



## Práctica 5.2

---



**Ciclo:** DAW  
**Módulo:** DWES  
**Curso:** 2020-2021  
**Autor:** César Guijarro Rosaleny



**Introducción.....3**

**películas\_ficha.phtml.....4**

**Listar críticas.....6**

**CRUD de críticas.....8**

    Ver críticas.....8

    Añadir crítica.....9

    Editar y borrar críticas.....13

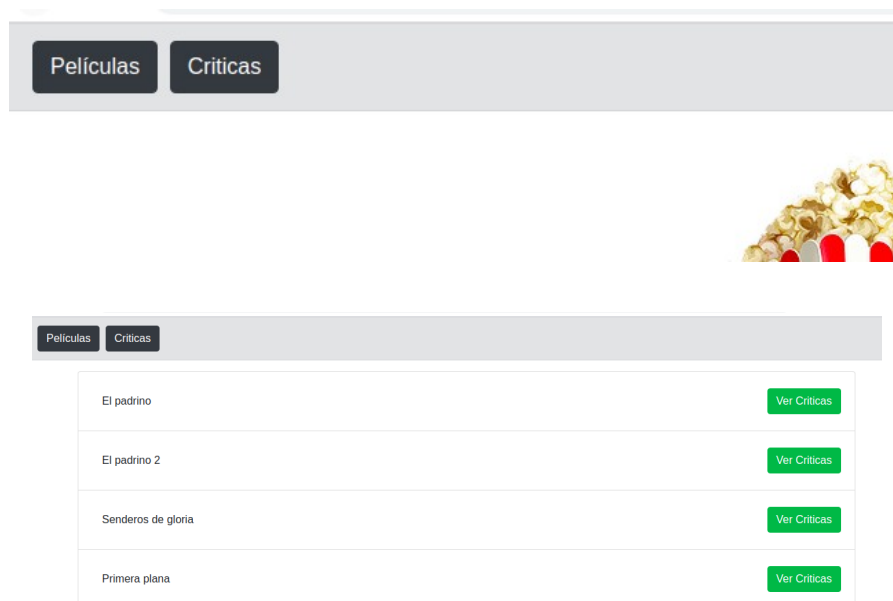
**Rúbrica.....18**

## Introducción

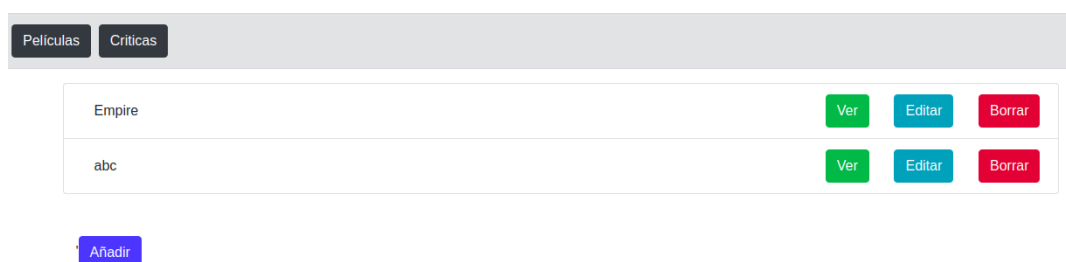
---

Vamos a continuar con nuestra web añadiendo las críticas de las películas, pudiendo verlas, insertar nuevas críticas, actualizarlas o borrarlas.

Añadiremos un nuevo botón *Críticas* que nos mostrará un listado con las películas.



Al seleccionar una, veremos el típico listado para ejecutar las operaciones CRUD (create, read, update y delete).



Además, la ficha de la película mostrará todas las críticas de la misma.

## películas\_ficha.phtml

Después de mostrar los datos de la película, sus directores y actores, se cargarán las críticas que tengamos almacenadas en nuestra bbdd para esa película.

Películas

Criticas

Título: El padrino  
 Año: 1972  
 Duración: 175  
 Dirección:  
 • Francis Ford Coppola  
 Actores:  
 • Marion Brando  
 • Al Pacino  
 • Robert Duvall  
 • James Caan  
 • Diane Keaton

Criticas:

Medio: Empire  
 Puntuación: 5.0  
 Crítica: Se podría argumentar que la película de Francis Ford Coppola basada en el bestseller de Mario Puzo, a la vez una película de arte y una superproducción comercial, marcó el comienzo de la era de la mega-película.

Medio: abc  
 Puntuación: 5.0  
 Crítica: Una labor de creación personal y un fresco, lleno de vivacidad y dramatismo, lo que se explica por el talento y la sensibilidad del joven realizador

Las plantillas ya están hechas, con lo que sólo tendremos que crear el modelo *Crítica*, y sacar los datos de las críticas de esa película en la función *getById()* del controlador *Pelicula* para pasárselo a la plantilla *películas\_ficha.phtml* igual que hacíamos con los actores y directores.

Pero antes vamos a añadir un nuevo método a nuestra clase *Model*. En la práctica anterior hicimos un método para sacar datos de relaciones N:M llamado *belongsToMany()*. Ahora vamos a crear otro más sencillo para leer datos asociados en una relación 1:N que llamaremos *hasMany()*.

Este método recibirá como parámetros la tabla secundaria, el nombre de la clave ajena que referencia a la tabla principal y el valor de la clave principal en esa tabla. En nuestro caso, la tabla principal será *Películas*, la tabla secundaria *Criticas* y la clave ajena de la tabla *id\_pelicula*. El valor de la clave principal de la tabla *Películas* vendrá en la URL (/películas/**1**).

```
protected function hasMany($t2, $fk_t2, $pk) {
    $sql = "SELECT t2.* FROM $t2 t2
           JOIN $this->table t1 ON t1.$this->primary_key = t2.$fk_t2
           AND t1.$this->primary_key = :id";
    $params = [
        ":id" => $pk
    ];
    return db\DB::execute($sql, $params);
}
```

El método es casi idéntico a *belongsToMany()*, lo único que cambia es la sentencia SQL.

El siguiente paso será crear el método *getCriticas()* en el modelo *Pelicula* similar a *getActores()* y *getDirectores()* pero usando el nuevo método *hasMany()* en lugar de *belongsToMany()*

Una vez lo tengamos, ya podemos utilizarlo en el controlador *Películas* para sacar sus críticas y pasárselas a la plantilla *pelicula\_ficha.phtml*.

```
$actores = modelPelícula::getActores($vars['id']);
$criticas = modelPelícula::getCriticas($vars['id']);
echo $this->templates->render('peliculas_ficha',
    ['pelicula' => $pelicula, 'directores' => $directores,
     'actores' => $actores, 'criticas' => $criticas]);
```

Ahora deberías ver las críticas cuando accedas a las fichas de las películas.

## Listar críticas

---

Vamos a crear ahora la pantalla donde se muestra el listado de las películas para ver sus críticas.

Lo primero será crear la ruta en nuestro archivo de rutas *web.php*:

```
$r->addRoute('GET', $basedir . '/criticas', 'critica@getAll');
```

El siguiente paso será crear el controlador *Critica* y la función *getAll()*, que será igual al método *getAll()* del controlador *Peliculas*, pero cambiando la plantilla que renderizará a *criticas.phtml*.

```
class Critica extends Controller{

    public function getAll(){
        $peliculas = modelPelicula::all();
        echo $this->templates->render('criticas', ['peliculas' => $peliculas]);
    }
}
```

De esta forma, cuando pinches al botón *Criticas* deberías ver un listado de películas con un botón para acceder a sus críticas:

El padrino	Ver Críticas
El padrino 2	Ver Críticas
Sanctus de gloria	Ver Críticas

Si te fijas en la plantilla, el botón *Ver Críticas* es un link que nos lleva a la ruta */peliculas/id\_pelicula/criticas*, con lo que tendremos que crear esa ruta en nuestro archivo *web.php*.

```
$r → addRoute('GET', $basedir . '/peliculas/{id:\d+}/criticas', 'pelicula@criticas');
```

En este caso, la ruta cargará el método *criticas()* del controlador *Pelicula*, método que tendremos que crear.

El método es muy sencillo. Utilizará el método *getCriticas()* para traer las críticas de la película y cargará la plantilla *peliculas\_critica.phtml* con esas críticas:

```
public function criticas($vars) {
    $criticas = modelPelicula::getCriticas($vars['id']);
    echo $this->templates->render('peliculas_criticas',
        ['id_pelicula' => $vars['id'], 'criticas' => $criticas]);
}
```

En este punto ya podrás ver el listado de las críticas de la película donde podremos realizar las operaciones CRUD:

Películas		Críticas		
Empire		Ver	Editar	Borrar
abc		Ver	Editar	Borrar
Añadir				

## CRUD de críticas

---

### Ver críticas

Para ver las críticas sólo tenemos que crear la ruta, añadir el método *getById()* al controlador *Critica* y cargar la plantilla *criticas\_ficha.phtml* con los datos de la crítica.

La ruta será la siguiente:

```
$r->addRoute('GET', $basedir . '/criticas/{id:\d+}', 'critica@getById');
```

El método *getById()* será igual que el del controlador *Pelicula*, cambiando los datos de películas por críticas y el nombre de la plantilla.

[Películas](#)[Críticas](#)

**Medio:** Empire

**Puntuación:** 5.0

**Crítica:** Se podría argumentar que la película de Francis Ford Coppola basada en el bestseller de Mario Puzo, a la vez una película de arte y una superproducción comercial, marcó el comienzo de la era de la mega-película.



## Añadir crítica

Para añadir críticas necesitaremos dos rutas nuevas. Una para mostrar el formulario para poder introducir los datos y otra que se encargará de recoger esos datos e insertarlos en la tabla.

```
$r->addRoute('GET', $basedir . '/peliculas/{id:\d+}/criticas/insertar',  
'critica@insertForm');  
$r->addRoute('POST', $basedir . '/peliculas/{id:\d+}/criticas', 'critica@insert');
```

Fíjate que las dos rutas son del estilo */pelicula/id\_pelicula/criticas*. Eso es porque necesitamos el id de la película para saber a cual pertenece la crítica que vamos a insertar.

Además, la segunda ruta ya no tiene el método GET como el resto. Ahora utilizamos el método POST. Lo habitual es utilizar métodos HTTP para distinguir entre acciones:

- GET: listar
- POST: insertar
- PUT: Actualizar
- DELETE: Borrar

Existen otros métodos HTTP, como PATCH que se utiliza para actualizaciones parciales, pero nosotros vamos a usar sólo esos 4.

Si te das cuenta, la segunda ruta está repetida en nuestro archivo de rutas, pero no hay conflicto, ya que los métodos de ambas rutas son diferentes (GET y POST).

```
$r->addRoute('GET', $basedir . '/criticas', 'critica@getAll');
$r->addRoute('GET', $basedir . '/peliculas/{id:\d+}/criticas', 'pelicula@');
$r->addRoute('GET', $basedir . '/criticas/{id:\d+}', 'critica@getId');
$r->addRoute('GET', $basedir . '/peliculas/{id:\d+}/criticas/insertar', '');
$r->addRoute('POST', $basedir . '/peliculas/{id:\d+}/criticas', 'critica@');
```

La primera ruta ejecutará la acción *insertForm()* del controlador *Critica*. Este método lo único que hará será mostrar la plantilla *criticas\_insertar.phtml* pasándole el id de la película como variable.

En nuestro formulario de inserción, meteremos ese *id\_pelicula* como un campo oculto, para que a la hora de recoger los datos del formulario tengamos el id de la película.

```
<form action="<?=$ENV['APP_URI']>/peliculas/<?=$id_pelicula>/criticas" method="POST">
  <input type="hidden" value="<?=$id_pelicula>" name="id_pelicula">
  <div class="form-group">
    <label for="medio">Medio:</label>
    <input type="text" value="" name="medio">
```

La segunda ruta será la encargada de insertar los datos en nuestra base de datos. Pero antes necesitamos crear dos métodos, uno en *Model.php* y otro en *DB.php* que será el encargado de introducir los datos de un formulario en una tabla de la bbdd.

Primero creamos el método en *DB.php*. Lo creamos aquí y no directamente en *Model.php* porque vamos a montar una sentencia básica (INSERT INTO...), con lo que, al igual que hicimos con el método *select()*, lo creamos en nuestra clase base de gestión de la base de datos.

Este método se encargará de montar y ejecutar la sentencia INSERT de SQL. Recibirá en el nombre de la tabla y un *array* en formato [nombre\_campo1 => valor\_campo1, nombre\_campo2 => valor\_campo2...].

Hay que tener en cuenta que para que todo funcione correctamente, los nombres de los controles del formulario (*input*, *textarea*, *select*...) tienen que ser los mismos que los nombres de los campos de la tabla, ya que así podremos montar la sentencia de forma automática:

```
protected function insert($table, $insertValues) {
    $fields = '(';
    $values = '(';
    $params = array();
    foreach ($insertValues as $key => $value) {
        $fields .= $key . ',';
        $param = ':' . $key;
        $values .= $param . ',';
        $params[$param] = $value;
    }
    $fields = \substr($fields, 0, -1) . ')';
    $values = \substr($values, 0, -1) . ')';
    $sql = "INSERT INTO $table $fields VALUES $values";
    return db\DB::execute($sql, $params);
}
```

Una vez hemos creado el método, ya podemos crear los métodos en el modelo y el controlador:

```
protected function insert() {
    $result = db\DB::insert($this->table, $_POST);
    return $result;
}
```

```
public function insert($vars) {  
    modelCritica::insert();  
    header('Location: ' . $_ENV['APP_URL'] . '/peliculas/' . $vars['id'] .  
    '/criticas');  
}
```

El método **header()** con la opción (*Location: url*) lo utilizamos para redirigir con PHP (redireccionar a la ruta que nos interese).

**NOTA:** El método **header()** es muy útil para redireccionar nuestra página, pero hay una cosa importante a tener en cuenta. Sólo funcionará si no se ha mostrado antes ningún código HTML (ni siquiera espacios en blanco). Por ejemplo, si antes de usar **header()** utilizamos algún **echo**, PHP nos dará un error al intentar redirigir nuestra web a la ruta indicada.

Esto puede ser útil a la hora de depurar nuestra aplicación, para mostrar, por ejemplo, el valor de alguna variable sin que PHP nos redirija a la ruta deseada. Eso sí, a la hora de poner nuestra web en producción, tenemos que tener claro que no podemos enviar ningún código HTML al cliente si queremos que el método funcione.

Si todo ha ido bien, ya deberías ser capaz de introducir críticas a nuestra base de datos.

## Editar y borrar críticas

El método para editar y borrar críticas será casi el mismo que el de inserción, aunque tendremos que solucionar un problema con las rutas.

Vamos a programar el método de edición. Como hemos dicho, será casi igual que el de inserción, con la diferencia que a la hora de editar el formulario tiene que mostrar los datos de la crítica antigua.

Primero creamos las dos rutas (la que muestre el formulario de edición y la que edite los datos).

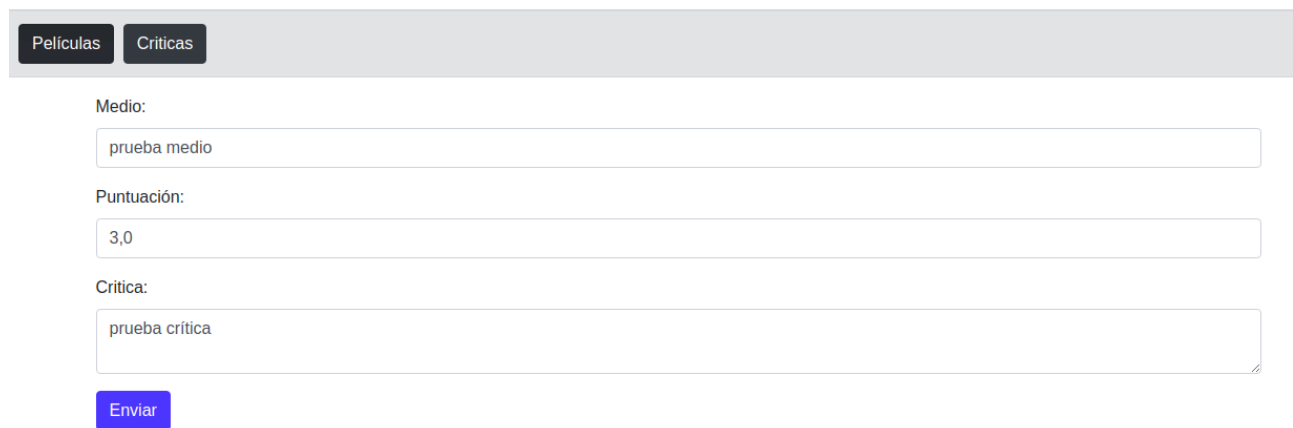
```
$r → addRoute('GET', $basedir .  
'/peliculas/{id_pelicula:\d+}/criticas/editar/{id_critica:\d+}', 'critica@editForm');  
$r->addRoute('PUT', $basedir .  
'/peliculas/{id_pelicula:\d+}/criticas/{id_critica:\d+}', 'critica@edit');
```

En realidad no necesitaríamos el id de la película, ya que podemos editar las críticas teniendo sólo su *id\_critica*. Lo hacemos para mantener la coherencia de las rutas y, sobre todo, para poder volver a la pantalla de ver las críticas de la película de la cual hemos editado una de ellas mediante el método **header()**.

El siguiente paso será crear la acción *editForm()* en el controlador *Critica* que nos mostrará el formulario de edición.

```
public function editForm($vars) {  
    $critica = modelCritica::find($vars['id_critica']);  
    echo $this->templates->render('criticas_editar',  
        ['id_pelicula' => $vars['id_pelicula'], 'critica' => $critica]);  
}
```

La función buscará la crítica por su id y cargará la plantilla *criticas\_editar.phtml*, pasándole los parámetros *id\_pelicula* y la critica encontrada.



El formulario muestra una barra superior con dos pestañas: 'Películas' y 'Críticas'. Debajo, hay tres campos de entrada etiquetados como 'Medio:', 'Puntuación:' y 'Crítica:'. El campo 'Medio:' contiene el texto 'prueba medio'. El campo 'Puntuación:' contiene el texto '3,0'. El campo 'Crítica:' contiene el texto 'prueba crítica'. Debajo de los campos, hay un botón azul con el texto 'Enviar'.

El siguiente paso es crear los 3 métodos *edit()*, uno en *DB.php* que será el que monte y ejecute la sentencia SQL (como antes lo creamos en *DB.php* por ser una sentencia SQL básica), otro en *Model.php* y el último en el controlador.

Empecemos con el método de *DB.php*. Necesitaremos 4 parámetros: el nombre de la tabla, los campos que vamos a editar (en nuestro caso serán siempre todos), y el nombre y valor de la clave principal para saber qué crítica editar.

Con todos esos datos ya podemos montar y ejecutar nuestra sentencia UPDATE.

```
private function edit($table, $editValues, $pkName, $pkValue) {
    $fields = "";
    $params = array();
    foreach ($editValues as $key => $value) {
        if ($key !== '_method') {
            $fields .= "$key = :$key,";
            $param = ':' . $key;
            $params[$param] = $value;
        }
    }
    $fields = \substr($fields, 0, -1);
    $where = "$pkName = :id";
    $params[":id"] = $pkValue;
    $sql = "UPDATE $table SET $fields WHERE $where";
    return $this->execute($sql, $params);
}
```

En el cuerpo del bucle **foreach** comprobamos si el nombre del campo que viene del formulario es distinto a `_method`. Cuando hagamos las rutas entenderás el porqué.

El siguiente será el método del modelo.

```
protected function edit($pk) {
    $result = db\DB::edit($this->table, $_POST, $this->primary_key, $pk);
    return $result;
}
```

Y, por último, el del controlador:

```
public function edit($vars) {
    modelCritica::edit($vars['id_critica']);
    header('Location: ' . $_ENV['APP_URL'] . '/peliculas/' .
        $vars['id_pelicula'] . '/criticas');
}
```

En principio ya estaría todo hecho. Prueba a editar una crítica. Como ves, nuestra web no reacciona a la edición y muestra la pantalla en blanco.

¿Porqué no funciona? El problema está en el campo *method* de la etiqueta *form* de nuestros formularios. HTML sólo permite definir métodos GET y POST, y nuestra ruta espera el método PUT para actualizar los datos.

Por fortuna, hay una forma muy sencilla de hacerle saber a PHP que queremos utilizar el método PUT. Fíjate en el formulario de la plantilla *criticas\_editar.phtml*. Hemos añadido un campo oculto llamado *\_method* cuyo valor es PUT.

```
<form action="<?= $ ENV['APP URL']?>/peliculas/<?=$id pelicula?>/criticas/
  <input type="hidden" name="_method" value="put" />
  <div class="form-group">
    <label for="medio">Medio:</label>
    <input type="text" class="form-control" name="medio">
```

Ahora podemos comprobar en nuestro archivo de rutas *web.php* si existe ese campo oculto y cambiar el método al que toca:

```
// Fetch method and URI from somewhere
// $httpMethod = $_SERVER['REQUEST_METHOD'];

// Ver si existe el campo _method para sobrescribir los métodos put y
// delete ya que los formularios sólo permiten POST y GET
$metodosPermitidos = ['GET', 'POST', 'PUT', 'DELETE'];
$httpMethod = strtoupper($_POST['_method'] ??
  $_SERVER['REQUEST_METHOD']);
if (!in_array($httpMethod, $metodosPermitidos)) {
    $httpMethod = 'GET';
}
```



Incluso podemos definir los métodos que vamos a permitir en nuestra web. Si alguien intenta usar otro método lo cambiaremos a GET.

Intenta crear la acción de borrado de críticas tal y como hemos hecho con la inserción y la edición. Fíjate que el formulario de borrado está en la plantilla *peliculas\_critica.phtml* y que ya lleva el campo oculto *\_method* con el valor DELETE.

# Rúbrica

APARTADO/ÍTEM	Puntuación máxima obtenible	Nota
<b>Bloque 1. Mostrar críticas en películas</b>	<b>1,5</b>	<b>2</b>
Se crea el método <i>hasMany()</i> de forma correcta	0,5	0,5
Se crea el método <i>getCriticas()</i> de forma correcta en el modelo <i>Pelicula</i>	0,5	0,5
Se traen las críticas en el controlador <i>Pelicula</i>	0,5	0,5
Las críticas se visualizan correctamente en la ficha de películas	0,5	0,5
<b>Bloque 2. Listar críticas</b>	<b>1,25</b>	<b>1,75</b>
Se crea la ruta para mostrar el listado de críticas por películas	0,25	0,25
Se crea el controlador <i>Criticas</i>	0,25	0,25
Se crea el método <i>getAll()</i> correctamente	0,5	0,5
Se crea la ruta para ver una crítica	0,25	0,25
Se crea el método <i>criticas()</i> correctamente	0,5	0,5
<b>Bloque 3. Ver crítica</b>	<b>0,75</b>	<b>0,75</b>
Se crea la ruta para ver las críticas individualmente	0,25	0,25
Se crea el método <i>getById()</i> correctamente	0,5	0,5
<b>Bloque 4. Añadir crítica</b>	<b>1,75</b>	<b>2,25</b>
Se crean las 2 rutas correctamente	0,25	0,25
Se crea el método que muestra el formulario de inserción	0,5	0,5
Se crea el método para insertar datos en <i>DB.php</i>	0,5	0,5
Se crea el método <i>insert()</i> en el modelo	0,5	0,5
Se crea el método <i>insert()</i> en el controlador	0,5	0,5
<b>Bloque 5. Editar crítica</b>	<b>4,5</b>	<b>2</b>
Se crean las 2 rutas correctamente	0,25	0,25
Se crea el método que muestra el formulario de edición	0,25	0,25
Se crea el método para editar datos en <i>DB.php</i>	0,5	0,5
Se crea el método <i>edit()</i> en el modelo	0,25	0,25
Se crea el método <i>edit()</i> en el controlador	0,25	0,25
Se añaden las líneas necesarias en <i>web.php</i> para leer el método adecuado	0,5	0,5
<b>Bloque 5. Borrar crítica</b>	<b>1,25</b>	<b>1,25</b>
Se crean la ruta correctamente	0,25	0,25
Se crea el método para borrar datos en <i>DB.php</i>	0,5	0,5
Se crea el método <i>delete()</i> en el modelo	0,25	0,25
Se crea el método <i>delete()</i> en el controlador	0,25	0,25