Encontrar servicios web.
9. ¿Cuál de los siguientes no es un formato para canales web?
   - ☐ *a)* UDDI.
   - ☐ *b)* ATOM.
   - ☐ *c)* RSS.

10. La API de Google Maps se utiliza:

- ☐ *a)* Desde el servidor.
- ☐ *b)* Desde el cliente.
- ☐ *c)* Se puede elegir.

SOLUCIONES:

1. a b **c**	**5.** **a** b c		
2. a b **c**	**6.** **a** b c	**9.** **a** b c	
3. a b **c**	**7.** **a** b c	**10.** a **b** c	
4. **a** b c	**8.** **a** b c		