

UNIDAD DIDÁCTICA 6:

FORMULARIOS Y PHP. MÉTODOS POST Y GET

Módulo profesional:
Desarrollo Web en Entorno Servidor

Índice

RESUMEN INTRODUCTORIO	3
INTRODUCCIÓN	3
CASO INTRODUCTORIO	3
1. RESUMEN FORMULARIOS SOBRE HTML	4
1.1. Tipo text	6
1.2. Elemento select	7
1.3. Tipo submit	9
1.4. Otros elementos	10
2. MÉTODOS HTTP	11
2.1. Protocolo HTTP y modelo cliente-servidor	11
2.2. HTTP Request	14
2.3. Métodos dentro de HTTP	15
2.4. Formularios. Método GET y POST	16
2.5. Método GET	17
2.6. Método POST	17
2.7. GET vs POST	18
3. FORMULARIOS Y PHP	19
3.1. SuperGlobal \$_POST, \$_GET	19
3.2. Funciones de comprobación y verificación	21
3.3. Formularios y objetos	23
RESUMEN FINAL	26

RESUMEN INTRODUCTORIO

En esta unidad comenzaremos a usar los formularios para recibir información por parte del usuario.

Igual que ocurre en otros lenguajes, resulta imprescindible tener elementos de interacción con el usuario para recibir datos e información.

INTRODUCCIÓN

El lenguaje PHP está inevitablemente unido al lenguaje HTML, hemos visto y utilizado el PHP para poder modificar el código final HTML, mediante estructuras de decisión o de repetición.

Llega el momento de interactuar con el usuario, y para ello utilizaremos los formularios en concreto la etiqueta `<form>` justamente para recibir esta información.

Es por lo tanto el camino contrario, donde a partir de código html podemos recoger información y ser utilizada sobre todo para interactuar con las bases de datos, este punto se abordará en la unidad didáctica siguiente donde se sumarán e interactuarán objetos, formularios y estructuras PHP

CASO INTRODUCTORIO

Al finalizar la unidad seremos capaces de programar un formulario para poder calcular la masa corporal de una persona a partir de una serie de información requerida.

1.RESUMEN FORMULARIOS SOBRE HTML

No es el objetivo de este módulo profundizar en el código HTML y por lo tanto en el código sobre formularios. Sin embargo tal y como estamos viendo a lo largo del curso, el lenguaje PHP se sirve del HTML como pasarela para mostrar resultados y visualizarlos de una forma correcta. Por lo tanto toca repasar y sentar al menos las bases del mínimo necesario para poder representar un formulario escrito con HTML.

Comencemos con un código base extraído de W3Schools:

Ejemplo de formulario básico

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>
  <body>

    <form action="/action_page.php">
      First name:<br>
      <input type="text" name="firstname" value="Mickey">
      <br>
      Last name:<br>
      <input type="text" name="lastname" value="Mouse">
      <br><br>
      <input type="submit" value="Submit">
    </form>

    <p>If you click the "Submit" button, the form-data will be sent to
    a page called "/action_page.php".</p>

  </body>
</html>
```

First name:

Last name:

Imagen 1: Formulario

Fuente de la imagen: https://www.w3schools.com/html/html_forms.asp

En concreto la parte importante y concreta referida al formulario es:

Etiqueta form

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" name="firstname" value="Mickey">
  <br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse">
  <br><br>
  <input type="submit" value="Submit">
</form>
```

La etiqueta HTML que define un formulario es **<form></form>**, la cual nos permite definir el resto de atributos y elementos dentro de un formulario. Dentro de la etiqueta form podemos seguir escribiendo y utilizando etiquetas html para dar formato a nuestro formulario, pero existen etiquetas específicas que nos permiten realizar esto, tales como **<label>**.

De las etiquetas más importantes nos encontramos con la etiqueta **<input>**, que nos va a permitir definir los diferentes elementos de entrada y por lo tanto de interacción con el usuario. Dependiendo del atributo **type**, el elemento tendrá una funcionalidad u otra. Algunos ejemplos:

tipo	descripción
<input type="text">	Define una caja de texto libre
<input type="radio">	Define un tipo selector de tipo <i>radio button</i>
<input type="submit">	Define un tipo botón

Veamos la explicación completa a través de un vídeo:



En el vídeo que encontrarás una introducción a los formularios

✓ <https://youtu.be/diEDoKJ-c9k>

1.1. Tipo text

Uno de los tipos más importantes es el tipo text. Veámoslo a través de un ejemplo que nos proporciona W3Schools:

Tipo text

```
<form>
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

Es el mismo ejemplo que antes habíamos visto en la introducción con los formularios, sin embargo nos vamos a centrar en uno de los tipos más utilizados para recoger información por parte del usuario, tipo="text".


Este tipo define una caja de texto, alfanumérica de entrada libre. Esto es importante ya que después en el momento de recoger la información introducida por parte del usuario, tendremos que tener en cuenta que tiene esas dos características:

- Es libre, y por lo tanto deberemos realizar una comprobación de errores y/o de datos.
- Es alfanumérica y por lo tanto deberemos realizar conversiones. Tenemos la suerte de que el lenguaje php se encarga de forma automática de esto.

En segundo lugar debemos fijarnos en un atributo MUY IMPORTANTE para nuestras necesidades de recolección de información desde la parte de

servidor. El atributo **name**. El atributo name va a permitir identificar una caja de texto con un identificador que debe ser UNICO, y que después utilizaremos dicho identificador para recoger dicha información.

Cuando liguemos esta parte con la recogida de información en PHP veremos de la importancia de este atributo.

	<p>Tras iniciarnos en la creación de los formularios vamos a generar un formulario que permita recoger la información de inscripción a nuestra web de un usuario:</p> <ol style="list-style-type: none">1) Generamos la estructura de ficheros dentro con una nueva carpeta y un html que denominaremos inscripción.html2) Crearemos la estructura básica con html3) Crearemos un nuevo formulario con la etiqueta form4) Introduciremos dos campos del tipo input text para recoger el Nombre y los Apellidos
--	---

1.2. Elemento select

Otro de los grandes elementos para la recepción de información es el elemento *select*. Veamos un ejemplo y su función:

Tipo elect
<pre><select> <option value="volvo">Volvo</option> <option value="saab">Saab</option> <option value="mercedes">Mercedes</option> <option value="audi">Audi</option> </select></pre>

Este elemento produciría el siguiente bloque en la visualización:

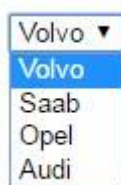


Imagen 2: Select

Fuente de la imagen:

https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_select

Nuevamente toca analizar el código que hemos utilizado:

- En primer lugar tenemos el elemento `<select>` que define una estructura desplegable de datos
- Para poder introducir cada uno de los datos que aparecerán en el desplegable utilizaremos las etiquetas `<option>`
- Dentro de la etiqueta `<option>` tendremos el atributo *value* que almacenará el valor que después utilizaremos o recogeremos desde PHP.

En el anterior ejemplo no tenemos uno de los atributos imprescindibles desde PHP, el atributo **name** y que nuevamente nos va a identificar el nombre de la variable desde el punto de vista de PHP. Volvemos a escribir de forma correcta el seleccionable de datos:

Tipo select

```
<select name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="mercedes">Mercedes</option>
  <option value="audi">Audi</option>
</select>
```

Otro de los aspectos importantes que vemos en el anterior ejemplo es la diferenciación entre la información presentada, Volvo y el valor de dicha option "volvo". Esto es importante ya que perfectamente podríamos presentar la siguiente selección de información:

Tipo select

```
<select name="cars">
  <option value="1">Volvo</option>
  <option value="2">Saab</option>
  <option value="3">Mercedes</option>
  <option value="4">Audi</option>
</select>
```

Vemos que el valor (que es el que nosotros recogeremos con el PHP) y el valor impreso por pantalla son totalmente diferentes, lo que nos proporciona una flexibilidad enorme este elemento.



COMPRUEBA LO QUE SABES:

Una vez estudiado cómo crear un formulario y diferentes elementos, ¿serías capaz de crear un formulario html con dos elementos input text y un select?

Coméntalo en el foro.

1.3. Tipo submit

Otro de los grandes y necesarios elementos, al menos desde el punto de vista de nuestras necesidades dentro de la programación de backend. Ya veremos más adelante el proceso completo de envío y recepción de información mediante un formulario. Pero para que se produzca dicho envío es necesario que exista un botón que permita al usuario enviar:

Ejemplo de formulario básico

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>
  <body>

    <form action="/action_page.php">
      First name:<br>
      <input type="text" name="firstname" value="Mickey">
```

```
<br>
Last name:<br>
<input type="text" name="lastname" value="Mouse">
<br><br>
<input type="submit" value="Submit">
</form>

<p>If you click the "Submit" button, the form-data will be sent to
a page called "/action_page.php".</p>

</body>
</html>
```

Vemos el elemento `<input type="submit" value="Submit">`, el cual no es difícil de explicar: el resultado es un botón el cual al ser apretado redigirá los datos introducidos a "action_page.php" en el ejemplo anterior, es decir hacia la página que definamos en el atributo action.



Ampliaremos nuestro formulario de inscripción antes iniciado:

- 1) Abriremos el documento inscripción.html
- 2) Añadiremos una nueva caja seleccionable para recoger el rango de edad (15 a 25 años, 26 a 35 años, 36 a 45 años, mas de 46 años)
- 3) Incluiremos un botón de submit para enviar la información.

Comenta en el foro el resultado de tu formulario.

1.4. Otros elementos

Tenemos la suerte de contar actualmente con muchos elementos dentro de un formulario, que podemos dividir en dos grandes apartados:

- Elementos que mejoran la visualización de la información, como la etiqueta `<label>`
- Elementos utilizados para recoger información como por ejemplo el elemento `type=radio`



Formularios

https://www.w3schools.com/html/html_forms.asp

Muchas son las documentaciones y código que podréis encontrar sobre forms, pero quizá una fácil y bien estructurada en la que nos proporciona w3 schools.



COMPRUEBA LO QUE SABES:

Una vez estudiado cómo crear un formulario y diferentes elementos, ¿serías capaz de complementar el anterior formulario creado con elementos label? Coméntalo en el foro.

2. MÉTODOS HTTP

Llega el momento de unir los formularios HTML a nuestro código en servidor PHP. En todo lenguaje de programación existe la forma y los mecanismos de interactuar con el usuario:

- Recogiendo información y/o estados del usuario, esto lo realizaremos a través de los formularios vistos en el apartado anterior.
- Realizando una acción a partir de la información recibida.

Nos queda por lo tanto conocer cómo somos capaces a través de la información que el usuario nos facilita, poder realizar una acción que puede implicar interactuar con una base de datos (principalmente), transformar la información facilitada, o simplemente cambiar la presentación de la aplicación web.

2.1. Protocolo HTTP y modelo cliente-servidor

El protocolo HTTP permite la comunicación entre clientes y servidores. En esta comunicación se puede intercambiar información, de hecho el

mecanismo habitual al cual el estándar WWW funciona es en el que el cliente realiza una petición mediante una URL y el servidor contesta con datos que habitualmente es un código HTML que el navegador sabe interpretar.

**RECUERDA... UD0, PROTOCOLO HTTP**

Según Ilya Grigorik (Grigorik, 2013) en su libro "High Performance Browser Networking: What every web developer should know about networking and web performance", el protocolo de transferencia de hipertexto (HTTP) es uno de los protocolos de aplicación más omnipresentes y ampliamente adoptados en Internet: es el lenguaje común entre clientes y servidores, lo que permite la web moderna. Es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.

Como ya vimos también, el protocolo HTTP funciona en un formato cliente-servidor

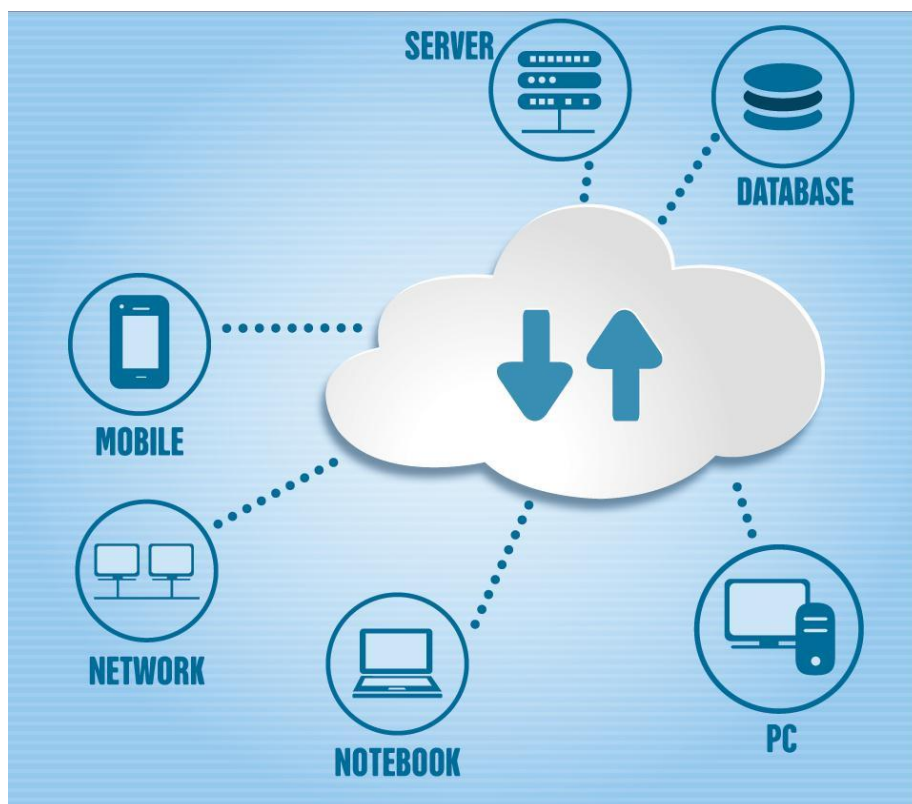


Imagen 3: Cliente-Servidor

Fuente de la imagen:

<https://www.vexels.com/vectors/preview/72547/remote-server-infographics>

¿Qué observamos en la imagen? Por un lado tenemos elementos que están realizando consultas, pidiendo información, visitando páginas, también todos ellos generando tráfico y nueva información. Son los denominados clientes dentro de este binomio que estamos analizando. En el desarrollo web, en el mundo de aplicaciones web, los clientes actualmente han dejado de ser elementos fijos, un *desktop* por ejemplo, como se puede observar en la imagen ahora mismo existen múltiples orígenes que actúan como clientes, un móvil, una tableta, un portátil, un coche, un reloj, ...

En el otro lado tenemos los servidores. Estos reciben las peticiones y/o demandas de los clientes, y dependiendo de diferentes variables, como quien solicita la pregunta, desde dónde la solicita, el navegador que está utilizando u otros parámetros es capaz de procesarla y devolver una respuesta. Las respuestas a las que estamos acostumbrados en un entorno

web, son las páginas web, pero hoy en día, cualquier aplicación de móvil está interactuando con un servidor para pedirle datos, sólo datos, que la aplicación transformará en la aplicación para presentarlos de una forma atractiva y útil.

2.2. HTTP Request

Una petición por parte de un cliente, un navegador, una aplicación, un dispositivo, ... utilizará el protocolo HTTP tal y como hemos visto anteriormente.



Imagen 4: Petición HTTP Request

Fuente de la imagen: propia

Como vemos en la imagen propuesta, y sin entrar en detalle dentro del protocolo, vemos que cuando el cliente realiza una petición utiliza el método GET, para determinar que lo que quiere es una devolución de información o de un recurso por parte del servidor.



En el vídeo que encontrarás una explicación del request y los métodos post y get

✓ <https://youtu.be/gQPI695unzg>

2.3. Métodos dentro de HTTP

Veremos más adelante que existen dentro del protocolo HTTP muchos métodos utilizables para indicar al servidor que acción a realizar.

En concreto los más utilizados son:

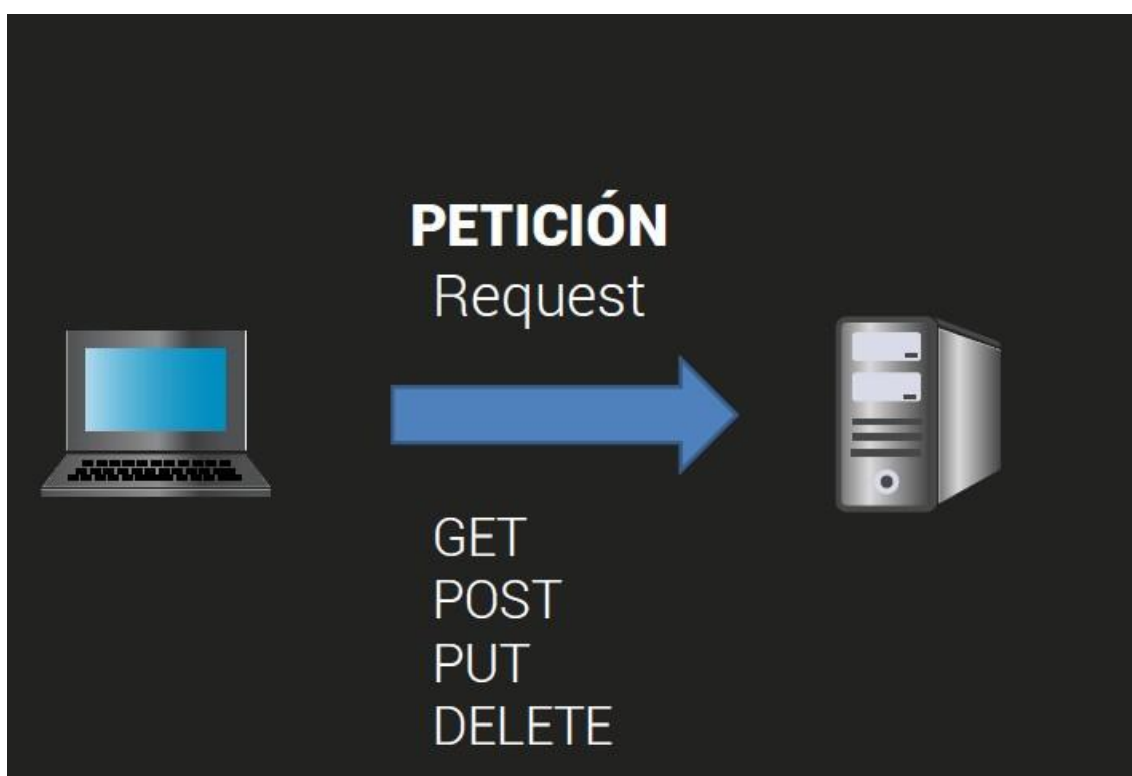


Imagen 5: Métodos más usados en HTTP para el desarrollo Web

Fuente de la imagen: propia



En el vídeo que encontrarás una explicación de los métodos más utilizados

✓ <https://youtu.be/gQPl695unzg>

2.4. Formularios. Método GET y POST

En el momento en el que utilizamos formularios, el usuario no sólo solicita datos al servidor sino que envía información mediante una petición que puede ser GET o POST:

- El método GET, habitualmente utilizamos este método en todas las peticiones, solicitamos información o un/os recurso/s al servidor
- El método POST sirve para enviar datos al servidor

Veámoslo a través de un ejemplo qué ocurre durante el uso envío de información:

Ejemplo de formulario básico

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>
  <body>

    <form action="/action_page.php" method="get">
      First name:<br>
      <input type="text" name="firstname" value="Mickey">
      <br>
      Last name:<br>
      <input type="text" name="lastname" value="Mouse">
      <br><br>
      <input type="submit" value="Submit">
    </form>

    <p>If you click the "Submit" button, the form-data will be sent to
    a page called "/action_page.php".</p>

  </body>
</html>
```


1. Tenemos el anterior formulario. El usuario rellenará la información de los dos campos propuestos "firstname" y "lastname".
2. El usuario aprieta el botón y el formulario reacciona enviando la información a la página que aparece en el atributo *action*, en este caso la página *action_page.php*
3. Dependiendo del valor *method*, la información se enviará mediante el método get o post

Veamos a continuación que significa la utilización de GET o de POST

2.5. Método GET

La información del formulario viajará a través de la URL en el navegador. Veremos entonces que la dirección de nuestra URL cambia a, *action_page.php?*

firstname=value1&lastname=value2

.

Como se observa todos los campos del formulario aparecen en dicha URL, comienzan los valores a partir del símbolo "?", cada dupla viene representada por nombre=valor, y además las duplas se separan con el símbolo &.

2.6. Método POST

En este caso la información no viaja a través de la URL, sino a través del cuerpo de la petición HTTP:

Cuerpo de la petición

```
POST action_page.php HTTP/1.1
Host: w3schools.com
firstname=value1&lastname=value2
```

Como se puede observar, los parámetros recogidos se encapsulan dentro de la cabecera HTTP, y por lo tanto no tendremos las limitaciones de tamaño que en el caso del método GET teníamos, sin embargo también perdemos otras ventajas como la de poder utilizar en llamadas con la etiqueta <a> mediante el método GET.

2.7. GET vs POST

Es importante conocer ambos métodos, sus diferencias y cuando utilizarlos.

	GET	POST
Historial	Los parámetros se guardan en el historial del buscador (browser) porque son parte del enlace (URL)	No se guardan los parámetros en el historial del buscador
Archivación (bookmark)	Puede ser archivado	No puede ser archivado
Parámetros	Puede enviarlos pero los datos de parámetros está limitado a lo que cabe en el URL. Es más seguro usar menos de 2K de los parámetros (algunos servidores llegan a 64K)	Puede enviar parámetros, incluyendo subir archivos al servidor
Vulnerabilidad	Más fácil de atacar	Más difícil de atacar
Restricciones en el tipo de data en el formulario	Sí, sólo se permiten caracteres ASCII	No tiene restricciones. Se permite data binaria.
Restricciones en la cantidad de data en formularios	Sí porque la data del formulario está en el URL y hay un límite para URLs de 2,048 caracteres, aunque puede variar	No tiene restricciones
Visibilidad	GET está visible para todo el	La variables en POST no se

	mundo (en la barra de URLs del buscador) y tiene límites a la cantidad de data que puede enviar	ven en los URLs
Valores de variables grandes	Límite de 7,607 caracteres	Límite de 8 MB

Utilizaremos en la gran mayoría de ocasiones el método POST, debido a que no tiene limitaciones en el tamaño de información, y tiene mayor seguridad. Aún así, el método GET tiene sus usos, como el de poder ser incluido en un enlace <a>.

3.FORMULARIOS Y PHP

Llegamos por fin al punto en el que unimos lo aprendido sobre la creación sobre formularios, sobre la utilización de métodos HTTP y recogida de dicha información mediante código PHP.

Libro recomendado, *Hacking with PHP* (Hudson, 2015)

En el capítulo 7 encontramos la referencia a ejemplos y documentación sobre los formularios

<http://www.hackingwithphp.com/7/0/0/html-forms>



3.1. SuperGlobal \$_POST, \$_GET

Recoger la información enviada por un formulario HTML a través de un documento PHP es supersencillo, ya que nos proporciona un array asociativo que recogerá de forma automática todo el contenido de un formulario.



En el vídeo que encontrarás una explicación cómo recoger la información desde un formulario utilizando la superglobal \$_POST

✓ <https://youtu.be/ejuYsNakmIE>



En el siguiente enlace tienes el código utilizado:



<https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/formularios>

Como podemos observar en el ejemplo y explicación, php nos facilita enormemente el trabajo ya que nos proporciona dos arrays asociativos denominados superglobals, en los cuales tendremos toda la información de los formularios html.



Variables predefinidas

<http://php.net/manual/es/reserved.variables.php>

PHP proporciona una gran cantidad de variables predefinidas para todos los scripts. Las variables representan de todo, desde variables externas hasta variables de entorno incorporadas, desde los últimos mensajes de error hasta los últimos encabezados recuperados.

En el caso del uso de método GET, la recepción será exactamente igual solo que utilizaremos la superglobal \$_GET para recoger toda la información. Pero lo mejor es utilizar ejemplos y ejercicios para ir comprendiendo el funcionamiento de dichas superglobals:



Añadiremos funcionalidad a nuestro formulario de inscripción en web:

- 1) Abriremos el documento inscripción.html
 - 2) Añadiremos el método post al formulario
 - 3) Añadiremos como action el fichero que crearemos en el siguiente punto inscripcion.php
 - 4) Dentro del mismo directorio crearemos un nuevo documento denominado inscripcion.php
 - 5) Mostraremos la información enviada por el formulario mediante la función var_dump
- Comenta en el foro el resultado de tu formulario.

3.2. Funciones de comprobación y verificación

Cuando realizamos envío de información a través de formularios mediante cualquiera de los métodos propuestos anteriormente necesitamos de métodos de control y verificación antes de realizar cualquier acción, ya que pueden darnos errores no controlados.

Las tres funciones más utilizadas cuando realizamos comprobaciones de los parámetros enviados a través de formularios son:

función	definición	ejemplo
isset	Determina si una variable está definida y no es NULL	<pre>// Esto evaluará a TRUE así que el texto se imprimirá. if (isset(\$var)) { echo "Esta variable está definida, así que se imprimirá"; }</pre>
is_null	Comprueba si la variable dada es NULL.	<pre>\$var = NULL; if (is_null(\$var)) { echo "Esta variable está definida pero su valor es nulo"; }</pre>
empty	Determina si una	<?php

variable es considerada `$var = 0;`

	<p>vacía. Una variable se considera vacía si no existe o si su valor es igual a FALSE. <code>empty()</code> no genera una advertencia si la variable no existe.</p>	<pre>// Se evalúa a true ya que \$var está vacía if (empty(\$var)) { echo '\$var es o bien 0, vacía, o no se encuentra definida en absoluto' ; } // Se evalúa como true ya que \$var está definida if (isset(\$var)) { echo '\$var está definida a pesar que está vacía'; } ?></pre>
--	---	--

Además de las funciones antes descritas puede resultar interesante consultar:

Otras funciones interesantes
<ul style="list-style-type: none"> is_bool() - Comprueba si una variable es de tipo booleano is_numeric() - Comprueba si una variable es un número o un string numérico is_float() - Comprueba si el tipo de una variable es float is_int() - Comprueba si el tipo de una variable es integer is_string() - Comprueba si una variable es de tipo string is_object() - Comprueba si una variable es un objeto is_array() - Comprueba si una variable es un array

En el caso de formularios, las funciones de comprobación más utilizadas son tanto `isset` como `empty`. Imaginemos que tenemos el siguiente formulario:

Formulario solicitado

```
<form action="conversion.php" method="post">
  <input type="text" name="nombre" value="">
  <br><br>
  <input type="submit" name="enviar" value="ENVIAR">
</form>
```

La combinación que deberíamos utilizar para asegurarnos que el valor nos llega correctamente en creación y contenido sería:

Función de comprobación

```
<?php
if(isset($_POST["nombre"])&&!empty($_POST["nombre"])){
?>
```



COMPRUEBA LO QUE SABES:

Una vez estudiado cómo comprobar y verificar la información de un formulario, ¿serías capaz de crear un formulario que envíe la información de 3 elementos tipo text y que al recoger la información se compruebe si los campos están vacíos?

Coméntalo en el foro.

3.3. Formularios y objetos

Hasta este punto, hemos utilizado los formularios escritos en HTML para poder recibir información del usuario y poder realizar cálculos, sacar por pantalla la información o realizar cambios en la página visible. También hemos visto que podíamos utilizar dos métodos de comunicación por debajo

del protocolo HTTP, como eran el método POST y el método GET para el envío de la información.

Es el turno de dar un paso más en nuestro conocimiento del PHP y unir la programación orientada a objetos que hemos estado viendo en anteriores lecciones para añadirsele al envío de información a través de formularios.

Pasaremos por lo tanto de utilizar dos ficheros a utilizar 3 y más ficheros que se relacionarán entre ellos para dar un resultado. Este proceso hará que nuestro desarrollo sea mucho más modular, mantenible y ampliable en un futuro.

En realidad no vamos a aprender o realizar nada nuevo, sino que vamos a integrar lo que sabíamos de clases y objetos para añadirlo a la recogida de información con formularios:



En el vídeo que encontrarás un primer ejemplo entre el uso de formularios y objetos

✓ <https://youtu.be/9URXva3YZ-0>



En el siguiente enlace tienes el código utilizado:



<https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/formsObjetos>

Como hemos visto en este primer ejemplo, el uso de los objetos nos permite tener el código mucho más claro, dividido en varios documentos y poder ser reutilizables.

Un segundo ejemplo:



En el vídeo que encontrarás un segundo ejemplo entre el uso de formularios y objetos

✓ <https://youtu.be/ua233-4n56Q>



En el siguiente enlace tienes el código utilizado:



<https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/formsObjetos>

En este segundo ejemplo está mucho más claro el ejemplo del uso de la reutilización de código, una de las grandes ventajas en la utilización de objetos.

RESUMEN FINAL

En esta unidad nos hemos adentrado en el uso de formularios como el mecanismo más importante para comunicarse con el usuario.

Los formularios son elementos de *cliente*, que el usuario utiliza para rellenar la información, por lo tanto para poder ser accesible por nuestro código que se encuentra en el servidor necesitamos conocer los mecanismos de comunicación y recepción.

En nuestro caso nos hemos centrado en los dos mecanismos más importantes, el método GET y el método POST que nos permiten enviar/recibir la información. El primero será utilizado principalmente para la comunicación con pocos datos o bien como enlace en la URL. El segundo será el utilizado principalmente en los formularios.

PHP, posee superglobals, \$_GET y \$_POST entre otras que nos permitirán recibir esta información para después poder manejar.

UNIDAD DIDÁCTICA 7:

BBDD, MySQL y PHP.

Módulo profesional:
Desarrollo Web en Entorno Servidor

Índice

RESUMEN INTRODUCTORIO	3
INTRODUCCIÓN	3
CASO INTRODUCTORIO	3
1. INTRODUCCIÓN AL CONCEPTO DE BASE DE DATOS	4
1.1. Base de datos Mysql.....	4
1.2. Servidor Mysql.....	6
1.3. Consideraciones sobre seguridad	8
1.4. Bases de datos en la nube	8
2. BASES DE DATOS Y PHP.....	9
2.1. Paso 1. Creando una base de datos y una tabla con PHPMYADMIN 10	
2.2. Paso 2. Crear conexión	11
2.3. Mysql frente a mysql	14
2.4. Paso 3. Lectura de datos.....	14
RESUMEN FINAL	16

RESUMEN INTRODUCTORIO

En esta unidad nos introduciremos en el uso de las bases de datos para el almacenamiento, lectura y modificación permanente de información.

Igual que ocurre en otros lenguajes, resulta imprescindible el uso de las bases de datos para mantener información y poderla utilizar, en este caso de forma individual, pero más delante de forma compartida entre varios usuarios.

INTRODUCCIÓN

Las bases de datos son el mecanismo para poder organizar y mantener de una forma estructurada y permanente información dentro de una aplicación.

Las aplicaciones Web son aplicaciones que necesitan de una forma intensiva el uso de dichas bases de datos para la realización de cualquier aplicación por muy pequeña que sea.

Al igual que ocurre en otros lenguajes de programación, para poder hacer uso de una base de datos, necesitaremos seguir los pasos de preparación, conexión y definitivamente de uso de las mismas. Pero una vez aprendidos dichos pasos, veremos que es muy fácil el uso de las bases de datos, ya que dentro de la programación web, y en concreto php, todo se resumirá al uso de arrays asociativos para moverse por la información.

CASO INTRODUCTORIO

Queremos realizar un informe de situación de una pequeña empresa a partir de la información almacenada en la base de datos de la misma.

1.INTRODUCCIÓN AL CONCEPTO DE BASE DE DATOS

Igual que ha ocurrido con el resto de conceptos y dentro del desarrollo en servidor, en nuestro caso con PHP, la creación de una aplicación depende de parámetros y tecnologías que se integran, se complementan o utilizan el PHP:

- PHP utiliza HTTP para la recepción y devolución de la información, por ello introducimos cómo funciona esta arquitectura. El modelo cliente servidor, la interpretación por parte del servidor Apache y por último los métodos POST y GET, son conceptos que hemos estudiado sobre la arquitectura HTTP.
- PHP se ejecuta junto y por encima del lenguaje HTML. PHP puede escribir HTML como se nutre de los formularios de HTML para recibir información y actuar de acuerdo a ella.

Le toca el turno de las bases de datos. No es el objetivo de este módulo estudiar en profundidad los conceptos sobre bases de datos, o sobre su tecnología, estos conceptos ya se consideran en 1º en el módulo específico sobre BBDD. Sin embargo tendremos una pequeña introducción sobre base de datos y porqué utilizamos estas junto a PHP.

1.1. Base de datos Mysql

Una de las bases de datos más utilizadas dentro del desarrollo Web es MySQL.

	<p>Mysql https://dev.mysql.com/doc/refman/5.7/en/introduction.html</p> <p>Mysql es un servidor de gestión de bases de datos relacional. Históricamente, Mysql y PostgreSQL se han repartido el monopolio del uso como gestor de bases de datos dentro de los desarrollos de aplicaciones web debido a su licencia opensource.</p> <p>Para más información sobre la última versión y los diferentes tipos de licencias, es conveniente visitar el enlace al fabricante facilitado.</p>
---	--

¿Cómo se incorpora un servidor de base de datos a nuestro modelo de arquitectura cliente-servidor? Veamoslo a través de una imagen:

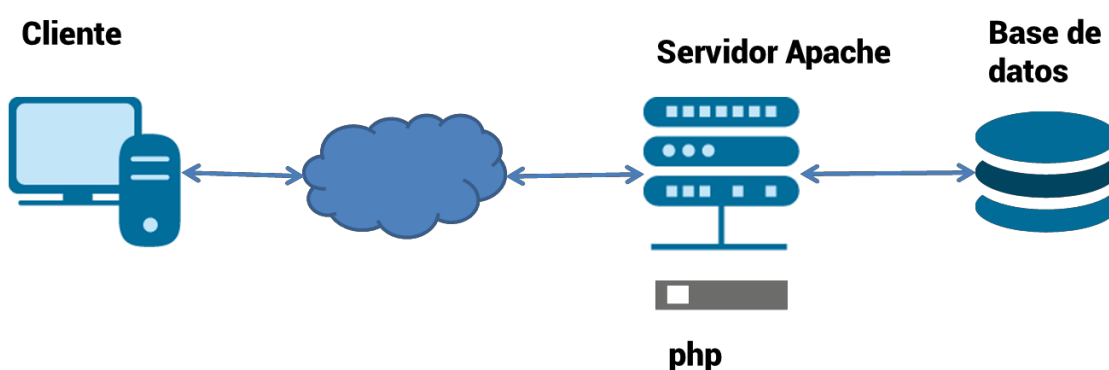


Imagen 1: Arquitectura cliente-servidor

Fuente de la imagen: propia

Podemos ver que aparece un nuevo elemento dentro de nuestro modelo, la base de datos. Al servidor de base de datos se accederá a través de nuestro ya conocido servidor Apache que será a su vez quien va a interpretar el código php.

1.2. Servidor Mysql

Por lo tanto a la vista del anterior diagrama vemos que como cualquier otro servidor, la base de datos sobre MySQL se gestionará y será accesible mediante unos parámetros de conexión que deberemos conocer:

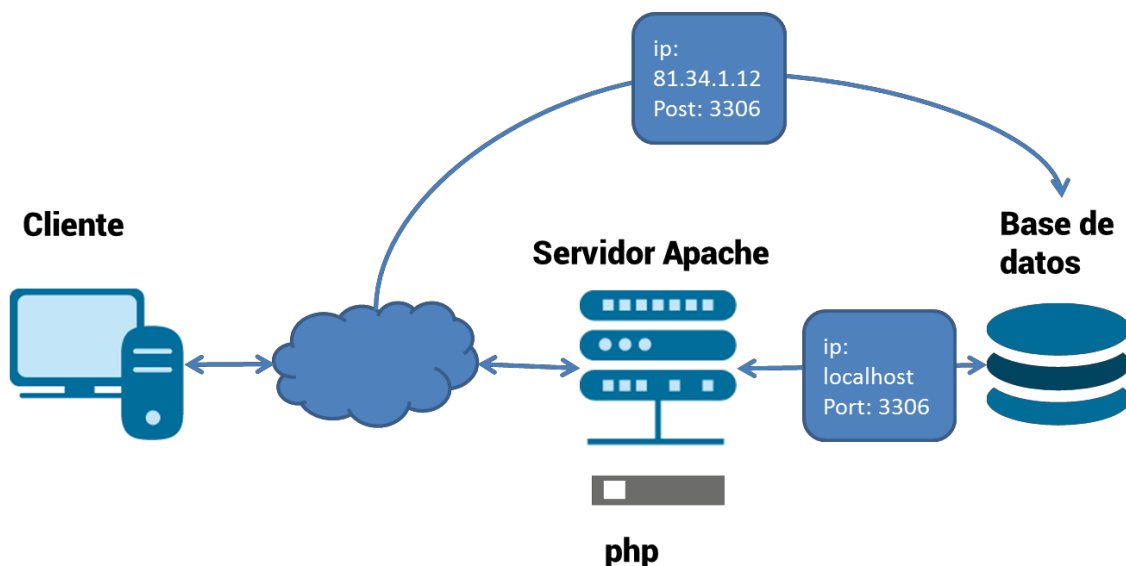


Imagen 2: Parámetros de conexión

Fuente de la imagen: propia

Como se puede observar, el servidor mysql va a tener una serie de parámetros básicos de conexión al servidor:

- **ip** o dirección de internet, dependerá desde dónde estemos "atacando" a nuestra base de datos, esta dirección será local o pública. Como podemos observar en la imagen, el proceso habitual es comunicarnos con nuestra base de datos a través del servidor Apache, y por lo tanto la dirección será "localhost".
- **Puerto**, el puerto por defecto de una base de datos mysql es el 3306, aunque es configurable.

Una vez que tenemos claros estos parámetros básicos para la conexión contra el **servidor** de base de datos debemos considerar que la interacción de nuestra aplicación se va a realizar contra una base de datos. Por ese motivo hemos querido destacar la palabra servidor:

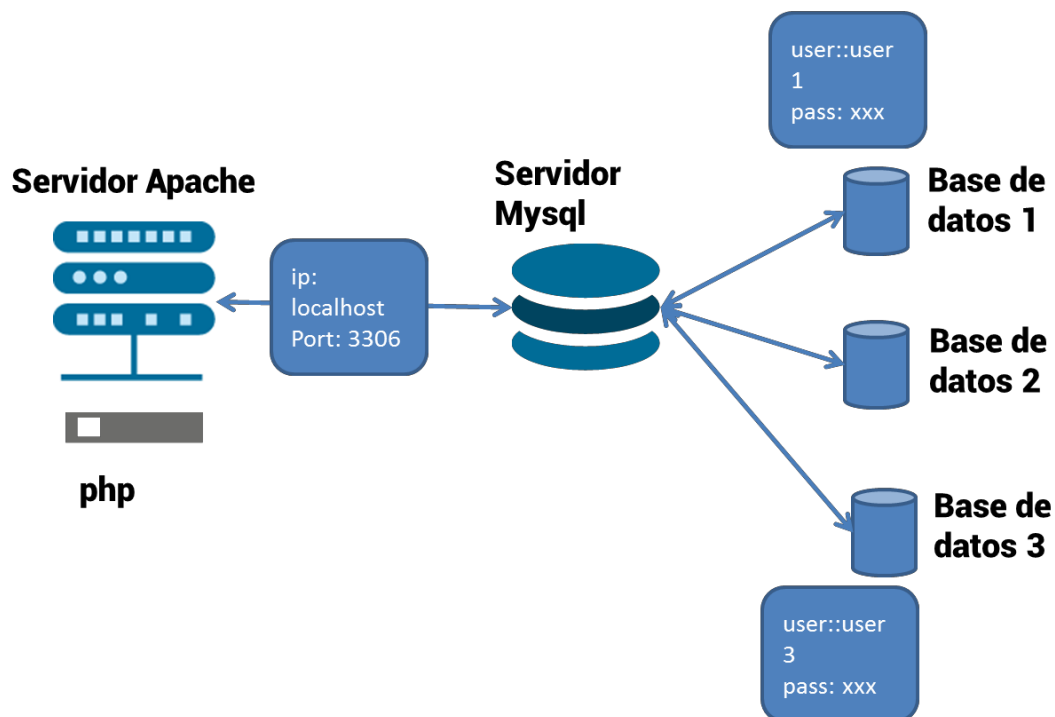


Imagen 3: Gestión de bases de datos

Fuente de la imagen: propia

Como observamos en esta imagen, al aparecer las bases de datos que gestiona nuestro servidor MySQL, ya tendríamos todos los parámetros para una conexión contra una base de datos:

- ✓ **ip**
- ✓ **puerto**
- ✓ **base de datos**
- ✓ **user**
- ✓ **pass**



COMPRUEBA LO QUE SABES:

Una vez estudiado qué es un servidor MySQL, ¿serías capaz utilizar el servidor PostgreSQL (<https://www.postgresql.org/>)? Coméntalo en el foro.

1.3. Consideraciones sobre seguridad

Tampoco queremos entrar en el detalle sobre configuración y seguridad sobre bases de datos y en concreto sobre MySQL que ya se estudió en módulos anteriores. Sin embargo sí que queremos incidir en algunos aspectos:

- El servidor de base de datos no debería ser accesible de forma directa desde el exterior o de forma pública.
- El usuario root, no debería poderse utilizar de forma pública
- Todos los usuarios deben tener contraseña
- Crear usuarios para cada una de las bases de datos o aplicaciones que se vayan a utilizar con diferentes niveles de seguridad.

1.4. Bases de datos en la nube

Actualmente con sistemas como AWS, el concepto de hosting y servidor ha evolucionado enormemente. Simplemente mostrar otra alternativa que es ahora mismo muy factible, escalable y mantenible:

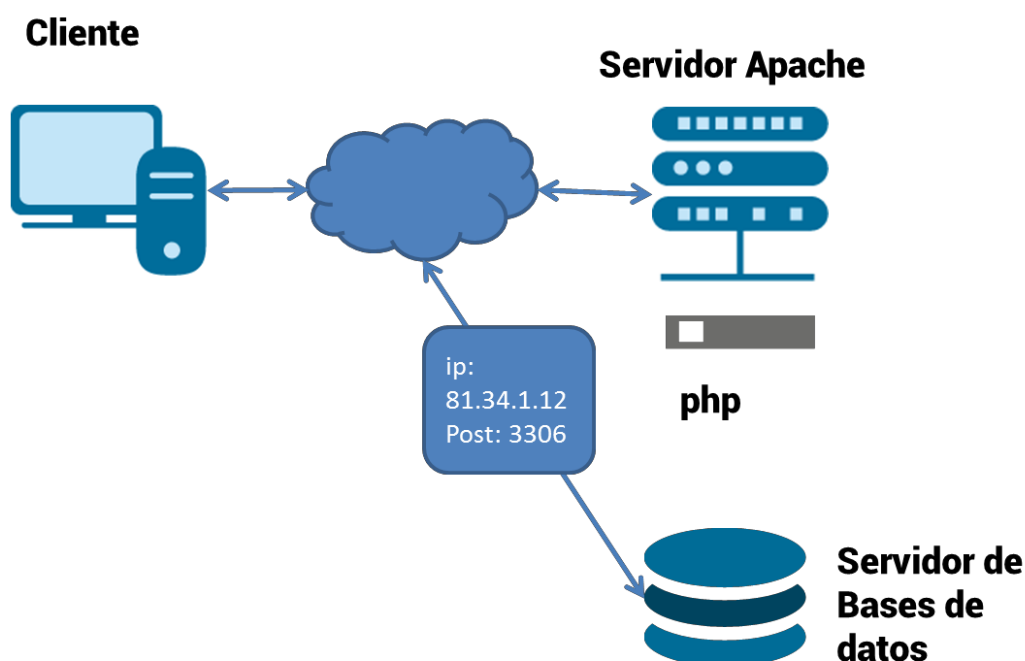


Imagen 3: Gestión de bases de datos

Fuente de la imagen: propia

Como vemos el sistema en la nube es transparente para el desarrollador, ya que va a necesitar de los mismos parámetros que con una base de datos en local. Las mejoras que introduce este sistema son:

- ✓ Si nuestro sistema está diseñado suficientemente independiente de la tecnología de base de datos, podemos escalar o incluso cambiar las bases de datos de una forma inmediata
- ✓ Los sistemas de backup o de seguridad se mejoran, ya que los servicios en la nube de sistemas de bases de datos están adaptados y configurados al mantenimiento de bases de datos

2. BASES DE DATOS Y PHP

El uso de bases de datos con PHP es muy sencillo, y más aún si ya se ha realizado programación orientada a objetos y con estructuras de datos permanentes y bases de datos relacionales como MySQL.

Los pasos serán prácticamente los mismos en cualquier lenguaje de programación orientado a objetos:

- ✓ Diseñar e implementar la estructura dentro de nuestra base de datos.
- ✓ Crear la conexión.
- ✓ Implementar las funciones necesarias para poder realizar Lectura, Escritura, Actualización y Borrado (CRUD), con una base de datos.

Libro recomendado, *Hacking with PHP* (Hudson, 2015)

En el capítulo 9 encontramos la referencia a ejemplos y documentación sobre el manejo de bases de datos con PHP

<http://www.hackingwithphp.com/9/0/0/databases>



Libro recomendado, PHP 7 Programming CookBook

(Bierer, 2016)

En el capítulo 5 encontraremos una fantástica referencia la interacción entre PHP 7 y las Bases de datos



2.1. Paso 1. Creando una base de datos y una tabla con PHPMYADMIN

Como hemos dicho anteriormente, no es el objetivo centrarnos en el uso de phpmyadmin, una herramienta fantástica para poder interactuar con nuestra base de datos en mysql. Pero sí recordaremos o realizaremos acciones básicas con esta herramienta para poder crear nuestra base de datos y nuestra primera tabla.



phpmyadmin

<https://www.phpmyadmin.net/>

phpmyadmin es una herramienta libre programada con PHP y que nos sirve para gestionar bases de datos en mysql por ejemplo

Veamos la explicación completa a través de un vídeo:



En el vídeo que encontrarás una introducción a los formularios

✓ <https://youtu.be/lisvmzREY4E>



Tras iniciarnos en la creación las bases de datos con mysql y phpmyadmin:

- 1) Abriremos un navegador y escribiremos la dirección a phpmyadmin
- 2) Crearemos una nueva base de datos con el nombre Alumnos
- 3) Crearemos una nueva tabla denominada alumnos con los siguientes campos:
 - a. Nombre, tipo varchar 64
 - b. Apellidos, tipo varchar 128
 - c. Edad, int
- 4) Introduciremos al menos 4 nuevos alumnos

2.2. Paso 2. Crear conexión

El primer paso en nuestro proceso es el de la conexión con la base de datos, y para ello necesitamos:

- Utilizar las librería mysqli (mysql mejorada)
- Enviar la información básica para poder conectarnos:
 - host, o máquina donde queremos conectarnos
 - db, o base de datos
 - user, usuario con permisos
 - password, la contraseña
- Veamos la explicación completa a través de un vídeo:



RECUERDA... UD5, OBJETOS EN PHP

La palabra reservada para la creación de un objeto es *new*, sea a partir de una clase creada por nosotros o proporcionada por PHP.



En el vídeo que encontrarás cómo generar la conexión a una base de datos

✓ https://youtu.be/aU_weN3azZ0

Y el código utilizado es:



En el siguiente enlace tienes el código utilizado:



<https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/db>

Como podemos observar tanto en el vídeo como en el código utilizado, la base de la conexión a la base de datos consiste en utilizar las librerías de mysqli:

Conexión a base de datos

```
<?php
//PLANTILLA

$mysqli = new mysqli("localhost", "usuario", "contraseña",
"basedatos");
if ($mysqli->connect_errno) {
    echo "Fallo al conectar a MySQL: (" . $mysqli->connect_errno . ")
" . $mysqli->connect_error;
}
echo $mysqli->host_info . "\n";

//BASE DE DATOS LOCAL

$mysqli = new mysqli("127.0.0.1", "usuario", "contraseña",
"basedatos", 3306);
if ($mysqli->connect_errno) {
    echo "Fallo al conectar a MySQL: (" . $mysqli->connect_errno . ")
" . $mysqli->connect_error;
}

echo $mysqli->host_info . "\n";
?>
```

Como podemos observar en el anterior código:

- Utilizamos librerías dentro de php que nos permiten la conexión a la base de datos
- El constructor de mysqli permite introducir todos los parámetros necesarios para la conexión a la base de datos, `mysqli("localhost", "usuario", "contraseña", "basedatos")`
- Mediante el método `connect_errno` podemos comprobar si la conexión se ha realizado con éxito.



Mysql mejorada

<http://php.net/manual/es/book.mysqli.php>

Podemos ver todas los métodos y clases dentro de las librerías mysql mejorada



Utilizando la base de datos creada realizaremos los siguientes pasos para la conexión:

- 1) Generamos la estructura de ficheros dentro con una nueva carpeta y un php que denominaremos alumnos.html
- 2) Realizaremos una nueva conexión creando un nuevo objeto mysqli con los parámetros necesarios para su conexión (IP,nombre de la base de datos, usuario y pass)
- 3) Comprobaremos si se ha conectado con éxito contra la bbdd mediante `connect_errno`

2.3. Mysqli frente a mysql

Como podemos observar en el anterior código y ejemplo, directamente utilizamos mysqli frente a mysql (obsoleto pero mantenido por compatibilidad hacia atrás).

Mysql es una librería no orientada a objetos y mantenida por php para desarrollos más antiguos pero que no va a ser ni mejorada y mantenida.

Por lo tanto, para nuevos desarrollos siempre utilizaremos mysqli.

2.4. Paso 3. Lectura de datos

Para la devolución de datos almacenados en una determinada tabla, deberemos utilizar la función query, la cual a su vez devolverá un resultado que es a su vez otro objeto. Un ejemplo lo tenemos en el siguiente código:

Lectura de datos

```
<?php
//Hacemos una consulta
$resultado = $mysqli->query("SELECT * FROM jugadores");
//Cuántas filas nos devuelve
echo "el número de jugadores es: ".$resultado->num_rows."<br>";
for($i=0;$i<$resultado->num_rows;$i++){
    $fila=$resultado->fetch_assoc();
    echo "El jugador ".$fila['id']." se
llama: ".$fila['nombre']."<br>";
}
?>
```

Una vez lanzada la consulta y almacenada en una variable, según la documentación, vemos que el resultado es a su vez otro objeto, mysqli_result, el cual contiene una propiedad super importante, num_rows que me permitirá conocer el número de filas de la tabla, y también el método fetch_assoc que me permitirá devolver una a una todas las filas que tiene la misma.

**Objeto `mysqli_result`**

<http://php.net/manual/es/class.mysqli-result.php>

Representa el conjunto de resultados obtenidos a partir de una consulta en la base de datos.



En el vídeo que encontrarás cómo leer datos desde la base de datos

✓ <https://youtu.be/L1-95frfJuY>



En el siguiente enlace tienes el código utilizado:



<https://github.com/pacogomezarnal/codigoEjemploCursosPHP/tree/master/db>

RESUMEN FINAL

En esta unidad nos hemos adentrado en la interacción entre BBDD y lenguaje de servidor, en nuestro caso PHP. Las aplicaciones web y en concreto el código de servidor tiene su principal objetivo en interactuar contra una BBD de datos.

Junto con los conocimientos adquiridos en la unidad anterior, interacción con formularios, queda prácticamente cerrada las posibilidades que nos permite PHP para realizar un proceso completo de interacción con el usuario:

- El usuario introduce información a través de un formulario HTML
- El lenguaje de cliente, javascript, junto con el de servidor, PHP verifican que toda la información escrita es correcta
- PHP recibe la información verificada e interactúa con la BBD
- PHP realiza una respuesta al cliente

UNIDAD DIDÁCTICA 8:

Sistema MVC.

Módulo profesional:
Desarrollo Web en Entorno Servidor

Contenido

Resumen Introductorio	3
Introducción.....	3
Caso Introductorio	3
1. Patrón MVC	4
1.1. Porqué utilizar el Patrón MVC	4
1.2. Patrón MVC.....	5
1.3. Modelo	7
1.4. Vista	7
1.5. Controlador.....	8
1.6. Patrón MVC. Ejemplo de Modelo con PHP	9
2. Vista y Formularios	11
2.1. Primer paso. Estructura de ficheros modelo, vista de resultados	13
2.2. Segundo paso. Formularios	16
3. CRUD.....	18
3.1. Primer paso. Modificación del modelo para la incorporación de la inserción.....	18
3.2. Segundo paso. Incluimos los formularios necesarios para enviar la información	20
3.3. Tercer paso. Update y delete.....	21
4. Funciones, POO y MVC	24
4.1. Parámetros por defecto	24
4.2. MVC y funciones	26
5. Herencia y MVC.....	29
5.1. Accediendo a métodos padre.....	32
5.2. Aplicándolo al Modelo	33
Resumen final:	37

Resumen Introductorio

En esta unidad aprenderemos los conceptos básicos del Patrón MVC, y del acrónimo CRUD, bases para los frameworks de desarrollo modernos y de desarrollo en equipo.

Introducción

La programación independiente, la programación procedimental, la programación tradicional basada en fichero hoy en día deja de tener sentido en un momento en el que las aplicaciones cada vez son más grandes, más dependientes del trabajo de más personas y en general necesitadas de manejar mucha información.

PHP desde su desarrollo hacia a un mundo más orientado a objetos y por lo tanto orientado hacia una programación más mantenible y escalable no es menos en introducirse en el mundo del desarrollo mediante FrameWorks como Symfony y Laravel.

Con la orientación a objetos damos un salto hacia modelos, arquitecturas y patrones de desarrollo como es el patrón MVC, Modelo-Vista-Controlador, que nos proporciona un grado importante de mejora en la organización de nuestro código, una mejora en la separación de las funciones de las clases y por lo tanto una impresionante mejora en las posibilidades de escalabilidad y mantenimiento de nuestras aplicaciones.

Caso Introductorio

Queremos realizar una aplicación que maneje diversas tablas dentro de una base de datos. Interactuar con el usuario y mantener la información mediante diversas pantallas de entrada y listado de la información.

1. Patrón MVC

1.1. Porqué utilizar el Patrón MVC

¿Por qué debemos programar orientado a objetos? ¿Por qué debemos utilizar un patrón tipo Modelo Vista Controlador (MVC)? Creo que es importante contestar a estas preguntas antes de comenzar a utilizar un patrón y un cambio de metodología de programación como el MVC ya que en un principio resulta más aparatoso o costoso utilizarlo. Y qué mejor que utilizar un ejemplo para poder ver el por qué, para ello utilizaremos el código que os presento en github y una base de datos que también os proporciono:



VIDEO DE INTERÉS

En el vídeo encontrarás una explicación de porqué utilizar el patrón MVC.



<https://youtu.be/jhw1zCec6VI>



ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:



<https://goo.gl/gtVwKq>

¿Qué ha ocurrido en el ejemplo que hemos visto? Si desarrollamos una ÚNICA página que se conecta a una ÚNICA tabla dentro de la base datos, de una ÚNICA base de datos, seguramente no tendrá ningún sentido ni la utilización del patrón MVC ni la utilización de objetos.

Pero actualmente el desarrollo de software ha crecido y se ha hecho muy muy complejo, y una aplicación no se basa en una sola página, ni en 10 ni en 20, tampoco se basa en una base de datos de 2 o 3 tablas, y por último tampoco nos encontramos habitualmente.

El patrón MVC da respuesta a las problemáticas planteadas:

- Reutilización de código de una forma ordenada y clara.
- Trabajo en equipo de trabajo con la división de tareas.
- Mantenimiento y escalabilidad del código.
- Eliminación de dependencias fuertes entre código.

1.2. Patrón MVC

Muchas y variadas son las definiciones que podemos encontrar al respecto de la arquitectura MVC. Según (Fowler, 2002), el *Model View Controller* (MVC) es uno de los patrones más citados y utilizados alrededor del mundo. Comenzó como un marco desarrollado por Trygve Reenskaug para la plataforma *Smalltalk* a finales de los años setenta. Desde entonces, ha desempeñado un papel influyente en la mayoría de los marcos de la interfaz de usuario y en el pensamiento sobre el diseño de la interfaz de usuario.

El MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

BIBLIOGRAFÍA RECOMENDADA

Libro recomendado, *Patterns of Enterprise Application Architecture* (Fowler, 2002)



En el MVC, tenemos dos separaciones principales: separar la presentación del modelo y separar el controlador de la vista. De estos la separación de la presentación y el modelo es uno de los principios de diseño más importantes en el software, y la única vez que no debe seguirlo es en sistemas muy simples, donde el modelo no tiene ningún comportamiento real de todos modos. Tan pronto como obtenga alguna lógica no visual debe aplicar la separación. Por desgracia, muchos de los marcos de la interfaz de usuario hacen que sea difícil, y los que no lo son a menudo se enseña sin una separación.

La separación de la vista y el controlador es menos importante, por lo que sólo recomendaría hacerlo cuando sea realmente útil.

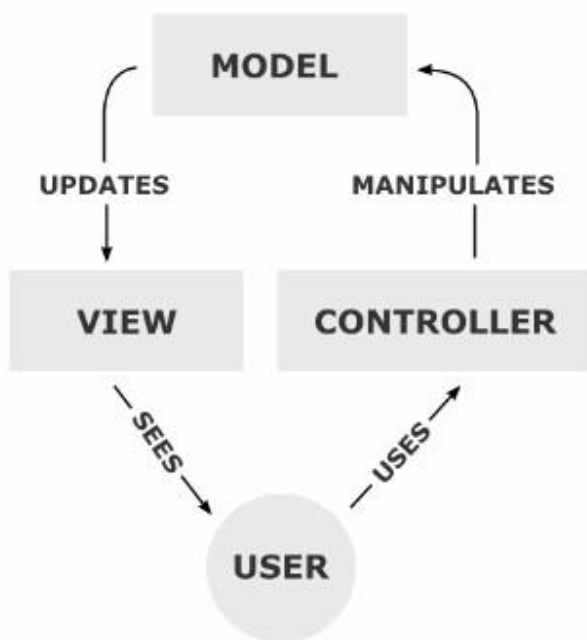


Imagen 1: Patrón MVC. Fuente de la imagen: propia.

1.3. Modelo

Es la representación de la información con la cual el sistema opera, por lo tanto, gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).

En esta parte nos encontraremos principalmente todo el desarrollo que interactúe con la base de datos.

1.4. Vista

Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por lo tanto requiere de dicho 'modelo' la información que debe representar como salida.

En esta parte nos encontraremos con toda la parte que tenga que ver con visualización y “pintar” la información.

1.5. Controlador

Es quizá la parte más compleja tanto de entender como de programar, ya que es la parte que más complicada se hace abstraer. En un principio, podríamos pensar que únicamente realiza las labores de enrutador, es decir, recibe las peticiones de los usuarios y redirige hacia donde corresponda:



Imagen 2: Controlador como enrutador. Fuente de la imagen: propia.

Y aunque es una de las funciones del controlador, no es la única función. Siguiendo con el anterior esquema, lo que podríamos ampliar como funciones del controlador sería:

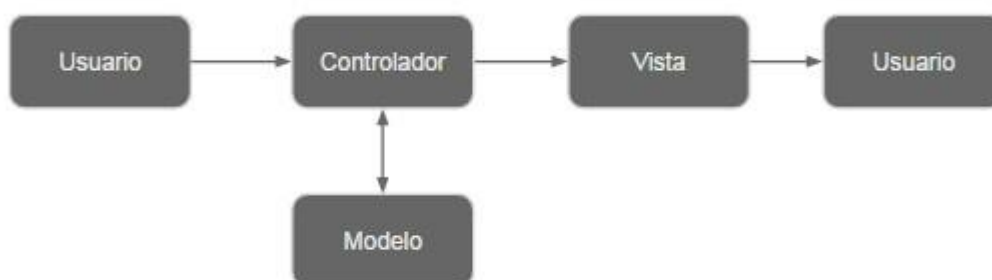


Imagen 3: Controlador. Fuente de la imagen: propia.

Esta ya es una aproximación más certera de la funcionalidad de un controlador:

- El controlador es quien recibe la petición del usuario.
- Será quien sepa qué información se necesita y por lo tanto de la necesidad o no de un Modelo.
- Por último, enrutará la información a través de una vista que le llegará al usuario.

1.6. Patrón MVC. Ejemplo de Modelo con PHP

Como siempre, un ejemplo vale mucho más que mil explicaciones teóricas, y para ello a partir del primer código realizado, modificaremos los tres ficheros y crearemos una clase que será el núcleo del cambio ya que nos permitirá esa distinción entre la Vista y el Modelo.



VIDEO DE INTERÉS

En el vídeo encontrarás un ejemplo de creación de modelo siguiendo el patrón MVC.



<https://youtu.be/CNws4qBji1M>



ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:



<https://goo.gl/AMEDf5>

Poco a poco iremos mejorando la estructura de ficheros, pero ya en un primer momento lo que observamos es que:

- Abstraemos todas las operaciones que realizábamos en múltiples ficheros contra nuestras bases de datos en un único fichero.
- Es mucho más sencillo la reutilización ya que únicamente deberemos incluir nuestra nueva clase.
- Es mucho más mantenible, ya que realizar aplicaciones se realizarán en un único fichero.

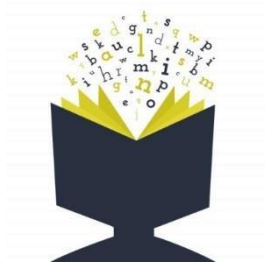
COMPRUEBA LO QUE SABES




Una vez estudiado qué es una arquitectura MVC, ¿serías capaz de encontrar 3 frameworks escritos con PHP que utilicen arquitectura MVC?

Coméntalo en el foro.

ARTÍCULO DE INTERÉS



Interesantísimo artículo sobre diferentes frameworks PHP.

 <https://www.genbetadev.com/frameworks/y-6-frameworks-php-mas-que-te-haran-la-vida-mas-simple>

COMPRUEBA LO QUE SABES



Una vez estudiado qué es una arquitectura MVC ¿serías capaz utilizar el servidor PostgreSQL (<https://www.postgresql.org/>)?

Coméntalo en el foro.

2. Vista y Formularios

Hemos aprendido a trabajar con el patrón Modelo, Vista y Controlador, y en concreto hemos comenzado con la parte de Modelo para poder encapsular y después generalizar nuestro objeto para posteriores páginas.

También aprendimos en una UD anterior, Formularios y objetos, cómo trabajar con formularios para poder interactuar con el usuario y así recoger información de forma dinámica.

Ahora pretendemos unir ambos aspectos, el de patrón MVC y el de Formularios y objetos para poder avanzar en el desarrollo de nuestras páginas. En este caso estamos hablando de la **VISTA**.



VIDEO DE INTERÉS

En el vídeo encontrarás el paso a paso de la creación de la vista y la incorporación de los formularios.



<https://youtu.be/dcbX44yzSiE>



ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:



<https://goo.gl/MzaAzi>



Nombre	Conferencia
76ers	East
Bobcats	East
Bucks	East
Bulls	East
Cavaliers	East
Celtics	East
Clippers	West
Grizzlies	West
Hawks	East
Heat	East
Hornets	West
Jazz	West
Kings	West
Knicks	East
Lakers	West
Magic	East
Mavericks	West
Nets	East
Nuggets	West
Pacers	East
Pistons	East
Raptors	East
Rockets	West
Spurs	West
Suns	West

Imagen 5: Resultado de ejecución. Fuente de la imagen: propia.

2.1. Primer paso. Estructura de ficheros modelo, vista de resultados

Usando como referencia el vídeo y el código propuesto, vemos que en primer lugar generaremos en la parte de modelo para interactuar con la base de datos.

Modelo

```
<?php
/**
 * Permitir la conexión contra la base de datos
 */
class dbNBA
{
    //Atributos necesarios para la conexión
    private $host="localhost";
    private $user="root";
    private $pass="";
    private $db_name="nba";

    //Conector
    private $conexion;

    //Propiedades para controlar errores
    private $error=false;

    function __construct()
    {
        $this->conexion = new mysqli($this->host, $this->user, $this->pass, $this->db_name);
        if ($this->conexion->connect_errno) {
            $this->error=true;
        }
    }

    //Función para saber si hay error en la conexión
    function hayError() {
        return $this->error;
    }

    //Función para devolver los equipos y su conferencia
    public function devolverEquiposConf($conferencia) {
        if($this->error==false) {
            $resultado = $this->conexion->query("SELECT nombre, conferencia
FROM equipos WHERE conferencia='".$conferencia."'");
            //echo "SELECT nombre, conferencia FROM equipos WHERE
conferencia='".$conferencia."'";
            return $resultado;
        } else {
            return null;
        }
    }
}
```

Como podemos observar:

1. Tenemos todo lo necesario para la administración de la conexión contra la base de datos:
 - a. Los parámetros
 - b. El control de errores
2. En segundo lugar, tenemos todo el código necesario para poder interactuar con la base de datos como es la parte de consultas

Una vez tenemos este código podemos realizar la parte de VISTA, y en primer lugar implementaremos la visualización de los datos sacados por el listado contra la base de datos:

Vista

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Listado de equipos por conferencia</title>
  </head>
  <body>
    <!-- Esqueleto de info-->
    <?php
      include "dbNBA.php";
      //Crear el objeto de conexion
      $nba=new dbNBA();
      //Comprobar que llega la variable conferencia
      if(isset($_POST["conferencia"])){
        //Recuperar los equipos y sus conferencias
        $resultado=$nba->devolverEquiposConf($_POST["conferencia"]);
        ?>
        <table>
          <tr>
            <th>Nombre</th>
            <th>Conferencia</th>
          </tr>
          <?php
            while ($fila = $resultado->fetch_assoc()) {
              echo "<tr>";
              echo "<td>".$fila["nombre"]."</td>";
              echo "<td>".$fila["conferencia"]."</td>";
              echo "</tr>";
            }
          ?>
        </table>
        <?php }?>
        <!-- Esqueleto de info-->
      </body>
    </html>
```


EJEMPLO PRÁCTICO

Una vez que hemos visto el inicio de la creación de la estructura mvc, vamos a crear un ejemplo sobre una base de datos de alumnos:

- 1) Para nuestro ejemplo necesitaremos una nueva base de datos:
 - a. Con la ayuda de Phpmyadmin crearemos una base de datos denominada escuela.
 - b. Crearemos una nueva tabla denominada alumnos con los siguientes campos:
 - i. Nombre, varchar 64
 - ii. Apellidos, varchar 128
 - iii. Edad, int
- 2) Crearemos una nueva carpeta para nuestro proyecto, y dentro de este:
 - a. listado.php
 - b. lib/escuela.php
- 3) Utilizando como modelo el ejemplo anterior llamado dbNBA, generaremos una nueva clase denominada Escuela.



2.2. Segundo paso. Formularios

¿Cómo incorporar interacción con el usuario? Los formularios son la clave en esto. Veámoslo con el vídeo explicativo:



VIDEO DE INTERÉS

En el vídeo encontrarás la incorporación del formulario a nuestro patrón mvc.



<https://youtu.be/zzEWhHE0vXk>



ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:



<https://goo.gl/B1pnQ8>

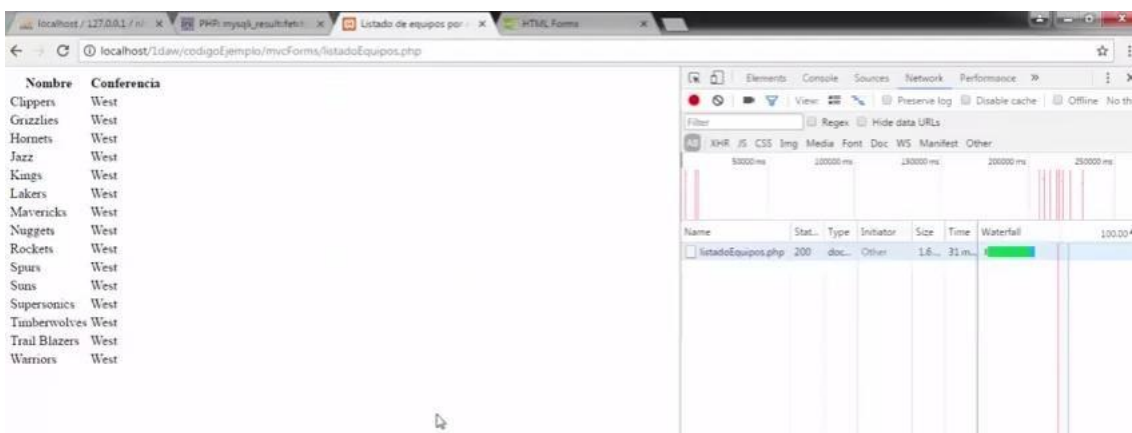


Imagen 6: Resultado de ejecución. Fuente de la imagen: propia.

Tal y como se explica, hemos incorporado un formulario que recoge nuestras peticiones:

Formulario

```
<!DOCTYPE html>

<html>
  <head>
    <meta charset="utf-8">
    <title>Pagina principal</title>
  </head>
  <body>
    <h2>MOSTRAR LISTADO DE TODOS LOS EQUIPOS</h2>
    <form action="listadoEquipos.php" method="POST">
      Conferencia:<br>
      <input type="text" name="conferencia"><br>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

COMPRUEBA LO QUE SABES



Una vez estudiado como incluir una vista y formulario a partir de una tabla de una base de datos, ¿puedes crear un ejemplo propio que presente los datos de la forma vista?

Coméntalo en el foro.

Esto permitirá que nos llegue el parámetro elegido a nuestro listadoEquipos.php, lo que nos permitirá seleccionar la conferencia, y por lo tanto deberíamos cambiar el método tal y como el vídeo nos indica.

3.CRUD

El término CRUD proviene del acrónimo, CREATE, READ, UPDATE y DELETE, es decir, todas aquellas operaciones básicas y esenciales que podemos realizar contra una base de datos:

- Crear un nuevo registro con INSERT.
- Leer registros con SELECT.
- Actualizar un registro almacenado con UPDATE.
- Borrar un registro con DELETE.

Después de haber aprendido a realizar consultas con una base de datos, y comprender y crear un patrón MVC, es muy sencillo comprender y utilizar el resto de acciones. Pero lo mejor, igual que en otras ocasiones, será realizar ejemplos que nos ayuden a comprender cada uno de las acciones.

3.1. Primer paso. Modificación del modelo para la incorporación de la inserción

La inserción de nuevos registros en una base de datos, después de la lectura de información, es la acción casi más importante para comenzar a interactuar con nuestras tablas. No resulta complejo ni complicado, pero será importante puesto que el proceso es un poco más largo realizarlo poco a poco y comprobando cada uno de los pasos:

VIDEO DE INTERÉS



En el vídeo encontrarás la explicación de la modificación del modelo para la incorporación del INSERT.



<https://youtu.be/dMqKVxvLz1I>



ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:

 <https://goo.gl/wk3sg3>



Imagen 7: Resultado de ejecución del INSERT. Fuente de la imagen: propia.

En concreto, el código que incluimos nuevo en nuestro modelo es:

INSERT

```
//function insercion contra la tabla usuarios
public function insertarUsuario($nombre,$apellidos,$edad){
    if($this->error==false)
    {
        $insert_sql="INSERT INTO usuario (id, nombre, apellidos, edad)
VALUES (NULL, '$nombre', '$apellidos', '$edad')";
        if (!$this->conexion->query($insert_sql)) {
            echo "Falló la insercion de la tabla: (" . $this->conexion->errno . ") " . $this->conexion->error;
            return false;
        }
        return true;
    }else{
        return false;
    }
}
```

3.2. Segundo paso. Incluimos los formularios necesarios para enviar la información

El siguiente paso una vez que tenemos creada una función que nos permite insertar información en la base de datos consiste en generalizar la función y hacerla más flexible. Para ello en primer lugar añadiremos los parámetros necesarios en la función para recibir los datos y en segundo lugar incluir un formulario que nos permita enviar la información (OJO, aún no realizaremos comprobación de la información):



VIDEO DE INTERÉS

En el vídeo encontrarás la explicación de la inclusión del formulario.



<https://youtu.be/NMonB4hVPs4>



ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:



<https://goo.gl/rTm2U1>

EJEMPLO PRÁCTICO

Seguiremos con el ejemplo creado sobre la base de datos de alumnos:



- 1) Usaremos la base de datos alumnos
- 2) En escuela.php incluiremos una nueva función que realice la inserción de un nuevo alumno en la tabla alumnos.
- 3) Crearemos un nuevo documento denominado alumno.php. En este fichero crearemos un formulario con 3 campos: nombre, apellidos y edad. Este formulario enviará la información a alumnoNuevo.php que recibirá la información y utilizará la nueva función para la inserción.

3.3. Tercer paso. Update y delete

La actualización de registros va a ser muy, muy parecida a la inserción. Aquí la gran dificultad es actualizar el registro correcto, ya que el resto de comandos y código es exactamente igual a los vistos:



VIDEO DE INTERÉS

En el vídeo encontrarás la explicación del update.



<https://youtu.be/XWLqGN5Y3II>



Imagen 8: Resultado de ejecución. Fuente de la imagen: propia.

En concreto, el código que incluimos nuevo en nuestro modelo es:

UPDATE

```
//function actualizar contra la tabla usuarios
public function actualizarUsuario($id,$nombre,$apellidos,$edad){
    if($this->error==false)
    {
        $insert_sql="UPDATE usuario SET nombre='".$nombre."',
apellidos='".$apellidos."', edad='".$edad.'" WHERE id='".$id;
        if (!$this->conexion->query($insert_sql)) {
            echo "Falló la actualizacion de la tabla: (" . $this->conexion->errno . ") " . $this->conexion->error;
            return false;
        }
        return true;
    }else{
        return false;
    }
}
```

Por último, el borrado de un determinado registro será muy parecido a la actualización, salvo que no es necesario todos los datos para realizar la consulta:

En concreto, el código que incluimos nuevo en nuestro modelo es:

DELETE

```
//function borrar contra la tabla usuarios
public function borrarUsuario($id){
    if($this->error==false)
    {
        $insert_sql="DELETE FROM usuario WHERE id=".$id;
        if (!$this->conexion->query($insert_sql)) {
            echo "Falló el borrado del usuario: (" . $this->conexion->errno . ") " . $this->conexion->error;
            return false;
        }
        return true;
    }else{
        return false;
    }
}
```

COMPRUEBA LO QUE SABES



Una vez estudiado el CRUD a partir de la información, ¿puedes crear un ejemplo propio que contenga un CRUD siguiendo el modelo estudiado?

Coméntalo en el foro.

4. Funciones, POO y MVC

4.1. Parámetros por defecto

Han sido ya muchas las entradas donde hemos hablado de funciones, su uso dentro de las páginas, su uso dentro de las clases, sus variantes, el ámbito y visibilidad, ... Cuando comenzamos a trabajar la conexión a base de datos embebida con la Programación orientada a objetos y la arquitectura MVC, se nos plantean situaciones donde necesitaríamos tener mayor flexibilidad en el uso de las funciones. Este es el punto en el que nos puede ayudar muchas reglas de php en este sentido.

RECUERDA...

Un método dentro de una clase tiene... (UD5)



- La palabra reservada *function*.
- El nombre de la función definida por el usuario (en este caso se denomina *foo*).
- Los argumentos de entrada de parámetros, los cuales pueden ser tantos como se necesiten o se requieran para el buen funcionamiento de la función.
- El cuerpo de la función.
- La devolución del resultado mediante la palabra reservada *return*.

Los parámetros por defecto en una función nos permiten definir funciones donde una, varias, o todos los parámetros de entrada puedan coger un parámetro por defecto cuando este no se defina. Vamos a verlo con el siguiente código y ejemplo:

Parámetros por defecto

```
<?php
function devolverUsuarioId($id,$ultimo=false){
    if($this->error==false){
        if($ultimo==true) $resultado = $this->conexion->query("SELECT
* FROM usuario ORDER BY id DESC LIMIT 1");
        else $resultado = $this->conexion->query("SELECT * FROM
usuario WHERE id=".$id);
        return $resultado;
    }else{
        return null;
    }
}

//USOS
$resultado=$usuarios->devolverUsuarioId(4);
$resultado=$usuarios->devolverUsuarioId(null,true);
?>
```



VIDEO DE INTERÉS

En el vídeo encontrarás la explicación sobre los parámetros por defecto.



<https://youtu.be/JdJiIlubXBA>

EJEMPLO PRÁCTICO



Seguiremos con el ejemplo creado sobre la base de datos de alumnos:

- 1) Usaremos la base de datos alumnos.
- 2) En escuela.php modificaremos nuestra función de inserción de datos, haciendo que el parámetro de entrada, edad, no sea obligatorio y que sea un parámetro con una edad 18 por defecto.

4.2. MVC y funciones

Hasta la fecha, nuestras funciones dentro de la clase devolvían objetos del tipo mysqli, algo un poco incoherente respecto a la arquitectura MVC, donde queremos que la parte de Modelo se quede en las clases y la parte de vista en el frontal. Para cambiar esta dinámica simplemente cambiaremos el resultado de nuestras funciones dentro de la definición de las clases.



VIDEO DE INTERÉS

En el vídeo encontrarás la explicación de la inclusión del formulario.



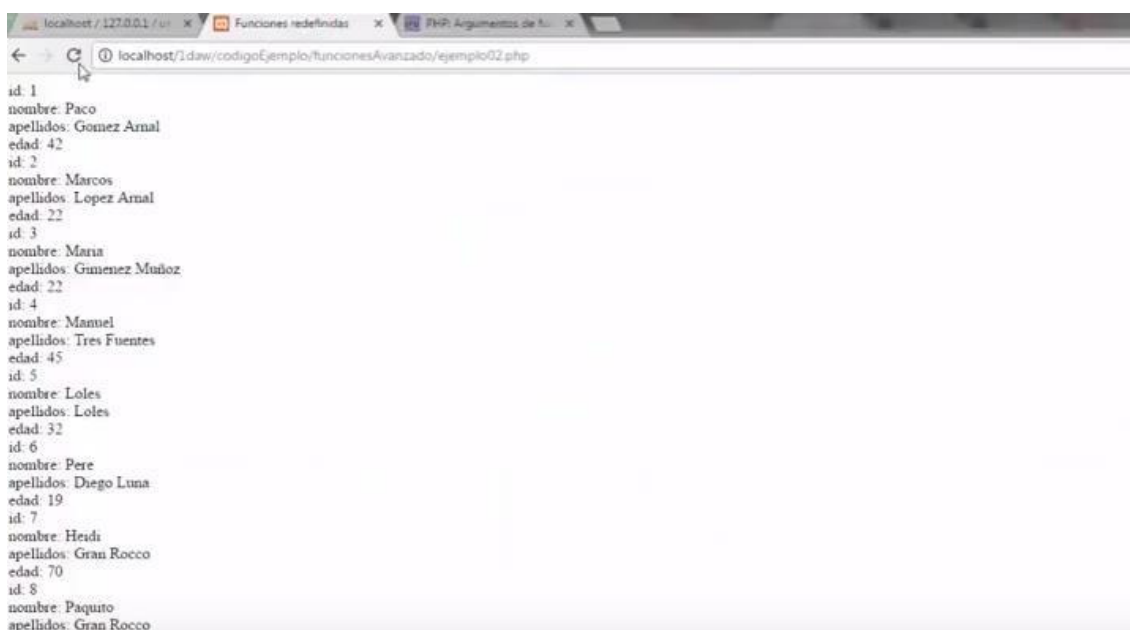
<https://youtu.be/4oBjcNmdM9w>



ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:

 <https://goo.gl/6LEzgG>



```
id: 1
nombre: Paco
apellidos: Gomez Arnal
edad: 42
id: 2
nombre: Marcos
apellidos: Lopez Arnal
edad: 22
id: 3
nombre: Maria
apellidos: Gimenez Muñoz
edad: 22
id: 4
nombre: Manuel
apellidos: Tres Fuentes
edad: 45
id: 5
nombre: Loles
apellidos: Loles
edad: 32
id: 6
nombre: Pere
apellidos: Diego Luna
edad: 19
id: 7
nombre: Heidi
apellidos: Gran Rocco
edad: 70
id: 8
nombre: Paquito
apellidos: Gran Rocco
```

Imagen 9: Resultado de ejecución. Fuente de la imagen: propia.

Cuando en la vista representábamos los datos de nuestra base de datos, utilizábamos el siguiente código:

Vista

```

<!DOCTYPE html>

<html>
  <head>
    <meta charset="utf-8">
    <title>Listado de equipos por conferencia</title>
  </head>
  <body>
    <!-- Esqueleto de info-->
    <?php
      include "dbNBA.php";
      //Crear el objeto de conexion
      $nba=new dbNBA();
      //Comprobar que llega la variable conferencia
      if(isset($_POST["conferencia"])){
        //Recuperar los equipos y sus conferencias
        $resultado=$nba->devolverEquiposConf($_POST["conferencia"]);
        ?>
        <table>
          <tr>
            <th>Nombre</th>
            <th>Conferencia</th>
          </tr>
          <?php
            while ($fila = $resultado->fetch_assoc()) {
              echo "<tr>";
              echo "<td>".$fila["nombre"]."</td>";
              echo "<td>".$fila["conferencia"]."</td>";
              echo "</tr>";
            }
          ?>
        </table>
        <?php }?>
      <!-- Esqueleto de info-->
    </body>
  </html>

```

Como vemos, utilizamos la función `fetch_assoc` de la librería `mysqli`. Esto provoca un "acople" y una gran dependencia de la base de datos que estamos utilizando.

Justamente lo que estamos intentando es producir un código en la vista lo más independiente del modelo, de la base de datos posible. Por este motivo es muy interesante “desacoplar”, y no hacerlo dependiente mediante la devolución de un array:

Devolviendo un array

```
<?php
//Devolver arrays
function devolverUsuarios() {
    $tabla=[];
    if($this->error==false) {
        $resultado = $this->conexion->query("SELECT * FROM usuario");
        while($fila=$resultado->fetch_assoc()) {
            $tabla[]=$fila;
        }
        return $tabla;
    }else{
        return null;
    }
}
?>
```

5. Herencia y MVC

Una de las propiedades más importantes y características de la programación orientada a objetos es la Herencia.

En POO, mediante la herencia, los diseñadores pueden crear nuevas clases partiendo de una clase o de una jerarquía de clases preexistente evitando con ello el rediseño, la modificación y verificación de la parte ya implementada. La herencia facilita la creación de objetos a partir de otros ya existentes e implica que una subclase obtiene todo el comportamiento (métodos) y eventualmente los atributos (variables) de su superclase.

PARA SABER MÁS...

Herencia



El fantástico libro gratuito sobre Java, **Thinking in Java**, 3rd ed. Revision 4.0, (Eckel, 2002), aunque verse sobre el lenguaje Java, nos proporciona una visión muy bien documentada de lo que es y significa la programación orientada a objetos.

Una de las propiedades más interesantes de la POO, la Herencia, la cual permite la reutilización del código. Por sí mismo, la idea de un objeto es una herramienta conveniente. Nos permite empaquetar los datos y la funcionalidad por concepto. Estos conceptos se expresan como unidades fundamentales en el lenguaje de programación utilizando la palabra clave class (en el caso de PHP también).

La herencia, como muy bien introducíamos, permite la reutilización de esa unidad fundamental, sus datos y sus funcionalidades, creando una estructura modular y arborescente de padres e hijos.

Utilizando una imagen que puede clarificar este concepto:

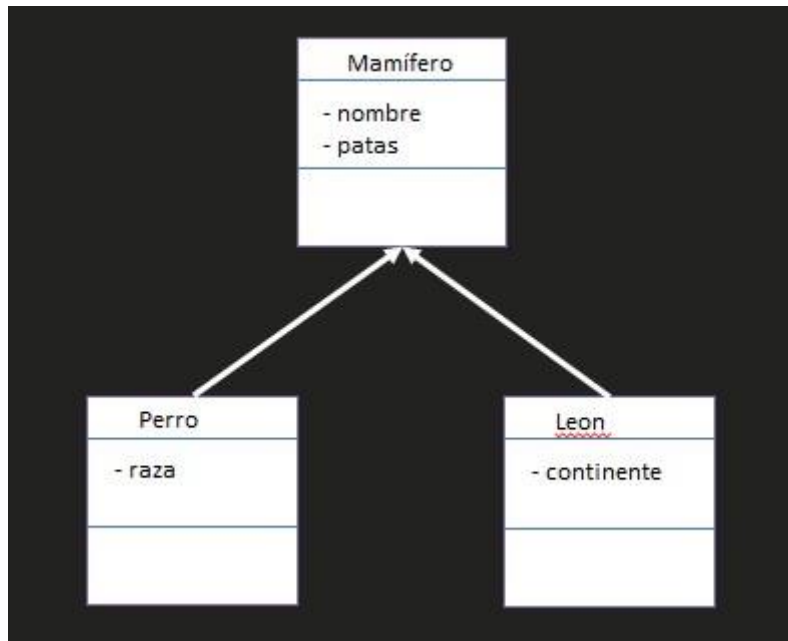


Imagen 4: Herencia. Fuente de la imagen: propia.

Son conceptos que en 1º y con otros lenguajes de programación como Java, están muy marcados. Pero no olvidemos que PHP pivotó hacia un modelo y por lo tanto orientado a objetos.



VIDEO DE INTERÉS

En el vídeo encontrarás la explicación la independización de código entre vista y modelo.



<https://youtu.be/ct7pReCvz0g>



ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:

 <https://goo.gl/8X8tW6>

La identificación la realizamos con la palabra reservada `extends`, el resto, su funcionamiento y significado será el mismo que con cualquier otro lenguaje de programación.

5.1. Accediendo a métodos padre

Cuando realizamos una estructura de clases padre-hijo, una estructura con diferentes propiedades, métodos, constantes, ..., aparecen ocasiones donde se pueden redefinir los métodos (esto se denomina sobrecarga), ahora quedémonos con que un mismo método del padre se puede redefinir en el hijo, y sería imposible poder acceder al del padre a no ser que utilicemos el operador de ámbito `::` (doble punto), seguido de la palabra reservada `parent`. Como siempre lo mejor es que lo veamos con un ejemplo:



VIDEO DE INTERÉS

En el vídeo encontrarás la explicación sobre el acceso a los métodos padre.



<https://youtu.be/370MMacFhzc>

5.2. Aplicándolo al Modelo

La herencia tiene una aplicación directa y muy fácil de entender en el modelo, ya que podemos realizar la siguiente división de las tareas:

- Una clase padre encargada de la conexión e interacción con la base de datos.
- Unas clases encargadas de la interacción con las tablas y el manejo de la información.

En esta tipología muy clásica dentro de los frameworks basados en MVC, las clases encargadas de la interacción con las tablas, normalmente tienen el mismo nombre que tienen las tablas con las que están relacionadas, mientras que la clase padre suele tener un nombre más generalista como db y esta clase es la encargada en última instancia de estar relacionada con la tecnología de base de datos elegida para nuestra aplicación.

Modificando db

```
<?php
/**
 * Permitir la conexión contra la base de datos
 */
class db
{
    //Atributos necesarios para la conexión
    private $host="localhost";
    private $user="root";
    private $pass="";
    private $db_name="usuarios";
    //Conector
    private $conexion;
    //Propiedades para controlar errores
    private $error=false;
    private $error_msj="";
    function construct()
    {
        $this->conexion = new mysqli($this->host, $this->user, $this->pass, $this->db_name);
        if ($this->conexion->connect_errno) {
            $this->error=true;
            $this->error_msj="No se ha podido realizar la conexión a la bd. Revisar base de datos o parámetros";
        }
    }
    //Función para saber si hay error en la conexión
    function hayError(){
        return $this->error;
    }
    //Función que devuelve mensaje de error
    function msjError(){
        return $this->error_msj;
    }
    //Metodo para la realización de consultas a la bd
    public function realizarConsulta($consulta){
        if($this->error==false){
            $resultado = $this->conexion->query($consulta);
            return $resultado;
        }else{
            $this->error_msj="Imposible realizar la consulta: ".$consulta;
            return null;
        }
    }
}
?>
```

Vemos en el anterior código dos nuevas modificaciones:

- No aparece código con las consultas específicas SQL, esto aparecerá en las clases hijas.
- Aparece una nueva función genérica `realizarConsulta($consulta)`, que es la encargada de realizar las consultas y comprobar si se puede realizar.

Clase hija

```
<?php
include "db.php";
/**
 *
 */
class Usuario extends db
{
    function __construct()
    {
        //De esta forma realizamos la conexion a la base de datos
        parent::__construct();
    }

    //Devolvemos todos los usuarios
    function devolverUsuarios() {
        //Construimos la consulta
        $sql="SELECT * from usuario";
        //Realizamos la consulta
        $resultado=$this->realizarConsulta($sql);
        if($resultado!=null){
            //Montamos la tabla de resultados
            $tabla=[];
            while($fila=$resultado->fetch_assoc()){
                $tabla[]=$fila;
            }
            return $tabla;
        }else{
            return null;
        }
    }
}
```

Aquí aparece nuestras consultas SQL, ya que tiene sentido que sea en la clase que va a estar relacionada con la tabla en cuestión la que tenga la información de cómo manejar dicha información.

Por otro lado, necesita de la clase padre db, para poder realizar las conexiones a través del conector y de una conexión ya abierta.

COMPRUEBA LO QUE SABES



Una vez estudiada la Herencia, ¿puedes modificar tu ejemplo anterior para que contenga al menos una clase padre y otra hija de acuerdo a los ejemplos presentados?

Coméntalo en el foro.

EJEMPLO PRÁCTICO



Seguiremos con el ejemplo creado sobre la base de datos de alumnos:

- 1) Usaremos la base de datos alumnos.
- 2) Crearemos una nueva clase denominada Alumnos.php dentro del directorio lib/alumnos.php.
- 3) De acuerdo a lo aprendido con herencia y PHP, modificaremos nuestros ficheros para que:
 - a. Escuela.php solo tenga los métodos de conexión.
 - b. Alumnos.php contenga los métodos CRUD.

Resumen final:

En esta importantísima unidad hemos descubierto y aprendido el funcionamiento principal de los *frameworks* de desarrollo modernos. Actualmente el desarrollo ha evolucionado, tanto en tecnología como metodología.

La POO y la metodología MVC, proporcionan esos mecanismos modernos de trabajo en equipo para el desarrollo de aplicaciones web, pero sobre todo mejoran unos aspectos importantísimo en el desarrollo software:

- Mejora la relación con el cliente, ya que se puede recibir un feedback continuo
- Mejora el diseño y desarrollo ya que permite modularizar las tareas
- Mejora el mantenimiento y mejoras del desarrollo ya que no es necesario un cambio de todo el desarrollo sino de determinadas partes

La metodología MVC, Modelo-Vista-Controlador, puede ser implementada de múltiples formas, la vista en esta unidad es una de ellas, y es el inicio para entender otras opciones desarrolladas o implementar el propio desarrollo a partir de las necesidades del producto.

UNIDAD DIDÁCTICA 9:

Sesiones, seguridad y autenticación

Módulo profesional:
Desarrollo Web en Entorno Servidor

Contenido

RESUMEN INTRODUCTORIO	3
INTRODUCCIÓN	3
CASO INTRODUCTORIO	3
1. SESIONES	4
1.1. Superglobal.....	7
1.2. Sesiones.....	8
1.3. Destruyendo sesiones.....	10
1.4. PASO 1: Objeto Seguridad	12
1.5. PASO 2: El formulario de registro.....	13
1.6. PASO 3: La codificación de la contraseña	14
1.7. PASO 4: El formulario de acceso	16
1.8. PASO 5: Activación de la sesión	17
1.9. PASO 6: Proteger una página	18
1.10. PASO 7: LogOut.....	19
2. COOKIES.....	20
2.1. Creando y recuperando cookies	22
2.2. Aplicación de las cookies al login.....	24
Resumen final:	26

RESUMEN INTRODUCTORIO

En esta unidad aprenderemos los conceptos necesarios para poder manejar sesiones y cookies, así como las diferencias entre ambos conceptos.

Una vez comprendidos los conceptos básicos podremos aplicarlos en la seguridad y control de acceso básico para nuestras aplicaciones.

INTRODUCCIÓN

Cualquier aplicación moderna tiene una zona de acceso restringido así como un registro e incluso una diferenciación de uso por roles de usuario. No es nada complejo implementarlo y con PHP es muy sencillo con la combinación y uso de sesiones y cookies.

Las sesiones nos permiten controlar desde la parte de servidor información referente para una petición de cliente. Las cookies nos permiten controlar información desde la parte de navegador o de cliente. La combinación de ambas propiedades nos permite realizar un control de acceso y seguridad completo para una determinada aplicación.

El uso de sesiones y cookies no será complejo ya que pertenecen al entorno de superglobals, es decir variables que no hace falta crear ni referencias y que son arrays asociativos que nos permiten almacenar y gestionar información, de igual forma a como realizábamos con GET o POST.

CASO INTRODUCTORIO

Queremos realizar introducir un control de acceso y una gestión de perfil a cualquier aplicación realizada.

1. SESIONES

Las sesiones, al igual que las cookies que veremos en el siguiente apartado, se encuentran dentro de lo que PHP denomina superglobals. En este epígrafe las veremos y estudiaremos su significado.

Es importante tener en cuenta la diferencia entre una sesión y las variables que intervienen, y una cookie y su uso.

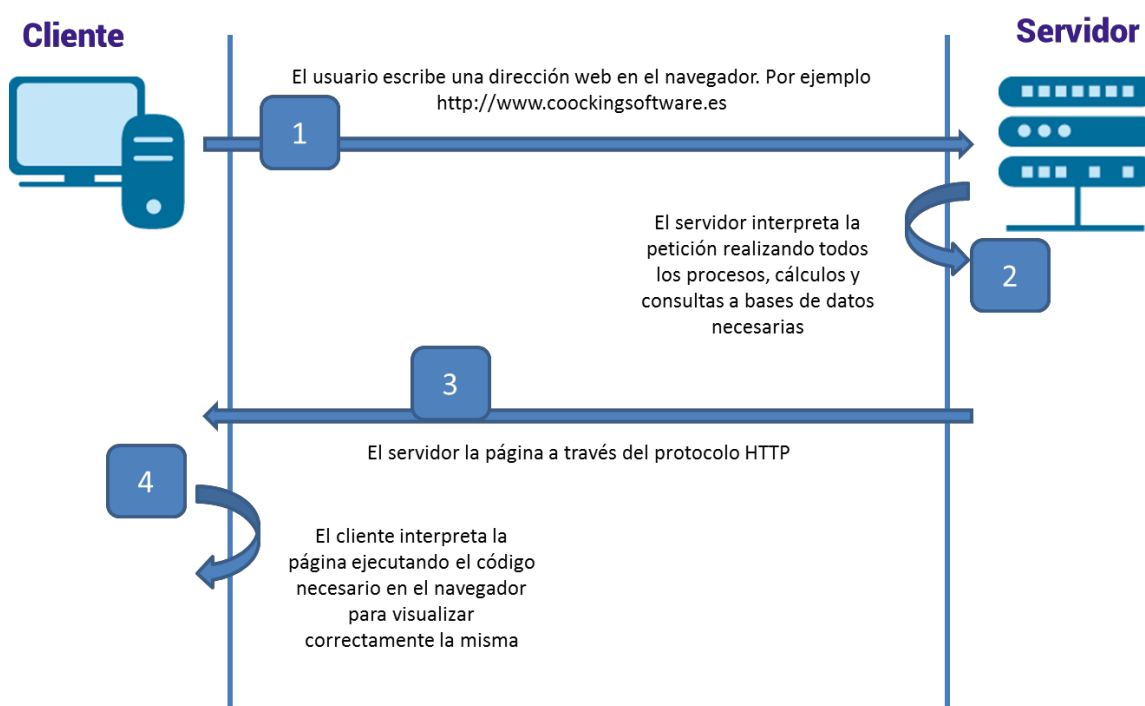


Imagen 1: Secuencia de petición HTTP. Fuente de la imagen: propia.

Recordamos y retomamos el esquema que en las primeras Unidades utilizábamos para describir una secuencia de petición por parte de un cliente de una página HTTP.

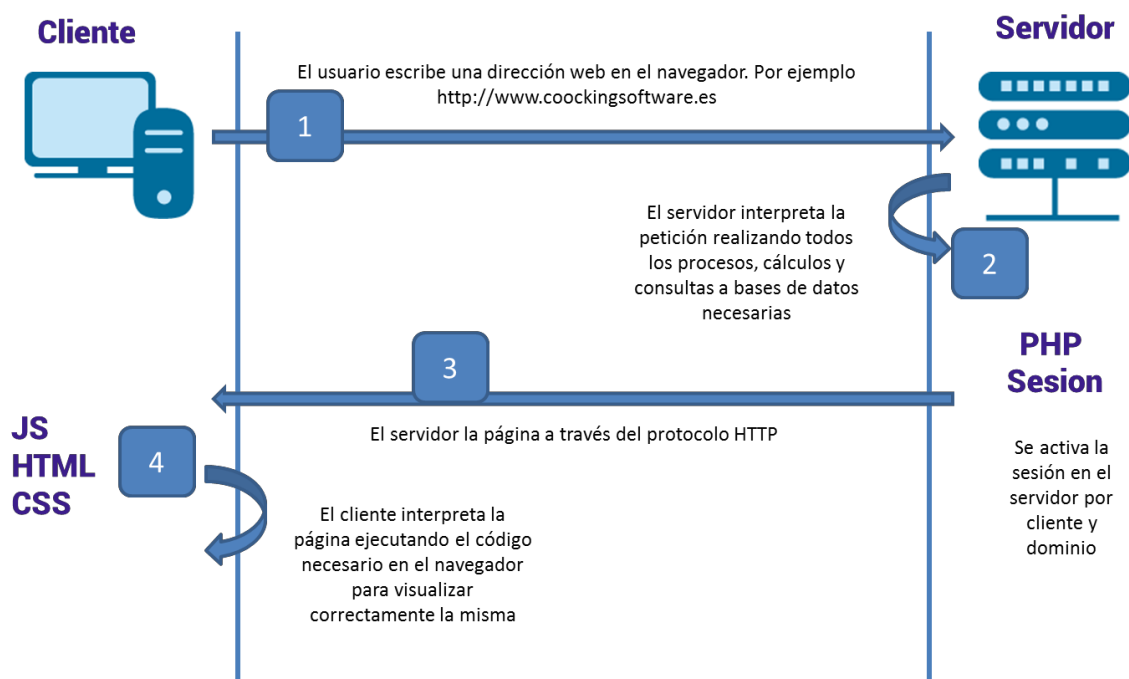


Imagen 2: Activación de la sesión. Fuente de la imagen: propia.

Como vemos en el anterior ejemplo, una vez que el cliente realiza la petición y esta alcanza el servidor, se activa una nueva sesión EN EL SERVIDOR, por cliente y por dominio. Es importante, y por eso lo remarcamos, que la sesión se controla por y en el SERVIDOR. Esta sesión estará por lo tanto activa por dos motivos:

- Siempre que el cliente mantenga el navegador abierto, independientemente de que todas las pestañas o conexiones estén cerradas contra ese servidor. Es decir, imaginemos que yo me tengo Chrome abierto, me conecto a la página web www.php.net, abro otras pestañas a otras URL, y cierro la pestaña de www.php.net. La sesión seguiría estando activa y abierta.
- La sesión se cerrará en el anterior caso cuando el navegador se cierre.
- El servidor cierra la sesión, bien porque haya pasado un tiempo programado, bien porque el cliente envíe esa acción (el típico *logout*), bien porque el servidor decida por funcionamiento de la aplicación cerrar la sesión.

El caso de las cookies es diferente:

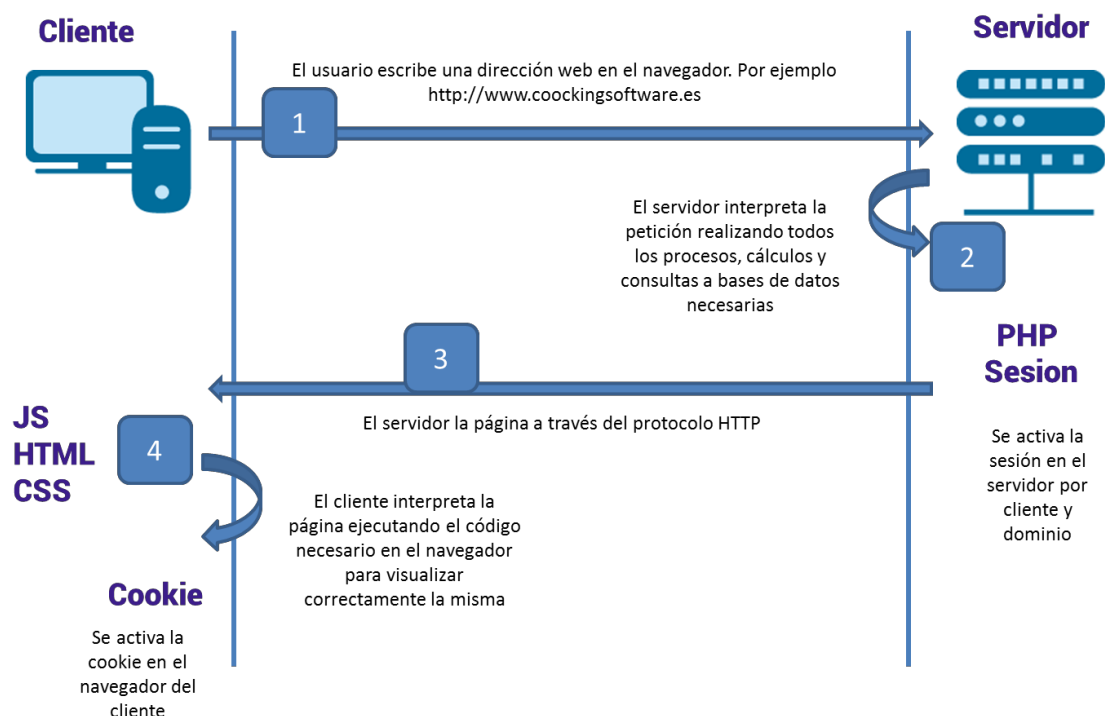


Imagen 3: Cookies vs sesiones. Fuente de la imagen: propia.

Como observamos, las cookies se crean y se mantienen en el cliente, es más, la cookie se crea en un navegador en concreto, esto es, si nosotros tenemos abiertos dos navegadores diferentes, Chrome y Firefox, por ejemplo, se pueden perfectamente tener dos cookies diferentes.

Así pues:

- Las sesiones están relacionadas con el servidor, las cookies con el cliente.
- Las sesiones se cierran al cerrar el navegador, las cookies no se destruyen al cerrar el navegador.
- Las sesiones son temporales (mientras dura la sesión o bien el navegador abierto), las cookies son permanentes hasta que son destruidas.

BIBLIOGRAFÍA RECOMENDADA



Libro recomendado: *Hacking with PHP* (Hudson, 2015).

En el capítulo 10 encontramos la referencia a ejemplos y documentación sobre el manejo de sesiones y cookies.

<http://www.hackingwithphp.com/10/0/0/cookies-and-sessions>

1.1. Superglobal

Algunas variables predefinidas en PHP son "superglobales", lo que significa que están disponibles en todos los ámbitos a lo largo del script. No es necesario emplear global \$variable; para acceder a ellas dentro de las funciones o métodos.

Según la documentación de php.net tenemos las siguientes superglobals:

SUPERGLOBAL

```
$GLOBALS  
$_SERVER  
$_GET  
$_POST  
$_FILES  
$_COOKIE  
$_SESSION  
$_REQUEST  
$_ENV
```

Vemos en este listado algunas de esas variables, como \$_POST y \$_GET, y como veremos el uso y acceso de las mismas será idéntico.

1.2. Sesiones

Las sesiones son una forma sencilla de almacenar datos para usuarios de manera individual usando un ID de sesión único. Esto se puede usar para hacer persistente la información de estado entre peticiones de páginas. Los ID de sesiones normalmente son enviados al navegador mediante cookies de sesión, y el ID se usa para recuperar los datos de sesión existente. La ausencia de un ID o una cookie de sesión permite saber a PHP para crear una nueva sesión y generar un nuevo ID de sesión.

Veamos cómo podemos usar la superglobal `$_SESSION` para el almacenamiento de información:



VIDEO DE INTERÉS

En el vídeo encontrarás una explicación de creación y uso de `$_SESSION`



<https://youtu.be/eW23BMX3eMM>



ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:



<https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/sesiones>

Como podemos ver, en el código y en el vídeo:

Crear sesión

```
<?php
    session_start();
    if (!isset($_SESSION['count'])) {
        $_SESSION['count']=0;
        $_SESSION['usuario']="Paco";
        print_r($_SESSION);
    } else {
        $_SESSION['count']++;
        print_r($_SESSION);
    }
?>
```

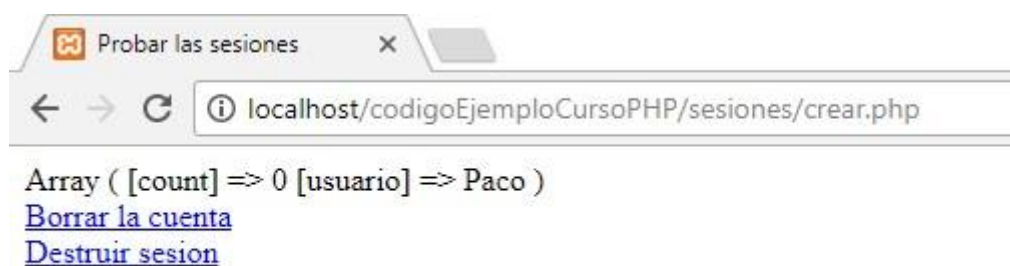


Imagen 1b: Resultado de ejecución. Fuente de la imagen: propia.

1. Para activar la sesión desde PHP y por lo tanto unir la sesión del usuario con la sesión almacenada en el servidor debemos utilizar la función `session_start()`; Si no utilizamos esta función, la sesión no se relacionará con el cliente.
2. Utilizando el array `$_SESSION`, podremos añadir, modificar y/o eliminar datos a dicha sesión.

EJEMPLO PRÁCTICO

Una vez que hemos visto el inicio de creación de las sesiones:



1. Crearemos dentro de una estructura de directorios dentro de nuestro servidor un nuevo fichero denominado numVisitas.php
2. Iniciamos la sesión con `sesión_start`
3. Añadimos una variable `numVisitas` dentro de la superglobal `$_SESSION`
4. Añadimos 1 siempre y cuando la sesión esté iniciada.

1.3. Destruyendo sesiones

Hemos antes dicho que una de las tres formas que teníamos para destruir una sesión era siempre desde el servidor, bien porque el usuario nos indique que quiere eliminar la sesión y toda la información relacionada, o bien porque la aplicación así lo requiera.

VIDEO DE INTERÉS



En el vídeo encontrarás una explicación de destrucción de la sesión.



<https://youtu.be/FHGRk31OS8A>

ENLACE DE INTERÉS



En el siguiente enlace tienes el código utilizado:



<https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/sesiones>



Imagen 2b: Resultado de ejecución. Fuente de la imagen: propia.

El primer y gran uso que le damos al uso de las sesiones es integrarlo con el control de acceso a las aplicaciones, los roles y la seguridad. Aunque encontrareis infinidad de código e información al respecto sobre el control de acceso y ejemplos con php, vamos a dar las pautas más importantes para la incorporación de las sesiones al sistema de seguridad.

1.4. PASO 1: Objeto Seguridad

El primer paso será seguir el modelo orientado a objetos que estamos utilizando a lo largo de todo el contenido y generar una nueva clase que controle y gestione la parte de sesión. Esta clase se encontraría dentro de lo que se denomina Controlador en el Patrón MVC.

Si entrar a describir por completo la clase las partes importantes que tendría serían:

El Constructor

```
function construct()  
{  
  
    //Arrancamos la sesion  
    session_start();  
    if(isset($_SESSION["usuario"])) $this->usuario=$_SESSION["usuario"];  
}
```

Como vemos, al concentrar en una sola clase el manejo de las sesiones, colocaremos en el constructor de esta clase la activación de la clase, de esta forma se realizará siempre y sólo se realizará una vez.

Usando la clase

```
$seguridad=new Seguridad();
```

Al tener la clase definida, la utilización será muy sencilla, creando un nuevo objeto allí donde se vaya a necesitar.

1.5. PASO 2: El formulario de registro

El formulario de registro puede ser tan sencillo como el que se muestra en la imagen o tan complejo como la aplicación necesite información del usuario:

Formulario de registro

Email

Contraseña

Confirmar contraseña

☒ Deseo recibir noticias, actualizaciones de software, y la última información relativa a productos y servicios.

Registrarse

Imagen 4: Formulario de registro. Fuente de la imagen:

<http://blog.openalfa.com/wp-content/uploads/sites/2/2014/11/formulario-de-registro.jpg>

Una de las particularidades que vemos en el anterior formulario y que sucede en un 100% de formularios de registro es la aparición de un doble campo de comprobación de la contraseña de registro del usuario. Esto se consigue a nivel de HTML de la siguiente forma:

Doble campo de contraseña en el formulario

```
<label for="pass0">Contraseña</label>
<input type="password" id="pass0" name="pass0"
placeholder="Contraseña..">

<label for="pass1">Repite Contraseña</label>
<input type="password" id="pass1" name="pass1"
placeholder="Contraseña..">
```

Como observamos, los campos son de tipo password, pero lo importante en este caso es que el *name* de ambos campos sea diferente para poder recogerlos mediante `$_POST` y poderlos comparar.

COMPRUEBA LO QUE SABES



Una vez visto el formulario de registro ¿serías capaz de añadir el campo usuario además del de email para ser almacenado?

Coméntalo en el foro.

1.6. PASO 3: La codificación de la contraseña

Como observamos en el anterior ejemplo la contraseña es texto plano que recibimos a través de los campos *pass0* y *pass1*. ¿Cómo se codifican dichas contraseñas en nuestra base de datos para que sea seguro?

Utilizaremos funciones ya programadas en php tipo hash como es sha o md5.

ARTÍCULO DE INTERÉS



Seguramente todos conocemos y usamos stackoverflow para la resolución de nuestros problemas incluso como propuesta de código realizado.

Aquí tenemos otro uso de stackoverflow como fuente de información (siempre que sepas filtrar y seguir informándonos al respecto)

① <https://stackoverflow.com/questions/16713810/how-secure-is-md5-and-sha1>

Ejemplo sha

```
<?php
$str = 'apple';

if (sha1($str) === 'd0be2dc421be4fcd0172e5afceea3970e2f3d940') {
    echo "Would you like a green or red apple?";
}
?>
```

Ejemplo md5

```
<?php
$str = 'apple';

if (md5($str) === '1f3870be274f6c49b3e31a0c6728957f') {
    echo "Would you like a green or red apple?";
}
?>
```

Como vemos, el resultado es muy parecido, produciendo en ambos casos hash que ahora sí almacenaremos en la base de datos. A su vez en la base de datos tendremos preparado un campo para almacenar el password que será de tipo VARCHAR si estamos trabajando con MySQL.

1.7. PASO 4: El formulario de acceso

El Formulario de acceso nos permite recoger la información de un usuario ya registrado en la base de datos y comprobar que existe en la misma con dicho usuario y dicha password. El único proceso a tener en cuenta en este paso es que en el momento de comprobar la contraseña, la misma se encuentra codificada en la base de datos, normalmente con un sistema de encriptación tipo MD5 o SHA, tal y como hemos visto en el PASO3.

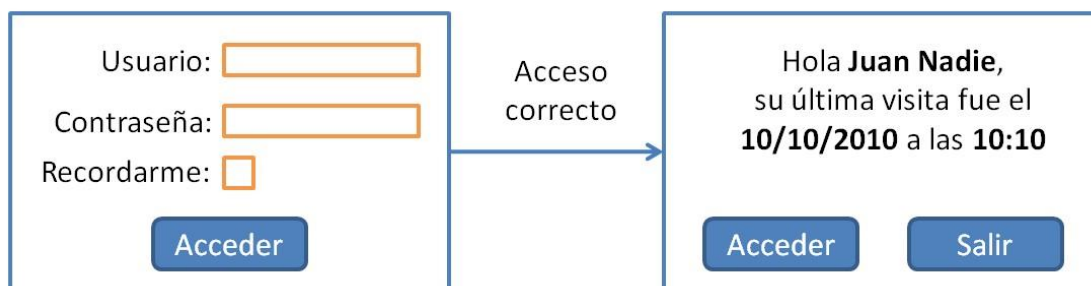


Imagen 5: Formulario de login. Fuente de la imagen:

<http://idesweb.com/img/proyecto/prac09/acceso-recordatorio.png>

Así pues, en este punto:

1. Recogeremos los datos del formulario.
2. Recogeremos los datos del usuario en la BBDD.
3. Comprobaremos la contraseña almacenada con la contraseña que hemos recogido del formulario (recordemos que esa contraseña está como texto plano y por lo tanto deberemos aplicarle la misma función de codificación que en el caso de registro, sha1 o md5).

COMPRUEBA LO QUE SABES



Una vez visto el formulario de acceso ¿serías capaz de utilizar el campo usuario o email dependiendo de lo que rellene el usuario?

Coméntalo en el foro.

1.8. PASO 5: Activación de la sesión

Una vez que el usuario se encuentra logado ya que lo hemos comprobado en el paso anterior, deberemos activar la sesión y añadir el usuario a dicha sesión:

Ejemplo md5

```
function construct()
{
    //Arrancamos la sesion
    session_start();
    if(isset($_SESSION["usuario"])) $this->usuario=$_SESSION["usuario"];
}

public function getUsuario(){
    return $this->usuario;
}

public function addUsuario($usuario,$pass,$remember=false){
    //Generando la variable de sesion
    $_SESSION["usuario"]=$usuario;
    $this->usuario=$usuario;
    //Almacenaremos el user/pass cookies
    if($remember)
    {
        setcookie("usuario",$usuario,time()+(60*60));
        setcookie("pass",$pass,time()+(60*60));
    }
}
```


Vemos en el anterior código varias funciones que intervienen en este paso:

- La primera ya la habíamos visto anteriormente ya que el constructor activa la sesión y comprueba si el \$_SESSION["usuario"] se encuentra activo.
- addUsuario permite añadir el usuario a la sesión una vez que se ha logado.
- getUsuario permite recoger el usuario.

1.9. PASO 6: Proteger una página

Una vez que tenemos implementado todo el mecanismo anterior es muy sencillo proteger cualquier página con el siguiente código:

Proteger página

```
<?php
include "../lib/Seguridad.php";
$seguridad=new Seguridad();
if($seguridad->getUsuario()==null){
    header('Location: login.php');
    exit;
}
?>
```

Como vemos, antes de cargar absolutamente nada de la página a proteger, antes de la etiqueta <HTML>, introducimos el anterior código y en el que:

- Primero incluimos la clase Seguridad y creamos el objeto (por lo que se activa la sesión).
- Comprobamos que exista el usuario.
- Si no existe redirigimos a una zona de la aplicación no protegida.

COMPRUEBA LO QUE SABES



Una vez visto el acceso y las sesiones ¿serías capaz de mostrar por pantalla el usuario que se ha registrado a tu aplicación?

Coméntalo en el foro.

1.10. PASO 7: LogOut

Realizar el logout es muy sencillo ya que únicamente debemos crear una función dentro de la clase seguridad que destruya la información dentro de la sesión:

LogOut

```
public function logout () {  
    $_SESSION=[];  
    session_destroy();  
}
```

2. COOKIES

Mediante las sesiones, podemos generar información particular/individual de un usuario que mantenemos lo que dure la sesión abierta. Es por lo tanto una información temporal que perderemos o bien cuando el usuario cierre la sesión, o bien cuando queramos borrar dicha sesión borrando todos los datos dentro de la sesión generados.

Tenemos un segundo mecanismo de interacción con el navegador, utilizar las cookies. Según la definición de php net, las cookies son un mecanismo por el que se almacenan datos en el navegador remoto para monitorizar o identificar a los usuarios que vuelvan al sitio web. En este caso, SI almacenamos información en el equipo del usuario, y por lo tanto aunque el usuario cierre el navegador dicha información permanecerá durante cierto tiempo esto tiene sus partes positivas y sus partes negativas:

- Tenemos un mecanismo sencillo para poder almacenar información y además que sea dependiente del usuario que realiza la petición.
- La información podrá ser alcanzada por cualquier otro software, y por lo tanto NO debemos almacenar información personal, sensible y que pueda ser utilizada posteriormente para el uso malicioso contra el usuario.

Recordemos el diagrama que teníamos al principio de la Unidad:

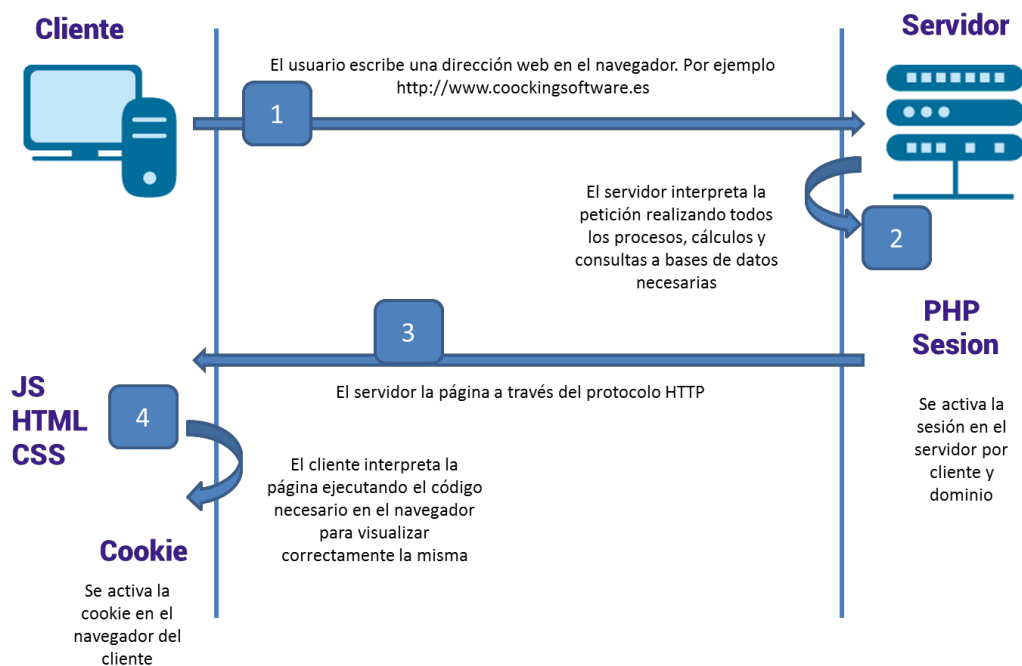


Imagen 3: Cookies vs sesiones. Fuente de la imagen: propia.



VIDEO DE INTERÉS

En el vídeo encontrarás una explicación sobre las cookies.



<https://youtu.be/IsHEBpZpSJA>

2.1. Creando y recuperando cookies

Crear una nueva cookie y recuperar esta va a ser una tarea tremendamente sencilla. Para ello seguiremos las indicaciones y el ejemplo que la misma documentación oficial de php.net no ofrece:

Manejo de cookies

```
<?php
Setting new cookie
=====

<?php
setcookie("name", "value", time()+$int);
/*name is your cookie's name
value is cookie's value
$int is time of cookie expires*/
?>

Getting Cookie
=====

<?php
echo $_COOKIE["your cookie name"];
?>

Updating Cookie
=====

<?php
setcookie("color", "red");
echo $_COOKIE["color"];
/*color is red*/
/* your codes and functions*/
setcookie("color", "blue");
echo $_COOKIE["color"];
/*new color is blue*/
?>

Deleting Cookie
=====

<?php
unset($_COOKIE["yourcookie"]);
/*Or*/
setcookie("yourcookie", "yourvalue", time()-1);
/*it expired so it's deleted*/
?>

?>
```



VIDEO DE INTERÉS

En el vídeo encontrarás una explicación sobre el uso de las cookies.



<https://youtu.be/lsAJFPyG1AQ>



ENLACE DE INTERÉS

En el siguiente enlace tienes el código utilizado:



<https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/cookies>



Imagen 3b: Resultado de ejecución. Fuente de la imagen: propia.

Como podemos observar en los ejemplos y en la documentación, una cookie permanecerá activa en nuestro navegador el tiempo que hayamos programado.

EJEMPLO PRÁCTICO



Una vez que hemos visto el inicio de creación de las cookies:

1. Crearemos dentro de una estructura de directorios dentro de nuestro servidor un nuevo fichero denominado numVisitas.php
2. Iniciamos la cookie numVisitas con el valor 0.
3. Añadimos 1 siempre y cuando la cookie exista.

2.2. Aplicación de las cookies al login

Un uso de las cookies muy típico es dentro del formulario de login de acceso a nuestra aplicación. Adelanto que la implementación realizada es MUY INSEGURA, ya que almacena el usuario y la contraseña, pero que nos sirve para comenzar a utilizar las cookies dentro de nuestras aplicaciones:



VIDEO DE INTERÉS

En el vídeo encontrarás una explicación sobre el uso de las cookies.



<https://youtu.be/TwfMR47i-98>

ENLACE DE INTERÉS



En el siguiente enlace tienes el código utilizado:



<https://github.com/pacogomezarnal/codigoEjemploCursoPHP/tree/master/seguridad>

COMPRUEBA LO QUE SABES



Una vez visto el uso de sesiones y cookies ¿serías capaz de crear un sistema de control de visitas por página y que un mismo usuario no contabilice dos veces la misma página?

Coméntalo en el foro.

Resumen final:

En esta unidad hemos aprendido y utilizado dos de los conceptos más importantes en el desarrollo de aplicaciones web: sesiones y cookies.

Mediante la combinación de ambos conceptos podemos implementar la seguridad de usuario y la particularización de los desarrollos. Con las sesiones y las cookies podemos desarrollar muchas otras acciones y utilidades a partir de los conocimientos aprendidos.

Las sesiones se inician, gestionan y finalizan desde el servidor, por lo que nos proporciona un mecanismo para poder realizar la autenticación y gestión de la seguridad. Las cookies se gestionan desde el cliente, por lo que son más inseguras, pero por otro lado proporcionan la permanencia de datos tan importante para almacenar información con la dupla usuario-navegador.

UNIDAD DIDÁCTICA 10:

API con PHP. JSON

Módulo profesional:
Desarrollo Web en Entorno Servidor

Duración: 160 Horas.
Código: 0613

Ciclo Formativo de Grado Superior:
Desarrollo Aplicaciones Web

Contenido

Resumen Introductorio	3
Introducción.....	3
Caso Introductorio	3
Desarrollo:.....	4
1. Introducción al API	4
1.1. API de datos	5
2. Construcción de un sencillo API.....	7
1.2. MVC	7
1.3. CRUD	10
1.4. API. Request METHOD	12
1.5. Plugin POSTMAN de Google Chrome	14
1.6. JSON.....	15
1.7. Estructura final	17
Resumen final:	19

Resumen Introductorio

En esta unidad aprenderemos los conceptos necesarios para poder realizar un API sencillo sobre una o todas las tablas de nuestra base de datos.

Introducción

Hoy en día donde la tecnología móvil ha crecido exponencialmente y donde los dispositivos en general requieren de información para su buen funcionamiento, se ha convertido a su vez en imprescindible el conocer los fundamentos básicos de un API para poder servir dicha información.

Un API contra una base de datos no es ni más ni menos que una aplicación que se ejecuta en servidor y que va a permitirnos comunicarnos e interactuar con las tablas de nuestra base de datos. Nos va a permitir por lo tanto realizar las cuatro acciones contra las tablas o CRUD.

Además es importante conocer el estándar JSON, que se ha convertido en prácticamente el mejor y más sencillo mecanismo de comunicación de información entre dispositivos.

Caso Introductorio

Queremos realizar un API contra la base de datos de cliente, para que un comercial con su móvil y una aplicación instalada en el mismo pueda acceder a esta información.

Desarrollo:

1.INTRODUCCIÓN AL API

Del inglés Interfaz de programación de aplicaciones, el API de una aplicación es un conjunto de clases, algoritmos y funcionalidades que nos proporcionan principalmente un acceso a información, en bases de datos normalmente, controlando y gestionando los permisos, accesos y niveles de seguridad para un *pull* de clientes.

Si lo aterrizamos en el entorno de programación, resulta que es una aplicación que nos va a permitir realizar una serie de acciones y de actividades de una forma sencilla y transparente para el usuario o máquina externa que utiliza ese API.

Nosotros nos vamos a centrar mucho en la creación y uso de un API para el acceso a información y datos en base de datos. Recordemos cómo accedíamos a una base de datos MySQL:

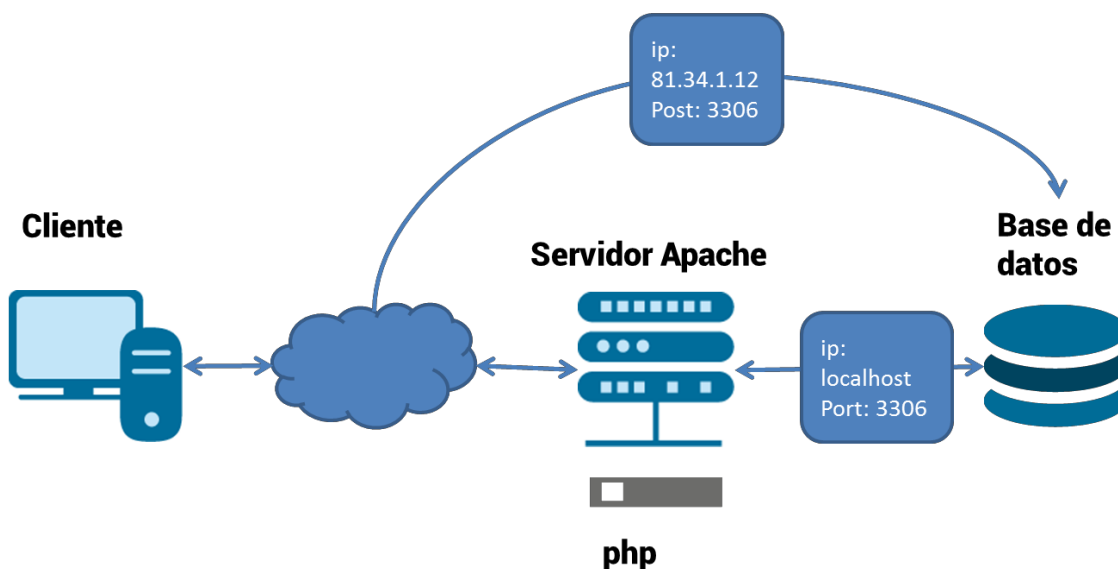


Imagen 1: Acceso a datos

Fuente de la imagen: propia

Como vemos en la anterior imagen, el cliente accedía por dos mecanismos a la información de la base de datos, bien a través del servidor Apache y de una aplicación realizada, en nuestro caso con PHP, o bien directamente.

¿Qué ocurre si el cliente no es un ser humano, sino un móvil o cualquier otro dispositivo automático? ¿Qué ocurre si quien accede no necesita disponer de una visualización de datos en formato HTML, es más en ese formato no sabría interpretar la información?

Podemos dar como respuesta que podríamos utilizar el mecanismo de la conexión directa con la base de datos, sin embargo ese mecanismo tiene dos problemas:

- Es muy dependiente del tipo de la base de datos y por lo tanto no estándar necesitando un conector para cualquier transacción. Por ejemplo en dispositivos móviles no es una alternativa la utilización de conectores para el acceso a las bases de datos.
- El acceso a la base de datos de forma directa es muy peligroso a nivel de seguridad ya que deja el acceso directo a la base de datos.

1.1. API de datos

La respuesta a las anteriores preguntas es crear un API, una app intermedia que acceda a los datos de una base de datos y que devuelva la información. Esto da solución a ambas problemáticas:

- El API utiliza un mecanismo estándar de comunicación como es el HTTP y por lo tanto no necesita de conectores privador, es el API el encargado de solucionar esta problemática
- El API se encarga de la seguridad

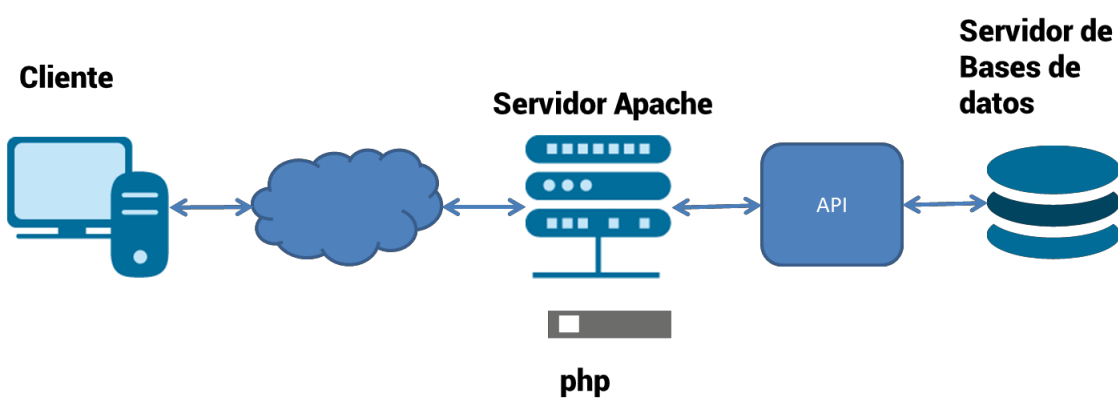


Imagen 2: Acceso a datos mediante API

Fuente de la imagen: propia

FORO 1
Título: Bases de datos para la realización de un API

2.CONSTRUCCIÓN DE UN SENCILLO API

En esta Unidad Didáctica vamos a construir un sencillo API, puesto que ya tenemos todas las herramientas y conocimientos para poder realizar un desarrollo que acceda a una serie de datos en unas tablas de la base de datos y dependiendo del método utilizado, devuelva información o bien interactúe con las tablas.

2.1. MVC

El primer paso será plantear la estructura siguiendo el Patrón MVC, así si lo que queremos es interactuar contra la tabla *usuarios*, deberemos crear la siguiente estructura:

modelo

```
|_____ db.php
          usuario.php
```



RECUERDA... MVC ... (UD8)

El MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

Si recordamos lo visto en la UD8:

- el fichero db.php será el encargado de la conexión contra la base de datos.
- El fichero usuario.php será el encargado de realizar todas las transacciones de información.

Un ejemplo de código de ambos ficheros los podemos recuperar también de la UD8.

Modificando db

```
<?php
/**
 * Permitir la conexión contra la base de datos
 */
class db
{
    //Atributos necesarios para la conexión
    private $host="localhost";
    private $user="root";
    private $pass="";
    private $db_name="usuarios";
    //Conector
    private $conexion;
    //Propiedades para controlar errores
    private $error=false;
    private $error_msj="";
    function construct()
    {
        $this->conexion = new mysqli($this->host, $this->user, $this->pass, $this->db_name);
        if ($this->conexion->connect_errno) {
            $this->error=true;
            $this->error_msj="No se ha podido realizar la conexión a la bd. Revisar base de datos o parámetros";
        }
    }
    //Función para saber si hay error en la conexión
    function hayError(){
        return $this->error;
    }
    //Función que devuelve mensaje de error
    function msjError(){
        return $this->error_msj;
    }
    //Método para la realización de consultas a la bd
    public function realizarConsulta($consulta){
        if($this->error==false){
            $resultado = $this->conexion->query($consulta);
            return $resultado;
        }else{
            $this->error_msj="Imposible realizar la consulta: ".$consulta;
            return null;
        }
    }
}
?>
```

Siendo el fichero usuario.php:

Clase hija

```
<?php
include "db.php";
/**
 *
 */
class Usuario extends db
{
    function __construct()
    {
        //De esta forma realizamos la conexion a la base de datos
        parent::__construct();
    }

    //Devolvemos todos los usuarios
    function devolverUsuarios() {
        //Construimos la consulta
        $sql="SELECT * from usuario";
        //Realizamos la consulta
        $resultado=$this->realizarConsulta($sql);
        if($resultado!=null){
            //Montamos la tabla de resultados
            $tabla=[];
            while($fila=$resultado->fetch_assoc()){
                $tabla[]=$fila;
            }
            return $tabla;
        }else{
            return null;
        }
    }
}
?>
```



EJEMPLO PRÁCTICO

Con lo visto y aprendido en Unidades anteriores y la actual podemos crear un API para una aplicación de un restaurante:

- 1) Crearemos una estructura de directorios y ficheros idéntica a la propuesta en la teoría:
 - a. Fichero padre db.php encargado de la conexión
 - b. Fichero hijo restaurante.php encargado de la interacción con una base de datos
- 2) Generaremos el contenido del fichero db.php de acuerdo al ejemplo anterior

2.2. CRUD

Deberemos crear el crud completo contra la base de datos y tablas contra las que queremos realizar la gestión de información. Esto lo realizaremos dentro del fichero usuario.php descrito en el apartado anterior.



RECUERDA... CRUD ... (UD8)

Deberemos crear el crud completo contra la base de datos y tablas contra las que queremos realizar la gestión de información. Esto lo realizaremos dentro del fichero usuario.php descrito en el apartado anterior.

Sería por lo tanto en este punto en el que desarrollaríamos todas las funciones necesarias para interactuar con nuestra base de datos:

- Funciones de lectura, tantas como necesidad tengamos en la aplicación. Como mínimo deberíamos tener tantas como tablas tengamos en nuestra aplicación.
- Funciones de escritura, para realizar la inserción de información en las tablas
- Funciones de actualización, para realizar la actualización de la información
- Funciones de borrado de información

En nuestros ejemplos no nos vamos a preocupar de la seguridad de nuestro API, pero ya podemos imaginar que uno de los problemas a atender más importante dentro del desarrollo del API es justamente limitar y/o permitir con unos determinados niveles de seguridad a la información justa.



EJEMPLO PRÁCTICO

Siguiendo con el ejemplo anterior:

- 1) Crearemos la tabla de la base de datos con los siguientes campos

	Restaurante	
	<ul style="list-style-type: none">- id- nombre- ciudad- tipo	
	<ul style="list-style-type: none">+ listarRestaurantes+ listaRestaurante(string:id)+ insertaRestaurante(string:nombre, string:ciudad,int:tipo)+ actualizaRestaurante(string:nombre, string:ciudad,int:tipo)+ borraRestaurante(string:id)	

2) Generaremos las funciones dentro de restaurante.php para poder acceder a la base de datos

3) Crearemos un fichero de prueba, debugRestaurante.php para realizar las diferentes pruebas contra el API generado

2.3. API. Request METHOD

Cuando vimos Formularios vimos los diferentes tipos de métodos request que podíamos utilizar:

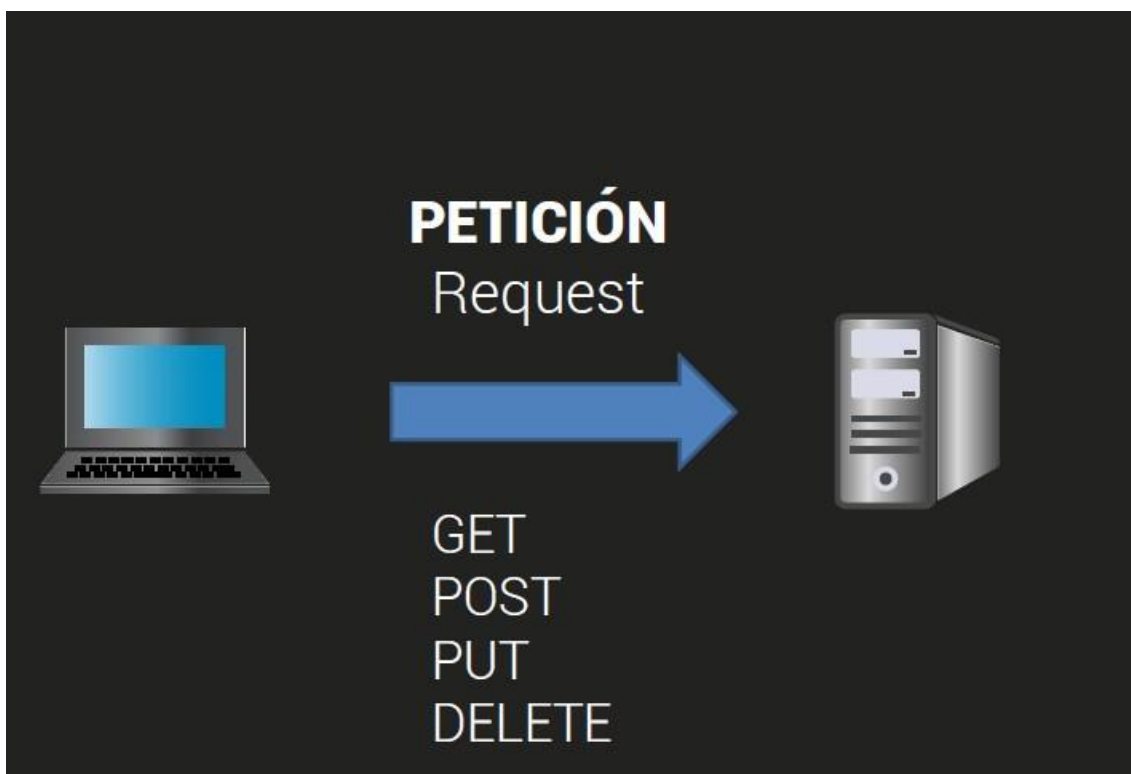


Imagen 3: Peticion Request

Fuente de la imagen: propia

En ese momento nos centrábamos únicamente en el uso de Formularios para la interacción con nuestras aplicaciones, pero en el momento que deja de ser un usuario final quien interactúa con nuestra aplicación y pasa a ser dispositivos que no pueden utilizar formularios, comienza a tener sentido el uso de los diferentes métodos de petición para poder discernir qué acción vamos a realizar con nuestra base de datos:

- Con el método GET realizaremos un SELECT de la base de datos
- Con el método POST realizaremos un INSERT
- Con el método DELETE realizaremos un DELETE
- Con el método PUT realizaremos un UPDATE

Este aterrizaje es mucho más complejo sobre todo si nos metemos dentro del mundo API RestFul, pero para no complicar la situación nos quedaremos en esta distinción.



PARA SABER MÁS

Encontraremos mucha información tanto sobre el API REstFul como sobre los diferentes métodos y sus usos.

<http://restcookbook.com/>

A través de la superglobal \$_SERVER podemos conocer cuál es el método en la petición por parte del cliente, el código es bien sencillo.

Según la documentación de php.net:

'REQUEST_METHOD'

Which request method was used to access the page; i.e. 'GET', 'HEAD', 'POST', 'PUT'.

```
$_SERVER['REQUEST_METHOD']
```



BIBLIOGRAFÍA RECOMENDADA

Libro recomendado, *Hacking with PHP* (Hudson, 2015)

En el capítulo 17 encontramos referencia a conceptos avanzados sobre seguridad muy interesantes cuando desarrollamos aspectos tan sensibles como un API

<http://www.hackingwithphp.com/17/0/0/security-concerns>



COMPRUEBA LO QUE SABES:

Una vez visto los diferentes tipos de HTTP Request ¿serías capaz de realizar un formulario dentro de un fichero php para realizar una consulta de tipo INSERT mediante el protocolo PUT?

Coméntalo en el foro.

2.4. Plugin POSTMAN de Google Chrome

En este punto, tener una aplicación que nos permita realizar comprobaciones y prueba de envío de información con diferentes métodos sería fantástica.

Existen diversas aplicaciones, pero la reina claramente es POSTMAN



PARA SABER MÁS

Aplicación que permite enviar y codificar datos para la comprobación, uso y análisis de APIs

<https://www.getpostman.com/>



En el vídeo encontrarás la explicación sobre qué es POSTMAN

✓ https://www.youtube.com/watch?v=ptvV_Fc3hd8&list=PLM-7VG-sgbtCJYpjQfmLCcJZ6Yd74oytQ&autoplay=1

TAREA 1
Título: Delete

TAREA 2
Título: Update

2.5. JSON

No es la intención ni la misión de este curso entrar en detalle del uso de JSON, pero para acabar de formar nuestro API, es imprescindible conocer generalidades del mismo.

JSON, acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

Se ha convertido en un estándar para la comunicación de información entre procesos, y sobre todo en la utilización de APIs. Veamos un ejemplo:

Ejemplo

```
{
  "glossary": {
    "title": "example glossary",
    "GlossDiv": {
      "title": "S",
      "GlossList": {
        "GlossEntry": {
          "ID": "SGML",
          "SortAs": "SGML",
          "GlossTerm": "Standard
Generalized Markup Language",
          "Acronym": "SGML",
          "Abbrev": "ISO 8879:1986",
          "GlossDef": {
            "para": "A meta-markup language, used to
create markup languages such as DocBook.",
            "GlossSeeAlso": ["GML",
"XML"]
          },
          "GlossSee": "markup"
        }
      }
    }
  }
}
```


Como podemos observar a diferencia con XML u otros Lenguajes de Marcas, el nivel de "burocracia" es mínima, ya que consiste en una estructura jerárquica de datos.



PARA SABER MÁS

Información del estándar y ejemplos de formación

<http://json.org/>

Cómo codificar y decodificar json con php va a ser bien sencillo también, ya que a partir de arrays asociativos podemos generar una cadena json fácilmente:

json_encode

```
<?php
$arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);

echo json_encode($arr);
?>
```

Y mediante la función decode tendremos el paso contrario:

json_decode

```
<?php
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

var_dump(json_decode($json));
?>
```



COMPRUEBA LO QUE SABES:

Una vez visto el estandar JSON ¿serías capaz de realizar un formulario dentro de un fichero php para realizar una consulta de tipo READ mediante el protocolo GET realizando la petición mediante JSON? Coméntalo en el foro.

2.6. Estructura final

Una vez visto tanto el modelo como la vista (en este caso la salida es un json), la estructura final quedaría de la siguiente forma:

modelo

```
|_____ db.php
                restaurante.php
```

api.php



ARTÍCULOS DE INTERÉS

Sobre PHP, y desarrollo PHP podemos encontrar una gran cantidad de información. No es menos el apartado del API desarrollado con PHP, donde podemos encontrar alternativas al desarrollo y metodologías utilizadas en esta unidad:

- <http://www.weblantropia.com/2016/08/30/restful-api-api-php-mysql/>
- <https://desarrolloweb.com/articulos/crear-api-rest-json-server.html>
- <http://frikibloggeo.blogspot.com.es/2017/01/crear-un-web-service-api-rest-con-php-y.html>

Donde api.php realizaría las siguientes labores:

- Recibe las peticiones por parte del cliente
- Distingue entre los métodos solicitados
- Recoge los datos si son necesarios a través de la superglobal `$_REQUEST`
- Realiza la llamada al método solicitado
- Devuelve la información a través de `json_encode`

En este caso la salida de datos no se realiza embebiendo el echo dentro de etiquetas `<html>` sino directamente con un echo del `json_encode`

api.php

```
<?php
include "../modelo/restaurante.php";

//distinguimos el tipo de peticion
$requestMode=$_SERVER['REQUEST_METHOD'];
if($requestMode=="GET") {
    ...
    echo json_encode($result);
}elseif($requestMode=="POST") {
    ...
    echo json_encode($result);
}elseif($requestMode=="PUT") {
    ...
    echo json_encode($result);
}elseif($requestMode=="DELETE") {
    ...
    echo json_encode($result);
}else{
    echo json_encode(["resultado"=>"Fallo"]);
}
?>
```



EJEMPLO PRÁCTICO

Siguiendo con el ejemplo anterior:

- 1) Crearemos un nuevo fichero, `api.php`, que será el encargado de recibir y devolver la información solicitada.
- 2) En dicho fichero crearemos una estructura sencilla con un `switch` o `if` y dependiendo del método usado, utilizaremos la función creada en `restaurante.php`
- 3) Incluiremos el código necesario para poder recibir/devolver la información con JSON

Resumen final:

En esta Unidad condensamos todo el conocimiento previamente acumulado para la desarrollo de un API de datos utilizando una arquitectura MVC, realizando el CRUD dentro de la aplicación y recibiendo y devolviendo la información mediante el standard JSON.

Un API de datos nos permite acceder de forma controlada a nuestras bases de datos y/o de información para poder usadas bien por nosotros mismos, o bien por otras aplicaciones como puede ser una aplicación móvil.

Para ello, un lenguaje PHP, como hemos utilizado en este módulo es perfecto para el desarrollo rápido y sencillo de la infraestructura necesaria de API de datos.