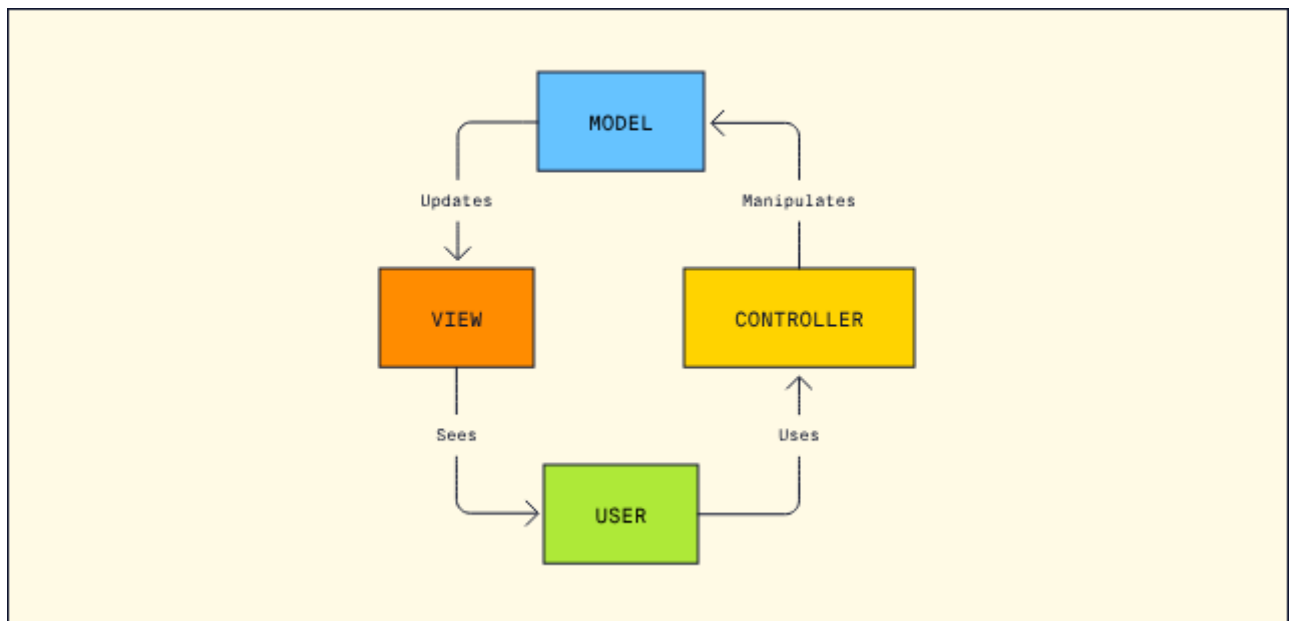


MODELO VISTA CONTROLADOR (MVC)



ALUMNO: DARIO ALEXANDER QUINDE QUIÑONEZ
ASIGNATURA: DESARROLLO WEB EN ENTORNO SERVIDOR
CICLO FORMATIVO: DESARROLLO DE APLICACIONES WEB
CURSO: 2ºDAW

ÍNDICE

1.Introducción.....	1
2.Estructura del proyecto.....	2
3. .htaccess (Rutas Amigables).....	4
4.Modelos.....	5
• 4.1 Clase Entidad.....	5
• 4.2 Clase Libro.....	6
• 4.3 Clase Disco.....	7
• 4.4 Clase Pelicula.....	8
5. Conexión a base de datos.....	9
6. Mapeo de objeto-relacional.....	10
7. Manipulación de datos databasefunctions.....	11
8 Rutas y constantes.....	13
9. Funcionalidad de la aplicación.....	14

1. INTRODUCCIÓN

Configuración inicial:

Para probar la aplicación, nos deberemos de asegurar de copiar la carpeta con todos los archivos necesarios en la raíz de la carpeta "htdocs" del entorno XAMPP.

Esto es importante para el funcionamiento de la pa.

Permisos de archivos:

Asegúrate de asignar los permisos correctos para la subida y bajada de imágenes. Esto es fundamental para que la aplicación pueda gestionar archivos de manera efectiva.

Documento SQL:

Antes de comenzar a utilizar la aplicación, se debe realizar la importación del documento SQL proporcionado. Esto es importante para la recopilación de datos y la importación de datos

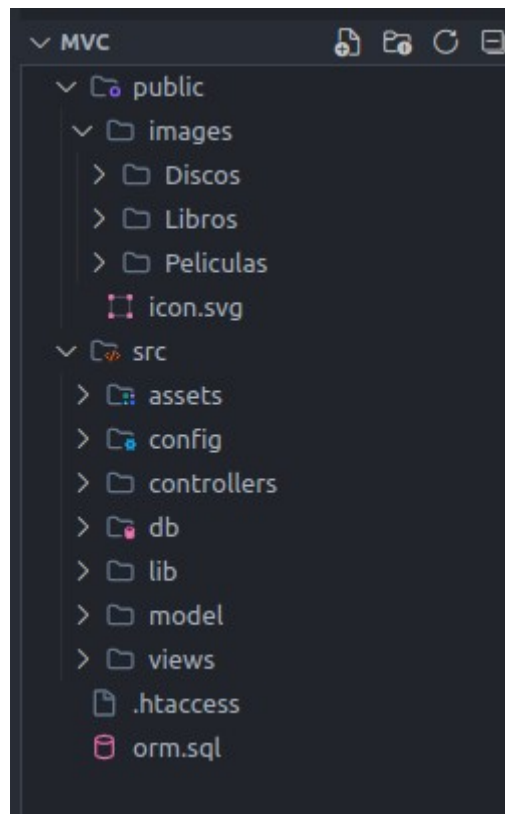
Modelo Vista Controlador (MVC):

Hemos implementado el patrón de diseño Modelo-Vista-Controlador en nuestra aplicación. Este enfoque separa la lógica de la aplicación en tres componentes principales: el Modelo (encargado de la gestión de datos), la Vista (encargada de la presentación) y el Controlador (gestiona la interacción entre el Modelo y la Vista). Este diseño facilita la escalabilidad y el mantenimiento de la aplicación.

Rutas Amigables con .htaccess: En nuestra aplicación, hemos implementado Rutas Amigables utilizando el archivo .htaccess. Aunque solo contamos con páginas principales como "home", "libros", "películas" y "discos", el archivo .htaccess nos permite controlar y gestionar estas rutas de manera efectiva. No encontrarás un archivo "index" en cada carpeta; en su lugar, utilizamos reglas de reescritura para manejar las distintas secciones de la aplicación.

Comunicación con la Base de Datos: Para la comunicación con la base de datos, hemos creado un Object-Relational Mapping (ORM). Este ORM nos permite realizar operaciones en la base de datos de manera sencilla y eficiente. Podemos actualizar, listar y realizar todas las consultas necesarias utilizando el ORM integrado en nuestra aplicación.

2. ESTRUCTURA DEL PROYECTO



public: En esta carpeta , almacenamos todas las imágenes del servidor , con su respectiva carpeta de cada modelo , para una mayor organización y legibilidad.

src : La estructura del proyecto sigue el patrón de diseño Modelo Vista Controlador (MVC), una arquitectura que organiza el código en tres componentes principales para mejorar la modularidad y la mantenibilidad del sistema. Vamos a analizar cada parte del proyecto en relación con el MVC, en src es donde implementamos esta arquitectura:

1. Modelo (model):

- **conexion.inc.php:** Gestiona la conexión a la base de datos.
- **`disco.inc.php`, `libro.inc.php`, `pelicula.inc.php`:** Representan entidades específicas con métodos relacionados.
- **`entidad.inc.php`:** Define una interfaz común para las entidades.
- **`orm.inc.php`:** Implementacion de Object-Relational Mapping (ORM) para interactuar con la base de datos.

2. Vista (`views`):

- **Archivos `.view.php`:** Contienen la presentación y estructura de las páginas web asociadas a cada controlador. Cada controlador tiene su vista correspondiente.

3. Controlador (`controllers`):

- **Archivos `.controller.php`**: Manejan la lógica de la aplicación y la interacción entre el modelo y la vista.

- **`404.controller.php`**: Controlador para manejar páginas no encontradas.

- **`add.controller.php`, `discos.controller.php`, `edit.controller.php`, `home.controller.php`, `libros.controller.php`, `peliculas.controller.php`**: Controladores asociados a funciones específicas de la aplicación.

**** Otros Componentes de src importantes en la aplicación:**

- Configuración (`config`):

- **`const.inc.php`**: Contiene constantes globales para el proyecto.

- **`database.inc.php`**: Configuración de la base de datos.

- Manejo de Base de Datos (`db`):

- **`database.functions.php`**: Funciones relacionadas con la base de datos la creación de queries , placeholder etc...

- Librerías (`lib`):

- **Archivos `.functions.inc.php`**: Contienen funciones utilitarias para diversos propósitos como manipulación de datos, manejo de imágenes, y paginación.

- Recursos Estáticos (`assets`):

- **Directorio `css`**: Contiene archivos CSS asociados a la presentación de diferentes vistas.

- Vistas Compartidas (`views`):

- **`nav.inc.php`**: Encabezado de la página común a todas las vistas.

En resumen, esta estructura organiza el proyecto de manera clara y separa las responsabilidades de presentación, lógica de la aplicación y acceso a datos, siguiendo el patrón MVC para una mejor mantenibilidad y escalabilidad. Cada componente desempeña un papel específico, facilitando la comprensión y el desarrollo del proyecto.

3. .HTACCESS (RUTAS AMIGABLES)

El archivo .htaccess se implementa en PHP con el objetivo de facilitar la implementación de rutas amigables en una aplicación web. Estas rutas amigables son una técnica que permite tener URLs más limpias y comprensibles, tanto para los usuarios como para los motores de búsqueda. En lugar de tener URLs con parámetros y extensiones, se utilizan rutas más descriptivas. El código proporcionado utiliza el módulo mod_rewrite de Apache para reescribir las URLs y direccionarlas hacia controladores específicos en una arquitectura MVC (Modelo-Vista-Controlador). Aquí está la explicación de cada parte del código:

Estas líneas habilitan el seguimiento de enlaces simbólicos y activan el motor de reescritura de Apache.

```
1 Options +FollowSymLinks
2 RewriteEngine On
```

```
RewriteRule ^$ src/controllers/home.controller.php [L]
RewriteRule ^Home src/controllers/home.controller.php [L]
```

Estas reglas manejan la ruta base y la ruta '/Home', rediriéndolas al controlador 'home.controller.php'.

```
RewriteRule ^Peliculas/([0-9]+) src/controllers/peliculas.controller.php?page=$1 [L]
RewriteRule ^Peliculas src/controllers/peliculas.controller.php [L]

#Paginacion de discos que funcione con el controlador discos
RewriteRule ^Discos/([0-9]+) src/controllers/discos.controller.php?page=$1 [L]
RewriteRule ^Discos src/controllers/discos.controller.php [L]
#Paginacion de libros que funcione con el controlador libros
RewriteRule ^Libros/([0-9]+) src/controllers/libros.controller.php?page=$1 [L]
RewriteRule ^Libros src/controllers/libros.controller.php [L]
```

Estas reglas manejan las rutas relacionadas con películas, discos y libros incluyendo paginación. La primera regla permite la paginación, mientras que la segunda maneja la ruta base.

```
#Seccion de editar el primer parametro sera la clase y el segundo el id
RewriteRule ^Editar/([a-zA-Z]+)/([0-9]+) src/controllers/edit.controller.php?class=$1&id=$2 [L]
#Seccion de editar se le pasa el parametro de la clase
```

Esta regla maneja la ruta para editar, donde el primer parámetro es la clase y el segundo es el ID.

```
#Seccion de añadir se le pasa el parametro de la clase
RewriteRule ^Anadir/([a-zA-Z]+) src/controllers/add.controller.php?class=$1 [L]
# Excluir archivos y directorios reales de la reescritura
```

Esta regla maneja la ruta para añadir, donde el parámetro es la clase.

```
RewriteRule ^Anadir/([a-zA-Z]+)/ src/controllers/add.controller.php?class=$1 [L]
# Excluir archivos y directorios reales de la reescritura
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

# Redirigir cualquier otra solicitud a tu controlador principal o página de error
RewriteRule ^(.*)$ src/controllers/404.controller.php
```

4. MODELOS

4.1 CLASE ENTIDAD

Esta clase abstracta, llamada Entidad, sirve como la base para las clases concretas Libro, Pelicula y Disco.

Clase Abstracta: La clase Entidad es abstracta, lo que significa que no se pueden crear instancias directas de esta clase. Su propósito es proporcionar una estructura común y propiedades compartidas, espera que las clases hijas concretas proporcionen implementaciones específicas.

Atributos: La clase tiene varios atributos que representan propiedades comunes de las entidades multimedia, como el **id**, **título**, **año**, **publicacion**, **genero**, **imagen** y **estado**.

Constructor: El constructor `__construct` se encarga de inicializar estos atributos. Se utiliza una sintaxis que permite valores predeterminados, por lo que si no se proporciona un valor para un atributo, se establecerá en un valor predeterminado (por ejemplo, cadena vacía para strings y 0 para enteros).

```
// Encapsulamos los atributos de la clase
3 references | 3 overrides
public function __construct(int $id = 0, string $titulo = '', DateTime $anio = null, string $publicacion = '', string $genero = '', string $imagen = '', bool $estado = true)
{
    $this->id = $id;
    $this->titulo = $titulo;
    $this->anio = $anio;
    $this->publicacion = $publicacion;
    $this->genero = $genero;
    $this->imagen = $imagen;
    $this->estado = $estado;
}
```

Métodos Getter y Setter: Se proporcionan métodos getter para obtener los valores de los atributos y métodos setter para establecerlos. Los métodos final indican que no pueden ser sobrescritos por clases hijas.

Método `__toString`: Se implementa un método `__toString` que devuelve una representación de cadena de la entidad, incluyendo el id, título, año, publicación y género.

4.2 CLASE LIBRO

Herencia de Entidad: La clase Libro extiende la clase abstracta Entidad. Esto significa que hereda las propiedades y métodos de la clase Entidad.

La clase Entidad sirve como una estructura base común para todas las entidades multimedia, mientras que la clase Libro agrega propiedades específicas de un libro.

Atributos Adicionales: Además de los atributos heredados de Entidad (como id, título, año, publicación, etc.), la clase Libro introduce atributos específicos para un libro, como autor, páginas e ISBN.

Constructor: La clase Libro tiene un constructor que inicializa sus atributos, utilizando la llamada al constructor de la clase padre (Entidad) para manejar los atributos comunes. Se establecen valores predeterminados para los atributos en caso de que no se proporcionen al crear una instancia de la clase.

```
3 references | 0 overrides | prototype
public function __construct(int $id = 0, string $titulo = '', DateTime $anio = null, string $publicacion = '', string $genero = '', string $imagen = '', bool $estado = true,
                             string $autor = '', int $paginas = 0, string $isbn = '')
{
    parent::__construct($id, $titulo, $anio, $publicacion, $genero, $imagen, $estado);
    $this->autor = $autor;
    $this->paginas = $paginas;
    $this->isbn = $isbn;
}
```

Métodos Getter y Setter: La clase proporciona métodos getter para obtener los valores de los atributos específicos de Libro (autor, páginas e ISBN) y métodos setter para modificar esos valores.

Método __toString: Se implementa un método __toString que devuelve una representación de cadena de la entidad Libro, incluyendo los atributos heredados de Entidad y los atributos específicos de Libro (autor, páginas e ISBN).

4.3 CLASE DISCO

Herencia de Entidad: La clase Disco extiende la clase abstracta Entidad , permitiendo compartir propiedades y métodos comunes definidos en Entidad.

Atributos Adicionales: Además de los atributos heredados de Entidad (como id, título, año, etc.), la clase Disco para representar información relacionada con discos, tales como el artista, la duración y el código ISWC.

Constructor: La clase cuenta con un constructor que acepta parámetros para inicializar los atributos tanto heredados como propios de Disco.

```
public function __construct(int $id = 0, string $titulo = '', DateTime $anio = null, string $publicacion = '', string $genero = '', string $imagen = '', bool $estado = true,
                           string $artista = '', int $duracion = 0, string $iswc = '') {
    // Llamada al constructor de la clase padre
    parent::__construct($id, $titulo, $anio, $publicacion, $genero, $imagen, $estado);
    $this->artista = $artista;
    $this->duracion = $duracion;
    $this->iswc = $iswc;
}
```

Métodos Getter y Setter: Se proporcionan métodos getter y setter para acceder y modificar los valores de los atributos específicos de Disco, como artista, duración e ISWC.

Método __toString: La clase implementa el método mágico __toString para generar una representación de cadena de la entidad Disco.

4.4 CLASE PELÍCULA

Herencia de Entidad: La clase Película extiende la clase abstracta Entidad, estableciendo una relación de herencia para compartir propiedades y métodos comunes.

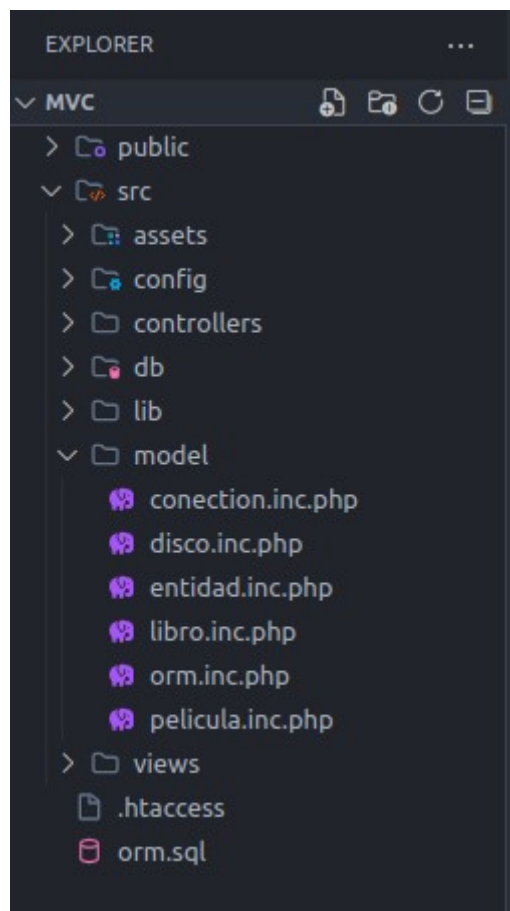
Atributos Adicionales: Además de los atributos heredados de Entidad, Película introduce atributos específicos para representar información relacionada con películas, como el director, el reparto, la duración y el código ISAN.

Constructor: La clase cuenta con un constructor que acepta parámetros para inicializar tanto los atributos heredados como los propios de Película.

```
3 references | 0 overrides | prototype
public function __construct(int $id = 0, string $titulo = '', DateTime $anio = null, string $publicacion = '', string $genero = '', string $imagen = '',
    bool $estado = true, string $director = '', string $reparto = '', int $duracion = 0, string $isan = '')
{
    parent::__construct($id, $titulo, $anio, $publicacion, $genero, $imagen, $estado);
    $this->reparto = $reparto;
    $this->director = $director;
    $this->duracion = $duracion;
    $this->isan = $isan;
}
```

Métodos Getter y Setter: Se proporcionan métodos getter y setter para acceder y modificar los valores de los atributos específicos de Película, como director, reparto, duración e ISAN. Estos métodos permiten un acceso controlado a los datos de la instancia.

Método __toString: La clase implementa el método mágico __toString para generar una representación de cadena de la entidad Película.



5. CONEXIÓN A BASE DE DATOS

Esta clase se encuentra en [src/model/connection.inc.php](#)

Esta clase abstracta proporciona funcionalidades para gestionar la conexión a la base de datos mediante PDO en una aplicación PHP.

Explicación de la clase Conection:

Atributo Estático: La clase tiene un atributo privado y estático llamado ``$connection``, que almacena la instancia de la conexión a la base de datos. Ser estático asegura que haya solo una instancia de la conexión compartida entre todas las instancias de la clase

Métodos:

Método ``openConnection``: Este método abre la conexión a la base de datos utilizando la configuración proporcionada en las constantes definidas en [/src/config/database.inc.php](#)

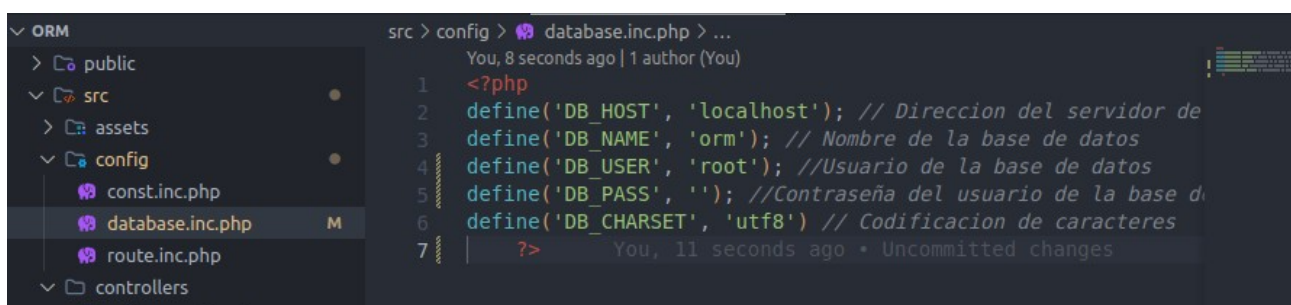
Se utiliza PDO para establecer la conexión y se configura para gestionar errores y establecer el conjunto de caracteres.

Método ``closeConnection``: Este método cierra la conexión a la base de datos. La conexión se establece a ``null``, indicando que no hay una conexión activa.

Método ``getConnection``: Este método estático permite obtener la instancia de la conexión a la base de datos. Es un getter para el atributo ``$connection``.

Método ``getNextAutoIncrement``: Este método estático devuelve el próximo valor AUTO_INCREMENT para una tabla específica. Utiliza una consulta SQL para obtener esta información desde la base de datos, y se espera que se proporcione el nombre de la base de datos y de la tabla como parámetros.

Constantes: Los valores como ``DB_HOST``, ``DB_NAME``, ``DB_USER``, ``DB_PASS``, y ``DB_CHARSET`` son constantes que deben estar definidas en [/src/config/database.inc.php](#). Estas constantes contienen la información necesaria para la conexión a la base de datos, como el nombre de host, nombre de la base de datos, usuario, contraseña y conjunto de caracteres.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders 'public', 'src', 'assets', 'config', and 'controllers'. Under 'config', there are files 'const.inc.php', 'database.inc.php' (marked with an 'M' for modified), and 'route.inc.php'. The code editor shows the content of 'database.inc.php' with the following code:

```
src > config > database.inc.php > ...
You, 8 seconds ago | 1 author (You)

1 <?php
2 define('DB_HOST', 'localhost'); // Direccion del servidor de
3 define('DB_NAME', 'orm'); // Nombre de la base de datos
4 define('DB_USER', 'root'); //Usuario de la base de datos
5 define('DB_PASS', ''); //Contraseña del usuario de la base d
6 define('DB_CHARSET', 'utf8') // Codificacion de caracteres
7 | ?> You, 11 seconds ago • Uncommitted changes
```

6. MAPEO OBJETO-RELACIONAL

La clase ORM (Mapeo Objeto-Relacional) es responsable de gestionar la comunicación entre los objetos de la aplicación y la base de datos.

Explicación de los métodos:

Método ``persist``: Este método se utiliza para insertar un objeto en la base de datos. Abre la conexión, asigna un nuevo ID al objeto utilizando el método ``getNextAutoIncrement`` de la clase ``Connection``, inicia una transacción, construye y ejecuta una consulta preparada para realizar la inserción, y finalmente realiza el commit de la transacción.

Método ``findAll``: Este método devuelve todos los objetos de una tabla específica. Abre la conexión, prepara y ejecuta una consulta simple para seleccionar todos los registros de la tabla, y devuelve el resultado como un array de asociaciones.

Método ``find``: Este método busca y devuelve un objeto específico de una tabla basándose en su ID. Abre la conexión, prepara y ejecuta una consulta preparada para seleccionar el registro con el ID proporcionado, y devuelve el objeto correspondiente.

Método ``updateState``: Actualiza el estado de un objeto en la base de datos, básicamente un borrado virtual.

Abre la conexión, inicia una transacción, construye y ejecuta una consulta preparada para actualizar el estado del objeto, y realiza el commit de la transacción.

Método ``flush``: Actualiza un objeto completo en la base de datos. Abre la conexión, inicia una transacción, obtiene los atributos del objeto, construye y ejecuta una consulta preparada para actualizar el objeto en la base de datos, y realiza el commit de la transacción.

Los métodos utilizan sentencias preparadas y PDO para garantizar la seguridad y prevenir inyecciones SQL.

La conexión a la base de datos se maneja mediante la clase ``Connection``, que proporciona funcionalidades básicas para abrir, cerrar y obtener la conexión.

Se utilizan varias funciones auxiliares definidas en archivos externos (como ``db.functions.inc.php`` y ``database.functions.php``) para realizar tareas específicas, como la construcción de consultas SQL y la manipulación de datos.

7. MANIPULACIÓN DE DATOS DATABASEFUNCTIONS

Esta clase se encuentra en [src/db/database.functions.php](#)

La clase encapsula funciones específicas de la base de datos, métodos para buscar objetos por ID, actualizar estados y objetos, y gestionar la redirección en la interacción con la base de datos.

En esta implementación, los controladores de la vista (por ejemplo, las páginas de Películas, Discos y Libros) no interactúan directamente con la clase ORM, sino que utilizan esta clase para gestionar la interacción con la base de datos.

Además, la vista de "editar" utiliza el método `update` para actualizar objetos en la base de datos, mientras que las acciones "eliminar" o "activar" utilizan el método `updateStateClass` para cambiar el estado del objeto y redirigir la vista en la que estemos

Esta clase fue creada , debido a la repetición de código , ya que Discos , Películas y Libros , compartían el mismo código con el mismo nombre de acciones de código.

```
/**
 * Busca un objeto en la base de datos por su id
 * @param object $orm - Objeto ORM
 * @param int $id - Id del objeto a buscar
 * @param string $class - Nombre de la clase del objeto
 * @return object|null
 */
4 references | 0 overrides
public static function findId($orm, $id, $class) : object|null
{
    return $orm->find($class, $id);
}
```

```

/**
 * Actualiza el estado de una clase o redirrecciona a la pagina de editar
 * @param object $orm - Objeto ORM
 * @param string $action - Accion a realizar
 * @param object $object - Objeto a actualizar
 * @param string $redirect - Url a la que redirigir
 * @return void
 */
3 references | 0 overrides
public static function updateStateClass($orm , $action , $object , $redirect) : void
{
    // Compruebo si alguno de los botones de la tabla ha sido pulsado
    // Si ha sido pulsado eliminar o activar , actualizo el estado del objeto
    if ($action == 'delete' || $action == 'enable') {
        // Actualizo el estado del objeto
        $object->setEstado($action == 'enable');
        // Actualizo el objeto en la base de datos
        $orm->updateState($object);
        header('Location: ' . $redirect); // Redirijo a la url obtenida por parametro
        exit();
    }
    // Si ha sido pulsado editar , redirijo a la pagina de editar con parametros get con el nombre
    // de la clase y el id del objeto
    if($action == 'edit'){
        header('Location: ' . $_SERVER["PHP_SELF"] . "?route=edit&class=" . ucfirst(get_class($object)) . "&id=" . $object->getId());
        exit();
    }
}
}

```

```

}

/**
 * Actualiza un objeto en la base de datos con los datos del formulario post recibido
 * @param object $orm - Objeto ORM
 * @param object $object - Objeto a actualizar
 * @param array $post - Array con los datos del formulario
 * @return void
 */
1 reference | 0 overrides
public static function update($orm, &$object , $post) : void
{
    // Almaceno en un array las propiedades del objeto con getPropiedadesEspecificas()
    $properties = array_merge(
        array_keys(getPropiedadesEspecificas($object)['Todos']),
        getPropiedadesEspecificas($object)[ucfirst(get_class($object))]
    );

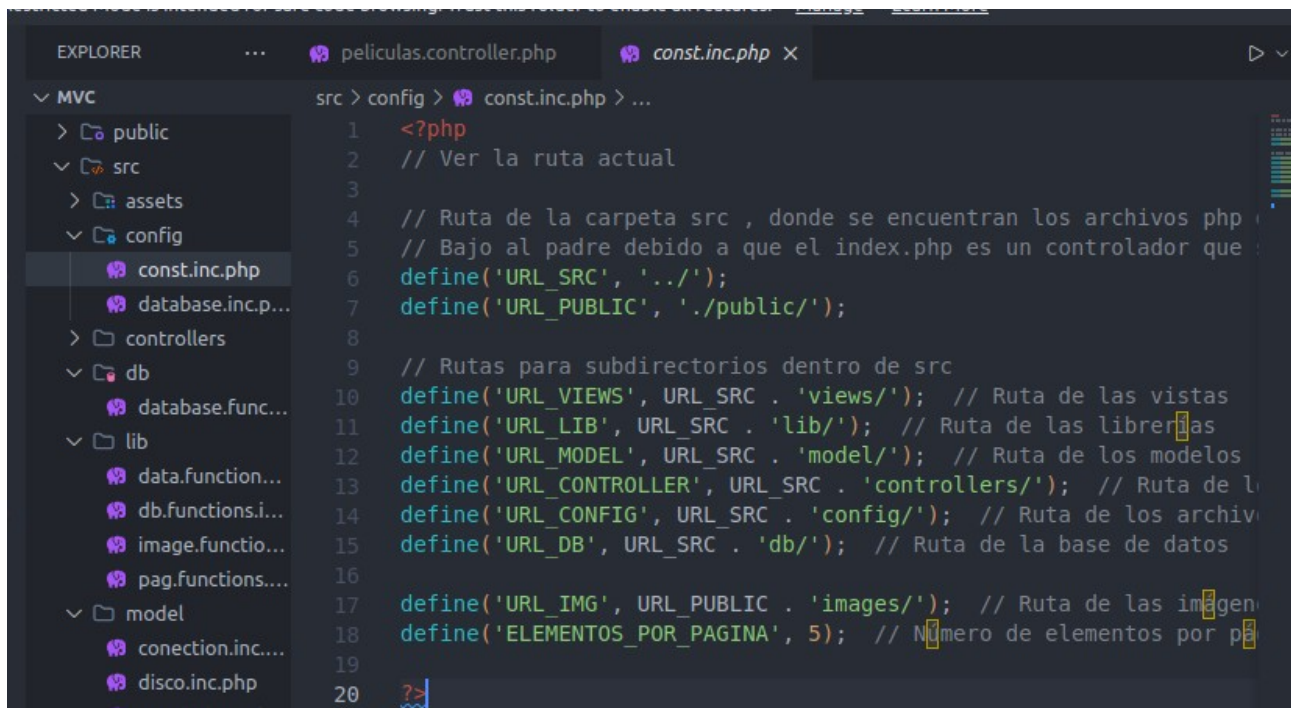
    $imagen = handleImageUpload(get_class($object), 'imagen'); //Llamo al metodo de imagenes para subir la imagen

    //Lo recorro y modifico el objeto con los datos del formulario
    foreach ($properties as $property) {
        if(strtolower($property) == 'imagen') continue; // Si la propiedad es imagen , continuo
        $metodo = 'set' . ucfirst($property); // Obtengo el nombre del metodo
        $object->$metodo($post[strtolower($property)]); // Modifico el objeto con los datos del formulario
    }
    //Evitar la perdida de la imagen
    if(!$imagen) $imagen = $object->getImagen();

    $object->setImagen($imagen); // Inserto la imagen en el objeto
    $orm->flush($object); // Actualizo el objeto en la base de datos
}
}

```


8.RUTAS Y CONSTANTES



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like public, src, assets, config, controllers, db, lib, and model. The file const.inc.php is selected in the config folder. The code editor shows the content of const.inc.php, which defines various URL constants for the application.

```
src > config > const.inc.php > ...
1  <?php
2  // Ver la ruta actual
3
4  // Ruta de la carpeta src , donde se encuentran los archivos php
5  // Bajo al padre debido a que el index.php es un controlador que
6  define('URL_SRC', '../');
7  define('URL_PUBLIC', '../public/');
8
9  // Rutas para subdirectorios dentro de src
10 define('URL_VIEWS', URL_SRC . 'views/'); // Ruta de las vistas
11 define('URL_LIB', URL_SRC . 'lib/'); // Ruta de las librerías
12 define('URL_MODEL', URL_SRC . 'model/'); // Ruta de los modelos
13 define('URL_CONTROLLER', URL_SRC . 'controllers/'); // Ruta de los controladores
14 define('URL_CONFIG', URL_SRC . 'config/'); // Ruta de los archivos de configuración
15 define('URL_DB', URL_SRC . 'db/'); // Ruta de la base de datos
16
17 define('URL_IMG', URL_PUBLIC . 'images/'); // Ruta de las imágenes
18 define('ELEMENTOS_POR_PAGINA', 5); // Número de elementos por página
19
20
```

He optado por crear constantes con el valor de las rutas que tiene la aplicación. Esto para tener una mayor legibilidad en el código y en caso de tener que cambiar rutas de carpetas etc... , solo tener que acceder a este único fichero que implementa la ruta de toda la aplicación.

9. FUNCIONALIDAD DE LA APLICACIÓN

Antes de explicar y ver la funcionalidad de la aplicación , explicar el por que de que el navegador sea un controlador.

El navegador tiene código embebido JavaScript , por que así ha sido el diseño , y tiene su respectivo controlador en el que básicamente obtengo la ruta actual para luego añadirlas en los hiperenlaces

```
<?php
require_once('../config/const.inc.php'); // Incluyo el archivo qu
$arrBasename = explode("/", $_SERVER["REQUEST_URI"]);
//... nos quedamos los dos primeros parámetros que son la raíz de
$ruta = "/" . $arrBasename[1] . "/";
?>

<?php
require_once(URL_VIEWS . 'nav.inc.php'); // Incluyo el archiv
?>
```

Lo he considerado controlador , de esa misma manera en vez de repetir código JS y código PHP y creando variables , he visto mas optimo crearlo a parte y hacer un require_once , en todas las vistas de la aplicación , siguiendo el **MODELO , VISTA , CONTROLADOR**

```
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Open+Sans:
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesom
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/sweetalert2@10.15.7/d
<link href="https://unpkg.com/boxicons@2.1.4/css/boxicons.min.css" rel="stylesh
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesom
<link rel="stylesheet" type="text/css" href="<?php echo $rutaCSS ?>">
<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>
</head>

<body>
<!-- Implemento el controlador del navegador para la navegacion de usuario -->
<?php require_once(URL_CONTROLLER . 'nav.controller.php') ?>
<div class="home">
```

EN TODAS LAS VISTAS TENDREMOS ESTA PARTE DONDE LA INCLUIAMOS

HOME

Como he explicado en el apartado del .htaccess , no he considerado tener un index.php , como tal el index seria en este caso home , que es a donde se redirige la pagina una vez ingresamos

En todos los controladores que veamos , veremos que estoy importando el archivo de configuración const.inc.php :

```
<?php
require_once('../config/const.inc.php'); // Incluyo el archivo qu
require_once(URL_MODEL . 'orm.inc.php'); // Incluyo el archivo qu
```

Como hemos explicado antes , considero tener las rutas en un fichero aparte , para tener una mejor estabilidad en el manejo en el caso de cambias rutas o crearlas o eliminarlas.

En el controlador de home , lo que he hecho es recopilar todos los datos que tienen y ordenarlas mediante la función usort , para poder ordenarlo en base a lo que busco que es que básicamente que el índice 0 sea la tupla obtenida de la base de datos más reciente para mostrarlo

Finalmente se importa la vista que es donde se hace visible esta información:

```
rc > controllers > home.controller.php
1  <?php
2  require_once('../config/const.inc.php'); // Incluyo el archivo que contiene las constantes
3  require_once(URL_MODEL . 'orm.inc.php'); // Incluyo el archivo que contiene la clase ORM
4  require_once(URL_MODEL . 'libro.inc.php'); // Incluyo el archivo que contiene la clase Libro
5  require_once(URL_MODEL . 'pelicula.inc.php'); // Incluyo el archivo que contiene la clase Pelicula
6  require_once(URL_MODEL . 'disco.inc.php'); // Incluyo el archivo que contiene la clase Disco
7  require_once(URL_LIB . 'data.functions.inc.php'); // Incluyo el archivo que contiene la funcion que compara
8  $orm = new ORM(); // Creo una instancia de la clase ORM
9  ?>
10
11  <?php
12
13  // Obtengo un array asociativo de los libros mediante el ORM , con el metodo findAll()
14  $dataModel = [
15      'Libros' => $orm->findAll("Libro"),
16      'Películas' => $orm->findAll("Pelicula"),
17      'Discos' => $orm->findAll("Disco")
18  ];
19
20  // Ordeno los arrays por fecha
21  usort($dataModel['Libros'], 'compareByDate');
22  usort($dataModel['Películas'], 'compareByDate');
23  usort($dataModel['Discos'], 'compareByDate');
24  ?>
25
26  <?php
27  // Creo variables para almacenar las rutas de las imagenes mas recientes y el titulo
28  $libroImg = URL_IMG . 'Libros/' . $dataModel['Libros'][0]['imagen'];
29  $libroTitle = $dataModel['Libros'][0]['titulo'];
30
31  $peliculaImg = URL_IMG . 'Películas/' . $dataModel['Películas'][0]['imagen'];
32  $peliculaTitle = $dataModel['Películas'][0]['titulo'];
33
34  $discoImg = URL_IMG . 'Discos/' . $dataModel['Discos'][0]['imagen'];
35  $discoTitle = $dataModel['Discos'][0]['titulo'];
36  ?>
37
38  <?php
39  require_once(URL_VIEWS . 'home.view.php');
40  ?>
```

DISCOS , PELÍCULAS Y LIBROS:

El código de estos 3 controladores , es igual excepto en algunas cosas como la redirecciones etc.

Como en home , se importa el fichero de configuración con las constantes y luego los requires necesarios en cada controlador :

```
<?php
require_once('../config/const.inc.php'); // Incluyo
require_once(URL_MODEL . 'orm.inc.php'); // Incluyo
require_once(URL_MODEL . 'disco.inc.php'); // Incluyo
require_once(URL_LIB . 'pag.functions.inc.php'); //
?>
```

Inicializamos el ORM , y utilizamos el método findAll() para obtener todos los datos de la tabla , luego la almacenamos en una variable , esta va a ser iterada después en una tabla de la vista para poder interactuar con ella , borrar , editar y reactivar.

Es importante la variable **\$redirect** es la que nos va a dar la ruta actual en este caso nos devuelve /MVC / , y a partir de ahí manipulamos

```
$orm = new ORM(); // Creo una instancia de la clase ORM
$discos = $orm->findAll('Disco'); // Obtengo un array asociativo de los discos mediante el ORM , con el metodo findAll()
$redirect = '/' . explode('/', $SERVER['REQUEST_URI'])[1] . '/'; // Obtengo la ruta de la url

// Compruebo si alguno de los botones de la tabla ha sido pulsado
if (isset($_POST['action'])) {
    $disco = DatabaseFunctions::findId($orm, $_POST['id'], 'Disco'); // Obtengo el disco mediante el ORM , con el metodo findId()
    $urlEdit = $redirect . 'Editar/Disco/' . $_POST['id']; // Url de redireccion a editar , con el id del disco
    DatabaseFunctions::updateStateClass($orm, $_POST['action'], $disco, $redirect . 'Discos', $urlEdit); // Actualizo el estado del disco mediante el ORM , con el metodo updateStateClass()
}

// Obtengo del array los elementos que tengan estado de boolean true
$discosActivos = array_filter($discos, function ($disco) {
    return $disco['estado'] == true;
});

// Obtengo del array los elementos que tengan estado de boolean false
$discosInactivos = array_filter($discos, function ($disco) {
    return $disco['estado'] == false;
});

$totalPaginas = ceil(count($discos) / ELEMENTOS_POR_PAGINA); // Numero total de las paginas
$paginaActual = isset($_GET['page']) ? $_GET['page'] : 1; // Obtengo la pagina actual

mostrarElementosPagina($discos, $totalPaginas, $paginaActual); // Muestro los elementos de la pagina actual
?>

<?php
$headers = ['TITULO', 'AÑO', 'GENERO', 'IMAGEN', 'ESTADO', 'ARTISTA', 'DURACION', 'ISWC', 'ACCIONES']; // Cabecera de la tabla
$rutaCSS = (isset($_GET['page'])) ? '../src/assets/css/index.css' : '../src/assets/css/index.css'; // Ruta del css
$url = (isset($_GET['page'])) ? "." . URL_IMG : URL_IMG; // Ruta de la imagen
// Incluyo el archivo que contiene la funcion que genera la vista
require_once(URL_VIEWS . 'discos.view.php');
?>
```

Como he explicado en DatabaseFunctions::updateStateClass , controla tanto la activacion/eliminación del elemento y tambien controla que si se ha pinchado editar hacia donde tiene que redireccionar , en este Editar/Disco/1 , esto lo conseguimos con las rutas amigables

Una cosa muy importante , de estas vistas es , la ruta **CSS** , la inclusión dinámica del archivo CSS en función de la URL actual es importante para adaptar el estilo de tu aplicación de manera específica para cada página, especialmente cuando se implementan rutas amigables.

La elección de que la ruta css sea dinámica ha sido por parte de las rutas amigables , debido a que si nosotros nos cambiábamos de página se perdía por completo los estilos CSS , y en la inspección de la página se podía observar que no encontraba la ruta es decir con la ruta GET['page'] , se tiene que bajar al padre de lo contrario no.

```
<link rel="icon" href="/public/icon.svg">
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Open+Sans:300,400,700&display=swap">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/sweetalert2@10.15.7/dist/sweetalert2.min.css">
<link href="https://unpkg.com/boxicons@2.1.4/css/boxicons.min.css" rel="stylesheet" />
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.1.0/css/all.min.css" />
<link rel="stylesheet" type="text/css" href="<?php echo $rutaCSS ?>">
<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>
</head>
```

AÑADIR:

```
controllers > add.controller.php

?>
<?php
$orm = new ORM(); // Creo una instancia de la clase ORM
$class = $_GET['class']; // Obtengo la clase que me pasan por parametro

// Juntamos las propiedades de la clase padre y de la clase hija
// mediante la funcion array_merge() , que une dos o mas arrays llamando a la funcion getPropiedadesEspecificas()
// obteniendo un array con las propiedades de la clase padre y de la clase hija
$propiedades = array_merge(
    // Obtenemos las propiedades de la clase padre que en este caso son las claves del array asociativo
    array_keys(getPropiedadesEspecificas(new $class)['Todos']),
    // Obtengo el array con los nombres de las propiedades de la clase hija , mediante la clase recibida por parametro
    getPropiedadesEspecificas(new $class)[ucfirst($class)]
);

//Compruebo si alguno de los botones del formulario ha sido pulsado
if (isset($_POST["addModel"])) {
    // Creo un objeto de la clase recibida por parametro
    $object = new $class();

    $imagen = handleImageUpload($class, 'imagen', '../' . URL_IMG); //Llamo al metodo de imagenes para subir la imagen

    if (!$imagen) { // Si la imagen no se ha subido correctamente
        // Redirijo a la pagina de home
        header("Location: " . $_SERVER['PHP_SELF']);
        exit();
    }

    //Recorro el array del objeto de la clase y mediante el setter asigno los valores al objeto
    foreach ($_POST as $key => $value) {
        if ($key != 'addModel') {
            $object->{"set" . ucfirst($key)}($value);
        }
    }
    $object->setImagen($imagen); // Inserto la imagen en el objeto
    $orm->persist($object); // Inserto el objeto en la base de datos

    // Redirijo a la pagina de la clase
    $redirect = '/' . explode("/", $_SERVER["REQUEST_URI"])[1] . '/' . $class . 's';
    header("Location: " . $redirect); // Redirijo a la url obtenida por parametro
}
?>
<?php require_once(URL_VIEWS . 'add.view.php'); //Incluyo la vista ?>
```

Este controlador es dinámico , es decir vale para cualquier clase (Disco , Pelicula y Libro) , esto es por la varibale \$propiedades en la que se pintara como formulario despues en la vista

Cuando el formulario ha sido enviado lo primero que hago es comprobar que la imagen se ha subido al servidor , de lo contrario redirige a home y no hace nada .

Si se ha subido la imagen recorro con un foreach el formulario POST , y con setter voy añadiendo los valores al objeto y finalmente lo inserto en la base de datos y redirigo a la vista de la clase creada.

```

src > views > add.view.php
23 <div class="home">
24 <p class="text"> SECCION PARA INSERTAR
25 <?php echo (strtolower($class) == "pelicula") ? "UNA NUEVA " . strtoupper($class) : " UN NUEVO " . strtoupper($class) ?>
26 </p>
27 <form method="post" action="<?php echo $_SERVER['PHP_SELF'] . '?class=' . $_GET['class'] ?>" class="formularioEditAnadir" enctype="multipart/form-data">
28 <?php
29     foreach ($propiedades as $value) {
30         $value = strtolower($value);
31         ?>
32         <label for="<?php echo $value ?>">Introduce :
33         <?php echo ucfirst($value) ?>
34         </label>
35         <?php
36
37         switch ($value) {
38             case 'imagen':
39                 ?>
40                 <input type="file" name="<?php echo $value ?>" id="<?php echo $value ?>" required>
41                 <?php
42                 break;
43             case 'estado':
44                 ?>
45                 <select name="<?php echo $value ?>" id="<?php echo $value ?>" required>
46                     <option value="1">Activo</option>
47                     <option value="0">Inactivo</option>
48                 </select>
49                 <?php
50                 break;
51             case 'anio':
52                 ?>
53                 <input type="date" name="<?php echo $value ?>" id="<?php echo $value ?>" required>
54                 <?php
55                 break;
56             case 'duracion':
57             case 'paginas':
58                 ?>
59                 <input type="number" name="<?php echo $value ?>" id="<?php echo $value ?>" required>
60                 <?php
61                 break;
62             default:
63                 ?>
64                 <input type="text" name="<?php echo $value ?>" id="<?php echo $value ?>" required>
65                 <?php
66                 break;
67         }

```

VISTA DE AÑADIR

EDITAR :

```
src > controllers > edit.controller.php
1 <?php
2 require_once('../config/const.inc.php'); // Incluyo el archivo que contiene las constantes
3 require_once(URL_MODEL . strtolower($_GET['class']) . '.inc.php'); // Incluyo el modelo que me pasan por parametro en la url
4 require_once(URL_MODEL . 'orm.inc.php'); // Incluyo el archivo que contiene la clase ORM
5 require_once(URL_LIB . 'image.functions.inc.php'); // Incluyo el archivo que contiene la funcion que gestiona las imagenes
6 ?>
7
8 <?php
9 $orm = new ORM(); // Creo una instancia de la clase ORM
10 // Obtengo un array asociativo de la clase que me llega por la url mediante el ORM , con el metodo findId()
11 $object = DatabaseFunctions::findId($orm, $_GET['id'], $_GET['class']);
12 ?>
13
14 <?php
15 // Si no hay ningun id || class en la url , redirijo a la pagina de home
16 if (!isset($_GET['id']) || !isset($_GET['class'])) {
17     header("Location: " . $_SERVER['PHP_SELF']);
18     exit();
19 }
20 // Compruebo si alguno de los botones del formulario ha sido pulsado
21 if (isset($_POST['editar'])) {
22     // Actualizo el objeto con los datos del formulario llamando al metodo update() de la clase DatabaseFunctions
23     DatabaseFunctions::update($orm, $object, $_POST, '../..' . URL_IMG);
24     $home = '/' . explode("/", $_SERVER['REQUEST_URI'])[1] . '/'; // Obtengo la ruta de la url de la raiz
25     // Redirijo a la pagina de home , la s es debido a que en la url de la pagina de home , se añade una s al final de la clase , por ejemplo libros -> libros
26     header("Location: " . $home . $_GET['class'] . 's');
27     exit();
28 }
29 ?>
30 <?php
31 // Incluyo el archivo que contiene la funcion que genera la vista
32 require_once(URL_VIEWS . 'edit.view.php');
33 ?>
```

Editar es similar a añadir , solamente que aquí tenemos que tomar dos parametros , un parametro `$_GET['class']` y otro parametro `$_GET['id']` , para obtener la tupla en base a la clase que sera la tabla y el identificador

```
</p>
<form method="post" action="php echo $_SERVER['PHP_SELF'] . '?class=' . $_GET['class'] . '&amp;id=' . $_GET['id']?&gt;" class="formularioEditAnadir"
enctype="multipart/form-data"&gt;
&lt;?php
foreach (get_class_methods(get_class($object)) as $metodo) { // Recorro los metodos de la clase
    // Compruebo si el metodo es un get
    if (substr($metodo, 0, 3) === 'get') {
        // Elimino "get" y convierto la primera letra a minúscula para obtener el nombre de la propiedad para mostrarla en el formulario
        $propiedad = lcfirst(substr($metodo, 3));

        // Compruebo que tipo es , para controlar , numeros , fechas etc...
        switch ($propiedad) {
            case 'id':
                ?&gt;
                &lt;input type="hidden" name="id" value="<?php echo $object-&gt;$metodo() ?&gt;"&gt;
                &lt;?php
                break;
            case 'imagen':
                ?&gt;
                &lt;label for="<?php echo $propiedad ?&gt;"&gt;
                &lt;?php echo ucfirst($propiedad) ?&gt;
                &lt;/label&gt;
                &lt;img src="<?php echo '../..' . URL_IMG . ucfirst($_GET['class']) . 's/' . $object-&gt;$metodo() ?&gt;"
                alt="<?php echo $object-&gt;$metodo() ?&gt;" width="100px" height="100px"&gt;
                &lt;input type="file" name="<?php echo $propiedad ?&gt;" id="<?php echo $propiedad ?&gt;"
                value="<?php echo $object-&gt;$metodo() ?&gt;"&gt;
                &lt;?php
                break;
            case 'estado':
                ?&gt;
                &lt;label for="<?php echo $propiedad ?&gt;"&gt;
                &lt;?php echo ucfirst($propiedad) ?&gt;
                &lt;/label&gt;
                &lt;select name="<?php echo $propiedad ?&gt;" id="<?php echo $propiedad ?&gt;"&gt;
                &lt;option value="1" &lt;?php echo ($object-&gt;$metodo() == 1 ? 'selected' : ''); ?&gt;&gt;Activo&lt;/option&gt;
                &lt;option value="0" &lt;?php echo ($object-&gt;$metodo() == 0 ? 'selected' : ''); ?&gt;&gt;Inactivo&lt;/option&gt;
                &lt;/select&gt;
                &lt;?php
                break;
            case 'anio':
                //Formateamos la fecha a tipo date de html con el metodo format()
                $fecha = new DateTime($object-&gt;$metodo());
                $fecha = $fecha-&gt;format('Y-m-d');
                ?&gt;
                &lt;input type="text" value="<?php echo $fecha ?&gt;"&gt;
                &lt;?php
                break;
        }
    }
}</pre
```

La condición if en la vista es importante para poder obtener el valor del objeto de la clase pasada por parámetro , no se puede hacer en el controlador debido a que en base a esto lo recopilo y los inputs tendrán estos valores.