



Universidad
Internacional
de Valencia

Implementación de validaciones del lado del servidor con Laravel

Titulación:

Máster Universitario en Desarrollo de
Aplicaciones y Servicios Web

Curso académico: 2022-2023

Alumno/a: Gómez Olivencia, Rubén

D.N.I.: 78910013-A

Asignatura: Seguridad Web

Índice general

1. Introducción	3
2. Laravel	3
3. Desarrollo realizado	3
3.1. Sistema de autenticación en Laravel	3
3.1.1. Seguridad durante el login	4
3.1.2. Bloqueo de cuentas por fallos de autenticación	4
3.2. Sistema de validación durante el registro	5
3.2.1. Creación de campos propios	6
3.2.1.1. Modificación de la base de datos	7
3.2.2. Reglas propias de validación	7
3.2.3. Mantener valores en el formulario tras un error	9
3.2.4. No permitir “copiar-pegar” la contraseña	10
3.3. Traducciones	11
4. Conclusiones	11

1. Introducción

La seguridad en un proyecto informático es algo que se debe pensar desde el inicio del mismo, ya que las decisiones tomadas pueden repercutir en las iteraciones posteriores y en la evolución del producto que estemos realizando.

A lo largo de este documento se van a explicar las decisiones tomadas, tanto en el ámbito de programación como de diseño, durante la creación de un formulario de registro para un portal web creado con el *framework* [Laravel](#).

2. Laravel

Laravel es un *framework* desarrollado en el año 2011 que hace uso del lenguaje de programación [PHP](#) para la creación de desarrollos basados en la tecnología web.

La idea de todo *framework* es la de crear un entorno de trabajo que unifique distintos conceptos, prácticas y criterios que sirvan como referencia para la creación de un proyecto.

Una de las muchas ventajas que nos proporciona Laravel es la de un sistema modular para la creación de validaciones en formularios, tal como nos indica su [documentación](#).

3. Desarrollo realizado

Una vez tenemos realizado el despliegue, se va a profundizar en el desarrollo realizado, destacando los siguientes aspectos:

3.1. Sistema de autenticación en Laravel

Laravel cuenta con un sistema de [autenticación](#) que nos permite la creación de usuarios de manera sencilla y también gestionar el contenido de las vistas dependiendo de si el usuario ha realizado *login* o es un invitado.

Aparte, gracias al sistema de [frontend scaffolding](#), podemos crear las típicas

vistas de registro de usuario, login, reseteo de contraseñas, etc., de manera automática, pudiendo elegir entre tres frameworks: [Bootstrap](#), [Vue](#) o [React](#).

Este sistema de autenticación ha sido modificado para que sea más seguro y realizando modificación en los campos que se piden durante el registro.

3.1.1. Seguridad durante el login

Una de las medidas de seguridad más sencilla de implementar es la de no dar información de más cuando existe algún error durante el login de un usuario.

Si un atacante consigue un listado de posibles e-mails que pertenecen a nuestra aplicación, si al intentar el login se le indica que esa cuenta no existe, pasará a realizar el ataque con otra cuenta. Mientras que si no se le indica nada, no obtendrá información y por tanto es posible que intente seguir con la misma cuenta más tiempo.

Error:

- Credenciales no válidas.

Para mostrar este mensaje, y los del registro, se ha creado un template en `resources/views/layouts/_errors.blade.php` que genera un listado con todos los errores, como aparece en la imagen.

Por otro lado, para no dar pistas, también se ha modificado el fichero `resources/views/auth/login.blade.php` quitando los bordes rojos que aparecen por defecto indicando si el error se ha producido en el login o en la contraseña.

3.1.2. Bloqueo de cuentas por fallos de autenticación

Para asegurar que no se realiza un ataque por fuerza bruta de contraseñas Laravel permite el bloqueo una vez realizado un número de intentos fallidos. Por defecto la configuración se encuentra en el fichero

`Illuminate/Foundation/Auth/ThrottlesLogins.php` dentro del directorio

`vendor/laravel/framework/src/`, donde aparece que el número máximo

de intentos es de 5 y el bloqueo se efectuará durante 1 minuto.

Para ajustar esta configuración a nuestras necesidades, se ha modificado el fichero `app/Http/Controllers/Auth/LoginController.php` en el que añadiremos las variables `$decayMinutes` y `$maxAttempts` quedando:

</> Añadiendo seguridad durante el login

```
<?php
...
class LoginController extends Controller
{
    use AuthenticatesUsers;
    protected $redirectTo = RouteServiceProvider::HOME;

    //variables modificadas por seguridad
    public $decayMinutes = 180;
    public $maxAttempts = 2;
    ...
}
```

De esta manera, si se intenta hacer login con una cuenta existente y se falla la contraseña 2 veces, se bloqueará el acceso apareciendo el siguiente error:

Error:

- Demasiados intentos de acceso. Inténtelo de nuevo en 10795 segundos.

Tal como aparece en el mensaje, ha habido demasiados intentos de acceso fallidos y habrá que esperar un número de segundos (los 180 minutos indicados en la configuración).

3.2. Sistema de validación durante el registro

A la hora de registrarse en una plataforma, hay que asegurarse que los datos introducidos son válidos. Esta validación no sólo se debe realizar en el frontend, ya que ese sistema es fácilmente traspasable con unos mínimos conocimientos.

Los datos deben de validarse antes de acceder a la base de datos, y por tanto se debe de realizar en el **backend** de la aplicación web.

Como ya se ha dicho previamente, Laravel cuenta con una [documentación extensa](#) acerca de su sistema de validación indicando cómo lo podemos ampliar para adecuarlo a nuestras necesidades, tal como veremos a continuación.

3.2.1. Creación de campos propios

El sistema de registro de Laravel (a través del sistema de autenticación propio) crea una página donde se puede insertar nombre, e-mail y contraseña.

Para añadir nuevos campos debemos modificar el fichero de la vista `resources/views/auth/register.blade.php` y añadir los campos que nos interese.

Modificaremos `app/Http/Controllers/Auth/RegisterController.php` para que estos campos posteriormente se inserten en la base de datos, quedando la función **create** de la siguiente manera:

</> Código para la creación de usuario

```
<?php
...
protected function create(array $data) {
    return User::create([
        'name'      => $data['name'],
        'surname'   => $data['surname'],
        'dni'       => $data['dni'],
        'email'     => $data['email'],
        'password'  => Hash::make($data['password']),
        'telephone' => $data['telephone'],
        'iban'      => str_replace(' ', '', $data['iban']),
        'about'     => $data['about'],
        'country_id' => $data['country_id'],
    ]);
}
```

3.2.1.1. Modificación de la base de datos

Para que los cambios del paso anterior funcionen, lógicamente hay que realizar la modificación de la base de datos para aceptar los nuevos campos.

Para ello se ha creado un fichero **migrate** en el que se indica los nuevos campos que se deben añadir a la tabla **users**.


3.2.2. Reglas propias de validación

Con los nuevos campos creados en la base de datos y con el formulario modificado, es momento de realizar las validaciones oportunas, paso previo a la inserción de datos en MySQL.

Laravel nos permite realizar validaciones sencillas teniendo en cuenta el campo, el tipo de datos, longitud...

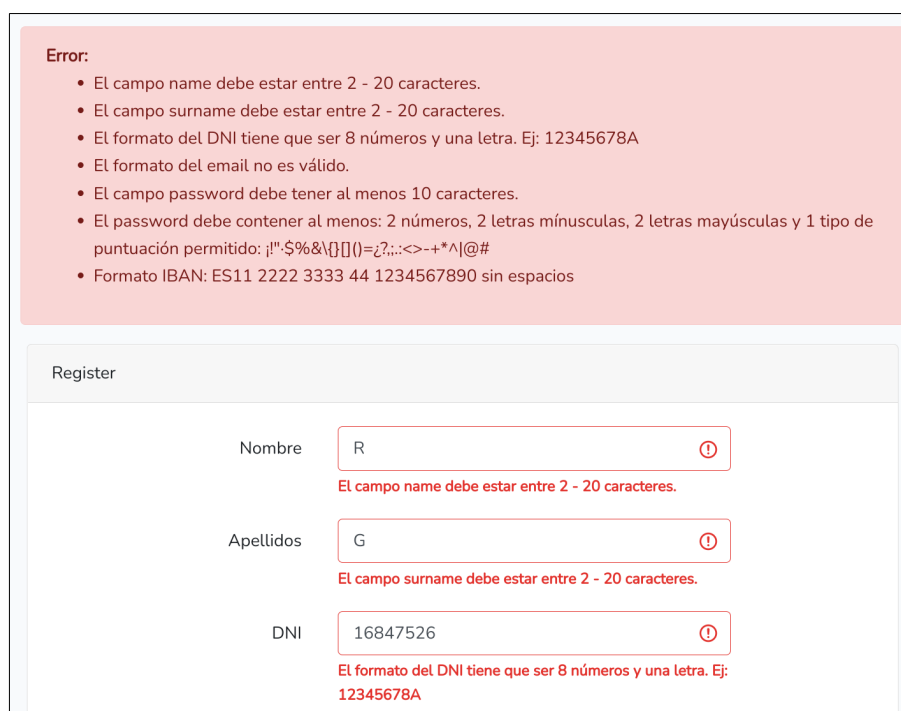
</> Validaciones realizadas

```
<?php
...
protected function validator(array $data)
{
    return Validator::make($data, [
        'name' => ['required', new name, 'between:2,20'],
        'surname' => ['required', new name, 'between:2,20'],
        'dni' => ['required', new dni, 'max:9'],
        'email' => ['required', 'email:filter', 'max:255', 'unique:users'],
        'password' => ['required', 'string', 'min:12', new password, 'confirmed'],
        'telephone' => ['nullable', new telefono, 'max:13'],
        'iban' => ['required', new iban, 'max:24'],
        'about' => ['nullable', 'string', 'between:20,250'],
        'country_id' => ['nullable', 'integer']
    ]);
}
```

En las validaciones expuestas, se han realizado validaciones propias para los campos **nombre**, **apellidos**, **dni**, **password**, **teléfono** e **IBAN**. Estas validaciones propias se han creado dentro del directorio  **app/rules**, donde cada fichero es una validación y contienen una clase con dos funciones principales:

- **passes:** función que recibe el contenido del campo del formulario y en el que realizaremos la validación. Las validaciones realizadas para este proyecto se han solucionado con una o varias expresiones regulares.
- **messages:** devuelve el mensaje de error en caso de que la función “passes” no devuelva true.

En caso de que existan varios mensajes de error, aparecerán todos seguidos e indicando cuáles son los campos que han dado error, a través de un borde rojo, así como el mismo mensaje de error de validación para que el usuario lo tenga en cuenta:



Error:

- El campo name debe estar entre 2 - 20 caracteres.
- El campo surname debe estar entre 2 - 20 caracteres.
- El formato del DNI tiene que ser 8 números y una letra. Ej: 12345678A
- El formato del email no es válido.
- El campo password debe tener al menos 10 caracteres.
- El password debe contener al menos: 2 números, 2 letras minúsculas, 2 letras mayúsculas y 1 tipo de puntuación permitido: !"\$.%& \{}()=¿?,:;<>-+*^|@#
- Formato IBAN: ES11 2222 3333 44 1234567890 sin espacios

Register

Nombre !
El campo name debe estar entre 2 - 20 caracteres.

Apellidos !
El campo surname debe estar entre 2 - 20 caracteres.

DNI !
El formato del DNI tiene que ser 8 números y una letra. Ej: 12345678A

Una de las validaciones más importantes realizadas ha sido la de la contraseña, la que se ha forzado para que tenga que ser:

- Al menos 10 caracteres, de los cuales:
- Al menos 2 tienen que ser letras minúsculas.
- Al menos 2 tienen que ser letras mayúsculas.
- Tiene que haber al menos 2 números.


- Tiene que incluir al menos 1 símbolo tipográfico.

A continuación se expone la función **passes** para la regla que valida si la contraseña se ha introducido de manera correcta.

En esta función se comprueba que la contraseña contiene los caracteres que se piden para que se considere una contraseña segura, ya que la longitud se ha comprobado en la función **validator** vista **previamente**.

Función **passes** del fichero app/Rules/password.php

```
<?php
...
public function passes($attribute, $value) {
    return
        // https://www.php.net/manual/en/regexp.reference.unicode.php
        preg_match('/(.*\d){2,}/u', $value) // dígitos
        && preg_match("/(.*\p{Ll}){2,}/u", $value) //letras minúsculas
        && preg_match("/(.*\p{Lu}){2,}/u", $value) //letras mayúsculas
        && preg_match('/(.*\W){1,}/u', $value) // \W -> signos de puntuación
    ;
}
```

De cara a futuro, se podría mejorar creando unas variables de entorno en el fichero  **.env** donde se indique el la dificultad de la contraseña a introducir.

El resto de validaciones se han realizado también mediante **expresiones regulares** en su fichero de reglas correspondiente.

3.2.3. Mantener valores en el formulario tras un error

Tal como se ha visto en el paso anterior, en caso de que el registro devuelva un error durante la validación, se le indicará al usuario, indicando los campos que debe corregir.

Es por eso que es importante mantener los valores que ha introducido previamente, ya que de esta manera estaremos facilitando la usabilidad de la aplicación.

Esto se ha conseguido en la vista del registro indicando en los parámetros **value**

el valor previo que nos permite obtener Laravel.

</> Valores antiguos en el formulario

```
<input id="dni" type="text" class="form-control
    @error('dni') is-invalid @enderror" name="dni"
    value="{{ old('dni') }}"
    required autocomplete="dni" autofocus>
```

Esto se ha repetido para todos los campos, a excepción de la contraseña.

3.2.4. No permitir “copiar-pegar” la contraseña

Para mejorar la seguridad y evitar que algún atacante, mediante ingeniería social delante del ordenador, pueda copiar la contraseña introducida, se han modificado los campos de contraseña y validación de contraseña para no permitir copiar, seleccionar o pegar la contraseña o su confirmación.

No permitiendo “copiar-pegar” también nos estamos asegurando que el usuario introduce de manera correcta la contraseña que ha elegido, minimizando el riesgo de equivocarse.

</> No permitimos “copiar-pegar” en el input

```
<input id="password" type="password" class="form-control
    @error('password') is-invalid @enderror"
    name="password" required autocomplete="new-password"
    onselectstart="return false"
    onpaste="return false;"
    onCopy="return false"
    onCut="return false"
    onDrag="return false"
    onDrop="return false"
    autocomplete=off/
>
```

Con el código expuesto, evitamos copiar, cortar, seleccionar, arrastrar y pegar el contenido de los textos que serán guardadas como contraseñas.

3.3. Traducciones

Tal como se ha visto en la imágenes, los mensajes de validación aparecen en castellano, ya que se han añadido traducciones de los mensajes originales y porque la aplicación ha sido configurada con el castellano como idioma principal.

Las traducciones de los mensajes se han añadido en el directorio `resources/lang/es`, manteniendo la misma estructura de ficheros existente en el idioma inglés.

Para realizar parte de las traducciones se ha utilizado como base las traducciones ya realizadas por Marco Gomes de su [proyecto de Github](#).

Las traducciones para los campos propios del proyecto se han añadido al fichero `formulario.php` dentro del directorio indicado previamente.

4. Conclusiones

Hoy en día es habitual que en cualquier portal web de servicios haya que realizar un registro con ciertos datos personales. Es importante que estos datos sean validados y que se compruebe que la seguridad del portal no se ve comprometida por los datos introducidos.

Es por eso que, previa a la inserción de los datos en la base de datos, es obligatorio realizar una validación de los datos introducidos en los formularios, tal como hemos visto a lo largo de este documento.

Aunque de primeras quizá pueda parecer una labor tediosa, los framework como Laravel facilitan esa labor. Por eso no hay excusa para no hacer uso de estos sistemas de validaciones, que por otro lado también ayudan al usuario a utilizar nuestras aplicaciones mejorando la usabilidad de la misma.