Rutas en Laravel

Las rutas (*routes*) son un mecanismo que permite a Laravel establecer qué respuesta enviar a una petición que intenta acceder a una determinada URL. Estas rutas se especifican en diferentes archivos dentro de la carpeta `routes` de nuestro proyecto Laravel.

Podríamos decir que existen dos tipos principales de rutas:

- Las rutas **web** (almacenadas en el archivo `routes/web.php` de la aplicación), que nos permitirán cargar distintas vistas en función de la URL que indique el cliente.
- Las rutas **API** (almacenadas en el archivo `routes/api.php`), a través de las cuales definiremos distintos servicios REST, como veremos también más adelante.

Nos vamos a centrar durante esta sesión en el primer grupo, por lo que editaremos el contenido del archivo `routes/web.php`. Estas rutas son las más habituales, y se utilizan para recuperar contenidos típicamente en formato HTML. Inicialmente ya existe una ruta predefinida hacia la raíz del proyecto, que carga la página de bienvenida al mismo

```php
<?php

use Illuminate\Support\Facades\Route;

Route::get('/', function() {
    return view('welcome');
});
```

Para definir una ruta en Laravel, se hace una llamada a un método estático de la clase `Route` (en el ejemplo anterior, al método `Route::get`). Como primer parámetro, especificaremos la URL de la ruta (la ruta raíz, en el ejemplo anterior), y como segundo parámetro, la función que se va a ejecutar cuando algún cliente haga una petición a esa ruta.

1. Rutas simples

Las rutas simples tienen un nombre de ruta fijo, y una función que responde a dicho nombre emitiendo una respuesta. Un ejemplo es la ruta raíz que viene por defecto en nuestro proyecto. Podríamos definir otra ruta a

continuación de esa, mediante la cual, si accedemos a la URL *http://biblioteca/fecha* nos muestre la fecha y hora actuales

```
Route::get('fecha', function() {
    return date("d/m/y h:i:s");
});
```

Si ponemos en marcha el servidor y accedemos a esa URL (o a *127.0.0.1:8000/fecha* si hemos lanzado la aplicación con `php artisan serve`), podremos ver esa fecha y hora actual.

2. Rutas con parámetros

Es también posible pasar **parámetros en la URL** de la ruta. Para ello, incluimos el nombre del parámetro entre llaves, y lo pasamos también a la función del segundo parámetro. Por ejemplo, si definimos una ruta para saludar al nombre que nos llega como parámetro, el código quedaría así:

```
Route::get('saludo/{nombre}', function($nombre) {
    return "Hola, " . $nombre;
});
```

En este caso, si el parámetro es obligatorio y no lo indicamos en la URL, nos redirigirá a una página de error 404. Para indicar que un parámetro no es obligatorio, se termina su nombre con un interrogante, y también conviene darle un valor por defecto en la función PHP de respuesta. Así modificaríamos la ruta anterior para que el nombre del usuario sea opcional y, en caso de no ponerlo, se le asigne el nombre "Invitado".

```
Route::get('saludo/{nombre?}', function($nombre = "Invitado") {
    return "Hola, " . $nombre;
});
```

2.1. Validación de parámetros

Algunos parámetros será preciso que sigan un determinado patrón. Por ejemplo, un identificador numérico sólo contendrá dígitos. Para asegurarnos de eso, podemos emplear el método `where` al definir la ruta. A este método le pasamos dos parámetros: el nombre del parámetro a validar, y la expresión regular que tiene que cumplir. En el caso del nombre anterior, si queremos que sólo contenga letras (mayúsculas o minúsculas), podemos hacer algo así:

```
Route::get('saludo/{nombre?}', function($nombre = "Invitado") {
    return "Hola, " . $nombre;
})->where('nombre', "[A-Za-z]+");
```

En caso de que la ruta no cumpla el patrón, se obtendrá una página de error. Más adelante se explicará cómo podemos personalizar estas páginas de error.

3. Rutas con nombre o *named routes*

En ocasiones puede ser conveniente asociar un nombre a una ruta. Especialmente, cuando esa ruta va a formar parte de un enlace en alguna página de nuestro sitio, ya que en un futuro la ruta podría cambiar, y de este modo evitamos tener que actualizar los enlaces con el nuevo nombre.

Para ello, al definir la ruta, le asociamos con la función `name` el nombre que queramos. Por ejemplo:

```
Route::get('contacto', function() {
    return "Página de contacto";
})->name('ruta_contacto');
```

Ahora, si queremos definir un enlace a esta ruta en cualquier parte, basta con emplear la función `route` de Laravel, indicando el nombre que le hemos asignado a esta ruta. Por lo tanto, en lugar de poner esto:

```
echo '<a href="/contacto">Contacto</a>';
```

Podemos hacer algo como esto otro, tal y como veremos más adelante cuando definamos nuestras vistas:

```
<a href="{{ route('ruta_contacto') }}">Contacto</a>
```

De este modo, ante futuros cambios en las rutas, sólo deberemos cambiar la URL en el correspondiente `Route::get` de `routes/web.php`.

4. Combinación de elementos en rutas

Podemos combinar varias cláusulas `where` en una ruta para validar distintos parámetros que pueda tener, y también enlazar estas llamadas con una a la función `name` para nombrar la ruta. Por ejemplo, la siguiente ruta espera recibir un nombre con caracteres, y un *id* numérico, ambos con valores por defecto:

```
Route::get('saludo/{nombre?}/{id?}',
function($nombre="Invitado", $id=0)
{
    return "Hola $nombre, tu código es el $id";
})->where('nombre', "[A-Za-z]+")
  ->where('id', "[0-9]+")
  ->name('saludo');
```

Si accedemos a cada una de las siguientes URLs, obtendremos cada una de las respuestas indicadas:

URL	Respuesta
/saludo	*Hola Invitado, tu código es el 0*
/saludo/Nacho	*Hola Nacho, tu código es el 0*
/saludo/Nacho/3	*Hola Nacho, tu código es el 3*
/saludo/3	Error 404 (URL incorrecta)

Notar que el último caso es incorrecto. No podemos especificar un *id* sin haber especificado un nombre delante, porque incumple el patrón de la URL. Se puede dejar un parámetro omitido, siempre y cuando los posteriores también lo estén.

5. Otros métodos de *Route*

Además de utilizar el método `get`, desde la clase `Route` también podemos acceder a otros métodos estáticos útiles, como `Route::post` (útil para recoger datos de formularios, por ejemplo), o también `Route::put`, `Route::delete` ... Los veremos con más detalle en secciones posteriores.