Desarrollo Web en Entorno Servidor

Programación en PHP

José A. Lara

Sistema MVC

- ¿Por qué debemos programar con orientación a objetos?
- ¿Por qué debemos utilizar un patrón tipo Modelo Vista Controlador (MVC)?

Es importante contestar a estas preguntas antes de comenzar a utilizar un patrón y un cambio de metodología de programación como el MVC ya que en un principio resulta **más aparatoso o costoso** utilizarlo.

Sistema MVC

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

Sistema MVC

¿Por qué utilizar el patrón MVC?

- ¿Por qué debemos programar orientado a objetos? ¿Por qué debemos utilizar un patrón tipo Modelo Vista Controlador (MVC)?
- Es importante contestar a estas preguntas antes de comenzar a utilizar un patrón y un cambio de metodología de programación como el MVC ya que en un principio resulta más aparatoso o costoso utilizarlo.

Sistema MVC

¿Por qué utilizar el patrón MVC?

- Si desarrollamos una ÚNICA página que se conecta a una ÚNICA tabla dentro de la base datos, de una ÚNICA base de datos, seguramente no tendrá ningún sentido ni la utilización del patrón MVC ni la utilización de objetos.
- Pero actualmente el desarrollo de software ha crecido y se ha hecho muy muy complejo, y una aplicación no se basa en una sola página, ni en 10 ni en 20, tampoco se basa en una base de datos de 2 ó 3 tablas, y por último tampoco nos encontramos habitualmente.

Sistema MVC

¿Por qué utilizar el patrón MVC?

El patrón MVC da respuesta a las problemáticas planteadas:

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

Sistema MVC

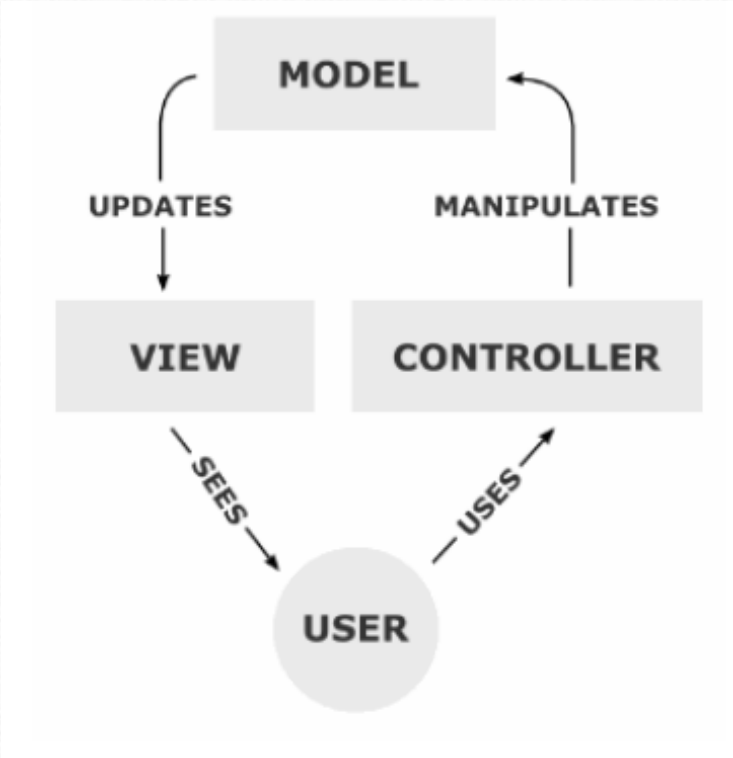
Patrón MVC

- El MVC propone la construcción de **tres componentes distintos** que son:
  - Modelo
  - Vista
  - Controlador

Este patrón de arquitectura de software se basa en las ideas de **reutilización de código** y la **separación de conceptos**, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

Sistema MVC

Patrón MVC



Sistema MVC

Patrón MVC: Modelo

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

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

Sistema MVC

Patrón MVC: Vista

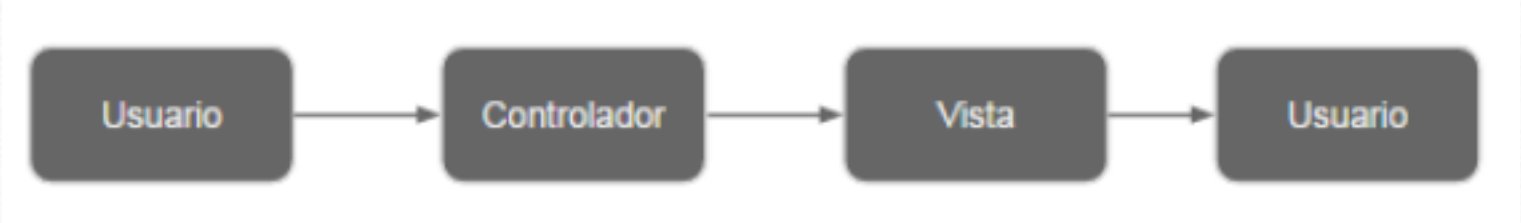
Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

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

Sistema MVC

Patrón MVC: Controlador

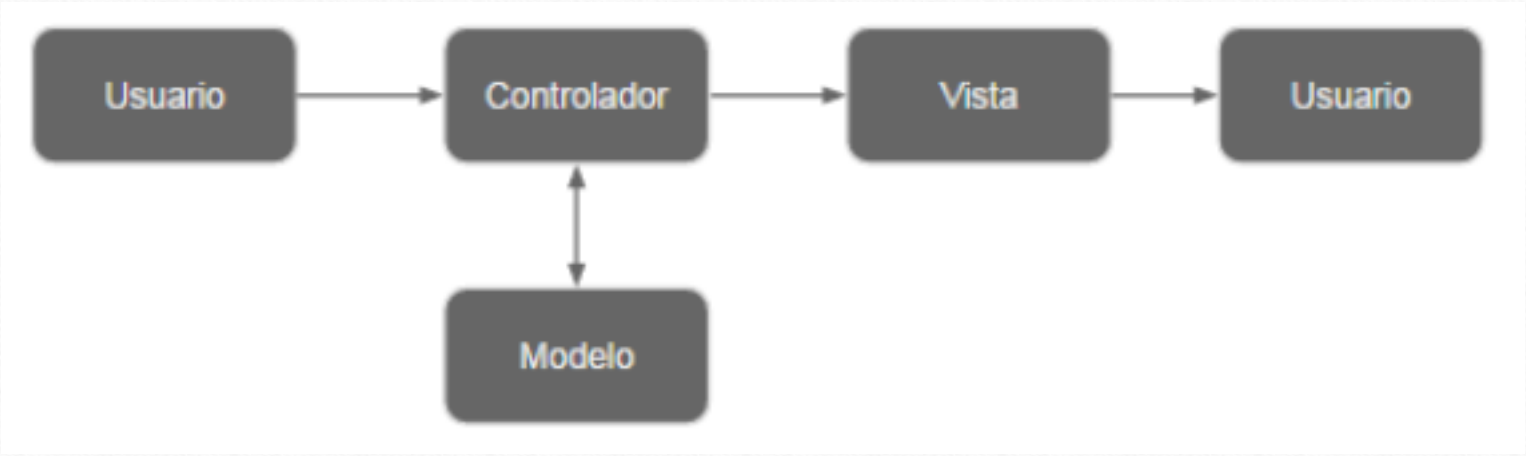
Es quizá la parte más compleja tanto de entender como de programar, ya que es la parte que más complicada se hace abstraer. En un principio, podríamos pensar que únicamente realiza las labores de enrutador, es decir, recibe las peticiones de los usuarios y redirige hacia donde corresponda:



Sistema MVC

Patrón MVC: Controlador

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



Sistema MVC

Patrón MVC

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

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

Sistema MVC

Patrón MVC: Ejemplo con PHP

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

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

Sistema MVC

Ejemplo

***Ver los archivos en la carpeta mvc*

Sistema MVC

Vista y Formularios

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

Sistema MVC

Vista y Formularios

Primer paso. Estructura de ficheros modelo, vista de resultados: en primer lugar generaremos en la parte de modelo para interactuar con la base de datos. Como podemos observar (***nba_db.php***):

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

Sistema MVC

Vista y Formularios

Una vez tenemos este código podemos realizar la parte de VISTA, y en primer lugar implementaremos la visualización de los datos obtenidos para el listado de la base de datos (***list_equipos_conferencia_mvc.php***)

Sistema MVC

Vista y Formularios

Segundo paso. Formularios

¿Cómo incorporar interacción con el usuario? Los formularios son la clave en esto. (***mvcForms/index.php***), hemos incorporado un formulario que recoge nuestras peticiones. Esto permitirá que nos llegue el parámetro elegido a nuestro listadoEquipos.php, lo que nos permitirá seleccionar la conferencia, y por lo tanto deberíamos cambiar el método.

Sistema MVC

Ejercicio

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

1. Para nuestro ejemplo necesitaremos una nueva base de datos:
    a. Con la ayuda de PHPMyAdmin crearemos una base de datos denominada escuela.
    b. Crearemos una nueva tabla denominada alumnos con los siguientes campos:
        i. Nombre, varchar 64
        ii. Apellidos, varchar 128
        iii. Edad, int

Sistema MVC

Ejercicio

2. Crearemos una nueva carpeta para nuestro proyecto, y dentro de este:
    a. listado.php
    b. lib/escuela.php
3. Utilizando como modelo el ejemplo anterior llamado dbNBA, generaremos una nueva clase denominada Escuela.

Sistema MVC

CRUD

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

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

Sistema MVC

CRUD

Después de haber aprendido a realizar consultas con una base de datos, y comprender y crear un patrón MVC, es muy sencillo comprender y utilizar el resto de acciones.

Sistema MVC

CRUD

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

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

Sistema MVC

CRUD

mvcCRUDClase/dbUsuarios.php

```php
//Funciones para la insercion
function insertarUsuario($nombre,$apellidos,$edad){
  $sqlInsercion="INSERT INTO usuario(id,nombre,apellidos,edad)
      VALUES (NULL, '".$nombre."', '".$apellidos."', ".$edad.")";
  $this->conexion->query($sqlInsercion);
}
```

Sistema MVC

CRUD

Segundo paso. Incluimos los formularios necesarios para enviar la información

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

Sistema MVC

Ejercicio

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

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

Sistema MVC

CRUD

Tercer paso. Update y delete

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

```php
//Funciones para la insercion
function actualizarUsuario($id,$nombre,$apellidos,$edad){
  $sqlInsercion="UPDATE usuario
  SET nombre='".$nombre."',apellidos='".$apellidos."',edad=".$edad.
  " WHERE id=".$id;
  $this->conexion->query($sqlInsercion);
}
```

Sistema MVC

CRUD

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

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

Sistema MVC

Funciones, POO y MVC

Parámetros por defecto

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

Sistema MVC

Funciones, POO y MVC

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

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

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

Sistema MVC

Ejercicio

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

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

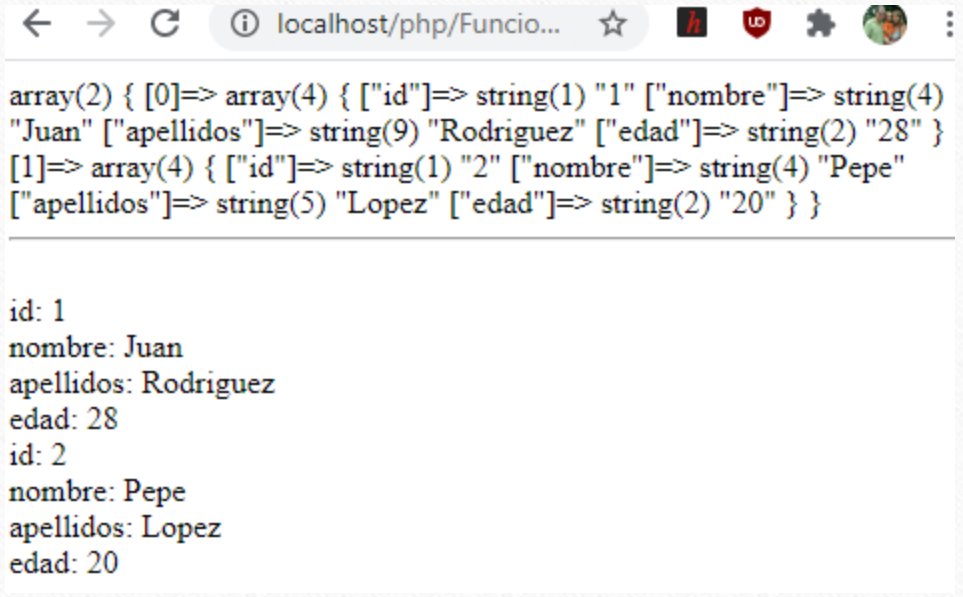
Sistema MVC

MVC y Funciones

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

Sistema MVC

MVC y Funciones




```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Funciones y MVC</title>
  </head>
  <body>
    <?php
    //Incluimos librerias
    include "dbUsuarios.php";
    $usuarios=new dbUsuarios();

    //DEvolverá el usuario 1
    $tabla=$usuarios->devolverUsuarios();
    var_dump($tabla);
    echo "</br><hr></br>";
    foreach ($tabla as $fila) {
      echo "id: ".$fila["id"]."</br>";
      echo "nombre: ".$fila["nombre"]."</br>";
      echo "apellidos: ".$fila["apellidos"]."</br>";
      echo "edad: ".$fila["edad"]."</br>";
    }
    ?>
  </body>
</html>
```

Sistema MVC

MVC y Funciones

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

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

Sistema MVC

MVC y Funciones

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

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

Sistema MVC

MVC y Funciones

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

Sistema MVC

Herencia y MVC

La herencia facilita la creación de objetos a partir de otros ya existentes e implica que una subclase obtiene todo el comportamiento (métodos) y eventualmente los atributos (variables) de su superclase.

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

Sistema MVC

Herencia y MVC: Aplicación al modelo

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

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

Sistema MVC

Herencia y MVC: Aplicación al modelo

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

Sistema MVC

Herencia y MVC: Aplicación al modelo

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

Sistema MVC

Herencia y MVC: Aplicación al modelo

Vemos en el anterior código dos nuevas modificaciones:

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

Sistema MVC

Herencia y MVC: Aplicación al modelo

Aquí aparecen nuestras consultas SQL, ya que tiene sentido que sea en la clase que va a estar relacionada con la tabla en cuestión la que tenga la información de cómo manejar dicha información. Por otro lado, necesita de la clase padre db, para poder realizar las conexiones a través del conector y de una conexión ya abierta.

```php
<?php
    include "db.php";
    /** * */
    class Usuario extends db {
        function __construct() {
        //De esta forma realizamos la conexion a la base de datos
            parent::__construct();
        }
        //Devolvemos todos los usuarios
        function devolverUsuarios(){
        //Construimos la consulta
            $sql="SELECT * from usuario";
            //Realizamos la consulta
            $resultado=$this->realizarConsulta($sql);
            if($resultado!=null){
            //Montamos la tabla de resultados
                $tabla=[];
                while($fila=$resultado->fetch_assoc()){
                 $tabla[]=$fila;
                }
                return $tabla;
            }else{
                return null;
            }
        }
    }
?>
```

Sistema MVC

Ejercicio

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

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