Controladores

Hasta el momento hemos visto solamente como devolver una cadena para una ruta y como asociar una vista a una ruta directamente en el fichero de rutas. Pero en general la forma recomendable de trabajar será asociar dichas rutas a un método de un controlador. Esto nos permitirá separar mucho mejor el código y crear clases (controladores) que agrupen toda la funcionalidad de un determinado recurso. Por ejemplo, podemos crear un controlador para gestionar toda la lógica asociada al control de usuarios o cualquier otro tipo de recurso.

Como ya vimos en la sección de introducción, los controladores son el punto de entrada de las peticiones de los usuarios y son los que deben contener toda la lógica asociada al procesamiento de una petición, encargándose de realizar las consultas necesarias a la base de datos, de preparar los datos y de llamar a la vista correspondiente con dichos datos.

Controlador básico

Los controladores se almacenan en ficheros PHP en la carpeta `app/Http/Controllers` y normalmente se les añade el sufijo `Controller`, por ejemplo `UserController.php` o `MoviesController.php`. A continuación se incluye un ejemplo básico de un controlador almacenado en el fichero `app/Http/Controllers/UserController.php`:

```php
<?php
namespace App\Http\Controllers;

use App\User;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Mostrar información de un usuario.
     * @param  int  $id
     * @return Response
     */
    public function showProfile($id)
    {
        $user = User::findOrFail($id);
        return view('user.profile', ['user' => $user]);
    }
}
```

Todos los controladores tienen que extender la clase base `Controller`. Esta clase viene ya creada por defecto con la instalación de Laravel, la podemos encontrar en la carpeta `app/Http/Controllers`. Se utiliza para centralizar toda la lógica que se vaya a utilizar de forma compartida por los controladores de nuestra aplicación. Por defecto solo carga código para validación y autorización, pero podemos añadir en la misma todos los métodos que necesitemos.

En el código de ejemplo, el método `showProfile($id)` lo único que realiza es obtener los datos de un usuario, generar la vista `user.profile` a partir de los datos obtenidos y devolverla como valor de retorno para que se muestre por pantalla.

Una vez definido un controlador ya podemos asociarlo a una ruta. Para esto tenemos que modificar el fichero de rutas `routes.php` de la forma:

```
Route::get('user/{id}', 'UserController@showProfile');
```

En lugar de pasar una función como segundo parámetro, tenemos que escribir una cadena que contenga el nombre del controlador, seguido de una arroba `@` y del nombre del método que queremos asociar. No es necesario añadir nada más, ni los parámetros que recibe el método en cuestión, todo esto se hace de forma automática.

Crear un nuevo controlador

Como hemos visto los controladores se almacenan dentro de la carpeta `app/Http/Controllers` como ficheros PHP. Para crear uno nuevo bien lo podemos hacer a mano y rellenar nosotros todo el código, o podemos utilizar el siguiente comando de Artisan que nos adelantará todo el trabajo:

```
php artisan make:controller MoviesController --plain
```

Este comando creará el controlador `MoviesController` dentro de la carpeta `app/Http/Controllers` y lo completará con el código básico que hemos visto antes. Al añadir la opción `--plain` le indicamos que no añada ningún método al controlador, por lo que el cuerpo de la clase estará vacío. De momento vamos a utilizar esta opción para añadir nosotros mismos los métodos que necesitemos. Más adelante, cuando hablemos sobre controladores tipo RESTful, volveremos a ver esta opción.

Controladores y espacios de nombres

También podemos crear sub-carpetas dentro de la carpeta `Controllers` para organizarnos mejor. En este caso, la estructura de carpetas que creemos no tendrá nada que ver con la ruta asociada a la petición y, de hecho, a la hora de hacer referencia al controlador únicamente tendremos que hacerlo a través de su espacio de nombres.

Como hemos visto al referenciar el controlador en el fichero de rutas únicamente tenemos que indicar su nombre y no toda la ruta ni el espacio de nombres `App\Http\Controllers`. Esto es porque el servicio encargado de cargar las rutas añade automáticamente el espacio de nombres raíz para los controladores. Si metemos todos nuestros controladores dentro del mismo espacio de nombres no tendremos que añadir nada más. Pero si decidimos crear sub-carpetas y organizar nuestros controladores en sub-espacios de nombres, entonces sí que tendremos que añadir esa parte.

Por ejemplo, si creamos un controlador en `App\Http\Controllers\Photos\AdminController`, entonces para registrar una ruta hasta dicho controlador tendríamos que hacer:

```
Route::get('foo', 'Photos\AdminController@method');
```

Generar una URL a una acción

Para generar la URL que apunte a una acción de un controlador podemos usar el método `action` de la forma:

```
$url = action('FooController@method');
```

Por ejemplo, para crear en una plantilla con *Blade* un enlace que apunte a una acción haríamos:

```
<a href="{{ action('FooController@method') }}">¡Aprieta aquí!</a>
```

Controladores implícitos

Laravel también permite definir fácilmente la creación de controladores como recursos que capturen todas las rutas de un determinado dominio. Por ejemplo, capturar todas las consultas que se realicen a la URL "users" o "users" seguido de cualquier cosa (por ejemplo "users/profile"). Para esto en primer lugar tenemos que definir la ruta en el fichero de rutas usando `Route::controller` de la forma:

```
Route::controller('users', 'UserController');
```

Esto quiere decir que todas las peticiones realizadas a la ruta "users" o subrutas de "users" se redirigirán al controlador `UserController`. Además se capturarán las peticiones de cualquier tipo, ya sean GET o POST, a dichas rutas. Para gestionar estas rutas en el controlador tenemos que seguir un patrón a la hora de definir el nombre de los métodos: primero tendremos que poner el tipo de petición y después la sub-ruta a la que debe de responder. Por ejemplo, para gestionar las peticiones tipo GET a la URL "users/profile" tendremos que crear el método "getProfile". La única excepción a este caso es "Index" que se referirá a las peticiones a la ruta raíz, por ejemplo "getIndex" gestionará las peticiones GET a "users". A continuación se incluye un ejemplo:

```
class UserController extends BaseController
{
    public function getIndex()
    {
        //
    }

    public function postProfile()
    {
        //
    }

    public function anyLogin()
    {
        //
    }
}
```

Además, si queremos crear rutas con varias palabras lo podemos hacer usando la notación "*CamelCase*" en el nombre del método. Por ejemplo el método "getAdminProfile" será parseado a la ruta "users/admin-profile".

También podemos definir un método especial que capture las todas las peticiones "perdidas" o no capturadas por el resto de métodos. Para esto simplemente tenemos que definir un método con el nombre `missingMethod` que recibirá por parámetros la ruta y los parámetros de la petición:

```
public function missingMethod($parameters = array())
{
    //
}
```

Caché de rutas

Si definimos todas nuestras rutas para que utilicen controladores podemos aprovechar la nueva funcionalidad para crear una caché de las rutas. Es importante que estén basadas en controladores porque si definimos respuestas directas desde el fichero de rutas (como vimos en el capítulo anterior) la caché no funcionará.

Gracias a la caché Laravel indican que se puede acelerar el proceso de registro de rutas hasta 100 veces. Para generar la caché simplemente tenemos que ejecutar el comando de *Artisan*:

```
php artisan route:cache
```

Si creamos más rutas y queremos añadirlas a la caché simplemente tenemos que volver a lanzar el mismo comando. Para borrar la caché de rutas y no generar una nueva caché tenemos que ejecutar:

```
php artisan route:clear
```

> La caché se recomienda crearla solo cuando ya vayamos a pasar a producción nuestra web. Cuando estamos trabajando en la web es posible que añadamos nuevas rutas y sino nos acordamos de regenerar la caché la ruta no funcionará.