VIDEOCLUB CAPÍTULO 3

Ejercicio 1 – Configuración de la BD y migraciones

Comprobamos que tenemos bien configurado nuestro fichero *.env* con los datos referentes al servidor de BD correctos.

A continuación, vamos a crear un modelo, una migración y un seeder para las películas de la aplicación. Para ellos vamos a usar un comando de artisan que nos crea todo: ***php artisan make:model Movie -ms***

Abrimos la migración llamada *create_movies_table* que se creó anteriormente y a la que habrá que añadir los siguientes campos:

Campo	Tipo	Valor por defecto
id	Autoincremental	
title	String	
year	String de longitud 8	
director	String de longitud 64	
poster	String	
rented	Booleano	False (->default(0))
synopsis	Text	
timestamps	Timestamps de Eloquent	
softDeletes	Marcado de borrado de Eloquent	

Por último, ejecutamos el comando de Artisan que ejecuta las migraciones y comprobamos en la BD que se ha creado la tabla con los campos correspondientes.

Ejercicio 2 – Configurar Seeder

El comando Artisan del ejercicio anterior nos habrá creado un fichero seeder llamado MovieSeeder.php. Lo usaremos para cargar en la BD los datos de las películas que tenemos en un array.

Para ello movemos el array que tenemos en CatalogController a la clase MovieSeeder como atributo privado.

Dentro del método *run()* de MovieSeeder realizamos las siguientes acciones:

1) Borramos el contenido de la tabla *movies*:
   DB::table('movies')->delete();
2) A continuación, añadimos el siguiente código:

```
DB::table('movies')->delete();
foreach ($this->arrayPeliculas as $pelicula){
    $p = new Movie();
    $p->title = $pelicula['title'];
    $p->year = $pelicula['year'];
    $p->director = $pelicula['director'];
    $p->poster = $pelicula['poster'];
    $p->rented = $pelicula['rented'];
    $p->synopsis = $pelicula['synopsis'];
    $p->save();
```

Por último, tenemos que llamar a este seeder que hemos creado desde el método *run* de la clase ***DatabaseSeeder*** de la siguiente forma:

$this->call([MovieSeeder::class]);

Y ejecutar el comando Artisan **php artisan db:seed**

Ahora comprobamos en la BD que la tabla *movies* se ha rellenado correctamente.

Ejercicio 3 – Uso de la base de datos

En este paso vamos a actualizar los métodos del controlador *CatalogController* para que obtengan los datos desde la base de datos:

1) Modificar el método *getIndex* para que obtenga toda la lista de películas desde la base de datos usando el modelo *Movie* y que se la pase a la vista.
2) Modificar el método *getShow* para que obtenga la película pasada como parámetro usando el método *findOrFail* y se la pase a la vista.
3) Modificar el método *getShow* para que obtenga la película pasada como parámetro usando el método *findOrFail* y se la pase a la vista.

En las vistas habrá que hacer los siguientes cambios:

1) En todas, en lugar de acceder a los datos como un array, hay que acceder como si fuese un objeto; para ello, hay que cambiar *$pelicula['campo']* por *$pelicula->campo*.
2) En la vista *index.blade.php* en vez de utilizar el índice del array *($key)* como identificador para crear el enlace a ***catalog/show/{id}***, tendremos que utilizar el campo **id** de la película (***$pelicula->id)***. Lo mismo en la vista *show.blade.php,* para generar el enlace de editar película tendremos que añadir el identificador de la película a la ruta ***catalog/edit***.

Ejercicio 4 – Añadir y editar películas

Lo primero que vamos a hacer es añadir las rutas que nos hacen falta para recoger los datos al enviar los formularios de añadir y editar película. Para ello, en el fichero *web.php* añadimos dos rutas (protegidas por el middleware auth):

- Una ruta de tipo POST para la url *catalog/create* que apuntará al método ***postCreate*** del controlador *CatalogController.*
- Una ruta de tipo PUT para la url *catalog/edit/{id}* que apuntará al método ***putEdit*** del controlador CatalogController.

A continuación, editamos la vista *catalog/edit.blade.php* con los siguientes cambios:

- Revisar que el método de envío del formulario sea de tipo PUT.
- Hay que modificar todos los *inputs* para que como valor del campo (en el *value*) ponga el valor correspondiente de la película que se quiere modificar *$pelicula->title* (no va entre {{ }} porque el componente ya las tiene, en caso de que no se usase el componente <x-text-input> si debería llevarlas).
- El formulario debe apuntar a la ruta *catalog/edit/{id}* por PUT creada anteriormente.

En la vista *catalog/create.blade.php* modificar el formulario para que apunte a la ruta *catalog/create* por POST creada anteriormente.

Por último, tenemos que actualizar el controlador *CatalogController* con los dos nuevos métodos. En ambos casos tenemos que usar la inyección de dependencias para añadir la clase ***Request*** como parámetro de entrada. Además, para cada método haremos lo siguiente:

- En el método ***postCreate*** creamos una nueva instancia del modelo **Movie**, asignamos el valor de todos los campos de entrada (menos rented) y guardamos. Por último, después de guardar, hacemos una redirección a la ruta */catalog*.
- En el método ***putEdit*** buscamos la película con el identificador pasado como parámetro, actualizamos sus campos y los guardamos. Por último, realizamos una redirección a la pantalla con la vista detalle de la película editada.

Ejercicio 5 – Completando el resto de botones

En este ejercicio vamos a añadir funcionalidad a los botones de alquilar, devolver y eliminar película. Todos estos botones están situados en la vista de detalle de una película (falta el botón de eliminar que hay que añadirlo). En todos los casos habrá que crear una nueva ruta, un nuevo método en el controlador y actualizar el botón en la vista.

Ruta	Tipo	Controlador/Acción
/catalog/rent/{id}	PUT	CatalogController@putRent
/catalog/return/{id}	PUT	CatalogController@putReturn
/catalog/delete/{id}	DELETE	CatalogController@deleteMovie

En primer lugar, hay que añadir las rutas anteriores al fichero *web.php* y posteriormente modificar el controlador *CatalogController* para añadir los nuevos métodos. Estos tres nuevos métodos son similares al método implementado antes para editar los datos de una película. En el caso de **putRent** y **putReturn** únicamente modificaremos el campo *rented* asignándole un valor *true* y *false* respectivamente, y una vez guardado realizaremos la redirección a la pantalla con la vista detalle de la película. Además, le mandaremos a la vista una variable para mostrarle un mensaje:

return redirect('catalog/show/' . $id)->with('msg',1);

Después en la vista de detalle de la película habrá que evaluar el valor de esa variable que se guarda en una sesión, usando: ***session('msg')==1*** y mostrar un mensaje indicando que todo ha ido bien.

En la vista de detalle de la película hay que modificar los botones y dado que las acciones se tienen que realizar usando peticiones HTTP de tipo PUT y DELETE, no podemos usar un enlace normal, sino que hay que meterlo en un formulario con el siguiente código:

```
Película disponible<br><br>
<form action="{{url('/catalog/rent/'.$pelicula->id)}}" method="POST" style="display: inline">
    @csrf
    @method('PUT')
    <button type="submit" class="text-white bg-blue-700 hover:bg-blue-800 focus:ring-4 fo
</form>
```

En el método **deleteMovie** habrá que llamar al método **delete()** una vez obtenido el registro que se desea borrar y redirigiremos al listado general mostrando un mensaje allí.