Métodos recurrentes

El método **set** asigna al objeto el valor de las variables por parámetro a las propiedades correspondientes. Es utilizado en todos los modelos.

El método **Insert** se comienza la conexión llamando al método **start**, con el método **prepare** con parámetro de una sentencia insert en la tabla correspondientes, con el primer campo nulo y el resto de los campos correspondiente recibidos en los parámetros, se llama el método **execute**, se prepara una consulta que devolverá el id máximo correspondiente de la tabla con **prepare** y se ejecuta. Se recuperan los datos de la consulta con **fetch,** se detiene la conexión y retorna el id correspondiente.

El método **Update** se llama **start**, se prepara la sentencia de Update en la tabla correspondiente se actualizan las columnas con los valores recibidos en los parámetros del método, post o get donde la clave primaria de la tabla sea igual al valor recibido en él parámetro, utilizando **prepare** y se ejecuta con **execute**, detiene la conexión con **stop**

El método **Agregar** llama el metodo **start** para iniciar la conexión, mediante post, get o parámetros recibe las variables que utilizara en una sentencia insert dentro de la tabla correspondiente, que es usada como parámetro de **prepare** y se ejecuta con **execute**, se llama a **stop** para detener la conexión.

El metodo **Eliminar** inicia la conexión con **start** mediante post, get o parámetros recibe la clave primaria de la tabla correspondiente con esta se prepara una sentencia delete con **prepare** y se llama a **execute**, se detiene la conexión con **stop.**

PDO

Se utiliza PDO(Objetos de Datos de PHP) para acceder a la base de datos.

El método **prepare** recibe como parámetro una sentencia y la prepara para su ejecución. Devuelve un objeto sentencia.

El método **execute** ejecuta la sentencia preparada. Devuelve true en caso de que se ejecutara correctamente o false en caso de un error.

El método **fetch** obtiene la siguiente fila del conjunto de resultados de la sentencia previamente ejecuta. Con el parámetro PDO::FETCH\_ASSOC devuelve un array indexado con los nombre de las columnas del conjunto de resultados.

El método **rowcount** devuelve el número de filas del conjunto de resultados de la última sentencia ejecutada.

Clase **DBConexion**

Esta clase se utiliza para realizar la conexión a la base de datos, se usara en el resto de las clases que interactúen con los datos utilizando pdo para realizar la conexión.

Se declaran las variables *host*, *base*, *user* y *pwd* como variable privadas estas corresponden al nombre del host y base de datos, al igual que el usuario y la contraseña con el que se realizara la conexión. Se declara la variable *charset* con el valor de “utf8” para que esta sea la decodificación. Se declara como variable protegida *pdo* y como variable publica *status.*

<?php

Class DBConexion {

private $host = "localhost";

private $base = "proyecto";

private $user = "root";

private $pwd = "";

private $charset = "utf8";

protected $pdo;

public $status;

En el método **start** dentro de un try se comienza la conexión creando una nueva instancia PDO con los parámetros que especifican el origen de la base de datos(variables declaradas en la parte superior) asignándola a la variable *pdo*. Después se utiliza el **setAttribute** como será manejada la base de datos, se devuelve un array indexado con el nombre de la columna y con el índice en el que fue devuelto el conjunto de resultados al terminar se asigna el valor de “202” a *status* y retorna true. En caso de que de error, en el catch se utiliza PDOException que asigna *status* el valor de “503” y retorna false.

protected function start(){

try {

$this->pdo = new PDO(

'mysql:host='.$this->host.';dbname='.$this->base.';charset='.$this->charset,

$this->user,

$this->pwd

);

$this->pdo->setAttribute(PDO::ATTR\_DEFAULT\_FETCH\_MODE, PDO::FETCH\_OBJ);

$this->status = 202;

return true;

} catch (PDOException $ex) {

$this->status = 503;

return false;

}

}

El metodo de **stop** le asigna a la instancia de *pdo* un valor nulo terminado la conexión.

protected function stop(){

$this->pdo = null;

}

}

?>

Clase **AmigosModelo**

Dentro de esta clase se declaran las variables con el mismo nombre de las columnas dentro de la tabla Amigo. Se utiliza para manejar los resultados de las consultas sobre la tabla de Amigos como un objeto. El método **set** se utiliza para establecer los valores dentro de las propiedades del objeto.

Dentro de la clase se declaran las variables *id\_amigos, estado*, *id\_usuario1* y *id\_usuario2* como variables públicas y el método **set**.

Class AmigosModelo{

public $id\_amigos;

public $estado;

public $id\_usuario1;

public $id\_usuario2;

Método **set**

El método **set** recibe como parámetros las variables *id\_amigos, estado, id\_usuario1* y *id\_usuario2*, es un método recurrente explicado en la parte superior.

public function set( $id\_amigos, $estado, $id\_usuario1, $id\_usuario2){

$this->id\_amigos = $id\_amigos;

$this->estado = $estado;

$this->id\_usuario1 = $id\_usuario1;

$this->id\_usuario2 = $id\_usuario2;

}

}

Clase **AmigosControlador**

Esta clase es utilizada para manejar toda la información que se relaciona con la tabla de Amigos. La clase se hereda de la clase **DBConexion** y contiene los métodos **ListaAmigos**, **Amigo**, **Estado**, **Solicitudes**, **Agregar**, **Eliminar**, **Aceptar** y **Amistad.**

Class AmigosControlador extends DBConexion

Metodo **ListaAmigos**

El método **ListaAmigos** utilizada para retornar una lista de objetos **AmigosModelo** de los amigos (solicitudes de amistad aceptadas) que tiene el usuario actual.

Primero se llama el método de **start** que inicia la conexión con la base de datos, a la variable *id\_usuario* se le asigna el valor de la variable de sesión *id\_usuario* Para saber el id del usuario actual*.* Con objeto stmt se llama a El método **prepare** que prepara la sentencia de la consulta a la tabla de amigosdonde el estado sea igual a 1 y el id\_usuario2 o id\_usuario1 sea igual a la variable *id\_usuario*, se llama a El método **execute** que ejecuta la sentencia preparada, se declara la variable *lista* como un array donde se guardara la lista de amigos del resultado de la consulta. Se abre un ciclo while que asigna la siguiente fila de stmt mediante El método **fetch** a la variable *Amigo*, dentro del ciclo se declara un objeto Amigos de la clase **AmigosModelo**, mediante El método **set** se le asignan los valores de *Amigo a* Amigos y este se agrega a la siguiente fila de *lista.*Se termina la conexión llamando a El método **stop** y El método retorna *lista.*

public function ListaAmigos(){

$this->start();

$id\_usuario = $\_SESSION['id\_usuario'];

$stmt = $this->pdo->prepare(

"SELECT \* FROM amigos where estado = 1 and (id\_usuario2 = $id\_usuario or id\_usuario1= $id\_usuario)"

);

$stmt->execute();

$lista = array();

while($Amigo = $stmt->fetch(PDO::FETCH\_ASSOC)):

$Amigos = new AmigosModelo();

$Amigos->set(

$Amigo["id\_amigos"],

$Amigo["estado"],

$Amigo["id\_usuario1"],

$Amigo["id\_usuario2"]

);

$lista[] = $Amigos;

endwhile;

$this->stop();

return $lista;

}

Metodo **Amigo**

El método **Amigo** es utilizada para comprobar si el usuario actual y otro usuario tienen una interacción de amistad registrada en la tabla de Amigos (son amigos o existe una solicitud de amistad pendiente).

Recibe como parámetro *id\_usuario2*, comienza la conexión y se asigna a la variable *id\_usuario1* el valor de la variable de sesión *id\_usuario*. Con objeto stmt se llama a el método **prepare** con el parámetro de la consulta a la tabla de amigos donde id\_usuario1 sea igual a *id\_usuario1* y id\_usuario2 sea igual a *id\_usuario2* o viceversa, se llama a el método **execute**. Se crea una instancia Amigos de **AmigosModelo** y se declara *Amigo* con el valor de stmt llamando el método **fetch**, mediante el método **set** se le asignan los valores de *Amigo a* Amigos. El método retorna Amigos.

public function Amigo($id\_usuario2){

$this->start();

$id\_usuario1 = $\_SESSION['id\_usuario'];

$stmt = $this->pdo->prepare(

"SELECT \* FROM amigos where ((id\_usuario1 = $id\_usuario1 and id\_usuario2 = $id\_usuario2) or (id\_usuario1 = $id\_usuario2 and id\_usuario2 = $id\_usuario1))"

);

$stmt->execute();

$Amigo = $stmt->fetch(PDO::FETCH\_ASSOC);

$Amigos = new AmigosModelo();

$Amigos->set(

$Amigo["id\_amigos"],

$Amigo["estado"],

$Amigo["id\_usuario1"],

$Amigo["id\_usuario2"]

);

return $Amigos;

}

Metodo **Estado**

El método **Estado** sirve para que cuando el usuario actual entre al perfil de otra persona, compruebe la relación de amistad que tienen. Si no hay alguna, puede enviar una solicitud para que pueda enviar una, si hay una solicitud pendiente, la marca como pendiente, o si ya son amigos, también muestra esto.

El método recibe como parametro *id\_usuario2, s*e comienza la conexión con el método **start**, se le asigna a *id\_usuario1* el valor de la variable de sesión *id\_usuario.* Se llama a el método **prepare** con el parámetro de la consulta a la tabla de amigos donde estado sea igual a 1, id\_usuario1 sea igual a id\_usuario1 y id\_usuario2 sea igual a id\_usuario2 o id\_usuario1 sea igual a id\_usuario2 y id\_usuario2 sea igual a id\_usuario1, se llama a el método **execute**. En caso de que el método **rowCount** devuelva un numero mayor a 0 se detiene la conexión con el método **stop** y retorna un 1, de lo contrario se prepara la consulta sobre la tabla amigos donde estado sea igual a 0 y id\_usuario1 sea igual a *id\_usuario2* y id\_usuario2 sea igual a *id\_usuario1* y se ejecuta, se comprueba que el numero de resultados de la consulta sea mayor a 0 con el método **rowCount** y se detiene la conexión con el método **stop** y retorna un 2 si el resultado es 0 se realiza la consulta en la table de amigos donde estado sea 0, id\_usuario1 sea igual a *id\_usuario1* y id\_usuario2 sea igual a *id\_usuario2* se comprueba el numero de resultados con *el método* **rowCount** y si es mayor a 0 se detiene la conexión y devuelve un 3 de lo contrario se detiene la conexión y devuelve un 0.

public function Estado($id\_usuario2){

$this->start();

$id\_usuario1 = $\_SESSION['id\_usuario'];

$stmt = $this->pdo->prepare(

"SELECT \* FROM amigos where estado = 1 and ((id\_usuario1 = $id\_usuario1 and id\_usuario2 = $id\_usuario2) or (id\_usuario1 = $id\_usuario2 and id\_usuario2 = $id\_usuario1))"

);

$stmt->execute();

if($stmt->rowCount() > 0){

$this->stop();

return 1;

}else{

$stmt = $this->pdo->prepare(

"SELECT \* FROM amigos where estado = 0 and id\_usuario1 = $id\_usuario2 and id\_usuario2 = $id\_usuario1"

);

$stmt->execute();

if($stmt->rowCount() > 0){

$this->stop();

return 2;

}else{

$stmt = $this->pdo->prepare(

"SELECT \* FROM amigos where estado = 0 and id\_usuario1 = $id\_usuario1 and id\_usuario2 = $id\_usuario2"

);

$stmt->execute();

if($stmt->rowCount() > 0){

$this->stop();

return 3;

}else{

$this->stop();

return 0;

}

}

}

}

Método **Solicitudes**

La método de **Solicitudes** se utiliza para que saber cuáles son las solicitudes de amistad que tiene el usuario que está utilizando la página en ese momento.

Se comienza la conexión y se le asigna el valor de la variable de sesión *id\_usuario* a *id\_usuario* se utiliza el método **prepare** con la consulta en la tabla de amigos donde estado sea igual a 0 y id\_usuario sea igual a *id\_usuario*, se ejecuta la consulta con el método **execute**. Se declara *lista* como array, dentro de un while que mediante el método **fetch** de stmt asigne la siguiente fila de resultados a *Amigo*se crea una instancia de **AmigosModelo** como Amigos, se establecen como parámetros del método **set** los valores de *Amigo, después se asigna Amigos* al siguiente lugar en *lista.* Se detiene la conexión y devuelve *lista.*

public function Solicitudes(){

$this->start();

$id\_usuario = $\_SESSION['id\_usuario'];

$stmt = $this->pdo->prepare(

"SELECT \* FROM amigos where estado = 0 and id\_usuario2 = $id\_usuario"

);

$stmt->execute();

$lista = array();

while($Amigo = $stmt->fetch(PDO::FETCH\_ASSOC)):

$Amigos = new AmigosModelo();

$Amigos->set(

$Amigo["id\_amigos"],

$Amigo["estado"],

$Amigo["id\_usuario1"],

$Amigo["id\_usuario2"]

);

$lista[] = $Amigos;

endwhile;

$this->stop();

return $lista;

}

Método **Agregar**

El método **Agregar** inserta una nueva relación de amistad en la tabla de Amigos cuando un usuario envía una solicitud de amistad a otro usuario.

Con el método start se inicia la conexión, se le asigna a *id\_usuario1* la variable de sesión *id\_usuario*, a *id\_usuario2* la variable get *id\_usuario.* Se prepara la sentencia de insert en la tabla de amigos los valores de *id\_usuario1* y *id\_usuario2 y* se ejecuta.Se le asigna a Notiuna nueva instancia de**NotificacionesControlado**, se llama al método **Insert.** Se detiene la conexióncon el método **stop** y se redirige al archivo Control.php por get *id\_usuario2.*

public function Agregar(){

$this->start();

$id\_usuario1 = $\_SESSION['id\_usuario'];

$id\_usuario2 = $\_GET['id\_usuario'];

$stmt = $this->pdo->prepare(

"INSERT INTO amigos VALUES( NULL,0,$id\_usuario1,$id\_usuario2)"

);

$stmt->execute();

$Noti = new NotificacionesControlador();

$Noti->Insert(6,$id\_usuario1,$id\_usuario1,$id\_usuario2);

$this->stop();

header("Location: Control.php?c=Perfiles&a=Perfiles&id=$id\_usuario2");

} }

Método **Eliminar**

El método **Eliminar** se utiliza para que cuando un usuario quiera deshacer la relación de amistad que tiene con otro usuario.

Si inicia la conexión con **start**, el valor de *id\_usuario* recibido por get se le asigna a *id\_usuario* y el valor de *id* se le asigna a *id\_amigos.* Se prepara la sentencia de eliminar de la tabla de amigos donde id\_amigos sea igual a *id\_amigos* y se envía como parámetro en el método **prepare**. Se llama al método **stop** y se redirige al perfil del usuario mandando por get *id\_usuario.*

public function Eliminar(){

$this->start();

$id\_usuario = $\_GET['id\_usuario'];

$id\_amigos = $\_GET['id'];

$stmt = $this->pdo->prepare(

"DELETE FROM amigos where id\_amigos= $id\_amigos"

);

$stmt->execute();

$this->stop();

header("Location: Control.php?c=Perfiles&a=Perfiles&id=$id\_usuario");

}

Método **Aceptar**

El método **Aceptar** se utiliza cuando existe una solicitud de amistad de un usuario a otro y el segundo la acepte.

Después de iniciar la conexión se asigna a *id\_usuario* el valor de la variable get *id\_usuario* y a *id\_amigos* el valor de *id. S*e prepara la sentencia que actualiza la tabla de amigos cambiando estado a un 1 donde id\_amigos es igual a *id\_amigos*, se ejecuta la sentencia, se ejecuta la sentencia. Se crea un objeto **AmigosControlador** como amigo se llama el método **Amistad** con *id\_amigos* como parámetro, se le asigna a *Noti* una nueva instancia de**NotificacionesControlado**, se llama al método **Insert.** Por último se detiene la conexión y se redirige al archivo Control.php mandando por get *id\_usuario.*

public function Aceptar(){

$this->start();

$id\_usuario = $\_GET['id\_usuario'];

$id\_amigos = $\_GET['id'];

$stmt = $this->pdo->prepare(

"UPDATE amigos SET estado = 1 where id\_amigos= $id\_amigos"

);

$stmt->execute();

$amigo = new AmigosControlador();

$a = $amigo->Amistad($id\_amigos);

$Noti = new NotificacionesControlador();

$Noti->Insert(5,$a->id\_usuario2,$a->id\_usuario2,$a->id\_usuario1);

$this->stop();

header("Location: Control.php?c=Perfiles&a=Perfiles&id=$id\_usuario"); }

Método **Amistad**

Este metodo se utiliza para recuperar la información de la relación entre un usuario y otro con un id\_amistad espesifico.

Se recibe como parámetro *id\_amigos, s*e prepara la consulta sobre la tabla amigos donde id\_amigos sea igual a *id\_amigos* y se ejecuta, con**fetch** se recuperan los resultados y se asignan *Amigo*, se crea una instancia**AmigosModelo** como Amigos se le asigna el valor de los resultados de *Amigo.* Se detiene la conexión y retornaAmigos.

public function Amistad($id\_amigos){

$this->start();

$stmt = $this->pdo->prepare(

"SELECT \* FROM amigos where id\_amigos = $id\_amigos"

);

$stmt->execute();

$Amigo = $stmt->fetch(PDO::FETCH\_ASSOC);

$Amigos = new AmigosModelo();

$Amigos->set(

$Amigo["id\_amigos"],

$Amigo["estado"],

$Amigo["id\_usuario1"],

$Amigo["id\_usuario2"]

);

$this->stop();

return $Amigos;

}

Clase **ArtistaModelo**

En esta clase se declaran las variables correspondientes a las columnas de la tabla artista. Se utiliza para guardar los resultados de las consultas sobre la tabla artista como un objeto. Con el método set se establecen las propedades del objeto.

Se tienen las variables *id\_artita, información\_contacto, técnica\_interes, id\_usuario, id\_diseno, id\_portafolio* y *id\_perfi,* y el metodo **set***.*

Class ArtistaModelo {

public $id\_artista;

public $informacion\_contacto;

public $tecnica\_interes;

public $id\_usuario;

public $id\_diseno;

public $id\_portafolio;

public $id\_perfil;

Método **set**

El método **set** recibe como parámetros las variables *id\_artita, información\_contacto, técnica\_interes, id\_usuario, id\_diseno, id\_portafolio* y *id\_perfil,* es un método recurrente explicado en la parte superior.

public function set( $id\_artista, $fn, $imagen\_perfil, $informacion\_contacto, $tecnica\_interes, $id\_pais, $id\_usuario, $id\_diseno, $id\_portafolio, $id\_perfil){

$this->id\_artista = $id\_artista;

$this->fn = $fn;

$this->imagen\_perfil = $imagen\_perfil;

$this->informacion\_contacto = $informacion\_contacto;

$this->tecnica\_interes = $tecnica\_interes;

$this->id\_pais = $id\_pais;

$this->id\_usuario = $id\_usuario;

$this->id\_diseno = $id\_diseno;

$this->id\_portafolio = $id\_portafolio;

$this->id\_perfil = $id\_perfil;

}

}

Clase **ArtistaControlador**

Esta clase se utiliza para manejar la información que entre y sal de la tabla de Artista, es una clase heredad de **DBConexion** y contiene los métodos **Artista**, **ArtistaUsuario, Insert** y **Update.**

class ArtistaControlador extends DBConexion

Método **Artista**

El método artista se utiliza para recuperar la información de la tabla artista mediante el id\_artista.

Se inicia la conexión, se prepara la consulta sobre la tabla de artista donde id\_artista sea igual a *id\_artista*  y se ejecuta. Mediante el método **fetch** se recupera los resultados de la consulta anterior y se asignan a *Artista,* se crea el objeto Artistas de la clase **ArtistaModelo**, se mandan como parámetros los valores de *Artista* al método **set**. Se detiene la conexión y retorna Artistas.

public function Artista($id\_artista){

$this->start();

$stmt = $this->pdo->prepare(

"SELECT \* FROM artista where id\_artista = $id\_artista"

);

$stmt->execute();

$Artista = $stmt->fetch(PDO::FETCH\_ASSOC);

$Artistas = new ArtistaModelo();

$Artistas->set(

$Artista["id\_artista"],

$Artista["informacion\_contacto"],

$Artista["tecnica\_interes"],

$Artista["id\_usuario"],

$Artista["id\_diseno"],

$Artista["id\_portafolio"],

$Artista["id\_perfil"]

);

$this->stop();

return $Artistas;

}

Método **ArtistaUsuario**

Este método se utiliza para recuperar la información de un usuario artista mediante el id\_usuario del artista.

Recibe como parámetro *id\_usuario, realiza la misma acción que el método de* **Artista** con la diferencia de que utiliza *id\_usuario* en lugar de *id\_artista.*

public function ArtistaUsuario($id\_usuario){

$this->start();

$stmt = $this->pdo->prepare(

"SELECT \* FROM artista where id\_usuario = $id\_usuario"

);

$stmt->execute();

$Artistas = new ArtistaModelo();

$Artista = $stmt->fetch(PDO::FETCH\_ASSOC);

$Artistas->set(

$Artista["id\_artista"],

$Artista["informacion\_contacto"],

$Artista["tecnica\_interes"],

$Artista["id\_usuario"],

$Artista["id\_diseno"],

$Artista["id\_portafolio"],

$Artista["id\_perfil"]

);

$this->stop();

return $Artistas;

}

Método **Insert**

El método se utiliza para agregar los datos del usuario artista, cuando se está registrando, a la tabla de artista, y devolver *id\_artista* que se acaba de insertar en la tabla.

El método **Insert** recibe los parámetros de *informacion, tecnica, id\_usuario, id\_diseno, id\_portafolo* y *id\_perfil.* Es un metodo recurrente explicado en la parte superiro que se aplica a la table de artista retornando *id\_artista.*

public function Insert($infomracion, $tecnica, $id\_usuario, $id\_diseno, $id\_portafolio, $id\_perfil){

$this->start();

$stmt = $this->pdo->prepare(

"INSERT into artista VALUES(NULL,'$informacion', '$tecnica', $id\_usuario, $id\_diseno,$id\_portafolio,$id\_perfil)"

);

$stmt->execute();

$stmt = $this->pdo->prepare(

"SELECT MAX(id\_artista) as id FROM artista"

);

$stmt->execute();

$artista = $stmt->fetch(PDO::FETCH\_ASSOC);

$this->stop();

return $artista["id\_artista"];

}

Método **Update**

Es utilizado cuando el usuario artista actualice los datos que estén dentro de la tabla artista.

Recibe los parámetros *información, técnica, id\_artista*, el método **Update** esta explicado en la parte superior en este caso se aplica a la tabla artista.

public function Update($informacion,$tecnica,$id\_artista){

$this->start();

$stmt = $this->pdo->prepare(

"UPDATE artista set informacion\_contacto ='$informacion', tecnica\_interes = '$tecnica' where id\_artista = $id\_artista"

);

$stmt->execute();

$this->stop();

}

Clase **BloqueadosModelo**

Dentro de la clase se declaran las variables con el nombre las columnas de tabla bloqueados. Es utilizada para manejar los resultados sobre las consultas de la tabla de bloqueados.

Se tienen las variables *id\_bloqueado, id\_usuario, inicio, fin* y *expirado* como públicas y el método **set.**

class BloqueadosModelo{

public $id\_bloqueado;

public $id\_usuario;

public $inicio;

public $fin;

public $expirado;

Método **set**

Recibe los prametros de *id\_bloqueado, id\_usuario, inicio, fin* y *expirado*.El metodo **set** es explicado en la parte de métodos recurrentes.

public function set($id\_bloqueado, $id\_usuario, $inicio, $fin, $expirado){

$this->id\_bloqueado = $id\_bloqueado;

$this->id\_usuario = $id\_usuario;

$this->inicio = $inicio;

$this->fin = $fin;

$this->expirado = $expirado;

}

Clase **BuscarControlador**

La clase es utilizada para realizar las funciones de buscar en el sistema. La clase es heredad de la clase **DBConexion** y contiene los métodos **Buscar, BuscarUsuario** y **BurcarPublicacion**.

Class BuscarControlador extends DBConexion{

Método **Buscar**

El método **Buscar** es utilizado en relación con la vista no cumple ninguna función aparte.

public function Buscar(){}

Método **BuscarUsuario**

Es utilizado para buscar entre los nombres de usuario lo que el usuario ingresada en el buscador, retorna la lista de usuarios que salgan dentro de la búsqueda.

Recibe como parámetro *usuario,* inicia la conexión llamando a **start** y prepara la consulta en la table de usuario donde el nombre de usuario coincida con *usuario* y el tipo de usuario sea distinto a 3 con **prepare** y **execute**. Se declara *lista* como un array y mediante **fetch** se asigna en un while a *Usuario*la fila siguiente del resultado de la consulta, se crea Usuarios como objeto de la clase **UsuarioModelo** y se asignan los valores de *Usuario a* Usuarios llamando a **set** y este se agrega al siguiente lugar en *lista.*Se detiene la conexión y retorna *lista.*

public function BuscarUsuario($usuario){

$this->start();

$stmt = $this->pdo->prepare(

"SELECT \* from usuario where nombre\_usuario LIKE '%$usuario%' and tipo\_usuario != 3 "

);

$stmt->execute();

$lista = array();

while($Usuario = $stmt->fetch(PDO::FETCH\_ASSOC)){

$Usuarios = new UsuarioModelo();

$Usuarios->set(

$Usuario["id\_usuario"],

$Usuario["contrasena"],

$Usuario["correo"],

$Usuario["nombre\_usuario"],

$Usuario["imagen\_perfil"],

$Usuario["fn"],

$Usuario["pais"],

$Usuario["estado"],

$Usuario["bloqueado"],

$Usuario["tipo\_usuario"],

$Usuario["permitir\_18"],

$Usuario["reset"],

$Usuario["auto5"],

$Usuario["auto10"],

$Usuario["auto15"],

$Usuario["auto20"]

);

$lista[] = $Usuarios;

}

$this->stop();

return $lista;

}

Método **BuscarPublicaciones**

Es utilizado para buscar entra las etiquetas y contenido de las publicaciones lo que el usuario ingreso en el buscador, retorna la lista de publicación que este dentro de la búsqueda.

Recibe como parámetro *palabra*, llama a **start,** se crea Usuarios como objeto de **UsuarioControlador** y se llama al m[etodo **Usuario** con la variable de sesión *id\_usuario* como parámetro, se comprueba si el usuario tiene activado el contenido explicito. Se realiza la consulta sobre la tabla de publicación donde ocultar sea 0, contenido\_explicito esté de acuerdo al usuario, privacidad sea 0 y contenido o etiquetas coincida con la *palabra* usando los metodos **prepara** y **execute.** Se declara *lista* como un array y mediante **fetch** dentro de un while se le asigna a *Publicacion*el valor de la fila siguiente del resultado de la consulta, se crea Publicaciones como objeto de la clase **PublicacionModelo** y se asignan los valores de *Publicacion a* Publicacion llamando a **set** y este se agrega al siguiente lugar en *lista.*Se detiene la conexión y retorna *lista.*

public function BuscarPublicacion($palabra){

$this->start();

$Usuarios = new UsuarioControlador();

$u = $Usuarios->Usuario($\_SESSION['id\_usuario']);

if($u->permitir\_18==1){

$c = "0,1";

}else{

$c = "0";

}

$stmt = $this->pdo->prepare(

"SELECT \* FROM publicacion where ocultar = 0 and contenido\_explicito in ($c) and privacidad = 0 and (contenido LIKE '%$palabra%' or etiquetas LIKE '%$palabra%') order by id\_publicacion DESC"

);

$stmt->execute();

$lista = array();

while($Publicacion = $stmt->fetch(PDO::FETCH\_ASSOC)){

$Publicaciones = new PublicacionModelo();

$Publicaciones->set(

$Publicacion["id\_publicacion"],

$Publicacion["fecha"],

$Publicacion["contenido\_explicito"],

$Publicacion["contenido"],

$Publicacion["etiquetas"],

$Publicacion["privacidad"],

$Publicacion["imagen"],

$Publicacion["id\_artista"],

$Publicacion["ocultar"]

);

$lista[] = $Publicaciones;

}

$this->stop();

return $lista;

}

Clase **ColeccionModelo**

La clase contiene las variables correspondientes al nombre de las columnas de la table colección*.* Se utiliza para manejar los resultados sobre las consultas como un objeto.

Se tienen las variables *id\_coleccion, descripción* y *id\_portafolios* como publicas al igual que el método ***set.***

Class ColeccionModelo{

public $id\_coleccion;

public $descripcion;

public $id\_portafolio;

Método  **set**

Recibe como parametros *id\_coleccion, descripción* y *id\_portafolios.* **Set** es explicado como método recurrente.

public function set( $id\_coleccion, $descripcion, $id\_portafolio){

$this->id\_coleccion = $id\_coleccion;

$this->descripcion = $descripcion;

$this->id\_portafolio = $id\_cportafolio;

}

Clase **ColeccionControlador**

La clasese utiliza parael manejo de datos de la tabla de colección e imagen\_coleccion. Se hereda de la **DBConexion** y contiene los métodos **Coleccions, ImagenesColeccion, AgregarColeccion, AgregarImgaen, EliminarImagen, EliminarColeccion, UpdateColeccion.**

Class ColeccionControlador extends DBConexion{

Método **Colecciones**

Este método se utiliza para recuperar las colecciones del usuario artista mediante el portafolio del artista

Recibe como parámetro *id\_portafolio,* llama al metodo **start** y realiza la consulta a la tabla de colección donde id\_portafolio sea igual *id\_portafolio* preparándola con **prepare** y ejecutando la consulta con **execute.** Se declara lista como un array. En un while se asigna a *c* los resultados de la consulta anterior usando **fetch,** dentrose declaraColecciones como objeto de la clase **ColeccionModelo**, se le asigna los resultado de *c* a Colecciones utilizando **set,** se agrega Colecciónes al ultimo lugar de *lista*, se detiene la conexión y se devuelve *lista.*

public function Colecciones($id\_portafolio){

$this->start();

$stmt = $this->pdo->prepare(

"SELECT \* FROM coleccion where id\_portafolio = $id\_portafolio"

);

$stmt->execute();

$lista = array();

while($c= $stmt->fetch(PDO::FETCH\_ASSOC)){

$Colecciones = new ColeccionModelo;

$Colecciones->set(

$c['id\_coleccion'],

$c['descripcion'],

$c['id\_portafolio']

);

$lista[] = $Coleccion;

}

$this->stop();

return $lista;

}

Método **ImagenesColeccion**

El método se utiliza para devolver una lista de las imágenes que pertenecen a una colección de un usuario artista.

El parámetro que recibe es *id\_coleccion*, se comienza la conexión con **start** y se prepara la consulta sobre la tabla imagen\_coleccion donde id\_colecion sea igual *id\_coleccion* con **prepare** y se llama a **execute.** Se declara *lista*  como array. Mientras exitan resultados de la consulta anterior se le asignan a *c* y estos se usan como parametros en el metodo **set** del objeto Colecciones de la clase **ImagenColeccionModelo**  y este se agrega a *lista*, se llama **stop** y se retorna *lista*

public function ImagenesColeccion($id\_coleccion){

$this->start();

$stmt = $this->pdo->prepare(

"SELECT \* FROM imagen\_coleccion where id\_coleccion = $id\_coleccion"

);

$stmt->execute();

$lista = array();

while($c= $stmt->fetch(PDO::FETCH\_ASSOC)){

$Colecciones = new ImagenColeccionModelo;

$Colecciones->set(

$c['id\_imagencoleccion'],

$c['imagen'],

$c['id\_coleccion']

);

$lista[] = $Colecciones;

}

$this->stop();

return $lista;

}

Metodo **AgregarColeccion**

Este método se utiliza cuando el usuario decide agregar una nueva colección a su portafolio, crea la colección sin imágenes.

Recibe mediante post *descripción* y *id\_portafolio*, el metodo **AgregarColeccion** se explica como metodo **Agregar** en los metodos recurrentes.

public function AgregarColeccion(){

$this->start();

$descripcion = $\_POST['descripcion'];

$id\_portafolio = $\_POST['id\_portafolio'];

$stmt = $this->pdo->prepare(

"INSERT into coleccion VALUES(NULL,'$descripcion',$id\_portafolio)"

);

$stmt->execute();

$this->stop();

}

Metodo **AgregarImagen**

Este método se utiliza cuando el usuario decide agregar imágenes a una colección dentro de su portafolio, pude agregar múltiples imágenes.

Se llama al método **start** para iniciar la conexión, mediant post se recibe *id\_coleccion* y se asigna a una variable con el mismo nombre. Se declara *total* con el valor del numero total de archivos subidos a *imagen* dentro de un for con valor máximo de *total* se mueve cada archivo recibido a la carpeta de ./Imágenes/imgColeccion/, y la dirección de la carpeta junto con el nombre del archivo es guardado en la variable *imagen* que junto con *id\_coleccion* es insertada en la tabla imagen\_coleccion utilizando los metodos **prepare** y **execute.**

public function AgregarImagen(){

$this->start();

$id\_colecion=$\_POST['id\_coleccion'];

$total = count($\_FILES["image"]["name"]);

for( $i=0 ; $i < $total ; $i++ ) {

$tmpFilePath = $\_FILES["image"]["tmp\_name"][$i];

$folder = "./Imagenes/imgColeccion/";

move\_uploaded\_file($tmpFilePath, $folder.$\_FILES["image"]["name"][$i]);

$imagen = $folder.$\_FILES["image"]["name"][$i];

$stmt = $this->pdo->prepare(

"INSERT into imagen\_coleccion VALUES(NULL,'$imagen',$id\_coleccion)"

);

$stmt->execute();

}

$this->stop();

}

Método **EliminarImagen**

Se utiliza para eliminar imágenes dentro de la colección del usuario artista.

Recibe mediante post *id\_imagen*y se le asigna a *id\_imagen.* El metodo **EliminarImagen**es explicado comometodo **Eliminar** en metodos recurrentes.

public function EliminarImagen(){

$this->start();

$id\_imagen =$\_POST['id\_imagen'];

$stmt = $this->pdo->prepare(

"DELETE FROM imagen\_coleccion where id\_imagencoleccion = $id\_imagen"

);

$stmt->execute();

$this->stop();

}

Método **EliminarColeccion**

Se utiliza cuando el usuario desea eliminar una colección completa. Elimina la colección y las imágenes dentro de la colección.

Recibe mediante get *id\_coleccion*y se le asigna a *id\_coleccion,* se prepara la sentencia delete en la tabla imagen\_coleccion donde id\_coleccion sea igual a *id\_coleccion,* se llama a **execute.** Se prepara la sentencia delete en la tabla colección donde id\_coleccion sea igual a *id\_coleccion* y se ejecuta.

public function EliminarColeccion(){

$this->start();

$id\_coleccion=$\_GET['id\_coleccion'];

$stmt = $this->pdo->prepare(

"DELETE FROM imagen\_coleccion where id\_coleccion = $id\_coleccion"

);

$stmt->execute();

$stmt = $this->pdo->prepare(

"DELETE FROM coleccion where id\_coleccion = $id\_coleccion"

);

$stmt->execute();

$this->stop();

}

Método **UpdateColeccion**

Se utiliza cuando el usuario quiere modificar la descripción de alguna de sus colecciones.

Se recibe mediante post *id\_coleccion* y se guarda en la variable *id\_coleccion,* al igual que se decibe *descripción* y se guarda en *descripción.* El metodo **UpdateColeccion** es explicado como **Update** en métodos recurrentes.

Clase **ComentarioModelo**

Esta clase contiene como variables los nombres de las columnas de la tabla comentario. Se utiliza para convertir los resultados de las consultas sobre esta tabla a objetos.

Se declaran públicas las variables *id\_comentario, fecha, contenido, id\_usuario, id\_pulicacion* y *ocultar* y el método **set.**

class ComentarioModelo{

public $id\_comentario;

public $fecha;

public $contenido;

public $id\_usuario;

public $id\_publicacion;

public $ocultar;

Metodo **set**

El metodo recibe como parametros *id\_comentario, fecha, contenido, id\_usuario, id\_pulicacion* y *ocultar*, se explica como metodo recurrente*.*

public function set( $id\_comentario, $fecha, $contenido, $id\_usuario, $id\_publicacion,$ocultar){

$this->id\_comentario = $id\_comentario;

$this->fecha = $fecha;

$this->contenido = $contenido;

$this->id\_usuario = $id\_usuario;

$this->id\_publicacion = $id\_publicacion;

$this->ocultar = $ocultar;

}

Clase **ComentarioControlador**

La clase es utilizad apara manejar la interacción de datos con la tabla de comentario.Es heredad de la clase **DBConexion** y contiene los cometidos **Comentarios, Comentario, Comentar, EliminarComentarioUsuario y EliminarComentario.**

class ComentarioControlador extends DBConexion

Metodo **Comentarios**

Se utiliza cuando se abre una publicación para ver los comentarios existentes dentro de la publicación.

Recibe como parámetros *id\_publicacion,* prepara la consulta sobre la tabla de comentario donde id\_publicacion sea igual *id\_publicacion* y se ejecuta. En un while se asignan los valores del resultado de la consulta obtenidos con **fetch** a *Comentario,* se crea un Comentarios como objeto de **ComentarioModelo** y se le asigna los valores de *Comentario* mediante **set**, y se agrega al array lista. Terminado el while se retorna *lista.*

public function Comentarios($id\_publicacion){

$this->start();

$stmt = $this->pdo->prepare(

"SELECT \* FROM comentario WHERE id\_publicacion = $id\_publicacion"

);

$stmt->execute();

$lista = array();

while($Comentario = $stmt->fetch(PDO::FETCH\_ASSOC)){

$Comentarios = new ComentarioModelo();

$Comentarios->set(

$Comentario["id\_comentario"],

$Comentario["fecha"],

$Comentario["contenido"],

$Comentario["id\_usuario"],

$Comentario["id\_publicacion"],

$Comentario["ocultar"]

);

$lista[] = $Comentarios;

}

$this->stop();

return $lista;

}

Metodo **Comentario**

Se utiliza para recuperar los datos de un comentario en específico usando su clave primaria.

Recibe como parametro, se inicia la conexión con **start**, se prepara una consulta en comentario donde id\_comentario es igual a *id\_comentario* se utiliza**prepare** y **execute** para realizar la consulta, se recuperan los resultado con **fetch** y se guardan dentro de *Comentario*y estos se guardan dentro de Comentarios con **set.** El metodo devuelve Comentarios.

public function Comentario($id\_comentario){

$this->start();

$stmt = $this->pdo->prepare(

"SELECT \* FROM comentario WHERE id\_comentario = $id\_comentario"

);

$stmt->execute();

$Comentario = $stmt->fetch(PDO::FETCH\_ASSOC);

$Comentarios = new ComentarioModelo();

$Comentarios->set(

$Comentario["id\_comentario"],

$Comentario["fecha"],

$Comentario["contenido"],

$Comentario["id\_usuario"],

$Comentario["id\_publicacion"],

$Comentario["ocultar"]

);

$this->stop();

return $Comentarios;

}

Método **Comentar**

Se utiliza cuando el usuario realiza un comentario en una publicación.

Se comienza la conexión con **start**, se guarda en *id\_publicacion* el valor de post *publicacionC* al igual que en en *comentario*  se guarda el valor de post *Comentario*, en *id\_usuario* se guarda el valor de la variable de sesión *id\_usuario*. Dentro de **prepare** se realiza la sentencia insert en la tabla de comentario con las variables en los campos correspondientes, se ejecuta la sentencia con **execute.** Se le asigna a Notiuna nueva instancia de**NotificacionesControlado** y a publicación una nueva instancia de **InicioControlador** se llama **PublicacionInfo** con el parametro de *id\_publicacion* y se guarda en *p*. Se crea ar como una nueva instancia de **ArtistaControlador** y llama al método de **Artista** con el parámetro *p->id\_artista* , se comprueba si el usuario actual no es el dueño de lapublicacion se llama al método **Insert.**

public function Comentar(){

$this->start();

$id\_publicacion = $\_POST['publicacionC'];

$comentario = $\_POST['Comentario'];

$id\_usuario = $\_SESSION['id\_usuario'];

$stmt = $this->pdo->prepare(

"INSERT into comentario values(NULL,NOW(),'$comentario',$id\_usuario,$id\_publicacion,0)"

);

$stmt->execute();

$Noti = new NotificacionesControlador();

$publicacion = new InicioControlador();

$p = $publicacion->PublicacionInfo($id\_publicacion);

$ar = new ArtistaControlador();

$a = $ar->Artista($p->id\_artista);

if($id\_usuario != $a->id\_usuario){

$Noti->Insert(2,$id\_usuario,$id\_publicacion,$a->id\_usuario);

}

$this->stop();

}

Método **EliminarComentarioUsuario**

Se utiliza cuando el usuario borra un comentario de una publicacion, pero no se borra de la tabla, cambia de estado ocultar para poder ser revisado por el moderador después.

Recibe por post *id\_c* y lo guarda en la variable *id\_comentario.* El metodo **EliminarComentarioUsuario** utiliza una sentencia Update que es explicado en el metodo recurrente **Update.**

public function ELiminarComentarioUsuario(){

$this->start();

$id\_comentario=$\_POST['id\_c'];

$stmt = $this->pdo->prepare(

"UPDATE comentario set ocultar = 1 WHERE id\_comentario = $id\_comentario"

);

$stmt->execute();

$this->stop();

}

Metodo **EliminarComentario**

Se utiliza independientemente de que el comentario fuera borrado por el usuario de la publicación, pueda ser borrado permanentemente la tabla comentario por el moderador.

El método recibe por post *id\_c* y guarda el valor en la variable *id\_comentario.* El método **EliminarComentario** es explicado como método **Eliminar** en métodos recurrentes.

public function EliminarComentario(){

$this->start();

$id\_comentario=$\_POST['id\_c'];

$stmt = $this->pdo->prepare(

"DELETE FROM reportes\_comentarios WHERE id\_comentario = $id\_comentario"

);

$stmt->execute();

$stmt = $this->pdo->prepare(

"DELETE FROM comentario WHERE id\_comentario = $id\_comentario"

);

$stmt->execute();

$this->stop();

}

Metodo **ValidarComentario**

Se utiliza cuando un usuario le da dislike a una publicación, comprueba si hay un comentario de este usuario en la publicación para permitirle o no dar dislike.

Se llama **start**, se le asigna a *id\_publicacion* el valor de la variable post *publicacionC* y a *id\_usuario* el valor de la variable de sesión *id\_usuario.* Se prepara y ejecuta la consulta en la tabla usuario donde id\_publicacion es igual a *id\_publicacion,* id\_usuario sea igual *id\_usuario* y ocultar sea igual a 0. Se comprueba con **rowCout** el numero de resultados si es mayor a 0 se imprime un 1, si no un 0.

public function ValidarComentario(){

$this->start();

$id\_publicacion = $\_POST['publicacionC'];

$id\_usuario = $\_SESSION['id\_usuario'];

$stmt = $this->pdo->prepare(

"SELECT \* FROM comentario WHERE id\_publicacion = $id\_publicacion AND id\_usuario = $id\_usuario AND ocultar = 0"

);

$stmt->execute();

if($stmt->rowCount() > 0){

$this->stop();

echo "1";

}else{

$this->stop();

echo "0";

}

}

Clase **ConversacionModelo**

En la clase declaran las variables que corresponden a las columnas dentro de la tabla de conversación. Se utiliza para manejar objetos con los resultados de las consultas en la tabla de conversación.

Se tienen las variables de *id\_conversacion, id\_usuario1* y *id\_usuario2* y el método **set¸** todo declarado como público.

Class ConversacionModelo{

public $id\_conversacion;

public $id\_usuario1;

public $id\_usuario2;

Método **set**

El método set establece las propiedades dentro de la clase. Recibe como parámetros *id\_conversacion, id\_usuario1* y *id\_usuario2* , como se explicó en métodos recurrentes, el método **set** asigna al objeto el valor de las variables por parámetro a las propiedades correspondientes.

public function set( $id\_conversacion, $id\_usuario1, $id\_usuario2){

$this->id\_conversacion = $id\_conversacion;

$this->id\_usuario1 = $id\_usuario1;

$this->id\_usuario2 = $id\_usuario2;

}

Método **Validar**

Se utiliza para comprobar si entre un usuario y otro existe una conversación en caso de que no exista se crea y retorna el id\_conversacion creado y si existe retorna el id\_conversacion.

Recibe como parámetro *id\_usuario2,* comienza la conexion y se le asigna a *id\_usuario1* el valor de la variable de sesión *id\_usuario****.*** Se prepara una consulta en la tabla conversación donde id\_usuario sea igual a *id\_usuario1* y id\_usuario2 sea igual a *id\_usuario2* o viceversa, se comprueban el numero de resultado llamando a **rowCount** si es mayor igual a cero, se aplica el método **Insert**  con las variables mencionadas, en caso de que sea distinto a cero se recuperan los datos con **fetch**, se guardan en *conversación* yretorna *id\_conversacion.*

public function Validar($id\_usuario2){

$this->start();

$id\_usuario1 = $\_SESSION["id\_usuario"];

$stmt = $this->pdo->prepare(

"SELECT \* FROM conversacion WHERE ((id\_usuario1 = $id\_usuario1 and id\_usuario2 = $id\_usuario2) or (id\_usuario1 = $id\_usuario2 and id\_usuario2 = $id\_usuario1))"

);

$stmt->execute();

if($stmt->rowCount() == 0){

$stmt = $this->pdo->prepare(

"INSERT INTO conversacion values(NULL,$id\_usuario1,$id\_usuario2)"

);

$stmt->execute();

$stmt = $this->pdo->prepare(

"SELECT MAX(id\_conversacion) as id FROM conversacion"

);

$stmt->execute();

$conversacion = $stmt->fetch(PDO::FETCH\_ASSOC);

$this->stop();

return $conversacion["id"];

}else{

$conversacion = $stmt->fetch(PDO::FETCH\_ASSOC);

$this->stop();

return $conversacion["id\_conversacion"];

}

}

Metodo **Conversaciones**

Se utiliza para saber que conversaciones tiene el usuario.

Incia la conexio llamndo a **start**, se asigna a *id\_usuario* el valor de la variable de sesión *id\_usuario.* Se prepara la consulta en la tabla de conversación donde *id\_usuario* sea igual a id\_usuario1 o id\_usuario2, se llama el método **execute** para ejecutar la sentencia. Se guardan los resultados en un array *lista* de objetos tipo **ConversacionModelo** y se retorna *lista.*

public function Conversaciones(){

$this->start();

$id\_usuario = $\_SESSION["id\_usuario"];

$stmt = $this->pdo->prepare(

"SELECT \* FROM conversacion WHERE id\_usuario1 = $id\_usuario or id\_usuario2 = $id\_usuario"

);

$stmt->execute();

$lista = array();

while($Conversacion = $stmt->fetch(PDO::FETCH\_ASSOC)){

$Conversaciones = new ConversacionModelo();

$Conversaciones->set(

$Conversacion["id\_conversacion"],

$Conversacion["id\_usuario1"],

$Conversacion["id\_usuario2"]

);

$lista[] = $Conversaciones;

}

$this->stop();

return $lista;

}

Clase **DisenoModelo**

La clase contiene variables con el nombre de las columnas de la tabla diseño, se utiliza para manejar los resultados de las consultas de esta tabla como objetos. Con el metodo **set** se establecen las propiedades del objeto.

Las variables son *id\_diseno, imagen\_fondo, color\_bordes, color\_titulo, color\_botones, color\_fondo* y *tipo\_perfil. El metodo* ***set*** *y las variables son públicas.*

*class DisenoModelo{*

*public $id\_diseno;*

*public $imagen\_fondo;*

*public $color\_bordes;*

*public $color\_titulos;*

*public $color\_botones;*

*public $color\_fondo;*

*public $tipo\_perfil;*

*Método* ***set***

*Recibe como parametros id\_diseno, imagen\_fondo, color\_bordes, color\_titulo, color\_botones, color\_fondo* y *tipo\_perfil.* El método **set** esta explicado en métodos recurrentes.

public function set( $id\_diseno, $imagen\_fondo, $color\_bordes, $color\_titulos,$color\_botones,$color\_fondo,$tipo\_perfil){

$this->id\_diseno = $id\_diseno;

$this->imagen\_fondo = $imagen\_fondo;

$this->color\_bordes = $color\_bordes;

$this->color\_titulos = $color\_titulos;

$this->color\_botones = $color\_botones;

$this->color\_fondo =$color\_fondo;

$this->tipo\_perfil = $tipo\_perfil;

}

Clase **DisenoControlado**

La clase se utiliza para manejar la información referente a la tabla de diseno. Es hereda de la clase **DBConexion** y contiene los métodos **Diseno, Insert** y **Update.**

Class DisenoControlador extends DBConexion{

Método **Diseno**

Se utiliza para recuperar los datos guardados en la tabla de diseño.

Como parametro tiene *id\_diseno,* inicia la conexión, prepara la consulta en la taba de diseño donde id\_diseno sea igual a *id\_diseno* y se ejecuta. Se guardas los seultado recuperado mediante **fetch** en *diseno* y estoy se mandan como parametro al metodo **set** de disenos instancia de **DisenoModelo**. El metodo retorna disenos.

public function Diseno($id\_diseno){

$this->start();

$stmt = $this->pdo->prepare(

"SELECT \* FROM diseno where id\_diseno = $id\_diseno"

);

$stmt->execute();

$disenos = new DisenoModelo;

$diseno = $stmt->fetch(PDO::FETCH\_ASSOC);

$disenos->set(

$diseno['id\_diseno'],

$diseno['imagen\_fondo'],

$diseno['color\_bordes'],

$diseno['color\_titulos'],

$diseno['color\_botones'],

$diseno['color\_fondo'],

$diseno['tipo\_perfil']

);

$this->stop();

return $disenos;

}

Método **Insert**

El método se utiliza cuando un usuario artista se esta registrando, el metodo guarda los datos que ingrese en la tabla de diseno.

Tiene los parámetros de *color\_bordes, color\_titulo, color\_botones, color\_fondo* y *tipo\_perfil*. El método **Insert** es explicado en métodos recurrentes.

public function Insert($color\_bordes,$color\_titulos,$color\_botones,$color\_fondo,$tipo\_perfil){

$this->start();

$stmt = $this->pdo->prepare(

"INSERT INTO diseno VALUES(NULL,NULL,'$color\_bordes','$color\_titulos','$color\_botones','$color\_fondo',$tipo\_perfil)"

);

$stmt->execute();

$stmt = $this->pdo->prepare(

"SELECT MAX(id\_diseno) as id FROM diseno"

);

$stmt->execute();

$diseno = $stmt->fetch(PDO::FETCH\_ASSOC);

$this->stop();

return $diseno["id"];

}

Metodo **Update**

Es utilizado cuando un usuario artista configura la apariencia de su perfil. Actualizando los colores, la imagen o el tipo de perfil.

Recibe como parametros *color\_bordes, color\_titulo, color\_botones, color\_fondo* , *tipo\_perfil* y *id\_diseno.* El metodo **Update** es explicado en la parte superior como metodo recurrente.

public function Update($color\_bordes, $color\_titulos, $color\_botones, $color\_fondo, $tipo\_perfil,$id\_diseno){

$this->start();

$stmt = $this->pdo->prepare(

"UPDATE diseno set color\_bordes = '$color\_bordes', color\_titulos ='$color\_titulos', color\_botones = '$color\_botones', color\_fondo = '$color\_fondo', tipo\_perfil = $tipo\_perfil where id\_diseno = $id\_diseno"

);

$stmt->execute();

$this->stop();

}

Clase **EstadoModelo**

Esta clase se utiliza para menejar los resultados de la tabla de estado como objetos por lo que tiene como variables las columnas de la respectiva table.

Contiene las variables de *id\_estado, id\_pais* y *estado*. Las variables y el metodos **set**son públicos.

Class EstadoModelo {

public $id\_estado;

public $id\_pais;

public $estado;

Metodo **set**

Recibe como parámetro *id\_estado, id\_pais* y *estado.* El método **set** esta explicado en la parte superior como método recurrente

public function set( $id\_estado, $id\_pais,$estado){

$this->id\_estado = $id\_estado;

$this->id\_pais = $id\_pais;

$this->estado = $estado;

}

Clase **FanModelo**

Se declaran las variables con los nombres de las columnas de la tabla fan. Es utilizada para manejar los resultados de las consultas realizadas en esta tabla como una instancia.

Tiene las variable de *id\_fan,información\_contacto, perfil* y *id\_usuario. Todas las variables y el metodo set son públicos.*

Class FanModelo{

public $id\_fan;

public $informacion\_contacto;

public $perfil;

public $id\_usuario;

Metodo **set**

Tiene como parámetros *id\_fan, información\_contacto, perfil* y *id\_usuario. El método* ***set*** *es explicado en métodos recurrentes.*

public function set( $id\_fan, $informacion\_contacto,$perfil,$id\_usuario){

$this->id\_fan = $id\_fan;

$this->informacion\_contacto = $informacion\_contacto;

$this->perfil =$perfil;

$this->id\_usuario = $id\_usuario;

}

}

Clase **FanControlador**

Es utilizada para la interacción de datos con la tabla Fan. Se hereda **DBConexion** y contiene los métodos **Insert, Update** y **Fan.**

Class FanControlador extends DBConexion

Método **Insert**

Se utiliza al momento de un usuario fan se esta registrando en el sistema, el método guarda en la tabla fan los datos registrados.

Recibe como parámetros *DatosFan, Perfil* y *id\_usuario.* El método **Insert** se aplica a la tabla de fan con la explicación de métodos recurrentes, retorna el valor de *id\_fan.*

public function Insert($DatosFan, $Perfil, $id\_usuario){

$this->start();

$stmt = $this->pdo->prepare(

"INSERT into fan VALUES(NULL,'$DatosFan','$Perfil',$id\_usuario)"

);

$stmt->execute();

$stmt = $this->pdo->prepare(

"SELECT MAX(id\_fan) as id FROM fan"

);

$stmt->execute();

$fan = $stmt->fetch(PDO::FETCH\_ASSOC);

$this->stop();

return $fan["id\_fan"];

}

Método **Update**

El método se utiliza cuando el usuario modifica sus datos en las configuraciones del sistema.

Recibe como parámetros *DatosFan* y *Perfil.*La explicación del método **Update** esta en métodos recurrentes.

public function Update($DatosFan, $Perfil ){

$this->start();

$id\_usuario = $\_SESSION['id\_usuario'];

$stmt = $this->pdo->prepare(

"UPDATE fan set informacion\_contacto ='$DatosFan',perfil = '$Perfil' where id\_usuario = $id\_usuario"

);

$stmt->execute();

$this->stop();

}

Método **Fab**

Se utiliza para recuperar la información específica de un usuario fan.

Recibe como parametro *id\_usuario,* realiza una consulta a la tabla fan donde id\_usuario sea igual a *id\_usuario,* se guardan los resultados de la consulta en *f* mediante **fetch**, se crea una intacia fan de **FanModelo** y se le asignan los valores de *f* mediante **set.** El método devuelve fan.

public function Fan($id\_usuario){

$this->start();

$stmt = $this->pdo->prepare(

"SELECT \* FROM fan where id\_usuario = $id\_usuario"

);

$stmt->execute();

$f = $stmt->fetch(PDO::FETCH\_ASSOC);

$fan = new FanModelo();

$fan->set(

$f['id\_fan'],

$f['informacion\_contacto'],

$f['perfil'],

$f['id\_usuario']

);

$this->stop();

return $fan;

}

Clase **ForoFavsModelo**

Se utiliza para manejar los resultados de las consultas que se hacen en la tabla de foro\_favs como un objeto de esta clase.

Se tiene las variables *id\_favs, id\_usuario* y *id\_forohilo como publicas al igual que el método* ***set.***

*class ForoFavsModelo{*

*public $id\_favs;*

*public $id\_usuario;*

*public $id\_forohilo;*

*Método* ***set***

*Recibe como parámetros id\_favs, id\_usuario* y *id\_forohilo*. El método **set** es el explicado en métodos recurrentes.

public function set($id\_favs, $id\_usuario, $id\_forohilo){

$this->id\_favs = $id\_favs;

$this->id\_usuario = $id\_usuario;

$this->id\_forohilo = $id\_forohilo;

}

}

Clase **ForoModelo**

Se tienen las variables correspondientes a cada una de las columnas de la tabla foro\_hilo. Se utiliza para generar objetos con las propiedades de esta tabla.

Se declaran las variables *id*\_forohilo*, fecha, contenido, titulo, id\_forotipo,id\_usuario* y *ocultar* como variables publicas junto con el método **set.**

class ForoModelo{

public $id\_forohilo;

public $fecha;

public $contenido;

public $titulo;

public $id\_forotipo;

public $id\_usuario;

public $ocultar;

Método **set**

Recibe como parámetros *id\_forohilo, fecha, contenido, titulo, id\_forotipo. Id\_usuario* y *ocultar.* Este método es explicado en métodos recurrentes.

public function set( $id\_forohilo, $fecha, $contenido, $titulo, $id\_forotipo, $id\_usuario, $ocultar){

$this->id\_forohilo = $id\_forohilo;

$this->fecha = $fecha;

$this->contenido = $contenido;

$this->titulo = $titulo;

$this->id\_forotipo = $id\_forotipo;

$this->id\_usuario = $id\_usuario;

$this->ocultar = $ocultar;

}

Clase **ForoControlador**

Se utiliza para manipular los datos de la tabla foro\_hilo y foro\_favs, Se hereda de la clase **DBConexion** y contiene los métodos **Foro, Hilo, HiloContenidos. NuevoHilo, ForoFavs, AgregarFavs, ConfirmarFavs, EliminarFavs, Foros, EliminarForoUsuario** y **EliminarForo.**

class ForoControlador extends DBConexion

{

Método **Foro**