Métodos Vacios

El MVC se maneja teniendo un método por vista, en caso de que no se necesite nada del método se crea un método vacio con el único propósito de abrir la vista.

Métodos recurrentes

En la mayoría de las clases existen métodos básicos que se utilizan repetitivamente que tienen la misma estructura cambiando las variables que reciben y la tabla a la que se aplica la sentencia sql, por lo que son explicados en este apartado como 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 método **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 método **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 usando el id recibido con **prepare** y se llama a **execute**, se detiene la conexión con **stop.**

El método **EliminarUsuario**, llama al método **start** mediante post, get o parámetros recibe la clave primaria de la tabla correspondiente con esta se prepara una sentencia Update que cambia la columna ocultar a 1 usando el id recibido, con **prepare** y se llama a **execute**, se detiene la conexión con **stop.**

El método **Lista** recibe por post, get o parámetros recibe un id para utilizar dentro de una consulta sobre la tabla correspondiente, Esta consulta se realiza utilicando **prepare** y **execute**. Se declara *lista* como array, utilizando **fetch** se recuperan los valores de la consulta y son guardados en una variable en un ciclo while, dentro se crea un objeto del modelo de la tabla correspondiente y se asignan los valores a esta utilizando **set**, el objeto es agregado a *lista*. Terminando el ciclo el método retorna *lista.*

PDO

Se utiliza PDO(Objetos de Datos de PHP) para acceder a la base de datos, por lo que los métodos PDO son usados repetitivamente en cada método.

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.*

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 tienen 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'];

//$id\_usuario2 = $\_GET['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"]

);

$this->stop();

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'];

//$id\_usuario2 = $\_GET['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**

El 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;

}

$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, $informacion\_contacto, $tecnica\_interes, $id\_usuario, $id\_diseno, $id\_portafolio, $id\_perfil){

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

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

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

$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[] = $Colecciones;

}

$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 cuando se requiere 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.

public function UpdateColeccion(){

$this->start();

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

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

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

"UPDATE coleccion set descripcion = $descripcion where id\_coleccion = $id\_coleccion"

);

$stmt->execute();

$this->stop();

}

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 publicación, 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** es explicado como **EliminarUsuario** en los métodos recurrentes,

        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;

}

Clase **ConversacionControlador**

Se utiliza para manejar los datos entrentes y salientes de la tabla de conversación en la base de datos, es heredada de la clase **DBConexion** y contiene los métodos **validar** y **conversaciones.**

Class ConversacionControlador extends DBConexion {

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 mostrar la lista de conversaciones que 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 utilizando id\_diseno como referencia.

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**

Este método es un método vacio que se usa en relación con la vista de Foro.

public function Foro(){   }

Método **Hilo**

El método es utilizado para recuperar los datos de un hilo en específico utilizando id\_forohilo recibido mediante un get.

Se llama al método **start.** Por medio de get se recibe el valor de *id* y se le asgina a *id,* con este valor se prepara la consulta sobre la tabla de foro\_hilo donde id\_forohilo sea igual sea igual a *id,* con **prepare** seguido de **execute**, se le asignan a *F* los resultados obtenidos con **fetch**, se crea Foro como instancia de **ForoModelo** y se le asignan los valores de *F* utilizando **set**. Se llama a **stop** y se detiene la conexión.

   public function Hilo(){

       $this->start();

       $id = $\_GET['id'];

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

"SELECT \* FROM foro\_hilo where id\_forohilo = $id "

);

        $stmt->execute();

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

$Foro = new ForoModelo();

$Foro->set(

$F['id\_forohilo'],

                        $F['fecha'],

                        $F['contenido'],

                        $F['titulo'],

                        $F['id\_forotipo'],

                        $F['id\_usuario'],

                        $F['ocultar']

             );

$this->stop();

return $Foro;

   }

Método **HiloContenido**

Se utiliza cuando los datos de un hilo en específico serán mostrados en la vista de hilo dentro de foro.

Este método realiza lo mismo que el método **Hilo** explicado anteriormente con la excepción de que recibe *id* como parámetro.

public function HiloContenido($id){

       $this->start();

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

"SELECT \* FROM foro\_hilo where id\_forohilo = $id"

);

        $stmt->execute();

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

$Foro = new ForoModelo();

$Foro->set(

$F['id\_forohilo'],

                        $F['fecha'],

                        $F['contenido'],

                        $F['titulo'],

                        $F['id\_forotipo'],

                        $F['id\_usuario'],

                        $F['ocultar']

             );

$this->stop();

return $Foro;

   }

Método **NuevoHilo**

Es utilizado cuando un usuario agrega un nuevo hilo, ya sea de tipo idea u objeto.

El método utiliza la extrucutra del método **Agregar** explicado en métodos recurrentes. Aplicado a la tabla foro\_hilo y recibiendo los datos de *descripción, titulo,* y *tipoF* por post. Terminando redirecciona al archivo Control.php?c=Foro&a=Foro.

   public function NuevoHilo(){

       if($\_SERVER['REQUEST\_METHOD']=='POST'){

       $this->start();

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

        $titulo = $\_POST['titulo'];

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

        $tipo = $\_POST['tipoF'];

        $stmt = $this->pdo->prepare(

"INSERT into foro\_hilo VALUES(NULL,NOW(),'$des','$titulo',$tipo, $id\_usuario,0)"

);

$stmt->execute();

        $this->stop();

        header("Location: Control.php?c=Foro&a=Foro");

            }

   }

Método **ForoFavs**

Se utiliza para mostrar en la vista de foro una lista de todos los foros que fueron agregados como favoritos por el usuario.

Recibe como parámetro *id\_usuario*, inicia la conexión llamando a **start** y prepara la consulta sobre la tabla foro\_favs donde id\_usuario sea igual a *id\_usuario* usándola como parámetro en **prepare** y llamando **ejecute***.* Se crea un array con el nombre *lista*, mientras que **fetch** devuelva una fila del resultado de la consulta anterior se guarda en *F* y se crea Foro como objeto de **ForoFavsModelo** guardando en este los valores de *F* como parámetro **set** se guarda Foro en el siguiente lugar dentro de *lista*. Se llama **stop** para detener la conexión y devulve el valor de *lista.*

   public function ForoFavs($id\_usuario){

            $this->start();

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

"SELECT \* FROM foro\_favs where id\_usuario = $id\_usuario ORDER BY id\_favs DESC"

);

        $stmt->execute();

             $lista = array();

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

$Foro = new ForoFavsModelo();

$Foro->set(

$F["id\_favs"],

$F["id\_usuario"],

$F["id\_forohilo"]

);

$lista[] = $Foro;

}

$this->stop();

return $lista;

    }

Método **AgregarFavs**

Se utiliza cuando un usuario agrega a favoritos un foro seleccionado.

El método es explicado como **Agregar** en métodos recurrentes es aplicado a la tabla de foro\_favs y recibe *id\_hilo* por get.

    public function AgregarFavs(){

            $this->start();

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

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

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

     "INSERT INTO foro\_favs VALUES (NULL, $id\_usuario, $id\_hilo)"

     );

         $stmt->execute();

    $this->stop();

    header("Location: Control.php?c=Foro&a=Hilo&id=".$id\_hilo);

    }

Método **ConfirmarFavs**

Es utilizado cuando un usuario entre a un hilo, confirma si el usuario ha marcado como favorito el hilo actual regresa un 1 si el usuario lo tiene como favorito y un 0 si no.

Recibe como parámetro *id\_hilo* y guarda en *id\_usuario* el valor de la variable de sesión *id\_usuario,*  con **prepare** y **ejecute** se realiza la consulta en la tabla foro\_favs donde id\_usuario sea igual a *id\_usuario* y id\_forohilo sea igual a *id\_hilo*. Si **rowCount** es mayor a 0 se deculve un 1, si no se devuelve un 0.

    public function ConfirmarFavs($id\_hilo){

        $this->start();

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

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

     "SELECT \* FROM foro\_favs WHERE id\_usuario = $id\_usuario and id\_forohilo = $id\_hilo"

     );

         $stmt->execute();

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

    $this->stop();

     return 1;

}else{

    $this->stop();

    return 0;

}

    }

Método **EliminarFavs**

Se utiliza cuando el usuario remueve un hilo de su lista de foros favoritos. Estando dentro del hilo.

Es explicado como un método **Eliminar** en la sección de métodos recurrentes, se aplica a la tabla foro\_favs y recibe mediante get el valor de *id\_forohilo*. Al terminar la sentencia redirige Control.php y mando por get *id\_forohilo.*

    public function EliminarFavs(){

        $this->start();

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

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

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

     "DELETE FROM foro\_favs where id\_usuario = $id\_usuario and id\_forohilo = $id\_forohilo"

     );

         $stmt->execute();

    $this->stop();

    header("Location: Control.php?c=Foro&a=Hilo&id=".$id\_forohilo);

    }

Método **Foros**

Retorna una lista de todos los foros existentes del tipo seleccionado que estén a la vista de los usuarios ordenados de manera descendiente comenzando por el ultimo foro publicado para ser mostrados en la vista de foro.

Recibe como parametro *id\_tipo*y comienza la conexión. Con prepare se realiza la consulta en la tabla de foro\_hilo donde id\_forotipo sea igual a *id\_tipo* y ocultar sea 0. Se llama a **execute** y con **fetch** se guardan los resultados en *F* dentro de un while, en Foro se guardan los valores utilizando el método **set** de la clase **ForoModelo**, sea agrega a *lista****,*** se detiene la conexión y se retorna *lista.*

    public function Foros($id\_tipo){

        $this->start();

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

"SELECT \* FROM foro\_hilo where id\_forotipo = $id\_tipo and ocultar = 0 ORDER BY id\_forohilo DESC"

);

        $stmt->execute();

             $lista = array();

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

$Foro = new ForoModelo();

$Foro->set(

$F['id\_forohilo'],

                        $F['fecha'],

                        $F['contenido'],

                        $F['titulo'],

                        $F['id\_forotipo'],

                        $F['id\_usuario'],

                        $F['ocultar']

             );

$lista[] = $Foro;

}

$this->stop();

return $lista;

    }

Método **ElimarForoUsuario**

Es utilizado cuando el usuario decide eliminar uno de los foros, permanece oculto para que el moderador aun pueda vesualizarrlo.

El método es explicado como **EliminarUsuario** en métodos recurrentes se aplica a la tabla foro\_hilo utilizando el valor recibido por get de *id\_f* guardado en *id\_forohilo,* con el excepción de que realiza una sentencia delete en la tabla de foro\_favs explicada como método **Eliminar**. Se redirige al archivo Control.php?c=Foro&a=Foro.

    public function EliminarForoUsuario(){

         $this->start();

            $id\_forohilo=$\_GET['id\_f'];

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

"UPDATE foro\_hilo SET ocultar = 1 WHERE id\_forohilo = $id\_forohilo"

);

$stmt->execute();

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

"DELETE FROM foro\_favs WHERE id\_forohilo = $id\_forohilo"

);

$stmt->execute();

$this->stop();

header("Location: Control.php?c=Foro&a=Foro");

    }

Método **EliminarForo**

Se utiliza cuando un moderador decide eliminar de la base de datos un foro, se elimina el foro, las respuestas y si es guardado como foro favorito por algún usuario.

Es explicado como el método recurrente **Eliminar** con la excepción de que realiza la sentencia en las tablas de foro\_respuesta, foro\_favs y foro\_hilo con el valor recibido por get en *id\_forohilo.*

public function ElimiarForo(){

        $this->start();

            $id\_forohilo=$\_GETT['id\_f'];

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

                "DELETE FROM foro\_respuesta WHERE id\_forohilo = $id\_forohilo"

            );

            $stmt->execute();

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

                "DELETE FROM foro\_favs WHERE id\_forohilo = $id\_forohilo"

            );

            $stmt->execute();

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

                "DELETE FROM foro\_hilo WHERE id\_forohilo = $id\_forohilo"

            );

            $stmt->execute();

$this->stop();

    }

Clase **ForoRespuestaModelo**

Dentro de la clase se tienen variables con el nombre de las columnas de la tabla foro\_respuesta. Se utiliza para manejar como objetos los resultados de las consultas sobre esta tabla. Se utiliza el método **set** explicado en la sección de métodos recurrentes.

Se declaran las varibales *id\_fororespuestas, fecha, contenido, id\_forohilo, id\_usuario* y *ocultar* como variables publicas al igual que el método **set**

    class ForoRespuestaModelo{

        public $id\_fororespuesta;

        public $fecha;

        public $contenido;

        public $id\_forohilo;

        public $id\_usuario;

        public $ocultar;

Método **set**

Recibe como parametro *id\_fororespuesta, fecha, contenido, id\_forohilo, id\_usuario* y *ocultar.* El método es explicado en métodos recurrentes.

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

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

    $this->fecha = $fecha;

    $this->contenido = $contenido;

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

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

$this->ocultar = $ocultar;

}

Clase **ForoRespuestaControlador**

Se utiliza para manejar el ingreso y la salida de los datos de la tabla foro\_respuesta creando objetos de la clase **ForoRespuestaModelo** para manejar los datos. La clase es heredada de **DBConexion** y contiene los métodos **Respuestas, Responder, EliminarRespuestaUsuario** y **EliminarRespuesta.**

class ForoRespuestaControlador extends DBConexion

Método **Respuestas**

Se utiliza para devolver las respuestas de un hilo en específico y mostrarlas en la vista de hilo.

Se recibe como parámetro *id\_forohilo*, se inicia la conexión llamando a **start**, se prepara la consulta sobre la tabla foro\_respuesta donde id\_forohilo sea igual a *id\_forohilo* ordenada de manera descendiente. Los valores se delas respuesta se vieron guardados en *lista* como un array de objetos **ForoRespuestaModelo** donde son guardados los valores recuperados de **fetch**, se terminar la conexión y el método devuelve *lista.*

    public function Respuestas($id\_forohilo){

        $this->start();

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

"SELECT \* FROM foro\_respuesta where id\_forohilo = $id\_forohilo ORDER BY id\_fororespuesta DESC"

);

        $stmt->execute();

                $lista = array();

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

$Res = new ForoRespuestaModelo();

$Res->set(

$F["id\_fororespuesta"],

$F["fecha"],

$F["contenido"],

$F["id\_forohilo"],

$F["id\_usuario"],

$F["ocultar"]

             );

$lista[] = $Res;

}

$this->stop();

return $lista;

    }

Método **Responder**

El método se utiliza cuando el usuario realiza una respuesta sobre un hilo.

Recibe por medio de post los valores que son asignados a las variables de *id\_hilo,*  y *cont*. Se realiza de la misma manera que **Agregar** explicado en métodos recurridos. Se declara los objetos Noti de la clase **NotificacionesControlador** y foro de la clase **ForoControlador**, se le asigna a *f* el resultado de **HiloContenido** con el parametro *id\_hilo* explicado en la parte superior en la clase de **ForoControlador**,en caso de que *id\_usuario* no coincida con el *id\_usuario* de *f.*

    public function Responder(){

        if($\_SERVER['REQUEST\_METHOD']=='POST'){

            $this->start();

                $id\_hilo = $\_POST['id\_forohilo'];

                $cont = $\_POST['contenido'];

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

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

     "INSERT INTO foro\_respuesta VALUES (NULL, NOW(), '$cont', $id\_hilo, $id\_usuario,0)"

     );

         $stmt->execute();

$Noti = new NotificacionesControlador();

$foro = new ForoControlador();

$f = $foro->HiloContenido($id\_hilo);

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

$Noti->Insert(3,$id\_usuario,$id\_hilo,$f->id\_usuario);

}

    $this->stop();

        }

    }

Método **EliminarRespuestaUsuario**

Se utiliza cuando el usuario desea eliminar su respuesta sobre un hilo.

Recibe mediante post el valor de *id\_fororespuesta* y se procede con la explicación de **EliminarUsuario** en métodos recurrentes aplicado a foro\_respuesta.

    public function ElimiarRespuestaUsuario(){

         $this->start();

            $id\_fororespuesta=$\_POST['id\_f'];

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

"UPDATE foro\_respuesta set ocultar = 1 WHERE id\_fororespuesta = $id\_fororesuesta"

);

$stmt->execute();

$this->stop();

    }

Método **EliminarRespuesta**

Se utiliza cuando el moderador elimina de forma definitiva de la base de datos una respuesta sobre un foro.

Mediante por recibe y asigna el valor a *id\_fororespuesta****.***Se explica como método **Eliminar** aplicado a foro\_respuesta.

public function ElimiarRespuesta(){

        $this->start();

            $id\_fororespuesta=$\_POST['id\_f'];

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

"DELETE FROM foro\_respuesta WHERE id\_fororespuesta = $id\_fororespuesta"

);

$stmt->execute();

$this->stop();

    }

}

Clase **ImagenColeccionModelo**

Los resultados de las consultas sobre la tabla de imagen\_coleccion son guardados como objetos de esta clase por lo que se tienen las misma variables que columnas y con el mismo nombre.

Se declaran como variables publicas *id\_imagencoleccion, imagen* y *id\_coleccion*,

Class ImagenColeccionModelo{

public $id\_imagencoleccion;

public $imagen;

public $id\_coleccion;

Método **set**

El método también es declarado como publico y es explicado como método recurrente en la parte superior, recibe los parámetros *id\_imagencoleccion, imagen* y *id\_coleccion*.

public function set( $id\_imagencoleccion, $imagen, $id\_coleccion){

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

    $this->imagen = $imagen;

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

}

Clase **InicioControlador**

Dentro de esta clase se manejan los datos de la tabla publicación. Se heredad de la clase **DBConexion**. Dentro se tienen los métodos **Inicio, InicioUsuario, Publicacion, PublicacionInfo, Publicar, EliminarPublicacioUsuario** y **EliminarPublicacion** y una variable global *publicacion* utilizada para mostrar valores en la vista de inicio**.**

Class InicioControlador extends DBConexion {

        public $publicacion;

Método **Inicio**

Se utiliza para mostrar las publicaciones de los amigos y seguidos del usuario, de acuerdo a la privacidad de la publicación y si el usuario tiene o no activado el contenido explicito.

Se declara *con* con el valor de 0 y se llama **start** para comenzar la conexión. Se delcara *ListaU* como un array y se crea Amigos como objeto de **AmigosControlador**, se llama **ListaAmigos** y se guarda el resultado en *am* mientras exista una fila en este se guardado en *a,* se aumenta *con* y se crea artista como objeto de **ArtistaControlador** se revisa si id\_usuario1 o id\_usuario2 son iguales a la variable de sesión *id\_usuario* y se manda como parámetro el contrario al método **ArtistaUsuario**, el valor devuelto de *id\_artista* es agregado a *listaU* en en lugar de *con*. Se crea Seguidores como objeto de **SeguidoresControlador,** se llama el método de ListaSeguiendo y lo que devuelve es guardado en *se* mientras exista valores dentro de *se*, se declara artista como objeto de **ArtistaControlador** se llama **ArtistaUsuario** con el valor de *id\_usuario2* de la fila de resultados, el valor devuelto de *id\_artista* se guarda en *listaU*. Mediante **implode** se convierte el array de *listaU* en string separando por una coma los lugares y esto se guarda en *x*. Se crea Usuarios como objeto de **UsuarioControlador** con el resultado de **Usuario** se comprueba si el usuario actual tiene permitido el contenido explicito. Con los valores obtenidos de los métodos anteriores se prepara una consulta en la tabla de publicacion donde ocultar sea 0, id\_artista se encuentre dentro de los valores de *x* y contenido\_explicito sea de acuerdo al usuario, con **prepare** y **execute** se realiza la consulta. Se declara *lista* como un array, utilizando **fetch** se guarda el siguiente resultado de la consulta en *Publicacion,* se crea Publicaciones como objeto de **PublicacionModelo,** con **set** se guardan los valores de *Publicacion* en el y se agrega a *lista.* Se detiene la conexión y se le asigna a la variable global *publicacion* los valores de *lista.*

public function Inicio(){

$con = 0;

$this->start();

$listaU = array();

$Amigos = new AmigosControlador();

$am = $Amigos->ListaAmigos();

if(isset($\_SESSION['id\_artista'])){

$listaU[$con] = $\_SESSION['id\_artista'];

}

foreach ($am as $a) {

$con++;

$artista = new ArtistaControlador();

if($a->id\_usuario1 == $\_SESSION['id\_usuario']){

$ar = $artista->ArtistaUsuario($a->id\_usuario2);

$listaU[$con] = $ar->id\_artista;

}

if($a->id\_usuario2 == $\_SESSION['id\_usuario']){

$ar = $artista->ArtistaUsuario($a->id\_usuario1);

$listaU[$con] = $ar->id\_artista;

}

}

$Seguidores = new SeguidoresControlador();

$se = $Seguidores->ListaSiguiendo();

foreach ($se as $s) {

$con++;

$artista = new ArtistaControlador();

$ar = $artista->ArtistaUsuario($s->id\_usuario2);

$listaU[$con] = $ar->id\_artista;

}

$x = implode(",", $listaU);

$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 id\_artista in ($x) and contenido\_explicito in ($c) 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();

$this->publicacion = $lista;

        }

Método **InicioUsuario**

Se utiliza en la pagina de inicio cuando no se ha registra o iniciado sesión dentro de la página, muestra las publicaciones que no tengan contenido explicito, no sean privadas y tengan imagen.

Se comienza la conexión con **start** y se prepara la consulta en publicacion donde contenido\_explicito y privacidad sea igual a 0 e imagen no sea nulo. Se realiza con **prepare** y **execute**. Se crea *lista* como array, mediante **fetch** se guarda los resultado en *Publicacion* mientras existan filas como resultado de la consulta. Publicaciones se crea como objeto de **PublicacionModelo** y con **set** se guarda los valores de *Publicacion* dentro de Publicaciones, el objeto se agrega a *lista.* Se detiene la conexión y devuelve *lista.*

public function InicioUsuario(){

$this->start();

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

"SELECT \* FROM publicacion where contenido\_explicito = 0 and privacidad = 0 and imagen is not null 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;

}

Método **Publicacion**

Devuelve la información de una publicación en específico con el id de una petición post de Ajax .

Se comprueba si se realizo una petición post, en caso de que sea verdad se le asigna a *id\_publicacion* el valor de *idp* pasado por post y se inicia la conexión. Con **prepare** y **execute** se realiza la consulta sobre publicacion donde id\_publicacion sea igual a *id\_publicacion*. Se guardan los resultados **fetch** en *Publicacion* y estos se guardan en Publicaciones utilizando **set**. Se detiene la conexión, se incluye el archivo de Publicacion.php y retorna Publicacion.

public function InicioUsuario(){

$this->start();

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

"SELECT \* FROM publicacion where contenido\_explicito = 0 and privacidad = 0 and imagen is not null 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;

}

Método **PublicacionInfo**

Devuelve la información de una publicación en especifico.

Realiza lo mismo que **Publicacion**explicado anteriormente con el excepción de que recibe *id\_publicacion* como parametro y no incluye ningún archivo.

public function PublicacionInfo($id\_publicacion){

$this->start();

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

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

);

$stmt->execute();

$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"]

);

$this->stop();

return $Publicaciones;

}

Método **Publicar**

Se utiliza cuando el usuario artista realiza una publicación, inserta los valores en la tabla de publicación y agrega el archivo a la carpeta correspondiente

En caso de que se haga una petición post, inicia la conexión y recibe los valores que son guardados en *des, edad* y *tipo.* Se crea art como objeto de **ArtistaControlador** y se llama **ArtistaUsuario** con el parámetro de *id\_usuario*, se guarda en *a*. Se comprueba si *imagen* no tiene valor, encaso de que no se realiza la sentencia insert en la tabla publicacion con los valores de las variables anteriores y la columna de imagen como nulo, con **prepare** y **execute**. En caso de que si tenga valor se mueve *imagen* a la carpeta de Imágenes/imgPublicacion después se realiza una sentencia insert con los valores de las variables el de *imagen* a la tabla publicacion con **prepare** y **execute.** Se termina la conexión y se redirige a inicio.

        public function Publicar(){

            if($\_SERVER['REQUEST\_METHOD']=='POST'){

            $this->start();

            $des = $\_POST['des'];

            if(isset($\_POST['edad'])){

                $edad = 1;

            }else{

                $edad = 0;

            }

            $tipo = $\_POST['tipoP'];

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

     $art = new ArtistaControlador();

     $a = $art->ArtistaUsuario($id\_usuario);

            if($\_FILES["image"]["name"]==''){

                $stmt = $this->pdo->prepare("INSERT into publicacion VALUES(NULL,NOW(),$edad,'$des',NULL,$tipo,null,$a->id\_artista,0)");

$stmt->execute();

            }else{

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

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

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

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

                $stmt = $this->pdo->prepare("INSERT into publicacion VALUES(NULL,NOW(),$edad,'$des',NULL,$tipo,'$imagen',$a->id\_artista,0)");

$stmt->execute();

            }

$this->stop();

            }

             header("Location: Control.php?c=Inicio&a=Inicio");

        }

Método **EliminarPublicacionUsuario**

Se utiliza cuando el usuario decide borrar su publicación, no se borra de la base de datos solo se actualiza ocultar y ya no es visible en la plataforma.

El método es explicado como **EliminarUsuario** en métodos recurrentes, se aplica a la tabla de publicacion y recibe *id\_publicacion*  por medio de get. Dependiendo de si es moderador o no se redirige al inicio correpondiente

public function EliminarPublicacionUsuario(){

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

$this->start();

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

"UPDATE publicacion set ocultar = 1 where id\_publicacion = $id\_publicacion"

);

$stmt->execute();

$this->stop();

if($\_SESSION['tipo\_usuario']==3){

header("location: Control.php?c=Moderador&a=Moderador");

}else{

header("location: Control.php?c=Inicio&a=Inicio");

}

}

Método **EliminarPublicacion**

Se utiliza cuando una publicacion es borrada de manera permanente de la base de datos, se borran los comentarios y me gusta de la publicacion al igual que la publicacion en sus respectivas tablas.

Recibe por medio de parametro *id\_publicacion* y se realiza el método recurrente **Eliminar** aplicado a las tablas comentario, me\_gusta y publicacion.

public function EliminarPublicacion($id\_publicacion){

            $this->start();

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

"DELETE FROM comentario where id\_publicacion = $id\_publicacion"

);

$stmt->execute();

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

"DELETE FROM me\_gusta where id\_publicacion = $id\_publicacion"

);

$stmt->execute();

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

"DELETE FROM publicacion where id\_publicacion = $id\_publicacion"

);

$stmt->execute();

$this->stop();

        }

Clase **InterfazControlador**

Se utiliza para el manejo de MVC, recibe por get los valores correspondientes al controlador y al método que se utilizara. Contiene los metodos **getControlador, getAccion, setControlador, setAccion** y  **render**  y las variables privadas *controlador, acción y ruta***.**

Class InterfazControlador {

private $controlador;

private $accion;

private $ruta;

public function \_\_construct(){

if(isset($\_GET["c"])):

$this->controlador = $\_GET["c"];

else:

$this->controlador = false;

endif;

if(isset($\_GET["a"])):

$this->accion = $\_GET["a"];

else:

$this->accion = false;

endif;

if(!isset($\_SESSION["id\_usuario"])):

if(!$this->controlador || !$this->accion):

header("Location: usuario.php");

endif;

else:

if(!$this->controlador || !$this->accion):

header("Location: Control.php?c=Inicio&a=Inicio");

endif;

endif;

}

Método **getControlador**

Se utiliza para obtener el valor de la variable *controlador* de una instancia creada de esta clase.

public function getControlador(){

return $this->controlador;

}

Método **getAccion**

Se utiliza para obtener el valor de la variable *acción* de la instancia.

public function getAccion(){

return $this->accion;

}

Método **setControlador**

Se utiliza para asignar a la propiedad *controlador* el valor recibido por el parametro *controlador*

public function setControlador($controlador){

$this->controlador = $controlador;

}

Método **setAccion**

Se utiliza igual que **setControlador** pero en la variable *acción.*

public function setAccion($accion){

$this->accion = $accion;

}

Método **render**

Se utiliza para incluir los archivos de las vistas en los archivos principales.

Si las variables de *controlador* y *acción* tienen valores las acumoda dentro de *ruta* como las carpetas correspondientes, si *ruta* existe retorna *ruta* si no en cualquiera de los casos retorna ./vistas/Usuario/Inicio.php.

public function render(){

if($this->controlador && $this->accion):

$ruta = "./vistas/".$this->controlador."/".$this->accion.".php";

if(file\_exists($ruta)):

return $ruta;

else:

return "./vistas/Usuario/Inicio.php";

endif;

else:

return "./vistas/Usuario/Inicio.php";

endif;

}

Clase **MeGustaModelo**

Se utiliza para guardar los resultas de las consultas sobre la tabla me\_gusta en un objeto, por lo que la variables coinciden con el nombre de las columnas de esta tabla.

Se tienen las variables publicas *id\_megusta, tipo\_me\_gusta, id\_publicacion* y *id\_usuario*.

Class MeGustaModelo{

public $id\_megusta;

public $tipo\_me\_gusta;

public $id\_publicacion;

public $id\_usuario;

Método **set**

Se utiliza para establecer los valores recibidos por parámetros en las propiedades de la instancia, se recibe como parametro *id\_megusta, tipo\_me\_gusta, id\_publicacion* y *id\_usuario*.

public function set( $id\_megusta, $tipo\_me\_gusta, $id\_publicacion, $id\_usuario){

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

$this->tipo\_me\_gusta = $tipo\_me\_gusta;

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

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

}

Clase **MeGustaControlador**

La clases es utilizada para manejar los datos referentes a la tabla me\_gusta, es heredada de **DBConexion** y tiene lo métodos **MeGusta, MeGustaConsulta** y **ListaMeGusta.**

Class MeGustaControlador extends DBConexion{

Método **MeGusta**

Se utiliza cuando un usuario da me gusta o no a una publicacion, se comprueba si ya tenia un registro en la tabla si no lo crea y si es el mismo tipo de registro lo elimna y si es el tipo contrario lo actualiza.

Recibe por medio de post los valores de *tipo\_me\_gusta* y *id\_publicacion,* se guarda en *id\_usuario* el valor de la variable de sesión *id\_usuario*. Se comienza la conexión y con **prepara** y **execute** se realiza la consulta sobre la tabla de me\_gusta donde id\_usuario y id\_publicacion coincidan. Con **rowCount** se comprueba el numero se resultado si es mayor a cero se guardan los resultado de **fetch** en *MeGusta****.*** Si el resultado *tipo\_me\_gusta* guardado en *MeGusta* es igual a *tipo\_me\_gusta* se elimina de la tabla con una sentencia delete usando **prepare** y **execute.** Si no se realiza una sentencia update con el valor de *tipo\_me\_gusta.* En caso de que el numero de resultados fuera cero se crea el registro una sentencia insert utilizando **prepare** y **execute.** Se crean objetos de las clases **NotificacionesControlador**, **InicioControlador** y **ArtistaControlador.** Se guarda en p los valores obtenidos de **PublicacionInfo** con el parámetro de *id\_publicacion* y se guarda en *a* los valores obtenidos de **Artista** con el parámetro de *id\_artista*. Si el *id\_usuario* de la publicacion es distinto al *id\_usuario* actual se llama **Insert** con los parámetros 1, *id\_usuario, id\_publicacion, a->id\_usuario.*

public function MeGusta(){

          $tipo\_me\_gusta= $\_POST['tipoLike'];

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

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

          $this->start();

$noti = new NotificacionesControlador();

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

"SELECT \* FROM me\_gusta WHERE id\_usuario = $id\_usuario and id\_publicacion = $id\_publicacion"

);

$stmt->execute();

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

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

if($MeGusta["tipo\_me\_gusta"]==$tipo\_me\_gusta){

      $stmt = $this->pdo->prepare(

"DELETE FROM me\_gusta WHERE id\_megusta = $MeGusta['id\_megusta']"

);

$stmt->execute();

} else{

      $stmt = $this->pdo->prepare(

"UPDATE me\_gusta set tipo\_me\_gusta = $tipo\_me\_gusta WHERE id\_megusta = $MeGusta['id\_megusta'] "

);

$stmt->execute();

}

}else{

 $stmt = $this->pdo->prepare(

"INSERT into me\_gusta VALUES(NULL,$tipo\_me\_gusta,$id\_publicacion,$id\_usuario)"

);

$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){

$Noti->Insert(1,$id\_usuario,$id\_publicacion,$a->id\_usuario);

}

}

$this->stop();

}

Método MeGustaConsulta

Es utilizado para saber si el usuario que abre una publicación tiene algún registro de interacción sobre esta. Retorna el registro si hay y en caso de que no retorna nulo.

Recibe como parámetro *id\_publicacion* y guarda en *id\_usuario* el valor de sesión *id\_usuario* y se comienza la conexión llamando a **start**. Se prepara la consulta sobre la tabla me\_gusta donde id\_usuario y id\_publicacion coincidan con las variables. Con **rowCount** se comprueba si hay algún resultado, si hay con **fetch** se recuperan los valores y se guardan en *MeGusta,* se crea MeGustas como instacia de **MeGustaModelo** y se guardan los valores de *MeGusta* dentro utilizando **set** y se retornaMeGustas, no hay resultados el método retorna nulo.

public function MeGustaConsulta($id\_publicacion){

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

             $this->start();

  $stmt = $this->pdo->prepare(

"SELECT \* FROM me\_gusta WHERE id\_usuario = $id\_usuario and id\_publicacion = $id\_publicacion"

);

$stmt->execute();

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

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

$MeGustas = new MeGustaModelo();

$MeGustas->set(

$MeGusta["id\_megusta"],

                  $MeGusta["tipo\_me\_gusta"],

                  $MeGusta["id\_publicacion"],

                  $MeGusta["id\_usuario"]

                  );

$this->stop();

return $MeGustas;

                    }else{

$this->stop();

                        return null;

                    }

        }

Método **ListaMeGusta**

Se utiliza para saber los registros que tiene la tabla me\_gusta sobre una pubicacion en especifico.

Recibe como parámetro *id\_publicacion*, se explica cómo **Lista** en la sección de métodos recurrentes aplicándolo a la tabla de me\_gusta.

     public function ListaMeGusta($id\_publicacion){

$this->start();

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

"SELECT \* FROM me\_gusta WHERE id\_publicacion = $id\_publicacion"

);

$stmt->execute();

$lista = array();

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

$MeGustas = new MeGustaModelo();

$MeGustas->set(

$MeGusta["id\_megusta"],

$MeGusta["tipo\_me\_gusta"],

$MeGusta["id\_publicacion"],

$MeGusta["id\_usuario"]

);

$lista[] = $MeGustas;

}

$this->stop();

return $lista;

}

Clase **MensajesModelo**

Se tienen las variables con el nombre correspondiente a cada columna de la tabla mensaje. Se utiliza para manejar los resultados de la tabla mensajes como objetos.

Se tienen las variables publicas *id\_mensaje, fecha, texto, id\_conversacion* y *id\_usuario*. Al igual que el método **set**.

    Class MensajesModelo{

            public $id\_mensaje;

            public $fecha;

            public $texto;

            public $id\_conversacion;

            public $id\_usuario;

Método **set**

Recibe como parámetros *id\_mensaje, fecha, texto, id\_conversacion* y *id\_usuario*. Es explicado en la sección de métodos recurrentes.

public function set($id\_mensaje, $fecha, $texto, $id\_conversacion, $id\_usuario ){

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

    $this->fecha = $fecha;

    $this->texto = $texto;

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

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

}

Clase **MensajesControlador**

Se utiliza para manejar toda la entrada y salida de información respectiva de la tabla mensaje, es heredad de la clase **DBConexion** y tiene los métodos **BandejaEntrada, Mensajes, Conversacion**  y  **Mandar**

Class MensajesControlador extends DBConexion

Método **BandejaEntrada** y **Mensajes**

Son métodos vacios utilizado en relación con sus respectivas vistas vistas.

public function BandejaEntrada(){

}

public function Mensajes(){

        }

Método MensajesNuevos

Se utiliza para realizar una petición Ajax e imprimir todos los mensajes de la conversación dentro del método.

Recibe *id\_usuario* por post y lo guarda en *id\_usuario*, crea Men como objeto de **MensajesControlador** llama al método **Conversacion** con el paramero *id\_usuario* y guarda el resultado en *me* por cada fila del resultado crea us como objeto de **UsuarioControlador** y se llama a Usuario con el *id\_usuario* de *me,* se imprime el contenido del mensaje, el nombre del usuario y la hora.

        public function MensajesNuevos(){

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

                        $Men = new MensajesControlador();

                        $me = $Men->Conversacion($id\_usuario);

                        foreach ($me as $m) {

                        $us = new UsuarioControlador();

                        $u = $us->Usuario($m->id\_usuario);

                            echo "<div class='Mensaje'><a href='Control.php?c=Perfiles&a=Perfiles&id=".$m->id\_usuario."'>$u->nombre\_usuario</a><div class='contenido'> $m->texto</div><div class='fecha'>$m->fecha</div></div>";

                        }

        }

Método **Conversacion**

Se utiliza para recuperar todos los mensajes de una conversación entre dos usuarios.

Recibe el paramaetro de *id\_usuario2,s*e comienza la conexión llamando a **start**, se crea co como instancia de **ConversacionControlador** se llama a **Validar**, con el valor devuleto se hace una consulta en la tabla de mensaje. Los resultados son guardos en una lista de objetos tipo **MensajesModelos** y el método devuelve la *lista.*

public function Conversacion($id\_usuario2){

             $this->start();

            $co = new conversacionControlador();

            $c = $co->Validar($id\_usuario2);

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

"SELECT \* FROM mensaje WHERE id\_conversacion = $c"

);

$stmt->execute();

$lista = array();

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

$Mensajes = new MensajesModelo();

$Mensajes->set(

$Mensaje["id\_mensaje"],

                    $Mensaje["fecha"],

                    $Mensaje["texto"],

                    $Mensaje["id\_conversacion"],

                    $Mensaje["id\_usuario"]

                    );

$lista[] = $Mensajes;

endwhile;

$this->stop();

return $lista;

        }

Método **Mandar**

Se utiliza cuando el usuario ingresa un nuevo mensaje en la conversación con otro usuario.

Recibe por medio de post *texto* y *id\_usuario2*, como prueba que existe la conversación llamando al método Validar con el parámetro *id\_usuario2* y se inserta en la tabla con los valores de las variables mencionadas.

public function Mandar(){

            $this->start();

            $texto = $\_POST['Texto'];

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

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

            $co = new conversacionControlador();

            $c = $co->Validar($id\_usuario2);

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

"INSERT into mensaje values(NULL,NOW(),'$texto',$c,$id\_usuario)"

);

$stmt->execute();

$this->stop();

        }

Clase **ModeradorControlador**

Se utiliza para manejar las funciones del usuario moderador respecto a los usuarios bloqueados.Se hereda de la clase **DBConexion** y tienes lo métodos **Moderador, BloquearUsuario, DesbloquearUsuario** y **ListaBloqueados.**

Class ModeradorControlador extends DBConexion

Método **Moderador**

Es un método vacío que solo tiene función en relación a la vista.

   public function Moderador(){    }

Método **BloquearUsuario**

Es utilizado cuando el moderador bloquea usuario de manera manual. Se inserta dependiendo si puso una fecha exacta o escogió entre las opciones de un día, una semana y un mes

En caso de que exista una petición post, se llama **start** y se asigna el valor de post *id\_usuario* a *id\_usuario* se revisa el valor de la fecha en caso de que sea cero, se inserta en la tabla de bloqueados la fecha ingresada. De lo contrario se le agregan los días pasado por el valor de *fecha* a la fecha actual y eso se inserta en la tabla de bloqueados. Se actualiza la tabla de usuario a bloqueado igual a i. Se termina la conexión y se redirige a reportes.

   public function BloquearUsuario(){

if($\_SERVER['REQUEST\_METHOD']=='POST'){

$this->start();

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

if($\_POST['fecha']==0){

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

$fecha = date\_create($fin);

$f = date\_format($fecha, 'Y-m-d H:i:s');

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

"INSERT into bloqueados VALUES(NULL,$id\_usuario,NOW(),'$f',0)"

);

$stmt->execute();

}else{

$fin = $\_POST['fecha'];

$x = $fin." days";

$fecha = date('Y-m-d H:i:s');

$fecha = date\_create($fecha);

$fecha = date\_add($fecha,date\_interval\_create\_from\_date\_string($x));

$f = date\_format($fecha, 'Y-m-d H:i:s');

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

"INSERT into bloqueados VALUES(NULL,$id\_usuario,NOW(),'$f',0)"

);

$stmt->execute();

}

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

"UPDATE usuario set bloqueado = 1 where id\_usuario = $id\_usuario"

);

$stmt->execute();

$this->stop();

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

}

}

Método **DesbloquearUsuario**

Se utiliza cuando el moderador desbloquea manual mente un usuario quitándolo de la tabla de bloqueados y actualizando la tabla de usuario.

Se comienza la conexión, recibe el valor de *id\_usuario* por get y realiza las sentencias de Update en la tabla de usuario cambiado bloqueado a 0 y delete en la tabla de bloqueados. Donde id\_usuario es igual a *id\_usuario* se utiliza **prepare** y **execute** para realizar las sentencias. Se redirige al inicio de moderador.

public function DesbloquearUsuario(){

$this->start();

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

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

"UPDATE usuario set bloqueado = 0 where id\_usuario = $id\_usuario"

);

$stmt->execute();

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

"DELETE from bloqueados where id\_usuario = $id\_usuario"

);

$stmt->execute();

$this->stop();

header("Location: Control.php?c=Moderador&a=Moderador");

}

Método **ListaBloqueados**

Se utiliza para mostrar todos los usuarios bloqueados al moderador

Es explicado como método recurrente **Lista** en la parte superior.

public function ListaBloqueados(){

$this->start();

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

"SELECT \* FROM bloqueados where expirado = 0"

);

$stmt->execute();

$lista = array();

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

$Bloqueados = new BloqueadosModelo();

$Bloqueados->set(

$B['id\_bloqueado'],

$B['id\_usuario'],

$B['inicio'],

$B['fin'],

$B['expirado']

);

$lista[] = $Bloqueados;

}

$this->stop();

return $lista;

}

Clase **NotificacionesModelo**

Se utiliza para manejar las notificaciones del usuario de manera sencilla como objetos. Las variables coinciden con el nombre de las columnas de la tabla notificaciones.

Se declaran las variables *id\_notificacion, tipo, contenido, id\_usuario, visto, fecha, id\_evento* y *id\_usuario* como publicas.

    Class NotificacionesModelo{

public $id\_notificacion;

public $tipo;

public $contenido;

public $id\_usuario;

public $visto;

public $fecha;

public $id\_evento;

public $id\_usuario1;

Método **set**

Recibe como parámetros *id\_notificacion, tipo, contenido, id\_usuario, visto, fecha, id\_evento* y *id\_usuario,*  es explicado como método recurrente.

public function set( $id\_notificacion, $tipo, $contenido, $id\_usuario, $visto, $fecha, $id\_evento, $id\_usuario1){

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

    $this->tipo = $tipo;

    $this->contenido = $contenido;

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

    $this->visto = $visto;

$this->fecha = $fecha;

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

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

}

Clase **NotificacionesControlador**

Se utiliza para manejar los registros de la tabla de notificaciones, muestra e inserta los valores dentro de la tabla. Hereda de la clase **DBConexion** y tiene los métodos **Notificaciones, ListaNotificaciones** e **Insert.**

    Class NotificacionesControlador extends DBConexion

Método **Notificaciones**

Es un método vacio que cumple función en relación con la vista de notificaciones.

public function Notificaciones(){}

Método **ListaNotificaciones**

Se utiliza para devolver las notificaciones del usuario actual y mostrarla en la vista de notificaciones de manera descendiente.

Es explicado como método recurrente **Lista**, se aplica a la tabla de notificaciones donde id\_usuario sea igual a *id\_usuario* de la variable de sesión.

public function ListaNotificaciones(){

            $this->start();

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

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

"SELECT \* FROM notificaciones where id\_usuario1 = $id\_usuario order by id\_notificaciones desc"

);

$stmt->execute();

$lista = array();

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

$Noti = new NotificacionesModelo();

$Noti->set(

$N['id\_notificacion'],

$N['tipo'],

$N['contenido'],

$N['id\_usuario'],

$N['visto'],

$N['fecha'],

$N['id\_evento'],

$N['id\_usuario1']

);

$lista[] = $Noti;

}

            $this->stop();

            return $lista;

        }

Método **Insert**

Se utiliza cuando hay un cambio en publicaciones, foro o en las relaciones de los usuarios, se llama para ingresar un nuevo registro sobre la tabla de notificaciones.

Recibe como parámetro *tipo, id\_usuario, id\_evento ­*y *x,* se tiene un array de contenido con la descripción de cada tipo de notificaciones. Con los valores anteriores si realiza un **Insert** explicado en métodos recurrentes.

public function Insert($tipo,$id\_usuario,$id\_evento,$x){

$this->start();

$contenido = array(

"",

"ha calificado tu publicaciones",

"ha comentado tu publicacion",

"ha respondido tu foro",

"te esta siguiendo",

"acepto tu solicitud de amistad",

"te envio solicitud de amistas",

"Tu comentario fue reportado",

"Tu publicacion fue reportada",

"Tu cuenta fue reportada");

$c = $contenido[$tipo];

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

"INSERT INTO notificaciones VALUES(NULL,$tipo,'$c',$id\_usuario,0,NOW(),$id\_evento,$x)"

);

$stmt->execute();

$this->stop();

}

Método **Update**

Se utiliza cuando el usuario abre una notificación comprobando que el usuario ha visto la notificación.

Recibe por medio de post el valor de *id\_noti*. Realiza el procedimiento explicado en el método recurrente **Update**.

public function Update(){

$this->start();

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

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

"UPDATE notificaciones set visto = 0 where id\_notificaciones = $id\_noti"

);

$stmt->execute();

$this->stop();

}

Clase **PaisModelo**

Se utiliza para mostras los valores de la tabla país como un objeto y crea listas de estos. Tiene variables con el mismo nombre de las columnas.

Se declaran las variables *id\_pais* y *nombre\_pais* como variables publicas.

Class PaisModelo {

public $id\_pais;

public $nombre\_pais;

Método **set**

Recibe como parámetros *id\_pais* y *nombre\_pais*, se asignanlos valores recibido por parámetro a las propiedades de la clase.

public function set( $id\_pais, $nombre\_pais){

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

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

}

Clase **PerfilModelo**

Se utiliza para manejar los datos del perfil del usuario artista guardados en la tabla perfil por lo tanto las variables tienen el mismo nombre de las columnas de esta tabla.

    class PerfilModelo{

        public $id\_perfil;

        public $metas;

        public $exper;

        public $otro;

        public $estudios;

Método **set**

Recibe como parámetro *id\_perfil, metas, exper, otro* y *estudios.* Es explicado como método recurrente utilizando las propiedades de la clase y los valores de los parametros.

public function set( $id\_perfil,$metas,$exper,$otro,$estudios){

    $this->id\_perfil = $id\_perfil;

    $this->metas = $metas;

    $this->exper = $exper;

    $this->otro = $otro;

    $this->estudios = $estudios;

}

Clase **PerfilControlador**

Se utiliza para manejar la información de perfil del usuario artista al momento de registrar, mostrase y modificarse.Se extiende de la clase **DBConexion** y tiene los métodos **Perfil, Insert** y **Update.**

    Class PerfilControlador extends DBConexion

Método **Perfil**

Es usada para devolver los valores guardados en la tabla perfil con un id especifico para mostrarlos en las vistas.

Recibe como parámetro *id\_perfil* que utiliza para hacer una consulta sobre la tabla perfil donde id\_perfil sea igual al parámetro. Se realiza la consulta con **prepare** y **excecute**. Los resultados obtenidos mediante **fetch** se guardan en el objeto perfiles de la clase **PerfilModelo** y se devuelve este valor.

        public function Perfil($id\_perfil){

                $this->start();

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

"SELECT \* FROM perfil where id\_perfil = $id\_perfil"

);

$stmt->execute();

$perfiles = new PerfilModelo;

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

$perfiles->set(

 $perfil['id\_perfil'],

                        $perfil['metas'],

                        $perfil['exper'],

                        $perfil['otro'],

                        $perfil['estudios']

);

$this->stop();

                return $perfiles;

        }

Método **Insert**

Se utiliza cuando un usuario artista se registra, el método guarda en la tabla perfil los datos ingresados por el usuario.

Recibe como parámetros *Metas, Exper, Otro* y *Estudios*.Es explicado en la sección de métodos recurrentes como **Insert.**

        public function Insert($Metas,$Exper,$Otro,$Estudios){

                $this->start();

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

"INSERT into perfil VALUES(NULL,'$Metas','$Exper','$Otro','$Estudios')"

);

$stmt->execute();

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

"SELECT MAX(id\_perfil) as id FROM perfil"

);

$stmt->execute();

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

$this->stop();

return $perfil["id"];

        }

Método **Update**

Se utiliza cuando el usuario artista configura sus datos y acutaliza los datos referentes a la tabla perfil.

Recibe como parámetros *Metas, Exper, Otro* y *Estudios*. Se aplica a la abla perfil y con los parámetros mencionados el método **Update** explicado en métodos recurrentes.

public function Update($Metas,$Exper,$Otro,$Estudios,$id\_perfil){

$this->start();

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

"UPDATE perfil SET metas= '$Metas', exper = '$Exper', otro ='$Otro', estudios ='$Estudios' where id\_perfil = $id\_perfil"

);

$stmt->execute();

$this->stop();

}

Clase **PortafolioModelo**

Se utiliza para devolver los valores de las consultas sobre la tabla de portafolio como un objeto utilizando el nombre de las columnas como propiedades. Las variables *id\_portafolio* y *descripción* se declaran como variables públicas.

    class PortafolioModelo{

        public $id\_portafolio;

        public $descripcion;

Método **set**

Recibe como parámetros *id\_portafolio* y *descripción*, se explica en métodos recurrentes.

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

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

    $this->descripcion = $descripcion;

}

Clase **PortafolioControlador**

Dentro de la clase se maneja la información de la tabla de portafolio, la que entra y sale como consultas. Se heredad de **DBConexion** y tiene los métodos de **Portafolio, Insert** y **Update.**

    Class PortafolioControlador extends DBConexion

Método **Portafolio**

Se utiliza para recuperar la información del portafolio de un usuario artista, utilizando el id\_portafolio correspondiente.

Recibe como parámetro *id\_portafolio*, se inicia la conexión llamando a **start**, se prepara la consulta sobre portafolio donde id\_portafolio sea igual a *id\_portafolio* con **prepare** y **execute**, se guardan los resultados obtenido con **fetch** en la variable *portafolio* y dentro del objeto Portafolios de la clase **PortafolioModelo** se guardan los valores de *portafolio* utilizando **set**, se detiene la conexión y el método devulve Portafolios.

        public function Portafolio($id\_portafolio){

                $this->start();

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

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

);

$stmt->execute();

$Portafolios = new PortafolioModelo;

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

$Portafolios->set(

 $portafolio['id\_diseno'],

                        $portafolio['descripcion']

);

$this->stop();

                return $Portafolios;

        }

Método **Insert**

Se utiliza para crear un portafolio al memento de que el usuario artista se esta registrando en la plataforma.

El método **Insert** se explica como método recurrente con la excepción de que todos los valores son nulos.

        public function Insert(){

                $this->start();

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

"INSERT into portafolio VALUES(NULL,NULL)"

);

$stmt->execute();

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

"SELECT MAX(id\_portafolio) as id FROM portafolio"

);

$stmt->execute();

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

$this->stop();

return $portafolio["id"];

}

Método **Update**

Se utiliza cuando el usuario artista modifica la descripción de su portafolio.

Recibe por medio de post *des* y *id\_portafolio*, se explica en métodos recurrentes como **Update** aplicado a la tabla de portafolio con las variables mencionadas.

public function Update(){

$this->start();

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

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

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

"UPDATE portafolio SET descripcion = '$des' WHERE id\_portafolio = id\_portafoli"

);

$stmt->execute();

$this->stop();

}

Clase **PublicacionModelo**

Se utiliza para manejar la información de las publicaciones registradas como objetos por lo que las propiedades llevan el mismo nombre que las columnas de la tabla publicación. Se declaran las variables *id\_publicacion, fecha, contenido\_explicito, contenido, etiquetas, privacidad, imagen, id\_artista* y *ocultar* como publicas*.*

class PublicacionModelo extends DBConexion{

       public $id\_publicacion;

public $fecha;

public $contenido\_explicito;

public $contenido;

public $etiquetas;

public $privacidad;

public $imagen;

public $id\_artista;

public $ocultar;

Método **set**

Recibe como parámetros *id\_publicacion, fecha, contenido\_explicito, contenido, etiquetas, privacidad, imagen, id\_artista* y *ocultar, se explica como método recurrente.*

public function set( $id\_publicacion,$fecha, $contenido\_explicito,$contenido,$etiquetas,$privacidad,$imagen,$id\_artista,$ocultar){

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

$this->fecha = $fecha;

$this->contenido\_explicito = $contenido\_explicito;

$this->contenido = $contenido;

$this->etiquetas = $etiquetas;

$this->privacidad = $privacidad;

$this->imagen = $imagen;

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

$this->ocultar = $ocultar;

}

Clase **ReportesComentariosModelo**

Se utiliza para guardas lo valores de las consultas sobre la tabla de reportes\_comentarios. Tiene las variables *id\_reporte, id\_comentario, id\_reportado, id\_reportero, razón, estatus* y *fecha* como variables publicas que corresponden a cada columna de la tabla de reportes\_comentarios.

    class ReportesComentariosModelo{

        public $id\_reporte;

        public $id\_comentario;

        public $id\_reportado;

        public $id\_reportero;

        public $razon;

        public $estatus;

        public $fecha;

Método **set**

El método **set** se explica como método recurrente en la sección de funciones compartidas del sistema.

public function set( $id\_reporte, $id\_comentario, $id\_reportado, $id\_reportero, $razon, $estatus, $fecha){

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

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

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

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

    $this->razon = $razon;

$this->estatus = $estatus;

    $this->fecha = $fecha;

}

Clase **ReportesComentariosControlador**

Se utiliza para el maneje de información entrante y saliente de la tabla de reportes\_comentario. Se hereda de **DBConexion** y tienes métodos **ReportarComentario, ListaComentarios** y **EstatusReporte.**

    Class ReportesComentariosControlador extends DBConexion

Método **ReportarComentario**

Se utiliza cuando uno usuario reporta el comentario relazado por otro usuario en cualquier publicación. Recibe como parámetros *idreporcom, idreportado* y *razón.* Inicia la conexión y se guarda en *idreportero* la variable de sesión *id\_usuario.* Con **prepare** y **execute** se realiza la sentencia insert sobre la tabla reportes\_comentarios con las variables mencionadas.

public function ReportarComentario($idreporcom, $idreportado, $razon){

                $this->start();

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

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

"INSERT into reportes\_comentarios VALUES(NULL, $idreporcom, $idreportado, $idreportero, '$razon',0,NOW())"

);

$stmt->execute();

$this->stop();

header("Location: Control.php?c=Inicio&a=Inicio");

    }

Método **ListaComentarios**

Se utiliza para devolver una lista de los reportes sobre comentarios que no hayan sido revisados y mostrarlos en la vista del moderador. Es explicado como el método **Lista** en métodos recurrentes.

public function ListaComentarios(){

       $this->start();

$stmt = $this->pdo->prepare("SELECT \* FROM reportes\_comentarios where estatus = 0");

$stmt->execute();

$lista = array();

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

$Reporte = new ReportesComentariosModelo();

$Reporte->set(

$R['id\_reporte'],

                        $R['id\_comentario'],

                        $R['id\_reportado'],

                        $R['id\_reportero'],

                        $R['razon'],

$R['estatus'],

$R['fecha']

);

$lista[] = $Reporte;

}

$this->stop();

return $lista;

   }

Método **ListaCoemntariosUsuario**

Se utiliza para mostrar al usuario y al moderador los reportes que tienen sobre comentarios. Recibe como parámetro *id\_usuario* que es utilizado para la consulta sobre la tabla reportes\_comentario, el método es explicado como método **Lista** en métodos recurrentes.

   public function ListaComentariosUsuario($id\_usuario){

       $this->start();

$stmt = $this->pdo->prepare("SELECT \* FROM reportes\_comentarios where id\_reportado = $id\_usuario");

$stmt->execute();

$lista = array();

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

$Reporte = new ReportesComentariosModelo();

$Reporte->set(

$R['id\_reporte'],

                        $R['id\_comentario'],

                        $R['id\_reportado'],

                        $R['id\_reportero'],

                        $R['razon'],

$R['estatus'],

$R['fecha']

);

$lista[] = $Reporte;

}

$this->stop();

return $lista;

   }

Método **EstatusReporte**

Se utiliza por el usuario moderador para cambiar el estado de los reportes sobre comentarios, se recibe por get los valores de *id\_reporte* y *estatus.* Se prepara una sentencia Update sobre la tabla de reportes comentario cambiando el estado a la variable *estatus,* con**prepare** y después **execute**. Se redirige a inicio de moderador.

public function EstatusReporte(){

$this->start();

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

$estatus = $\_GET['estatus'];

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

"UPDATE reportes\_comentarios set estatus = $estatus where id\_reporte = $id\_reporte "

);

$stmt->execute();

header("location: Control.php?c=Moderador&a=Moderador");

$this->stop();

}

Clase **ReportesControlador**

Se utiliza para el mejor manejo de reportes en general, dentro de los metodos se redirige a los controladores adecuados. Se tienen los métodos **Reportesusuario, ReportaComentario, ReportarPublicacion** y **ReportarUsuario.**

    Class ReportesControlador extends DBConexion

Método **ReportesUsuario**

Es un método vacio que solo cumple función con la vista de ReportesUsuarios.

        public function ReportesUsuario(){}

Método **ReportarComentario**

Se utiliza para registrar los datos en reportes\_comenterios mandando los datos al controlador y registra la acción en notificaciones.

Comprueba si existe una petición post, en caso que si se inicia la conexión y se recibe por post los valores de *razón, idreportado* e *idreporcom*, se crea co como objeto de **ComentarioControlador** y se guarda en *c* lo devuelto por **Comentario.** Se crea re como objeto de **ReportesComentarioControlador** y se llama **ReportarComentario** con los parámetros *idereporcom, idereportado* y *razón*, se crea Noti como objeto de **NotificacionesControlador** se llama **Insert** con los parámetros 7,*idreportado, id\_publicacion* y *idreportado*. Se detiene la conexión y se redirige a inicio.

        public function ReportarComentario(){

            if($\_SERVER['REQUEST\_METHOD']=='POST'){

                $this->start();

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

                $idreportado = $\_POST['id\_usuario'];

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

                $co = new ComentarioControlador();

                $c = $co->Comentario($idreporcom);

$re = new ReportesComentariosControlador();

$re->ReportarComentario($idreporcom, $idreportado, $razon);

$Noti = new NotificacionesControlador();

$Noti->Insert(7,$idreportado,$c->id\_publicacion,$idreportado);

$this->stop();

header("Location: Control.php?c=Inicio&a=Inicio");

            }

        }

Método **ReportarPublicacion**

Cumple la misma función que el método **ReportarComentario** con la excepción de que se hace sobre publicaciones. Se comprueba si existe la petición post, se comienza la conexión y se recibe por medio de post los valores *razón, idreportado, idreportero* y *idreporpub*. Se crea re como objeto de **ReportesPublicacionesControlador** y se llama **ReportarPublicacion** con los parámetros que se recibieron por post. Se crea Noti vomo jeto de **NotificacionesControlador** y se llama **Insert** con un 8 y los valores recibidos por post como parámetro. El archivo redirige a Inicio.

public function ReportarPublicacion(){

            if($\_SERVER['REQUEST\_METHOD']=='POST'){

                $this->start();

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

                $idreportado = $\_POST['id\_artista'];

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

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

                $re = new ReportesPublicacionesControlador();

$re->ReportarPublicacion($idreporpub, $idreportado, $razon);

$Noti = new NotificacionesControlador();

     $Noti->Insert(8,$idreportado,$idreporpub,$idreportado);

$this->stop();

header("Location: Control.php?c=Inicio&a=Inicio");

    }

        }

Método **ReportarUsuario**

Se utiliza cuando se genera un reporte sobre la cuenta de un usuario, realiza lo mismo que el método anterior con la excepción que el objeto es de la clase **ReportesUsuariosControlador**.

        public function ReportarUsuario(){

            if($\_SERVER['REQUEST\_METHOD']=='POST'){

                $this->start();

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

                $idreporusu = $\_POST['id\_usuario'];

                $re = new ReportesUsuariosControlador();

$re->ReportarUsuario($idreporusu, $razon);

$Noti = new NotificacionesControlador();

$Noti->Insert(9,$idreporusu,$idreporusu,$idreporusu);

$this->stop();

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

            }

        }

Clase **ReportesPublicaionesModelo**

Se declaran las variables *id\_reporte, id\_publicacion, id\_reporteado, id\_reportero, razón, estatus* y *fecha*, que corresponden a las columnas de la tabla de reportes\_publicaciones se utilizan como las propiedades de la clase para ser manejada como un objeto con los resultados de la consultas sobre esta tabla.

    class ReportesPublicacionesModelo{

        public $id\_reporte;

        public $id\_publicacion;

        public $id\_reportado;

        public $id\_reportero;

        public $razon;

        public $estatus;

        public $fecha;

Método **set**

Se explica como método recurrente, recibe como parámetro *id\_reporte, id\_publicacion, id\_reporteado, id\_reportero, razón, estatus* y *fecha*

public function set( $id\_reporte, $id\_publicacion, $id\_reportado, $id\_reportero, $razon, $estatus, $fecha){

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

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

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

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

    $this->razon = $razon;

    $this->estatus = $estatus;

    $this->fecha = $fecha;

}

Clase **ReportesPublicacionesControlador**

Se extiende de la clase **DBConexion**, se utiliza para manejar la información entrante y saliente de la tabla de reportes\_publlicaciones. Se tienen los métodos **ReportarPublicacion, ListaPublicaciones, ListaPublicacionesUsuario** y **EstatusReporte.**

    Class ReportesPublicacionesControlador extends DBConexion

Metodo **ReportarPublicacion**

Se utiliza cuando un usuario realiza un reporte sobre alguna publicación, crea un nuevo registro en la tabla de reportes\_publicacion. Recibe como parámetros *idreporpub, idreportado,*y *razón*. Se realiza el mismo procedimiento que el método **Insert** explicado en métodos recurrentes.

 public function ReportarPublicacion($idreporpub, $idreportado,$razon){

            $this->start();

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

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

"INSERT into reportes\_publicaciones VALUES(NULL, $idreporpub, $idreportado, $idreportero, '$razon',0,NOW())"

);

$stmt->execute();

$this->stop();

        }

Método **ListaPublicaciones**

Se utiliza para mostrar todos los reportes de la tabla repostes\_publicaciones que no han sido revisados por el moderador. Es explicado en métodos reurrentes como **Lista** aplicado a la tabla de reportes\_publicaciones donde estatus es igual a 0.

        public function ListaPublicaciones(){

        $this->start();

$stmt = $this->pdo->prepare("SELECT \* FROM reportes\_publicaciones where estatus = 0");

$stmt->execute();

$lista = array();

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

$Reporte = new ReportesPublicacionesModelo();

$Reporte->set(

$R['id\_reporte'],

                        $R['id\_publicacion'],

                        $R['id\_reportado'],

                        $R['id\_reportero'],

                        $R['razon'],

$R['estatus'],

$R['fecha']

);

$lista[] = $Reporte;

}

$this->stop();

return $lista;

   }

Metodo **EstatusReporte**

Se utiliza para manejar las decisiones del moderador. Recibe por get los valores de *id\_reporte* y *estatus,* realiza la misma función explicada en **EstatusReporte** explicada en la clase **ReportesComentariosControlador.**

    public function EstatusReporte(){

$this->start();

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

$estatus = $\_GET['estatus'];

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

"UPDATE reportes\_publicaciones set estatus = $estatus where id\_reporte = $id\_reporte "

);

$stmt->execute();

header("location: Control.php?c=Moderador&a=Moderador");

$this->stop();

}

Clase **ReportesUsuariosModelo**

Es utilizada para generar objetos con las propiedades de la tabla de reportes\_usuarios por lo que se declaran las variables *id\_reporte, id\_reportado, id\_reportero, razón, estatus* y *fecha,* que corresponden a las tablas.

    class ReportesUsuariosModelo{

        public $id\_reporte;

        public $id\_reportado;

        public $id\_reportero;

        public $razon;

        public $estatus;

        public $fecha;

Método **set**

Recibe como parámetros *id\_reporte, id\_reportado, id\_reportero, razón, estatus* y *fecha,* asigna los valores recibidos por parámetros a las propiedades de la clase.

public function set( $id\_reporte, $id\_reportado, $id\_reportero, $razon, $estatus, $fecha){

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

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

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

    $this->razon = $razon;

    $this->estatus = $estatus;

    $this->fecha = $fecha;

}

Clase **ReportarUsuarioControlador**

Se hereda de la clase **DBConexion** y se utiliza para el manejo de los registros de la tabla reportes\_usuarios. Tiene los métodos **ReportaUsuario, ListaUuarios** y **EstatusReporte.**

    Class ReportesUsuariosControlador extends DBConexion

Método **ReportarUsuario**

Se utiliza para generar un nuevo registro en la tabla de reposrtes\_usuarios. Recibe por parámetro *idreporusu* y *razón*. Se realiza el mismo procedimiento que el método **Insert** explicado en métodos recurrentes.

        public function ReportarUsuario($idreporusu, $razon){

                $this->start();

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

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

"INSERT into reportes\_usuarios VALUES(NULL, $idreporusu, $idreportero, '$razon',0,NOW())"

);

$stmt->execute();

$this->stop();

        }

Método **ListaUsuarios**

Se utiliza para mostrar en la interfaz de moderador los reportes realizados sobre usuarios que no hayan sido revisados por el moderador. Se explica como método **Lista** en métodos recurrentes.

        public function ListaUsuarios(){

       $this->start();

$stmt = $this->pdo->prepare("SELECT \* FROM reportes\_usuarios where estatus = 0");

$stmt->execute();

$lista = array();

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

$Reporte = new ReportesUsuariosModelo();

$Reporte->set(

$R['id\_reporte'],

                        $R['id\_reportado'],

                        $R['id\_reportero'],

                        $R['razon'],

$R['estatus'],

$R['fecha']

);

$lista[] = $Reporte;

}

$this->stop();

return $lista;

   }

Método **EstatusReporte**

Se utiliza para modificar el reporte sobre el usuario ya sea aprobarlo o denegarlo. Se explica en la clase **ReporteComentariosControlador** como **EstatusReporte** con la diferencia que se aplica a la tabla de reportes\_usuarios.

public function EstatusReporte(){

$this->start();

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

$estatus = $\_GET['estatus'];

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

"UPDATE reportes\_usuarios set estatus = $estatus where id\_reporte = $id\_reporte "

);

$stmt->execute();

header("location: Control.php?c=Moderador&a=Moderador");

$this->stop();

}

Clase **RetoAeptadoModelo**

Se utiliza para manejar los datos recuperados de las consultas sobre la tabla de retos\_aceptados. Se declaran *id\_aceptado, id\_artista* y *id\_reto* como variables públicas.

    class RetoAceptadoModelo{

        public $id\_aceptado;

        public $id\_artista;

        public $id\_reto;

Metodo **set**

Recibe como parametros *id\_aceptado, id\_artista* y *id\_reto,*  se explica cómo **set** en métodos recurrentes.

public function set( $id\_aceptado, $id\_artista, $id\_reto){

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

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

    $this->id\_reto = $id\_reto;

}

Clase **RetoAceptadoControlador**

Se extiende de la clase **DBConexion** y se utiliza para majear los datos de la tabla de retos\_aceptados. Contiene los métodos **AceptarRetos** y **RetosAceptados.**

    Class RetoAceptadoControlador extends DBConexion

Método **AceptarReto**

Se utiliza cuando el usuario sube una imagen al reto de la semana, crea un registro en retos\_aceptados. Se recibe por parámetro *id\_reto* y se realiza la misma función que el método **Insert** explicado en métodos recurrentes.

public function AceptarReto($id\_reto){

$this->start();

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

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

"INSERT into retos\_aceptados VALUES(NULL,$id\_artista,$id\_reto)"

);

$stmt->execute();

$this->stop();

}

Método **RetosAceptados**

Devuelve una lista de objeto tipo **RetoAceptadoModelo** para mostrar los retos hechos por el usuario.Se realiza el mismo procedimiento que el método **Lista** explicado en métodos recurrentes.

public function RetosAceptados(){

$this->start();

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

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

"SELECT \* FROM retos\_aceptados where id\_artista = $id\_artista"

);

$stmt->execute();

$lista = array();

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

$Retos = new RetoAceptadoModelo();

$Retos->set(

$Reto["id\_reto"],

$Reto["id\_artista"],

$Reto["id\_reto"]

);

$lista[] = $Retos;

}

$this->stop();

return $Lista;

}

Clase **RetoModelo**

Tiene las propiedades de *id\_reto, fecha* y *descripción* que corresponden a los nombres de las columnas de la tabla reto. Se utiliza para manejar los resultados de las consultas sobre esta tabla.

    class RetoModelo{

        public $id\_reto;

        public $fecha;

        public $descripcion;

Método **set**

Recibe como parámetros *id\_reto, fecha* y *descripción,* es explicado en la sección de métodos recurrentes.

public function set( $id\_reto, $fecha, $descripcion){

    $this->id\_reto = $id\_reto;

    $this->fecha = $fecha;

    $this->descripcion = $descripcion;

}

Clase **RetoControlador**

Es utilizado para manejar la información de la tabla retos e imagen\_reto. Se hereda de la clase **DBConexion** y contiene los métodos **Reto** y **SubirReto.**

    Class RetoControlador extends DBConexion

Método **Reto**

Se utiliza para mostrar el ultimo reto en la vista de reto. Realiza con **prepare** y **execute** una consulta donde el valor de fecha sea mayor o igual a la fecha actual y se ordena de manera descendiente por fecha. Se guardan los resultados obtenido con **fetch** en *Reto*, se crea Retos como objeto de **RetoModelo** y utilizando **set** se guardan los valores de *Reto.* Se devulve Retos.

public function Reto(){

    $this->start();

     $stmt = $this->pdo->prepare(

"SELECT \* FROM retos where fecha >= NOW() order by fecha DESC"

);

$stmt->execute();

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

$Retos = new RetoModelo();

$Retos->set(

$Reto["id\_reto"],

$Reto["fecha"],

$Reto["descripcion"]

);

$this->stop();

return $Retos;

}

Método **SubirReto**

Se utiliza para subir la imagen que el usuario realice como reto a la carpeta correspondiente y a la base de datos. Comprueba si existe la patecion post, en caso de que si recibe los datos de la imagen y utilizando **move\_uploaded\_file** se mueve a la carpeta ./Imágenes/imgReto, si esto se realizo correctamente se crea un objeto **RetoAceptadoControlador** para llamar a **AceptarReto** con el parámetro de *id\_reto* recibido por post, lo recuperado se inserta en la base de datos junto con la dirección del archivo, se redirige a reto.

public function SubirReto(){

if($\_SERVER['REQUEST\_METHOD']=='POST'){

$this->start();

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

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

if(!move\_uploaded\_file( $tmp\_name,"$folder".$\_FILES["image"]["name"])){

echo "error;";

}else{

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

$RetoA = new RetoAceptadoControlador();

$ra = $RetoA->AceptarReto($id\_reto);

$id\_aceptado = $ra->id\_aceptado;

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

"INSERT into imagen\_reto VALUES(NULL,'$folder".$\_FILES["image"]["name"]."',$id\_aceptado)"

);

$stmt->execute();

$this->stop();

header("Location: Control.php?c=Reto&a=Reto");

}

}

     }