Validaciones con Laravel 5

Introducción a las validaciones de datos del usuario, entrada mediante HTTP Request, con el framework PHP Laravel 5.

Una de las necesidades fundamentales de cualquier aplicación en general, y aplicaciones web en particular, son las validaciones de la entrada de datos del usuario. Obviamente en Laravel se toman muchas molestias por proporcionarnos métodos de validación sencillos y potentes. Además no existe una única vía, por lo que habrá distintos caminos para realizar una validación.

Para comenzar, vamos a analizar la manera más rápida de validar, manteniendo la lógica en el controlador. Más adelante veremos alternativas que varían un poco esta propuesta, permitiendo separar el código de validaciones a clases diferentes, pensadas para ello, que colocaremos en la carpeta Requests (fíjate la "s" al final para distinguirlo de Request).

El ejemplo completo de validación que vamos a realizar está basado en las prácticas que nos sugiere la documentación oficial de Laravel 5.1 en la sección "ValidationQuickstart".

Creamos las rutas

Vamos a necesitar dos rutas en nuestra aplicación. La primera de ellas nos mostraría un formulario donde estarán aquellos campos que necesitemos que el usuario rellene. La segunda para validar los datos y realizar las acciones necesarias si son correctos o no.

En cuanto a la primera ruta, donde estará el formulario, tendremos algo como esto:

```
Route::get('producto/crear','ProductoController@create');
```

Esta ruta llamará a la acción "create" sobre el controlador "ProductoController", siempre que se acceda a la URI "producto/crear" con el verbo HTTP get.

En la segunda ruta realizaremos las acciones de validación y la parte de insertar la información en la base de datos, en el caso que la validación fuera correcta.

```
Route::post('producto','ProductoController@store');
```

Como puedes ver, esa ruta se activa con un envío por post sobre la URI "producto". Entonces se invoca a la acción "store" sobre el controlador "ProductoController".

Como has podido comprobar, en "ProductoControler" concentramos todo el trabajo del ejemplo de validación, es decir, las dos rutas van a diferentes acciones del mismo controlador.

Controlador con validación ProductoController

Ahora vamos a observar el controlador donde vamos a realizar las acciones invocadas por las rutas.

```
classProductoControllerextendsController
{
 // Aquí van los métodos del controlador
 // el código de los métodos está más abajo en el artículo
}
```

Comenzaremos por ver el método create(), que es invocado por la primera ruta. En tal método tenemos que invocar la vista donde está el formulario mediante el cual el usuario insertará la información que luego vamos a validar.

```
publicfunctioncreate(Request $request)
{
returnview('producto.create');
}
```

Nota: De momento nos importa poco la vista del formulario. Puedes poner un archivo que tendrá básicamente el código HTML de cualquier formulario con cualquier número de campos. Al final del artículo os pasaré el código completo de la vista que he usado yo en mi ejemplo.

Este método del controlador no necesita más código, porque solo se encarga de mostrar la vista del formulario. Donde tenemos que hacer las validaciones será en el siguiente método, que enseguida os mostramos.

TraitValidatesRequests en el controlador base

Laravel 5 incluye un trait que se carga en el controlador base (clase Controller), llamado "ValidatesRequests". Ese trait contiene código que estará disponible en todos los controladores que nosotros creemos, puesto que todos extienden la clase Controller.

Nota: Puedes encontrar el código del controlador base (clase Controller) en la ruta "app/Http/Controllers/Controller.php" y encontrarás la declaración de uso del trait en la línea:

```
useDispatchesJobs,ValidatesRequests;
```

No confundas ese "use" que es la primera línea de código de una clase, con un "use" que aparece en cualquier lugar del código para definir que estás usando cosas declaradas en otros namespaces.

Dentro del trait "ValidatesRequests" hay un método que nos sirve para hacer validaciones de los datos que nos llegan mediante HTTP Request, llamado validate(). Por tanto, en todos nuestros controladores podremos invocar este método para realizar validaciones de datos de entrada.

El método validate() recibe dos parámetros:

1. El objeto Request
2. Las reglas de validación que queramos definir.

El objeto Request ya lo conocemos de sobra, puesto que ya lo hemos tratado muchas veces. Lo inyectarás en el método del controlador como ya hemos detallado en el artículo HTTP Request en Laravel 5.

Las reglas de validación son diversas y perfectamente configurables por nosotros mediante una sencilla sintaxis. De momento vamos a conocer algunas reglas elementales, pero Laravel tiene muchas otras que trataremos más adelante. Aunque este tema lo puedes consultar de una manera muy rápida en la documentación oficial de Laravel 5.1 en la sección AvailableValidation Rules.

Para especificar las reglas, como veremos en el código del método store(), un poco más abajo, las indicamos con un array asociativo. Como llaves (key) del array usamos el nombre del campo que deseamos validar (atributo "name" de los campos de formulario) y como valores indicamos todas las reglas que se deben comprobar en dicho campo para considerarlo correcto.

Este sería el código de nuestro método store() del controlador, donde puedes ver la llamada a validate() para la validación del formulario de producto que has debido de crear en la anterior acción.

```
publicfunctionstore(Request $request)
{
$this->validate($request,[
'nombre'=>'required|max:255',
'descripcion'=>'required',
'precio'=>'required|numeric',
]);

echo'Ahora sé que los datos están validados. Puedo insertar en la base de
datos';
}
```

Comportamientos ante un válido / inválido

Cuando se llama el método validate(), como resultado de pasar un proceso de comprobación de los datos de entrada, pueden ocurrir dos cosas:

1. **Los datos eran válidos:** en cuyo caso simplemente se continúa el procesamiento del método, ejecutando las instrucciones que se hayan después de la llamada a validate().
2. **Los datos eran inválidos:** en cuyo caso se hace una redirección a la ruta anterior, donde estaba el formulario. (Aunque cabe señalar que en conexiones Ajax este comportamiento puede tener variaciones, puesto que se tendrá que generar un JSON y enviar como respuesta al cliente, algo que no vamos a tratar ahora).

En el código del método store(), que presentamos unos párrafos atrás, comprobarás que después de invocar el método validate() se pone el código que realiza las acciones pertinentes. Nosotros hemos colocado un simple "echo", lanzando un mensaje, pero generalmente aquí habría que llamar al modelo que se encarga de guardar esos datos en la base de datos.

Mostrar los errores de validación

Con todo el código anterior ya tenemos implementado todo el proceso de validación, en esta dinámica de hacerlo dentro del controlador. Realmente has observado que el corazón del proceso es el método validate() presente en todos los controladores. Sin embargo, para completar este ejercicio debemos realizar algunas tareas sobre la vista para mejorar la usabilidad del formulario, ayudando al usuario a identificar qué es lo que ha ocurrido y volviendo a poblar el formulario. Serán por tanto dos tareas las que nos faltan:

- Mostrar los errores de validación
- Recuperar los datos en los campos de formulario que se habían escrito anteriormente

Los errores de validación los vamos a extraer de una variable llamada **$errors, que está disponible en toda vista.**

Laravel automáticamente crea la variable $errorsflasheando en la sesión todos los errores de validación que se hayan podido producir. Si no hubo tal error de validación Laravel crea igualmente esa variable, pero vacía, por lo que podemos usarla siempre que queramos, sin preocuparnos si está o no está definida, porque siempre lo va a estar.

Además el bindeo se realiza automáticamente a la vista, por lo que nosotros no tenemos que hacer absolutamente nada para pasar esa variable.

Por tanto, en nuestra vista podemos fácilmente recorrer los errores definidos en $errors y mostrarlos en el formato que nosotros deseemos. A continuación tendrías un pedazo de código que se encarga de recorrer esos errores, mostrando un elemento DIV con los errores en el caso que haya alguno, y listando cada uno en un elemento LI de una lista UL.

```
@if(count($errors)>0)
        <div class="errors">
                <ul>
                @foreach($errors->all()as$error)
                        <li>{{$error}}</li>
                @endforeach
                </ul>
        </div>
@endif
```

De momento este código te puede parecer un poco extraño, porque no hemos hablado de la sintaxis del sistema de templatesBlade, el oficial de Laravel. No te preocupes porque lo tocaremos más adelante y podrás entenderlo perfectamente. No obstante, **para que te**

funcione, es muy importante que nombres el archivo de esta vista con extensión ".blade.php", porque si no, el código "Blade" no se ejecutaría.

Nota: Los mensajes de error aparecen en inglés, pero es muy fácil obtener las traducciones. En Github hay un proyecto que te las ofrece ya listas en una serie de idiomas.

Poblar de nuevo los valores de los campos de formulario

Primero decir que esta otra parte del trabajo en la vista realmente nos la podríamos ahorrar en Laravel, ya que hay un método muy sencillo por el que Laravel nos lo hace automáticamente. Se trataría de usar la librería que viene con Laravel para generar código de los formularios, pero como no la hemos visto todavía, nos toca hacerlo "a mano", aunque comprobarás que tampoco es nada doloroso.

Simplemente en los campos INPUT hemos creado el value, asignando lo que nos devuelve el helperold(), indicando el campo que queremos recuperar. Esa función se encarga de traerse aquello que estaba escrito anteriormente en el campo del formulario, tal como explicamos en el artículo Volcado de la entrada de datos de usuario a la sesión.

```
<input type="text" name="nombre" value="{{old('nombre')}}">
```

Este código no necesita muchas más explicaciones, pero simplemente tómalo como una alternativa, sabiendo que hay mejores maneras de hacer esto, si le dejas a Laravel la responsabilidad de crear el HTML de tu formulario.

Código completo de la vista

Como prometí, aquí va el listado completo de la vista que he usado para este ejercicio.

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>Crear un producto</title>
        <style>
        .errors{
                background-color: #fcc;
                border: 1px solid #966;
        }
        form{
                margin-top: 20px;
                line-height:1.5em;
        }
        label{
                display: inline-block;
                width: 120px;
        }
        </style>
</head>
<body>
```

```
        <h1>Crear un producto</h1>
        @if(count($errors)>0)
                <div class="errors">
                        <ul>
                        @foreach($errors->all()as$error)
                                <li>{{$error}}</li>
                        @endforeach
                        </ul>
                </div>
        @endif

        <form action="/producto" method="post">
                <label for="nombre">Nombre:</label><input type="text"
name="nombre" value="{{old('nombre')}}">
                <br />
                <label for="descripcion">Descripción:</label><input
type="text" name="descripcion" value="{{old('descripcion')}}">
                <br />
                <label for="precio">Precio:</label><input type="text"
name="precio" value="{{old('precio')}}">
                <br />
                <input type="submit" value="Crear">
        </form>
</body>
</html>
```

Por favor, ten en cuenta que esta vista tiene un código muy elemental, simplemente para salir del paso. Observa estos detalles:

- No estoy pasando el token para protección CSRF, por lo que tendrás que desactivar el middleware asociado. Esto lo hemos explicado ya en varias ocasiones, así que debes saber a qué me refiero. Si no, revisa el artículo Verbos en las rutas de Laravel.
- Guarda el archivo en la carpeta correspondiente. Tal como quedó el código del método create() del controlador, sería en la carpeta "resources/views/producto".
- El nombre del archivo será create.blade.php. Ese nombre lo marcamos al invocar la vista, desde el controlador ProductoController, en la acción create().