UNIVERSIDAD DON BOSCO
FACULTAD DE INGENIERÍA
PHP
GRUPO 01T
CICLO 3-2021

PARCIAL TEORICO
“PHP - Programación Web avanzada para profesionales”

CASTELLANOS MARTINEZ, ANDREA ALEJANDRA 100% CM150518
MEJIA SOSA, MARLENY GUADALUPE 100% MS132476

CATEDRATICO: ALEXANDER SIGÜENZA

SOYAPANGO, MARTES 06 DE JULIO DE 2021

Capítulo 4

Trabajando con cookies

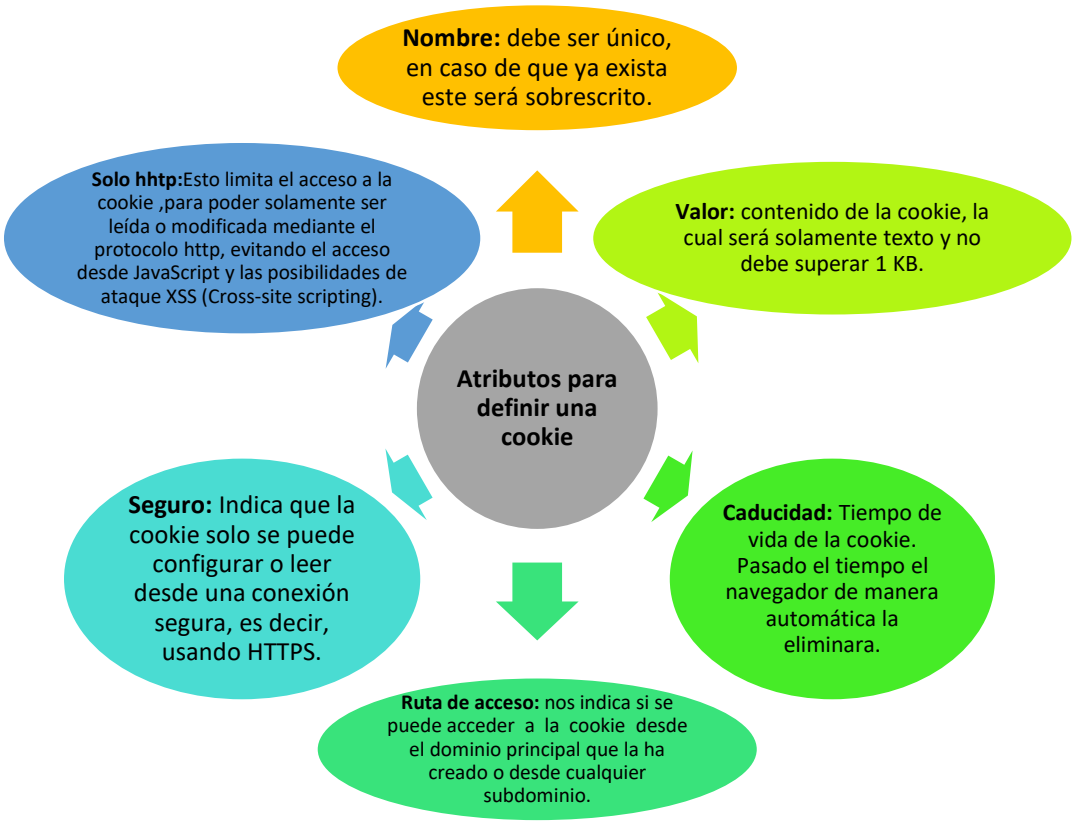
Recetas Web

Las cookies son simples archivos de texto que se almacenan en el disco duro de nuestros de nuestros visitantes, los cuales pueden ser recuperados en el próximo ingreso que realicemos.

Las cookies almacenan solamente texto dentro de ellas que no superen 1KB de tamaño, podemos almacenar restricciones técnicas. Se pueden acceder a ellas desde el equipo donde se almacenaron, incluso desde extensiones populares para Firefox, Chrome o internet Explorer.

Es fundamental tener en cuenta que solo se debe almacenar información que contengas preferencias del usuario o personalizaciones de los sitios, jamás debe ser información privada o de carácter personal ya que puede que terceros tengan acceso a esta.

En relacion a seguridad solo podemos acceder a las cookies por medio de el dominio o subdominio desdes el cual se ha dejado la cookie. Se puede definir la autoridad de acceso del dominio principal, permitiendo que todos los subdominios accedan.



Funcion php set cookie

|Función que devuelve un booleano. Sí la función nos devuelve un valor verdadero esto nos indica que se puede incluir la cookie en el navegador, de lo contrario si nos devuelve falso indica que no se pudo colocar la cookie en el sistema.

```php
<?php
setcookie("nombre1", "mivalor");
setcookie("nombre2", "mivalor2", time() + 3600);
echo "<pre>";
print_r($_COOKIE);
echo "</pre>";
?>
```

Este código en pantalla simplemente nos informa el valor de las cookies que se han guardado. En el navegador de nuestros visitantes y podemos acceder a ellos mediante el array $__COOKIE, indicando como componente el nombre.

Para poder eliminar una cookie, se utiliza la misma sentencia con la que se crea, pero con el valor o contenido vacío y el tiempo a configurar será una hora hacia atrás, esto hará que la cookie se sobrescriba vacía en su verificación de caducidad se eliminara por completo.

```php
<?php
setcookie("nombre2"," ", time() - 3600);
echo "<pre>";
print_r($_COOKIE);
echo "</pre>";
?>
```

Para poder reutilizarlo cuantas veces sea necesario este Código, podemos llevar a cabo el desarrollo de un manejador de cookies mediante clases y objetos .

```php
<?php
class CookieUtils
{
    const Defaultlife=3600;//segundos de duracion = una hora
    public function get($name){
        if(isset ($_COOKIE[$name]))
        {
            return $_COOKIE[$name];
        }
        else{
            return false;
        }
    }
}
?>
```

```php
<?php
$Cookie = new CookieUtils();

$Cookie->set('preferencias', 'color:azul|font:verdana|header:1.gif');

echo "Recuperamos el valor usando nuestro método get <br>";
echo $Cookie->get('preferencias');
echo "<br><br> Mostramos el contenido usando print_r <br>";
echo "<pre>";print_r($_COOKIE['preferencias']);
echo "</pre>";
public function getAllCookies() {
    if(!empty($_COOKIE)) {
        return array_keys($_COOKIE);
    } else {
        return false;
    }
}
public function deleteAllCookies() {
    $c = self::getAllCookies();
    if(!empty($c)) {
        for($i=0;$i<count($c);$i++) {
            self::delete($c[$i]);
        }
    }
?>
```

Contamos con una clase que nos permitirá manejar en forma centralizada todo lo relacionado con las cookies de nuestro desarrollo.
Nuestro método **getAllCookies** nos permitirá obtener un array con todas las cookies que se hayan seteado para nuestro dominio. Con este método tendremos acceso a todas las cookies incluso si no conocemos el nombre de una de ellas.

El método **deleteAllCookies** limpiar todas las cookies de una sola vez.

Seguridad en el uso de cookies

Cuando hablamos de seguridad es importante tomar en cuenta ciertos aspectos para manejar las cookies:

- No se debe almacenar información privada sin ser encriptada.
    - Siempre debemos encriptar los datos con un algoritmo que nos permita recuperar dicha información.
- Utilizar cookies de tipo HTTPOnly cuando sepamos que no será necesario leerlas desde algún lenguaje del lado cliente.

Sesiones

El tiempo que los usuarios pasan visitando nuestro sitio web, o el tiempo que dejan de visitar nuestro sitio web durante un tiempo prolongado, o el tiempo que cierran su navegador. Las sesiones aparecen en PHP como una alternativa a la persistencia de la información sin almacenarlas en la base de datos.

Php nos brinda la función sesión_start, que permite iniciar una sesión y en el array $_SESSION se almacenara la información. Para finalizar una sesión utilizamos la función session_destroy.

Autenticación de usuarios

Crearemos una clase extendida de CookieUtils, lo cual permitirá que reutilicemos todo lo relacionado con el manejo de cookies, se creara un Login de usuarios que haga uso de cookies y sesiones para autentificar usuarios.

Para poder autenticar usuario declararemos atributos de clase, una variable booleana para almacenar el estado de la autenticación $loggedin, y un array donde se guardarán los datos del usuario autentificado $arrUsuario.

Para poder llevar a cabo la autenticación tomaremos como prueba dos valores estáticos que serán nuestro usuario y password correctos.

Una vez verificado si existe información en nuestras sesiones o cookies, debemos validarla contra la información de nuestra base de datos; en este caso, respecto de los dos atributos de clase que hemos definido como testeo.

Creando un captcha con GD

La palabra captcha es el acrónimo de **Completely Automated Public Turing test to tell Computers and Humans Apart** (prueba de turing pública y automática para diferenciar máquinas y humanos).Este consiste en ingresar una serie de caracteres alfanuméricos que se muestran en una imagen,ya sea con un fondo de color ,que los caracteres se encuentren distorsionados. Esto permite evitar ataques de fuerza bruta o que una maquina no puede leerlos.

En Php para poder hacer uso de archivo de imágenes es necesario tener la librería GD. Al hacer uso de esta podemos crear una clase, que nos permita generar una imagen distorsionada, siendo congelable su longitud, tipo de caracteres, color de fondo, podremos definir todos aquellos elementos para poder dar complejidad ,y la longitud que el captcha tendrá.

```
class Captcha{
  private static $caracteres = 'abcdefghijklmno-pqrstuvwxyz0123456789';
  private function texto($long = 5) {
      for($i=0;$i<$long;$i++) {
          $texto .= self::$caracteres{
              rand(0,strlen(self::$caracteres))};
              }
              return $texto;
              }
              public function captcha($w=150,$h=30) {
                  session_start();
                  $_SESSION['captcha'] = self::texto();
                  $captcha = imagecreate( $w, $h );
                  $colorFondo = imagecolorallocate( $capt-cha, 0, 0, 255 );
                  $colorTexto = imagecolorallocate( $capt-cha, 255, 255, 0 );
                  $colorLinea = imagecolorallocate( $capt-cha, 255, 105, 180 );
                  imageline($captcha, 0, 9, 150, 9, $color-Linea);
                  imageline($captcha, 0, 15, 150, 15, $co-lorLinea);
                  imageline($captcha, 0, 20, 150, 20, $co-lorLinea);
                  imagestring($captcha, 5, 30, 5, $_SESSION['captcha'],
                  $colorTexto );
                  header( "Content-type: image/png" );
                  imagepng( $captcha );
                  imagedestroy( $captcha );
                  }
                  }
```

En esta clase encontramos la función “texto” la cual nos facilitará una cadena de texto aleatoria que estará formada por los caracteres previamente definidos. Sea asigna una variable que contiene la longitud por defecto de este texto.

La función captcha es de carácter público y estará actuando como un constructor de nuestra clase, y será la encargada de comparar la cadena de texto ingresada por el usuario y lo que nuestra función texto envió.

Dicha clase hace uso de la libraría GD, que nos permite realizar un manejo de gráficos desde PHP. **Imagecreate** nos permite crear un rectángulo con las dimensiones ya establecidas, **Imagecolorallocate** nos permite agregar un color de fondo por defecto a nuestra imagen captcha.

Con este script se nos devolverá una imagen que contenga el captcha, para esto se le indica al navegador mediante el envío de headers o cabeceras que el contenido a mostrar será una imagen de tipo PNG: header(“Content-type: image/png”); a continuación, la mostraremos en pantalla Es muy habitual destruir la imagen dinámicamente creada de la memoria una vez mostrada.

Para llevar a cabo la aplicación de dicha clase simplemente debemos utilizar el tag HTML img, indicando como origen de la imagen nuestro archivo PHP.

```php
<?php
require(“classes/clsCaptcha.php”);
$captcha = new Captcha();+
 ?>
```

Es importante tener en cuenta que utilizar un captcha podría limitar el acceso de personas con deficiencias visuales, puesto que no podrían interpretar los lectores automáticos que se instalan, bloqueando el acceso al recurso.

Creando firmas dinámicas para nuestros e-mails

Genere una firma que pueda cambiar con el tiempo y obtenga información de nuestra base de datos o archivo de configuración. De esta forma, podemos actualizar la información, y se verá reflejada en todas las llamadas que se hayan realizado.

Haciendo uso de la librería GD y la librería Freetype que nos permitirá el uso de tipografías del sistema para la firma. De no contar con esta librería se puede hacer uso de imagestring.

Se hará uso de dos recursos gráficos que deben estar disponibles en nuestro servidor, una imagen de fondo que podrá estar en cualquier formato soportado (PNG, JPG, GIF), y una tipografía cuyo archivo deberá ser de tipo TTF.

```
class Firmas {
  public function __construct($bgFile) {
$dim = getimagesize($bgFile);
if($dim) {
$firma = imagecreatetruecolor($dim[0],$dim[1]);
$firma = imagecreatefromjpeg($bgFile);
$color = imagecolorallocate($firma, 255, 255, 255);
$fuente = 'fuente.ttf';
imagettftext($firma, 14, 0, 10, 35, $color, $fuente, "Christian Cibelli");
imagettftext($firma, 11  , 0, 10, 55, $color, $fuente, "chcibelli@gmail.com");
imagettftext($firma, 11, 0, 10, 75, $color, $fuente, "http://www.christiancibelli.com");
header("Content-type: image/png");
imagepng($firma);
imagedestroy($firma);
} else {
    return false;
    }
        }
}
```

La clase firma cuenta con todas las definiciones en el constructor de manera que solo se deberá crear un nuevo objeto, el cual devolverá la firma generada en pantalla.

El uso de **imagettftext** nos permite añadir el texto que deseamos agregar a nuestra firma, tomando también como parámetros el tamaño en pixeles, el Angulo de rotación que esta tendrá, la posición del texto y el color que vamos a utilizar.

Con este archivo Php podríamos incluir en e-mails esta firma, al igual que podríamos agregar una frase de manera automática.

Protegiendo nuestras imágenes con marcas de agua

La técnica de la marca de agua consiste en insertar un mensaje oculto o no en el interior del objeto que deseamos proteger. En esta ocasión veremos como estampar un mensaje sobre una imagen.

```php
<?php
class Watermark {
    public function __construct($i,$wm) {
        $dim = getimagesize($i);
        if($dim) {
            $img = imagecreatetruecolor($dim[0],$dim[1]);
            $img = imagecreatefromjpeg($i);
            $color = imagecolorallocate($img, 211, 211, 211);
            $fuente = 'fuente.ttf';
            imagettftext($img, 20, 0, $dim[0]/3, $dim[1]/2, $color, $fuente, $wm);
            header("Content-type: image/png");
            imagepng($img);
            imagedestroy($img);
        } else {
            return false;
        }
        }
    }
        $img = new Watermark('./adjuntos/img/jpg/paisaje.jpg','(C) '.date("Y").' Demo ');
?>
```

Para hacer uso de la clase watermark reemplazamos los source de nuestro tag Img por algo similar a watermark.php?img=paisaje.jpg. Lo cual permitirá que nuestra clase busque la imagen y esta sea devuelta con la marca de agua, la desventaja de utilizar este tipo de clase es que cualquier usuario con conocimientos básicos de HTML podría tener acceso a la imagen sin marca de agua.

Una alternativa para evitar este inconveniente seria aplicar rewrite en nuestro servidor apache a través del archivo .htaccess, ordenaremos que todas las llamadas al archivo de imagen se reescriban (rewrite) en nuestro archivo PHP, que devolverá el archivo con la marca de agua.

Nuestro archivo .htaccess luciría de la siguiente manera:

```
RewriteEngine On
RewriteRule  ^(.*).jpg$ /wm/wm.php?i=$1.jpg [L,NS]
```

Al utilizar esta opción cuando el usuario solicite algún archivo que finalice en .jpg, el servidor internamente llamará a wm.php, enviándole como dato por URL en la variable $i, el nombre de la imagen solicitada. Esto generara una reescritura de la URL solicitada que hace internamente el navegador. De esta forma, se devolverá la imagen con la marca de agua impresa.

Generación de thumbnails usando GD

La redimensión de imágenes en forma dinámica y on-line, permite poder aligerar la carga del sitio Web. Se busca realizar esto para contar con sitios web de bajo peso ya que en algunas ocasiones se acceden a estos a través de celulares, PDAS, o en conexiones de baja velocidad.

Para redimensionar imágenes, podemos crear un método que modifique en forma dinámica y online, mostrando al usuario mediante el envió de headers imágenes redimensionadas.

```php
<?php class Thumb
{
    public function onlineThumb($img,$maxWidth,$maxHeight=0)
    {
        list($oWidth,$oHeight,$image_type) = getimagesize($img);
        switch($image_type)
        {
            case 1: $im = imagecreatefromgif($img);
            break;
            case 2: $im = imagecreatefromjpeg($img);
            break;
            case 3: $im = imagecreatefrompng($img);
            break;
        }
        if($maxHeight == 0)
        $maxHeight = $oHeight * $maxWidth / $oWidth;
        $newImg = imagecreatetruecolor($maxWidth, $maxHeight);
        if(($image_type==1) || ($image_type==3))
        {
            imagealphablending($newImg, false);
            imagesavealpha($newImg,true);
            $transparent=imagecolorallocatealpha($newImg,255,255,255,127);
            image□lledrectangle($newImg,0,0,$maxWidth,$maxHeight,$transparent);
        }
           imagecopyresampled($newImg,$im,0,0,0,0,$maxWidth,$maxHeight,$oWidth,$oHeight);
           switch ($image_type)
           {
               case 1:
                header("Content-type: image/gif");
                imagegif ($newImg);
                break;
                case 2:
                header("Content-type: image/jpeg");
                imagejpeg($newImg);
                break;
                case 3:
                header("Content-type: image/png");
                imagepng ($newImg);
                break;
           }
           imagedestroy($newImg);
    }
}
?>
```

El método **onlineThumb** será el encargado de generar la miniatura de la imagen solicitada, la cual podremos encontrar ya sea en nuestro servidor o a la cual se accede de manera remota.

Dicho método toma 3 parámetros, la imagen a redimensionar, el ancho máximo será de carácter obligatorio, del mismo modo podremos agregar el alto de la imagen de no hacerlo el método de manera automática calcular el alto que mejor convenga.

La función nativa de PHP getimagesize nos brindara información sobre la imagen. En este caso, pasaremos a variables utilizando list, el ancho, alto y el tipo que será un valor entero donde retornar 1 indicará gif, 2 jpg y 3 png. Este dato será fundamental para crear la miniatura en el mismo formato que en su original; y también, para enviar el header correspondiente a la hora de mostrarla en pantalla.

Para hacer uso de esta solo debemos indicar donde se encuentra la imagen a redimensionar y el tamaño que deseamos. Estos datos pueden agregarse a un archivo que genere la instancia del objeto y reciba por parámetro los datos, utilizando esta cadena como src de nuestro tag <img/>.

```
<img src="resize.php?i=http://img.genbeta.com/2009/01/mozilla-test-pilot.png&w=100" />
```

Y nuestro archivo luciría de la siguiente manera:

```
<?php
require("classes/clsThumbs.php");
$th = new Thumb();
$th->onlineThumb($_REQUEST['i'],$_REQUEST['w']);
?>
```

Esta alternativa se puede utilizar siempre y cuando nuestro sitio web no posea un alto trafico. Pero podemos adaptar dicha clase para generar las miniaturas y que estas sean almacenadas en el servidor. La modificación que realizaremos es en lugar de mostrar la imagen por pantalla, enviaremos un segundo parámetro a imagegif, imagejpeg o imagepng, donde indicaremos la ubicación y el archivo de destino.

```
switch ($image_type)
{
 case 1:
     imagegif ($newImg,$fileDest,100);
     break;
 case 2:
     imagejpeg($newImg,$fileDest,100);
     break;
 case 3:
     imagepng ($newImg,$fileDest,100);
     break;
 }
 imagedestroy($newImg);
 return $fileDest;
```

Se agrega una normalización que permitirá verificar si la imagen existe y está disponible; de ser así esta se redimensiona, sino que se toma de manera local.

```php
    private function thumbExists($img)
{
    if(@le_exists($img))
    {
        return true;
    }
    return false;
}
```

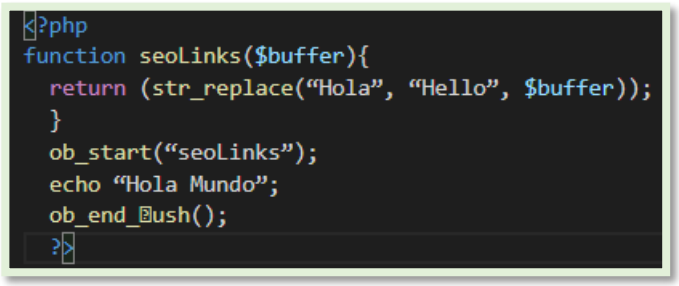
URL amigables con PHP

Las URL amigables o semánticas son las que permiten que el usuario pueda distinguir el contenido al que apunta dicha ruta con solo verla. El objetivo principal de una url amigable consiste en tener una mejor indexación de los motores de búsqueda, que en su gran mayoría otorgan peso al contenido de acuerdo con las palabras clave que encuentren, y mejorando la posición de los contenidos en las búsquedas orgánicas. Es decir que si se cuenta con una url amigable en los resultados de búsqueda se encontrara entre los primeros de arriba.

Estas url deberán ser interpretadas por el servidor, un ejemplo de este podría ser Apache, mediante el modulo mod_rewrite que simplemente reescribe la URL enviada internamente por otra. Se estima que una URL con más de 2000 caracteres puede no ser soportada por algunos navegadores. Internet Explorer de Microsoft tiene su límite en 2038, dando error pasado ese número.

Modificando contenido con Output Buffering

La función **ob_start** inicia el almacenamiento o buffering de la salida generada. Esta función admite el envío como parámetro de una función para la llamada de regreso o callback, la cual tomará todo el contenido almacenado en forma de string y devolverá un String procesado por la función que se ha indicado.



```php
<?php
function seoLinks($buffer){
  return (str_replace("Hola", "Hello", $buffer));
  }
  ob_start("seoLinks");
  echo "Hola Mundo";
  ob_end_flush();
  ?>
```

El código anterior mostrará en pantalla el texto Hello Mundo en lugar de Hola Mundo, dado que la función de callback reemplaza el contenido del buffer

La función **ob_end_flush** envía o vuelca en pantalla todo el contenido almacenado en el buffer, y deshabilita el almacenamiento previamente iniciado con ob_start.

Detectando crawlers de motores de búsqueda

Todos los sitios Web que se encuentran on-line son permanentemente visitados por procesos automáticos pertenecientes a los diferentes buscadores los cuales verifican el contenido, reindexan, comprueban que aún esté disponible, y la información que recolectan como resultado de estos procesos es aquello que vemos al hacer una búsqueda en Internet.

Performance, seguridad y buenas prácticas

Existen alternativas para que nuestro contenido dinámico se transforme ocasionalmente en contenido estático cuando sea conveniente, como es el caso de un sitio web con alta concurrencia.

Estatizando en archivos HTML

- **Estatizando contenido con funciones PHP para el manejo de sistema de archivos**

  PHP procesa los scripts del lado del servidor y esto devuelve simplemente contenido HTML. Es posible evitar ese procesamiento repetitivo para que el usuario vea directamente una página HTML, la cual sería actualizada cuando se realicen cambios o cada cierto período mediante una tarea programada en el servidor.

  El manejo de las estáticas se realiza con la clase Estáticas de la siguiente forma:

```
class Estáticas
{
   public function estatizar($url,$archivo)
   {
      $estatica = le_get_contents($url);
      if(empty($estatica))
      {
       return false;
      }
      else
      {
         $estatica .= ‘
         <!-- ‘ . date(‘d-m-Y H:i:s’) . ‘ -->’;
      }
      $handle = fopen($archivo, ‘w+’);
      if(!$handle)
      {
```

```
            return false;
        }
        else
        {
            fwrite($handle, $estatica);
            $handle = fclose($handle);
            return true;
        }
    }
}
```

- **Validando errores en los archivos estáticos**

  Mostrar contenido estatizado en HTML es mucho menos costoso que procesar llamadas a base de datos por cada solicitud. Por este motivo tenemos que asegurarnos que el contenido estático generado siempre se vea correctamente, y en caso de existir algún problema, no realizar la copia del archivo estático actualizado.

  Para validar que el archivo esté correcto, usaremos una expresión regular que busque una palabra clave dentro del contenido del archivo estático que se va a guardar.

```
public function estatizar($url,$archivo)
{
    $estatica = file_get_contents($url);
    if(empty($estatica))
    {
        return false;
    }
    else
    {
```

```
        $estatica .= '<!-- ' . date('d-m-Y H:i:s') . ' -->';
    }
    //Verificamos que no haya errores
    if (preg_match("/\bERRORWEB\b/", $estatica) || empty($estatica) )
    {   //Se encontró ERROR no copiamos
        $copiar = false;
    }
    else
    {
        $copiar = true;
    }
    if($copiar)
    {
        $handle = fopen($archivo, 'w+');
        if(!$handle)
        {
         return false;
        }
        else
        {
            fwrite($handle, $estatica);
            $handle = fclose($handle);
            return true;
        }
    }
    else
    {
        return false;
    }
}
```

- **Creando estáticas bajo demanda y/o como tareas programadas**

Hemos visto opciones para estatizar el contenido, pero necesitan ser llamadas para realizar la copia del contenido, para ello existen dos opciones

1. Ingresar una tarea programada en el servidor a través de cron, en el caso de sistemas operativos Linux o Unix y desde la gestión de tareas programadas, en el caso de usar Windows.
2. Que los usuarios soliciten y ejecuten esta tarea cuando necesiten ver el contenido.

En cada visita a un contenido definido como estático, deberemos verificar si ya existe la versión estática, en caso de existir incluiremos esa página. En el caso contrario, procederemos a crearla para luego incluirla.

Para asegurar la actualización programaremos una tarea que se ocupe de eliminar el contenido estático cada cierto tiempo. Entonces, cuando alguien visite una página y no la encuentre, la creará. Dentro de la ventana de tiempo definida el próximo visitante verá el archivo HTML que otro usuario en su primera visita acaba de crear por nosotros.

Manejo de errores

El manejo y control de los mensajes de errores es fundamental debido a la información potencial que dan estos mensajes a los usuarios cuando algún error inesperado ocurre.

El primer paso es lógicamente deshabilitar el muestreo de errores a los usuarios en ambientes de producción, o sea, en los servidores a los cuales tiene acceso el visitante. Esto podemos hacerlo mediante la configuración de PHP en su archivo de directivas php.ini o bien indicando al inicio de cada uno de nuestros archivos

```
<?php error_reporting(0);?>
```

De este modo el servidor no muestra los errores en pantalla, pero el usuario no verá nada o podrá ver el contenido hasta donde haya ocurrido el error.

Esto es una buena práctica que también se puede utilizar para dejar registro de los incidentes, indicar al usuario mensajes amigables o tomar acciones de acuerdo con el tipo

de error que se haya producido y además sirvan de alertas de que es necesario un mantenimiento.

A continuación, veremos cómo crear nuestro propio manejador o handler de errores.

- **Creando nuestra clase para el manejo de errores**

  Por defecto, PHP muestra los errores de acuerdo con el nivel de configuración que se haya indicado en la pantalla, y con los mensajes estándar que se hayan señalado.

  Para esto contamos con la posibilidad de asignar una función propia que reciba los errores y los maneje, esto lo hacemos mediante la función set_error_handler, que recibe el nombre de la función propia que queramos utilizar, y nos devuelve el gestor de errores que estaba vigente al momento de hacer el cambio. De este modo, podemos volver al anterior cuando sea necesario. En caso de que estuviéramos usando el propio del sistema, éste devolverá null.

  El uso básico consiste en principio en indicarle a PHP que no utilizaremos más su gestor de errores nativo:

```
<?php $gestor = set_error_handler(“gestorErrores”); ?>
```

  Siguiente paso es indicar que use un método de clase para poder mantener todo en clases y objetos.

```
<?php
  $gestor = set_error_handler(array(‘clsErrores’, ‘   gestorErrores’));
?>
```

  LA clase clsErrores sea quien se ocupará de modificar el manejador de errores y luego mostrar e informar los errores con otro formato.

```
class clsErrores
{
   public $errtype = array
   (
      1=> “ERROR”,
      2=> “WARNING”,
      4    => “PARSE ERROR”,
      8    => “NOTICE”,
```

```
        16   => “CORE ERROR”,
        32   => “CORE WARNING”,
        64   => “COMPILE ERROR”,
        128  => “COMPILE WARNING”,
        256  => “USER ERROR”,
        512  => “USER WARNING”,
        1024 => “USER NOTICE”,
    );

    private $mostrarErrores = true;

    public function __construct()
    {
        $gestor = set_error_handler(array($this, ‘gestorErrores’));
        error_reporting(E_ALL);
    }
}
```

Luego de la definición de la clase, generaremos un array, que asociará los números de errores internos de PHP con el tipo de error en texto.Otra variable indicará si se muestran o no los errores.

El siguiente paso es la creación del método gestorErrores, que indicamos como nuevo manejador de errores en el constructor.

Este método tomara los errores y transformara los mensajes, permitiendo mostrarlos en un formato amigable con información detallada, por ejemplo, el archivo donde ocurre, el número de línea, el tipo de mensaje de error y agregaremos también la IP del servidor.

```
public function gestorErrores($errno, $errstr, $□le, $line, $context)
{
    $strErr  = “<pre>”;
    $strErr .= “-- ERROR “.$this->errtype[$errno].” --” .PHP_EOL;
```

```
    $strErr .= “FECHA : “ . date(“Y-m-d H:i:s”) .PHP_EOL;
    $strErr .= “ARCHIVO : “ . $□le .PHP_EOL;
    $strErr .= “LINEA : “ . $line .PHP_EOL;
    $strErr .= “IP SERVIDOR : “ . $_SERVER[‘SERVER_ADDR’] .PHP_EOL;
    $strErr .= “IP USUARIO : “ . $_SERVER[‘REMOTE_ADDR’] .PHP_EOL;
    $strErr .= “MENSAJE : “ . $errstr . PHP_EOL;
    $strErr .= “-- ERROR “.$this->errtype[$errno].” --” .PHP_EOL;
    $strErr .= “</pre>”;

    if($this->mostrarErrores)
    {
        echo $strErr;
    }
    else
    {
        echo “ERROR”.PHP_EOL;
    }
}
```

Por último, agregaremos la opción de login en un archivo de texto y cierta clase de errores según el nivel o tipo indicado en el atributo privado de la clase. De este modo, podremos obtener un detalle de los mensajes archivados y tomar acciones correctivas en el código fuente.

```
private $loguearErrores = true;
private $archivoErrores = ‘/tmp/php_errores.log’;
```

Agregamos una condición más dentro del método controlador de errores para que añada al archivo indicado el mensaje de error detectado.
La clase para el control de errores completa se vería de la siguiente forma:

```
<?php
class clsErrores
{
    public $errtype = array
```

```
(
    1=> “ERROR”,
    2=> “WARNING”,
    4    => “PARSE ERROR”,
    8    => “NOTICE”,
    16   => “CORE ERROR”,
    32   => “CORE WARNING”,
    64   => “COMPILE ERROR”,
    128  => “COMPILE WARNING”,
    256  => “USER ERROR”,
    512  => “USER WARNING”,
    1024 => “USER NOTICE”,
);

private $mostrarErrores = true;
private $loguearErrores = true;
private $archivoErrores = ‘/tmp/php_errores.log’;

public function __construct()
{      ´
$gestor = set_error_handler(array($this, ‘gestorErrores’));
error_reporting(E_ALL);
}

public function gestorErrores($errno, $errstr, $file, $line,    $context)
{
    $strErr  = “<pre>”;
    $strErr .= “-- ERROR “.$this->errtype[$errno].” --” .PHP_EOL;
    $strErr .= “FECHA: “ . date(“Y-m-d H:i:s”).PHP_EOL;
    $strErr .= “ARCHIVO: “ . $file .PHP_EOL;
    $strErr .= “LINEA: “ . $line . PHP_EOL;
    $strErr .= “IP SERVIDOR: “ .$_SERVER[‘SERVER_ADDR’].PHP_EOL;
    $strErr .= “IP USUARIO: “ .$_SERVER[‘REMOTE_ADDR’].PHP_EOL;
    $strErr .= “MENSAJE: “ . $errstr.PHP_EOL;
```

```
        $strErr .= “--ERROR “.$this->errtype[$errno].”--”.PHP_EOL;
        $strErr .= “</pre>”;



        if($this->loguearErrores)
        {
            if(is_writable($this->archivoErrores))
            {
                $logTxt=file_get_contents($this->          archivoErrores);
                $logTxt .= $strErr . PHP_EOL;
                file_put_contents($this-> archivoErrores, $logTxt);
            }
        }
        if($this->mostrarErrores)
        {
            echo $strErr;
        }
        else
        {
            echo “ERROR” .PHP_EOL;
        }
    }
}
?>
```

Solo se necesita instanciar la clase para que se haga el cambio de manejador global de errores por el nuestro, se active el muestreo total de errores, y se comiencen a mostrar con nuestro propio formato.

```
<?
    phprequire(‘classes/clsErrores.php’);
    $errores = new clsErrores();
?>
```

Mejoras de seguridad y buenas prácticas

Algunas sugerencias que se deben tener en cuenta al iniciar un proyecto y al trabajar con PHP son:

- **Utilizar programación orientada a objetos**

  Esto permite tener un código ordenado, separado, reutilizable y simple de mantener en el futuro.
- **Leer el manual oficia**

  PHP cuenta con documentación oficial disponible on-line en múltiples idiomas y con un anexo donde los usuarios comentan sus dudas o mejoras, agregan opciones y alternativas mejoradas del uso de cada función.
- **No comenzar con un framework, pero sí tenerlo en cuenta**

  Un framework es un ambiente en el cual se puede enmarcar un proyecto y provee un conjunto de clases y métodos que resuelven muchas necesidades. Todo está bajo el paradigma de la Programación Orientada a Objetos y en general adoptan el patrón de arquitectura MVC (Modelo Vista-Controlador).
- **Utilizar un IDE**

  IDE es un software en el cual se encuentran integradas varias herramientas para programar. Principales características:
  - Coloreo de sintaxis.
  - Los métodos se autocompletan.
  - Indexación de métodos de clases y documentación.
  - Posibilidad de integrarse con sistemas de versionado de código como SVN o GIT.
  - Control de errores y alertas en tiempo de edición.
- **No mostrar los errores**

  Es importante que nunca mostremos a los usuarios los errores del sitio, ya que además de dar una mala impresión, otorgan valiosa información sobre nuestro desarrollo.
- **Desactivar register_globals**

  A partir de la versión 5 de PHP viene desactivado por defecto, es importan-te no activarlo. Esta característica activada lo que permite es que el servidor Web considere las variables que se envían por GET, POST, COOKIES como variables globales.

- **Deshabilitar las funciones peligrosas que no necesitemos**
  Nos permi-te deshabilitar el uso de ciertas funciones que puedan exponer nuestro sistema, y es ideal en ambientes compartidos.
- **No permitir la apertura e inclusión de archivos remotos**
  A menos que sea estrictamente necesario para el funcionamiento de nuestro desarrollo no tiene sentido permitir que se incluyan o abran archivos por URL.
- **Limitar los archivos y directorios que se pueden abrir**
  Es posible limitar el directorio del cual se puede tomar archivos.
- **Aplicar límites al tiempo de ejecución de nuestros scripts, uso de memoria y tamaño de datos a enviar, y recibir por POST o subida de archivos**
  Nuestros proyectos pueden necesitar en mayor o menor medida modificar los límites establecidos por defecto por PHP. Esto puede ser personalizado de acuerdo con cada necesidad, pero es importante tener fijado algún límite.
- **No dejar un archivo con phpinfo() en nuestro servidor**
  Es fundamental recordar la eliminación de este archivo antes de dejar nuestro sitio Web en producción y listo para usar.
- **Permitir la apertura de código PHP con tags “cortos”**
  Lo ideal es que todos los scripts PHP se indiquen con su marca de apertura extendida y estándar: <?php, pero es muy común que se utilice la versión reducida <?.