

PHP es la tecnología del lado del servidor para hacer páginas web dinámicas, más utilizada hoy en día.

Al llevar tanto tiempo en el mercado, se ha creado una comunidad de programadores en PHP inmensa y esto supone una gran cantidad de posibilidades de cara al desarrollo web, que requeriría un curso 3 veces más grande a este.

Y nos preguntamos por qué estudiar tanto PHP si se puede utilizar CMS.

¿Qué es un CMS?

Content Managemen Sistem. Es un Gestor de contenido, una especie de paquete o aplicación creada en PHP, que te instala una tienda, un blog, una página web...un foro...por defecto, sin que tu tengas que picar ni un solo código.....NO NECESITAS PROGRAMAR, SOLO INSTALAR UN CMS

¿Por qué aprender entonces a programar si existen los CMS? **No es lo mismo instalar un CMS que aprender a programarlo.** Imagínate que quieres poner el blog en la parte derecha de tu página web a modo de un mini foro. También crear un blog de forma personalizada...los CMS no siempre se adaptan a tus necesidades.

CMS hoy en día hay muchos: WORDPRESS, DRUPAL, JOOMLA....Paquetes programados en PHP. Cuando quieras añadir una funcionalidad que no traen, entonces se programaría en php o se utiliza los plugin

Un plugin no es más que un paquete programado en PHP, que agrega una funcionalidad a tu web, ejemplo: que aparezca tu página en una búsqueda cuando el cliente hace una petición entonces se instalará un plugin SEO (SEO técnica que hay que seguir para que tu página quede lo mejor posicionada posible en los resultados de búsqueda); plugin de contadores de visitas...

Hay plugin para casi todo, sin embargo cuando no hay plugin habrá que programarlo.

Otro gran motivo de porque debemos aprender a programar en PHP, es por utilizar los frameworks (o entorno de trabajo, como por ejemplo LARAVEL

Instalar un CMS no lleva más de 15 minutos.(Internet está lleno de tutoriales de cómo instalar un CMS)

Es verdad que las mediana empresas piden sobretodo instalar un CMS, pero ojo!, en las pimes no en grandes empresas. Y siempre pagan más por programar una página web que por hacerla desde un CMS

Vamos a crear un blog para repasar todo lo más importante de php: funcionalidad, seguridad....

CREACION DE UN BLOG

1. Primero crear la base de datos (**BDDDBLOG**) en PHPMyAdmin, donde almacenar las entradas de ese blog:
 - a. Título
 - b. Fecha (día y hora)
 - c. Comentarios
 - d. Imagen
 - e. Id (identificador único)
2. Creamos la tabla **contenido**

Si quisiéramos modificar el nombre de la tabla , una vez creadas , DESDE PESTAÑA OPERACIONES

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra	Acción
1	Id	int(11)			No	Ninguna	AUTO_INCREMENT	Cambiar Eliminar Primaria Único Índice Espacial Texto completo Más
2	Título	varchar(255)	latin1_swedish_ci		No	Ninguna		Cambiar Eliminar Primaria Único Índice Espacial Texto completo Más
3	Fecha	datetime			No	Ninguna		Cambiar Eliminar Primaria Único Índice Espacial Texto completo Más
4	Comentario	text	latin1_swedish_ci		No	Ninguna		Cambiar Eliminar Primaria Único Índice Espacial Texto completo Más
5	Imagen	varchar(35)	latin1_swedish_ci		No	Ninguna		Cambiar Eliminar Primaria Único Índice Espacial Texto completo Más

3. Una vez creada la base de datos para ir almacenando las entradas, se crea un formulario HTML que nos permita introducir los registros.



4. Tenemos que tener en cuenta el nombre que vamos a tener en cada uno de los apartados:
 - a. Por ejemplo título tendrá el **name campo_titulo**
 - b. Comentario tiene el **name area_comentario**
 - c. A continuación tenemos un texto cuya única función es informar “seleccione una imagen con tamaño inferior a 2MB
 - d. Un botón de examinar que va a ser el botón que nos va a servir para subir la imagen que queremos subir a la base de datos. Llamada **name imagen**
 - e. Botón de enviar el formulario
 - f. Un enlace a el blog para ver si se ha introducido correctamente la entrada

debemos tener en cuenta que a la hora de crear una aplicación en PHP. Podemos crearla por procedimientos o Orientado a Objetos

vamos a hacerlo primero por procedimientos :

el formulario llama a un documento PHP llamado insertar_contenido.php

Esta página será encargada de recepcionar los datos que le vamos a enviar desde el formulario, conectar con la base de datos, e insertar esos datos dentro de la tabla que creamos en nuestra base de datos

Creamos pues el documento insertar_contenido

1. Primero creamos la conexión con la BBDD,
 - a. Creamos la variable a la que llamamos miconexion y le asignamos el método mysqli_connect
 - b. Comprobamos la conexión

```
<?php

$miconexion=mysqli_connect("localhost", "root", "", "bbddBlog");

/*Comprobar conexión*/

if(!$miconexion){

    echo "La conexión ha fallado: " . mysqli_error();

    exit();

}
```

```
if($_FILES['imagen']['error']
```

```
?>
```

- c. Evaluar si se ha subido correctamente la imagen. Utilizamos la función superglobal \$_FILES para subir imágenes al servidor. si miramos la página de php.net, veremos que esta función puede dar algunos errores: valor 0, valor 1(demasiado tamaño), valor 2, valor 3, valor 4 (que no se ha subido ningún archivo).... vamos a evaluar los errores

```
}else{

    echo "Entrada subida correctamente<br/>";

    if((isset($_FILES['imagen']['name']) && ($_FILES['imagen']['error']==UPLOAD_ERR_OK))){

        $destino_de_ruta="imagenes/"

    }
```

2. Debemos tener en cuenta que cuando creamos subimos una foto del servidor , primero pasa a un directorio temporal y luego a un directorio escogido por nosotros (destino_ruta)
3. Así que en la carpeta raíz del proyecto, creamos una carpeta más a la cual voy a llamar imágenes. De tal manera que las images que se suban con éxito a la base de datos sean movidas desde el directorio temporal del servidor a la carpeta que acabo de crear images para tener las imágenes aquí, y que luego se almacene la ruta en la base de datos y podamos buscar la imágenes , a través de su ruta, en esta carpeta
- 4.
5. Utilizamos luego la función **move_uploaded_file** para mover la imagen desde el directorio temporal del servidor, al que nosotros queremos y vendrá determinada por el campo imagen del formulario a través de la super variable S_FILES. Además S_FILES es un array asociativo que guarda cierta información es el directorio temporal de la imagen, y eso se representa mediante nombre de dentro del array tmp_name (nombre temporal), a continuación utilizamos la variable destino_de_ruta y lo concatenamos con el nombre que tiene esa imagen almacenada en la base de datos \$_file[image],[name}
6. Con esto hemos conseguido mover la imagen del directorio temporal a nuestra carpeta.

```
}  
  
if($_FILES['imagen']['error']){  
    switch ($_FILES['imagen']['error']){  
        case 1: //Error exceso de tamaño de archivo  
            echo "El tamaño del archivo supera el límite permitido por el servidor (param upload_max_size de php.ini)";  
            break;  
        case 2: //Error exceso de tamaño de archivo  
            echo "El tamaño del archivo supera el tamaño permitido por el formulario (post_max_size de php.ini)";  
            break;  
        case 3: //Error interrupción durante subida  
            echo "El envío del archivo se ha interrumpido durante la transmisión";  
            break;  
        case 4: //Error de no se ha enviado archivo  
            echo "El tamaño del archivo es nulo o no se ha enviado archivo";  
            break;  
    }  
}  
  
}else{
```

Y en caso de que no haya error.....

```
}else{

    echo "No hay error en la transferencia del archivo. <br/>";

    if((isset($_FILES['imagen']['name']) && ($_FILES['imagen']['error'] == UPLOAD_ERR_OK))){

        $destino_de_ruta='imagenes/';

        move_uploaded_file($_FILES['imagen']['tmp_name'], $destino_de_ruta . $_FILES['imagen']['name']);

        echo "El archivo " . $_FILES['imagen']['name'] . " Se ha copiado en el directorio de imágenes";

    }else{

        echo "El archivo no se ha copiado en el directorio de imágenes";

    }

}
```

importante!

La propiedad name permite poner el valor max_file_size para dar el máximo permitido a través de VALUE

Código:

```
<form action='script' method='post' enctype='multipart/form-data'>
  <input type='hidden' name='MAX_FILE_SIZE' value='1000000' />
  <input type='file' name='archivo' />
</form>
```

, además, valida el tamaño en PHP así:

Código PHP:

```
if( $_FILES['archivo']['size'] > 1000000 ) {
    echo "No se pueden subir archivos con pesos mayores a 1MB";
} else {
    // tu código.
}
```

¡aludos.

Y por último la consulta

```
$miconsulta="INSERT INTO contenido (Titulo, fecha, Comentario, Imagen) VALUES ('Titulo', 'Fecha', 'Comentario',
'Imagen')"; +
```

Solo nos quedaría introducir la instrucción insert into para introducir nuestra información en la base de datos

Y modificamos cada valor por el recogido en la base de datos:

```
}

$eltitulo=$_POST['campo_titulo'];

$lafecha=date("Y-m-d H:i:s");

$elcomentario=$_POST['area_comentarios'];

$laimagen=$_FILES['imagen']['name'];

$miconsulta="INSERT INTO contenido (Titulo, fecha, Comentario, Imagen) VALUES ('" . $eltitulo .
'" . $elcomentario . "', '" . $laimagen . "')";

$resultado=mysqli_query($miconexion, $miconsulta);

/* Cerramos conexión*/

mysqli_close($miconexion);
```

Hasta ahora hemos utilizado un estilo de programación por procedimientos:

- Hemos creado la página del formulario
- Y la página que rescata la información del formulario y la envía a la base de datos
- Ahora hay que crear la página del blog, llamada “mostrar blog”

```
27 $miconsulta="SELECT * FROM CONTENIDO ORDER BY FECHA DESC";
28
29
30 if($resultado=mysqli_query($miconexion, $miconsulta)){
31
32     while($registro=mysqli_fetch_assoc($resultado)){
33
34         echo "<h3>" . $registro['Titulo'] . "</h3>";
35
36         echo "<h4>" . $registro['Fecha'] . "</h4>";
37
38         echo "<div style='width:400px;'>" . $registro['Comentario'] . "</div><br/><br/>";
39
40         if($registro['Imagen']!=""){
41
42             echo "<img src='imagenes/" . $registro['Imagen'] . "' width='300px' />";
43
44         }
45
46         echo "<hr/>";
47
48     }
49
50 }
51
```

1. Primero conectamos esta página con la base de datos para rescatar la información que hay almacenada en ella (copio-pegó de la página que rescata la información)
2. Una vez conectada, creamos una instrucción SQL que rescate la información que hay almacenada mediante SELECT, y lo almaceno en la variable myconsulta
3. Ahora tenemos que ordenar los registros. Las entradas de un blog suelen estar ordenadas por fechas. Lo normal es que aparezcan de la más reciente a la más antigua. ORDER BY por defecto es ascendente, si queremos que sea al revés debemos incluir la cláusula DESC después de fecha, para que sea descendente
4. Una vez almacenado en la variable “miconsulta”, los registros, se trata de extraerlos : si la variable resultado es igual a mysqli_query de mi conexión y la consulta miconsulta. A partir de aquí recorreremos el array , mediante un array asociativo , que es más cómodo porque hacemos mención a los nombres que tenemos almacenados en el array (mysqli_fetch_array)

5. Resaltamos título con h3,, fecha con h4, metemos los comentarios en un div, con un estilo
6. Con la imagen, al no ser obligatoria incluirla, puede que aparezca una cruz roja como que no encuentra la imagen. Para evitar que salga esta cruz roja de imagen vacía o imagen rota, con un condicional que evalúe si tenemos imagen almacenada o no la tenemos

```
105
106     <a href="Formulario.php">Añadir nueva entrada</a>
107
108     <a href="Mostrar Blog.php">Ver blog</a>
109 </body>
110 </html>
```

Vamos a crear un blog utilizando un Estilo de programación orientada a objetos

Siempre que hablamos de programación orientada a objetos, lleva implícito modular un programa o aplicación. Consiste en crear módulos o partes en nuestro programa, como si fuera una especie de puzle, para que interactúen entre ellos, y consigan poner en funcionamiento la aplicación.

Siempre que hablamos de modular, debemos seguir el modelo MVC (modelo vista controlador).

Consiste en dividir nuestro programa en tres partes:

- MODELO Encargado de gestionar los datos
- VISTA Encargado de la interface de nuestra aplicación
- CONTROLADOR encargado de interactuar entre el MODELO Y VISTA

Comencemos....

Dentro de la capa VISTA, podemos crear 2 archivos:

- Formulario
- Mostrar blog

Dentro de la vista CONTROLADOR, podemos crear un archivo llamado TRANSACCIONES, que sirva de puente entre **formulario** y **mostrar blog**.

Dentro de capa MODELO podemos crear dos archivos:

- Uno encargado de **crear** entradas de blog
- Otro encargado de **manipular** esas entradas de blog

Es decir , lo que pretendo es que cada entrada de blog que está compuesta por un título, una fecha, unos comentarios y una imagen...todo eso es un objeto que se representa con el OBJETO BLOG, y por otra parte con otro archivo que es el de manejo de los objetos, vamos a extraer la información de cada uno de esos objetos blog

Todos ellos se unirán en un puzle que formarán una unidad de la aplicación



Vamos a llevar esto al código

1. Primero hacemos una carpeta raíz que contenga:
 - a. Una carpeta imágenes
 - b. Una carpeta MODELO. Compuesto por dos archivos:
 - i. Objeto blog
 - ii. Manipula objeto blog
 - c. Una carpeta VISTA. Compuesta de dos archivos:
 - i. Formulario. Que ahora no apunta al documento “insertar_contenido”, sino que apunta a un documento de la carpeta “controlador”.
 - ii. Mostrar blog
 - d. Una carpeta **.../controlador/transacciones.php**

Comenzamos con el archivo Objeto Blog

- Tenemos cinco elementos o propiedades del objeto blog:
 - Visibles: TÍTULO, IMAGEN, COMENTARIO,
 - No visibles: ID y FECHA
- Utilizamos la propiedad private para que se puedan encapsular y que solo puedan encapsularse en el mismo objeto
- Luego necesitamos los métodos de acceso: los getters y los setters.
 - Los setters establecen los valores de los cinco elementos....
 - Los getters son los encargados de mostrarnos los valores de dichas propiedades
- Un getter seria por ejemplo

```
public función getId(){  
}
```

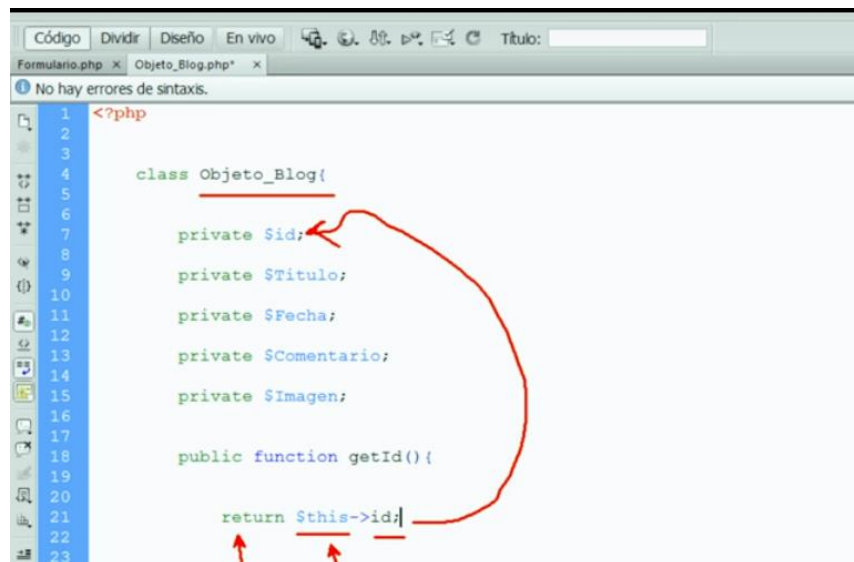

como el getter es el encargado de mostrarnos el valor de la propiedad, necesitará tener un return para que nos muestre la propiedad id. (Estamos diciendo, devuélveme el id de la entrada de blog donde me encuentro)

- Luego el setter

```
public function setId(){}
```

a este método le pasamos por parámetro un argumento (en el ejemplo \$id). Ese valor que pasemos por parámetro, es el que pasaremos a la propiedad ID. Y esto se pone diciendo this id es= al id que te paso por parámetro

- Con this id estoy haciendo referencia a la propiedad id del objeto
- Con el ID hacemos referencia al parámetro o argumento que le he pasado



```
<?php

class Objeto_Blog{

    private $id;
    private $Titulo;
    private $Fecha;
    private $Comentario;
    private $Imagen;

    public function getId(){

        return $this->id;
    }
}
```

Añadimos todos los getters y setters



```

    return $this->id;
}

public function setId($id){

    $this->id=$id;

}

public function getTitulo(){

    return $this->Titulo;

}

public function setTitulo($Titulo){

    $this->Titulo=$Titulo;

}

public function getComentario(){

    return $this->Comentario;

}

public function setComentario($Comentario){
}
```

Guardamos cambios en el archivo objeto blog

- El siguiente paso será crear el otro archivo de la capa MODELO: El encargado de manipular los datos , tanto de extraer la información de las entradas de blog, como de insertar nuevas entradas de blog en la base de datos
 - Archivo> Nuevo PHP> manejo_objetos.php
 - Creamos la clase que se va a encargar de hacer todas estas operaciones. En esta clase se necesitamos manipular los objetos creados en archivo objeto_blog. Por lo que tengo que meterlo en la clase manejo objetos, mediante include
 - Necesitamos crear una conexión con la base de datos. Como esta conexión la vamos a crear también en el archivo transacciones, lo que voy a hacer aquí es crear una variable encapsulada (private), a la que voy a llamar conexión.
 - Esta clase debe tener también un método constructor, que será el que primero se ejecuta cuando instanciamos esta clase, y que tendrá por objetivo establecer la conexión. Se podrá hacer así public function (el nombre de la clase) por qué se utiliza como nombre del constructor, el de la clase? Y no utilizo el nombre de la clase, __constructor hasta hace poco se permitía utilizar el nombre de la clase, como nombre del método constructor, sin embargo desde la última versión de php , se recomienda que no sean iguales, sino que se utilice la sintaxis constructor public function __construct (\$conexión **ojo! No confundir este parámetro o argumento con la variable**). Dentro de esta clase vamos a llamar a un método que se va a encargar de establecer la conexión. (This-> setConexion).
 - Nada más instanciar la clases se ejecuta el constructor. El constructor llama a este método que voy a crear ahora desde el que se establecerá la conexión. A este método les vamos a enviar un parámetro o argumento, que es a su vez el que hemos recibido con el constructor . Este método junto con el de método conexión PDO estaría establecida la conexión. En esta última asociamos la conexión con el argumento.
 - Ahora tenemos que crear los métodos correspondientes para obtener la información almacenada en la base de datos, es decir obtener las entradas de blog mediante un array asociativo y la instrucción SQL que nos permita ver las entradas de blog por fecha , y una instancia del objeto blog y luego ir almacenado cada objeto blog en el array(get contenidos por fechas) e insertar,
 - Creamos todos los métodos setters para ir acumulando la propiedad correspondiente
 - Asocio la matriz con el objeto blog. Y como quiero iniciar desde el valor del objeto contador , (0). Se lo doy.
 - Hasta ahora he almacenado un objeto con todas sus propiedades y todos sus métodos dentro de \$matriz
 - Después de haber almacenado este primer objeto, debemos incrementar la variable contador, para que a la siguiente vuelta de bucle, cuando esté evaluando el segundo registro de este array asociativo que nos ha devuelto la

consulta SQL, cree un segundo objeto, establezca las propiedades de este segundo objeto, que serán los datos que vengan de la consulta SQL, y almacene ese segundo en la posición 1. Y así vamos a ir almacenando en esa matriz los objetos. Tanto objetos blog como registros nos devuelva la consulta SQL. Return matriz, que es donde está almacenada toda la información.

- Con esto ya tendríamos terminado el método que nos permite obtener información que se encuentra almacenada en la base de datos.

```
1 <?php
2
3
4 include("Objeto_Blog.php");
5
6 class Manejo_Objetos{
7
8
9
10     private $conexion;
11
12     public function __construct($conexion){
13
14
15         $this->setConexion($conexion);
16
17
18     }
19
20
21
22     public function setConexion(PDO $conexion){
23
24         $this->conexion=$conexion;
25
26
27
28     }
29
30
31
32     public function getContenidoPorFecha(){
33
34
35         $matriz=array();
36
37         $contador=0;
38
39         $resultado=$this->conexion->query("SELECT * FROM CONTENIDO ORDER BY FECHA");
40
41
42         while($registro=$resultado->fetch(PDO::FETCH_ASSOC)){
43
44
45             $blog=new Objeto_Blog();
46
47             $blog->setId($registro["Id"]);
48
49
50
51
52         }
53
54
55
56 }
```

```

while($registro=$resultado->fetch(PDO::FETCH_ASSOC)){

    $blog=new Objeto_Blog();

    $blog->setId($registro["Id"]);
    $blog->setTitulo($registro["Titulo"]);
    $blog->setFecha($registro["Fecha"]);
    $blog->setComentario($registro["Comentario"]);
    $blog->setImagen($registro["Imagen"]);

    $matriz[$contador]=$blog;

    $contador++;

}

return $matriz;
}

```

- Ahora vamos a crear otro método el que se encarga de introducir la información en la base de datos y con el método `exec` ejecutamos la instrucción SQL.

```

return $matriz;
}

public function insertaContenido(Objeto_Blog $blog){

    $sql="INSERT INTO CONTENIDO (Titulo, Fecha, Comentario, Imagen) VALUES ('" . $blog->getTitulo() . "','" . $blog->getFecha() . "','" . $blog->getComentario() . "','" . $blog->getImagen() . "')";

    $this->conexion->exec($sql);

}

```

-

○

hasta aquí tenemos tres archivos:

- El formulario, que lo tenemos dentro de la VISTA, es a través del cual se van a insertar las entradas del blog
- Dos archivos en la parte de MODELO
 - archivo `OBJETO_BLOG` que se va a encargar de construir objetos de tipo blog, con su título, imagen, comentario... Aquí tenemos la clase blog, con sus setters (establece las propiedades de este objeto) y getters (permite obtener las propiedades)
 - archivo `MANEJO_OBJETOS` encargado de obtener y establecer dichas propiedades a través de los métodos correspondientes.

Nuestro formulario apuntaba, desde el atributo action, a el archivo transacciones. Este es el encargado de crear las transacciones entre el archivo formulario y los dos archivos de la capa MODELO,

Vamos a elaborar ahora el archivo transacciones

Funciones de este archivo:

- recoger la información recogida en los campos del formulario
- e interaccionando con MANEJO_OBJETOS, debe ser capaz de introducir la información en la base de datos. Recordemos que MANEJO_OBJETOS tiene un método llamado **Inserta_contenido** que pide por parámetros un objeto de tipo blog, y que se va a encargar con una sentencia SQL de tipo INSERT INTO, de insertar la información en la base de datos
- ¿Cómo ponemos en contacto a estos dos archivos?
 - a. Primero creamos el archivo php llamado transacciones y lo metemos dentro de la carpeta CONTROLADOR puesto que se va a encargar de interactuar entre la VISTA (formulario) y el MODELO (la base de datos)
 - b. El contenido de este archivo va a ser igual que el archivo insertar_contenido, creado cuando estuvimos programando el BLOG por procedimientos
 - i. Conexión con la base de datos
 - ii. Comprobar si la imagen subía correctamente, y sus posibles errores
 - iii. Movíamos esa imagen del directorio temporal al directorio imagen
 - iv. A continuación trabajábamos con la base de datos, rescatando la información de los campos del formulario , para luego hacer la consulta con SQL
- 2. Lo que varia es
 - a. La conexión que no se va a establecer de esta forma
 - b. la parte de establecer los valores que vienen del formulario

primero incluimos los dos archivos

- Cómo vamos a tener que empezar a crear **objetos_Blog** para determinar cómo van a ser nuestras entradas de blog, con la información que viene en el formulario,
- y también vamos a utilizar los objetos de **manejo-objetos** (puesto que tenemos la función inserta contenidos)

Lo importante será incluir ambos archivos

Establecemos conexión en modo Programación Orientada a Objeto

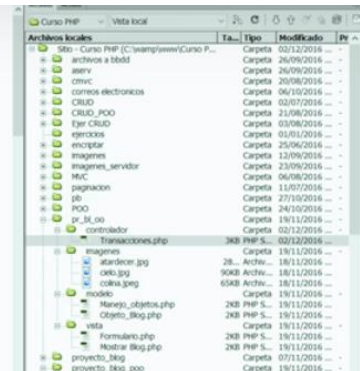
Una vez incluidos los dos archivos establecemos la conexión de este archivo transacciones con la base de datos mediante el **try**

- Creamos una variable `$miconexion`, y luego hacemos la conexión de siempre pero ahora utilizando la librería PDO, con la sintaxis POO
A continuación los atributos de esta conexión (set attribute)

```

1 <doctype html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>Documento sin título</title>
6 </head>
7
8 <body>
9
10
11 <?php
12
13 include_once("../modelo/Objeto_blog.php");
14
15 include_once("../modelo/Manejo_objetos.php");
16
17 try{
18
19     $miconexion=new PDO("mysql:host=localhost; dbname=bbdblog", "root", "");
20
21     $miconexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
22
23
24
25
26
27
28
29
30
31
32
33 }
34
35 }
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```



- Después del if-else se cierra el try

```

77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

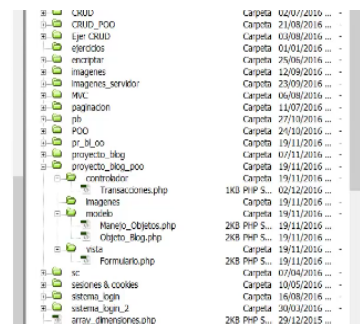


- Después se crea el catch, para programar lo que queremos que haga en caso de error.

```

67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```



- En último caso, tendría que rescatar del formulario, toda la información que el usuario ha ido introduciendo, y construir un objeto. Este objeto debe ser de tipo blog. Por esto hemos creado el archivo objeto-blog, con sus getters y sus setters:
¿cómo hacemos un objeto de los datos que ha introducido el usuario?
Antes del cierre de try, crearnos una instancia, un objeto, perteneciente a la clase `manejo-objetos`.

Luego creamos un objeto de tipo blog para poder acceder a todos los métodos y propiedades de la clase blog. (`new Manejo_objeto`) con esta instancia ya podemos crear todos los objetos pertenecientes a la clase `objeto_blog`, para ir estableciendo propiedades, por ejemplo para establecer el título, establecemos la función o método `setTitulo`, para el comentario `set comentario`.

así pues ponemos, nombre de instancia (\$blog->setTitulo(y debemos establecer la información que viene del campo título que será **campo_titulo**) para evitar la inyección (que permite acceder a la información de la base de datos), pondríamos el addslashes y el htmlentities que se introduce tanto en título como en comentario

```

    }

    $Manejo_Objetos=new Manejo_Objetos($miconexion);

    $blog=new Objeto_Blog();

    $blog->setTitulo(htmlentities(addslashes($_POST["campo_titulo"]), ENT_QUOTES));

    $blog->setFecha(Date("Y-m-d H:i:s"));

    $blog->setComentario(htmlentities(addslashes($_POST["area_comentarios"]), ENT_QUOTES));

    $blog->setImagen($_FILES["imagen"]["name"]);

} catch(Exception $e){

    die("Error: " . $e->getMessage());

}

```

- Y hasta aquí la creación del objeto
- Es decir lo que hemos conseguido es que la información que se ha introducido en el formulario + la fecha en concreto hemos construido el objeto del tipo blog
- El siguiente paso es coger ese objeto e insertarlo en la base de datos. Aquí entra en juego la función Inserta_contenido de la clase manejo_objetos. Esta clase, pide por parámetro un parámetro de tipo blog, entonces nos vamos a nuestro archivo transacciones y para poder acceder a ese método, lo único que podemos hacer es utilizar la instancia que hemos creado anteriormente de la clase manejo_objetos y el método inserta_contenido (y nos pide el objeto de tipo blog)

Y por último un echo que diga, entrada realizada con éxito!

```

$blog=new Objeto_Blog();

$blog->setTitulo(htmlentities(addslashes($_POST["campo_titulo"]), ENT_QUOTES));

$blog->setFecha(Date("Y-m-d H:i:s"));

$blog->setComentario(htmlentities(addslashes($_POST["area_comentarios"]), ENT_QUOTES));

$blog->setImagen($_FILES["imagen"]["name"]);

$Manejo_Objetos->insertaContenido($blog);

echo "<br/> Entrada de blog agregada con éxito <br/>";

} catch(Exception $e){

    die("Error: " . $e->getMessage());

}

```

- hasta aquí hemos conseguido que la información que el usuario introduce en el formulario y esa información pase a la base de datos
- Ahora nos queda rescatar esa información en la entrada de blog

Vamos a crear ahora ese archivo que se encarga de crear las entradas de blog en MODELO VISTA

En la clase manejo objetos tenemos dos funciones o métodos:

- inserta_contenidos, encargada de insertar contenidos en la base de datos
- get contenido por fecha, con una consulta SQL que rescataba o consultaba la información de la tabla contenido y la ordenaba por fecha

teniendo esto claro, creamos ahora el archivo PHP dentro de VISTA, y lo llamamos mostrar_blog

1. primero incluimos el archivo manejo_objetos puesto que vamos a utilizar una función que hay en él.
2. A continuación conectamos con la bbdd (con el try)
3. creamos el catch
4. ahora dentro del try creamos ese objeto perteneciente a la clase manejo_objetos. como el constructor de la clase manejo_objetos nos pide por parámetro la conexión por lo que pasamos también ahora la conexión \$miconexion

Archivo **mostrar_blog.php**

```
<body>
<?php
include("../modelo/Manejo_Objetos.php");

try{
    $miconexion=new PDO('mysql:host=localhost; dbname=bbddBlog', "root", "");
    $miconexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $manejo_objetos=new Manejo_Objetos($miconexion);
    $tabla_blog=$manejo_objetos->getContenidoPorFecha();

    if(empty($tabla_blog)){
        echo "No hay entradas de blog aún";
    }else{
        foreach($tabla_blog as $valor){

            echo "<h3>". $valor->getTitulo(). "</h3>";
            echo "<h4>". $valor->getFecha(). "</h4>";
            echo "<div style='width:400px'>";
            echo $valor->getComentario(). "</div>";
            if($valor->getImagen()!=""){
                echo "<img src='../imagenes/" . $valor->getImagen(). ".jpg' width='300px' height='200px'>";
            }
            echo "<br/>";
        }
    }
}

}catch(Exception $e){
    die("Error: ". $e->getMessage());
}
```

Y probamos todo


```

    }

    $Manejo_Objetos=new Manejo_Objetos($miconexion);
    $blog=new Objeto_Blog();
    $blog->setTitulo(htmlentities(addslashes($_POST["campo_titulo"]), ENT_QUOTES));
    $blog->setFecha(Date("Y-m-d H:i:s"));
    $blog->setComentario(htmlentities(addslashes($_POST["area_comentarios"]), ENT_QUOTES));
    $blog->setImagen($_FILES["imagen"]["name"]);

    }catch(Exception $e){

        die("Error: " . $e->getMessage());
    }
}

```

Y hasta aquí la creación del objeto

Es decir lo que hemos conseguido es que la información que se ha introducido en el formulario + la fecha en concreto hemos construido el objeto del tipo blog

- El siguiente paso es coger ese objeto e insertarlo en la base de datos. Aquí entra en juego la función Inserta_contenido de la clase manejo_objetos. Esta clase, pide por parámetro un parámetro de tipo blog, entonces nos vamos a nuestro archivo transacciones y para poder acceder a ese método, lo único que podemos hacer es utilizar la instancia que hemos creado anteriormente de la clase manejo_objetos y el método inserta_contenido (y nos pide el objeto de tipo blog)
- Y por último un echo que diga, entrada realizada con éxito!

```

$Manejo_Objetos=new Manejo_Objetos($miconexion);
$blog=new Objeto_Blog();
$blog->setTitulo(htmlentities(addslashes($_POST["campo_titulo"]), ENT_QUOTES));
$blog->setFecha(Date("Y-m-d H:i:s"));
$blog->setComentario(htmlentities(addslashes($_POST["area_comentarios"]), ENT_QUOTES));
$blog->setImagen($_FILES["imagen"]["name"]);

$Manejo_Objetos->insertaContenido($blog);

echo "<br/> Entrada de blog agregada con éxito |

}catch(Exception $e){

    die("Error: " . $e->getMessage());

}

```

hasta aquí hemos conseguido que la información que el usuario introduce en el formulario y esa información pase a la base de datos
Ahora nos queda rescatar esa información en la entrada de blog

