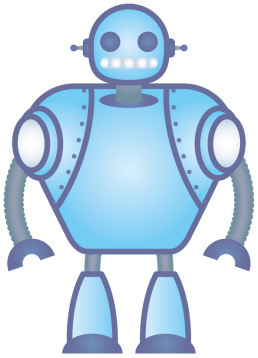


# Tema 11
# Acceso y creación de servicios

Comenzamos esta breve unidad introduciendo un nuevo concepto que, posiblemente ya conozcas. Definimos una **API** (**Application Pogramming Interface**) como un servicio que podemos consumir desde nuestras aplicaciones. Aunque muchas de las compañías más importantes, como Twitter, Amazon, eBay, Google, etc., prestan este tipo de servicios, también podremos crear nuestros propios servicios.

Este tipo de servicios se basan generalmente en la arquitectura **REST** (**Representational State Transfer**) que se originó a principios del siglo XXI tomando como base sistemas hipermedia distribuidos, entre los que se encuentra la World Wide Web. En la actualidad, esta terminología se emplea de manera mucho más amplia, describiendo cualquier interfaz que se base en el protocolo HTTP para el intercambio de información.

Los servicios RESTFUL definen un conjunto básico de operaciones: **POST**, **GET**, **PUT** y **DELETE**. Basándonos en estos verbos, podemos construir la interfaz de nuestro propio servicio, permitiendo que otros desarrolladores puedan hacer uso en sus aplicaciones de las funcionalidades que se les proporcione. No utilizaremos estos protocolos de cualquier manera, sino que se emplearán según las circunstancias.

| Protocolo | Descripción |
|---|---|
| **GET** | Lo utilizamos para recuperar información. |
| **POST** | Se emplea para indicar que queremos añadir información. |
| **PUT** | Indica que vamos a actualizar un dato. |
| **DELETE** | Se utiliza para indicar que se va a proceder a borrar información. |

Siempre que hacemos una petición a una API lo haremos a través de una **URI** (**Uniform Resource Identifier**), cuya estructura se especificará convenientemente en la documentación del servicio. Así, tras atender la petición, la API responderá proporcionando una respuesta utilizando generalmente **JSON** o **XML**.

## Accediendo a un servicio

Generalmente haremos uso desde PHP de la función **file_get_contents**. Ésta realiza una petición a una URI y, la respuesta obtenida nos la devuelve en forma de cadena de texto.

```
$response = file_get_contents( ... ) ;
```

Como se dijo en unidades anteriores, tras ejecutar un script PHP se genera normalmente un documento HTML. Sin embargo, nuestro lenguaje puede producir otro tipo de contenido, como documentos PDF, imágenes, XML, CSV, JSON, etc.

Recordemos que las API devolverán mayormente cadenas JSON. De esta manera, después de obtener una respuesta del servicio, tendremos que transformar dicha cadena a una estructura que nos sea mucho más manejable.

```
$data = json_decode( $response ) ;
```

La función anterior tomará la cadena JSON devuelta por la función **file_get_contents** y la transforma en un *array* que podremos manipular más fácilmente. Este método de acceso a servicios RESTFUL es el más empleado para recuperar información (protocolo **GET**) cuando desarrollamos en PHP aplicaciones sencillas.

## CURL

**cURL** es el nombre que recibe una librería que permite establecer una conexión y comunicarse con diferentes tipos de servidores utilizando protocolos como HTTP. PHP soporta por defecto esta librería, por lo que también podremos emplearla para acceder a servicios RESFTUL. Además, cURL también admite certificados HTTPS, aunque eso no será objeto de estudio en esta unidad.

Esta librería incorpora diferentes funciones que nos ayudarán a conseguir nuestro objetivo. Las más empleadas son: **curl_init**, **curl_setopt**, **curl_exec** y **curl_close**. Antes de poder acceder a cualquier API tendremos que inicializar la sesión cURL.

```
$curl_resource = curl_init() ;
```

Hecho esto, utilizamos la función **curl_setopt** para especificar la URI.

```
curl_setopt($curl_resource, CURLOPT_URL, "...") ;
```

La función anterior se utiliza también para configurar otros aspectos de la conexión, como el tipo de protocolo a utilizar, además de otras muchas más características. Recordemos que el

protocolo **GET** se emplea generalmente para solicitar información; éste será el valor por defecto, por lo que no tenemos que especificarlo. No será así cuando utilicemos los protocolos **POST**, **PUT** o **DELETE**. Teniendo todo esto en cuenta, lanzamos la petición y cerramos la conexión.

```
$data = curl_exec($curl_resource) ;
curl_close($curl_resource) ;
```

La variable **$data** debería contener la información devuelta por el servicio, pero no es así, ya que **curl_exec** retorna **true** en caso de éxito o **false** en caso de error, excepto si indicamos explícitamente que queremos obtener un resultado. Hacemos esto con la opción **CURLOPT_RETURNTRANSFER**.

```
curl_setopt($curl_resource, CURLOP_RETURNTRANSFER, true) ;
```

Ahora, **curl_exec** devolverá el resultado o **false** si se ha producido un error. La variable **$data** tendrá el resultado devuelto por la API, pero la tendremos en formato cadena, por lo que tendremos que convertirla en un objeto que sea más manejable.

```
$decodified = json_decode($data) ;
```

Aunque esto funciona y suele ser suficiente, no sería del todo correcto. Realmente , al hacer la petición, tendríamos que indicar de manera explícita que esperamos recibir un valor de tipo JSON.

```
curl_setopt($curl_resource, CURLOPT_HTTPHEADER, ["Accept: application/json"]) ;
```

Supongamos ahora que queremos realizar una petición **POST**. Recordemos que, generalmente utilizamos este protocolo para realizar una inserción de datos  por lo que, en primer lugar tendremos que definir la colección de valores que le proporcionaremos a la API para que los guarde convenientemente.

```
$data = [ "clave1" => valor1, "clave2" => valor2, ...] ;
```

Seguidamente, tendremos que transformar el *array* anterior y codificarlo en formato URL.

```
$query = http_build_query( $data ) ;
```

Suponiendo que el *array* tuviese únicamente dos parejas clave-valor, la función anterior genera la siguiente cadena.

```
clave1=valor1&clave2=valor2
```

Hecho esto, iniciamos la conexión y la configuramos convenientemente.

```
$curl_resorurce = curl_init() ;
curl_setopt($curl_resource, CURLOPT_URL, " ... ") ;
curl_setopt($curl_resource, CURLOPT_POST, true) ;
curl_setopt($curl_resource, CURLOPT_POSTFIELDS, $query) ;
```

Las dos últimas llamadas a **curl_setopt** configuran el protocolo y la información que se enviará a través de este. En última instancia, lanzamos la petición y cerramos la conexión.

```
$data = curl_exec($curl_resource) ;
curl_close($curl_resource) ;
```

Llegados a este punto cabría hacerse la siguiente pregunta: ¿cómo hacemos entonces si queremos utilizar los protocolos **PUT** y **DELETE**? Ambos funcionan como POST, esto es, envían la información codificada en el cuerpo del mensaje HTTP. Sin embargo, cuando utilicemos la librería cURL tendremos que configurar la conexión utilizando la opción **CURLOPT_CUSTOMREQUEST**.

```
curl_setopt($curl_resource, CURLOPT_CUSTOMREQUEST, protocolo ) ;
```

## Creando nuestra propia API

Hasta ahora hemos visto cómo acceder a un servicio RESFUL para obtener y, en ocasiones gestionar la información que estos contienen. En los ejemplos que hemos hecho en clase hemos utilizado una API en concreto para básicamente recuperar información, aunque también podríamos haberla modificado.

Sin embargo, como desarrolladores, también podemos crear nuestros propios servicios web. Cuando creemos nuestra propia API tendremos que pensar, en primer lugar, qué funcionalidades queremos proporcionar a aquellos usuarios que vayan a utilizar nuestra API. Supongamos que hemos desarrollado una aplicación para gestionar la información de la biblioteca pública de nuestra localidad y ahora queremos crear un servicio para que pueda ser utilizado por otros usuarios y/o aplicaciones.

| Código de operación | Descripción |
|---|---|
| **libros** | Proporciona una relación con información sobre todos los libros de la biblioteca. |
| **ejemplar** | Proporciona información sobre un determinado libro. |

Cuando realizamos una petición a una API ésta debe identificar de alguna manera la operación que debe realizar. Esto nos lleva a proporcionarle el código de operación a través de un parámetro, conjuntamente con aquellos que le sean necesarios para realizar la tarea que le estamos solicitando. Si queremos recuperar información sobre todos los libros de la biblioteca haremos:

```
www.mibiblioteca.com/api/index.php?operacion=libros
```

Y si, por el contrario, queremos recuperar información sobre un sólo ejemplar.

```
www.mibiblioteca.com/api/index.php?operacion=ejemplar&titulo=titulo_del_libro
```

Todo esto suponiendo que nuestra API esté accesible en la URL **www.mibiblioteca.com/api**. Esto es así generalmente, aunque también suelen agruparse según versiones.

```
www.mibiblioteca.com/api/v1
www.mibiblioteca.com/api/v2
...
```

Al recibir la petición, y antes que nada, lo más importante será conocer la operación que se nos está solicitando y, en función de esto realizar una u otra tarea.

```
$operacion = $_GET["operacion"] ;

if      ($operacion=="libros")   { ... }
else if ($operacion=="ejemplar") { ... }
...
```

Generalmente, la API devolverá una colección de datos. Supongamos que contamos con la siguiente información de nuestro libro.

```
["titulo" => "Leyendas", "autor" => "Gustavo Adolfo Bécquer", "genero" => "prosa"]
```

Sin embargo, esta información estará contenida principalmente en un objeto o array, como sucede en este caso. Transformaremos esta estructura de datos en la cadena JSON que se entregará como respuesta.

```
$respuesta = json_encode( array_de_datos ) ;
```

Finalmente, bastará con volcar el resultado en el documento que se genera a partir de la ejecución de nuestro script PHP. Sin embargo, dicho documento tiene por defecto formato

HTML y nosotros vamos a devolver una cadena JSON. Si no hacemos nada al respecto, el cliente interpretará la respuesta de manera errónea, así que tendremos que indicar explícitamente el formato de la información que se está devolviendo. Si recordamos, esta información se envía en la cabecera del paquete HTTP, por lo que tendremos que modificar el contenido de la misma.

```
header("Content-Type: application/json") ;
```

La función header de PHP nos permite modificar la información que se envía en la cabecera de un mensaje HTTP. Recuerda que la cabecera se envía siempre antes que el cuerpo, por lo que la línea anterior deberá aparecer en el código antes de volcar cualquier contenido en el documento generado por el script.

```
echo $respuesta ;
```

## API key

Si curioseamos por Internet encontraremos numerosas API de libre acceso. No obstante, muchas plataformas exigen que el usuario se registre, proporcionándole una clave que lo identifique unívocamente y le permita acceder al servicio. Hasta ahora, no hemos necesitado hacer esto, pero en este apartado veremos cómo trabajar con una **API key**.

Sin meternos en excesivas complicaciones utilizaremos la información del usuario para generar una **API key**. Obviamente, debemos garantizar que dicha clave es única, así que meteremos también en la coctelera una semilla aleatoria junto con un método de encriptación, agitamos y, voilá, obtenemos nuestra clave.

```
md5( time()."usuario@email.com") ;
```

Aquí vemos cómo hemos utilizado una marca de tiempo como semilla y la dirección de email del usuario para generar la clave, que ha sido codificada utilizando el algoritmo de encriptación MD5. PHP nos proporciona diferentes métodos de encriptación (**md5**, **sha1**, **hash**, **crypt** y **password_hash**) y su uso depende del nivel de seguridad que quedamos aplicar a nuestra clave.

Obviamente, la clave **debe ser persistente**, por lo que tendremos que guardarla convenientemente y asociarla al usuario. Generalmente, tal y como veremos más adelante, utilizaremos para ello una base de datos.

Hecho esto, se le exigirá al usuario que proporcione la API key en la URL junto con la operación a realizar y los parámetros necesarios. Nuestro script PHP recibirá la petición y, antes de atenderla tendrá que comprobar si la clave que se nos ha proporcionado es correcta y está vinculada a algún usuario. Si esto sucede, se realizará la tarea solicitada; en

otro caso, nuestra API devolverá un mensaje de error junto con el código **401 Unauthorized**. Recordemos que la función **header** nos permite definir el código de respuesta de la petición HTTP.

## Maquillando la URL

Supongamos que un usuario quiere recuperar información sobre todos los libros de la biblioteca. Como hemos dicho anteriormente, deberá realizar su petición a través de la URL:

```
www.mibiblioteca.com/api/index.php?operacion=libros
```

Sin embargo, si protegemos nuestro servicio a través de una API key, también deberemos incluirla en la petición anterior.

```
www.mibiblioteca.com/api/index.php?operacion=libros&key=97e152c7a1cf29658f7e1223268c23ee
```

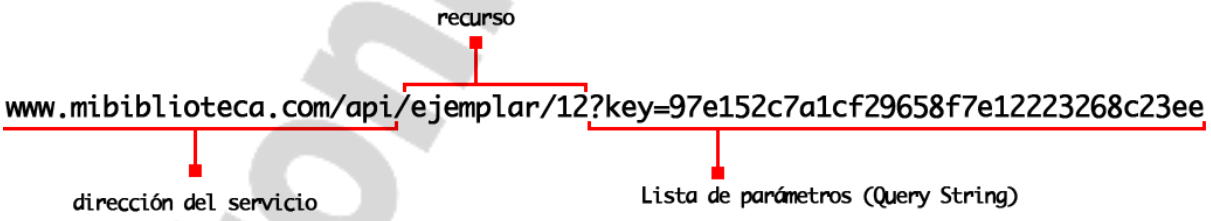
Escribir una URL de esta manera resulta poco amigable y, según el número de parámetros que sean necesarios, también será poco manejable. ¿Existe alguna manera de *maquillar* la URL para que esto no suceda? La respuesta es sí. Nuestra URL podría escribirse como sigue:

```
www.mibiblioteca.com/api/libros?key=97e152c7a1cf29658f7e1223268c23ee
```

O, si estamos solicitando información sobre un ejemplar:

```
www.mibiblioteca.com/api/ejemplar/12?key=97e152c7a1cf29658f7e1223268c23ee
```

Donde **libros** y **ejemplar/12** indican el recurso que se solicita y **key** la clave asociada al usuario, tal y como se ve en la siguiente imagen.




Pero, ¿cómo hacemos todo esto? La respuesta la encontramos en el archivo **.htaccess** (**HyperText Access**), un fichero de configuración de Apache con el que podemos modificar el comportamiento del servidor. Utilizamos este archivo para especificar qué puede hacer y qué no el usuario, restringir el acceso a carpetas, realizar redirecciones y un largo etcétera. Sin embargo, todo esto excede los contenidos de este módulo, por lo que nos centraremos únicamente en lo que realmente nos es de interés, y esto es la capacidad de **.htaccess** para traducir direcciones URL.

Realmente, lo que hacemos en este archivo es definir el comportamiento del servidor ante cualquiera de las URL anteriores e indicar cómo debe traducirlas. Debemos recordar que, para utilizar esta funcionalidad tendremos que activarla.

```
RewriteEngine On
```

Esta orden deberá ser la primera que encontremos en el archivo **.htaccess**. A continuación, definimos tantas reglas de traducción o **reescritura de URL** como necesitemos, siguiendo la siguiente sintaxis.

**RewriteRule url_patron url_destino**

La instrucción **RewriteRule** indicará al servidor que, cuando un visitante acceda una URL que coincida con el patrón indicado, ésta deberá traducirse internamente a una URL real. El patrón podrá definirse empleando expresiones regulares. De esta manera, gracias a este mecanismo, las direcciones serán mucho más amigables para los visitantes y buscadores que indexen los contenidos de nuestra web.

```
RewriteEngine On
RewriteRule ^libros$                  index.php?operacion=libros [QSA,L]
RewriteRule ^ejemplar/([1-9][0-9]*)$ index.php?operacion=ejemplar&id=$1 [QSA,L]
```

Escribir las expresiones regulares es quizá la tarea más compleja que nos vamos a encontrar a la hora de definir URLs amigables. El fragmento de código anterior nos muestra cómo tendríamos que definir las reglas correspondientes a los ejemplos que vimos anteriormente.

Como el patrón es una expresión regular deberá comenzar por **^** y finalizar con **$**. Seguidamente, escribimos la URL real y terminamos con una secuencia de flags encerrados entre corchetes. En este caso **QSA** indicará al servidor que anexe posibles parámetros al final de la URL real, en tanto que **L** hará que el servidor detenga el proceso de traducción de reglas.

En la segunda regla encontramos una expresión regular más compleja. El patrón indicará al servidor que la URL a traducir debe tener dos partes. La primera de ellas será la palabra **ejemplar** y, separada por **/** encontraremos un número mayor o igual a **1**. Como dicho valor es incierto, utilizaremos **$1** para referirnos a él si es necesario. Imagina ahora la siguiente regla.

```
RewriteRule ^foo/([A-Z]+)/([0-9]+)$ index.php?operacion=foo&a=$1&b=$2 [L]
```

En este caso, como tenemos dos valores indeterminados separados por sendas barras, nos referiremos al primero de ellos con **$1** y al segundo con **$2**. ¿Sabrías indicar qué definen

cada una de esas expresiones regulares? Ambas indican que en la URL deberán aparecer, como mínimo, una letra mayúscula entre **A** y **Z**, y un número entre **0** y **9** respectivamente.