UD10.9 – Laravel

Sistema de validación de usuarios propio

2º CFGS
Desarrollo de Aplicaciones Web

2022-23

1.- Introducción

Como se ha estudiado en el apartado anterior Laravel ofrece los sistemas Breeze y JetStream que implementan automáticamente un sistema de validación de usuarios con muchas características interesantes.

En ocasiones, personalizar los sistemas Breeze y JetStream es complicado, e incluso hay ocasiones en las que no son necesarias todas las características de estos sistemas.

Para estas ocasiones puede ser interesante crear un sistema de validación de usuarios propio.

2.- Sistema de validación de usuarios propio

Para añadir un sistema de validación propio los pasos a seguir son los siguientes:

- Definir la tabla para usuarios.
- Definir el modelo para usuarios.
- Definir una vista con el formulario de registro.
- Definir una vista con el formulario de login.
- Definir un controlador para gestionar las acciones del registro/login.
- Añadir rutas para el registro/login.
- Proteger rutas que sean para el acceso restringido.
- Añadir opciones de logout.

3.- Definir la tabla para usuarios

Se puede crear una tabla propia para ello o bien utilizar la tabla que ofrece Laravel a través de la migración que se crea por defecto.

Se recomienda hacer uso de la tabla Users que ofrece Laravel para simplificar procesos.

Si aún no se han ejecutado las migraciones, se pueden añadir los campos nuevos a la migración **Users** que Laravel crea por defecto al crear el proyecto.

Migración original:

```php
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('username');
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->enum('rol', ['user', 'admin'])
              ->default('user');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

3.- Definir la tabla para usuarios

Si ya se habían ejecutado las migraciones, se deberá crear una migración nueva que modifique la tabla Users.

La migración puede llamarse **add_fields_to_users_table**.

En esa migración en el **método up** se deben **añadir los campos** que se necesiten.

En el **método down** se debe **eliminar los campos** añadidos.

```php
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('username')->after('id');
        $table->enum('rol', ['user', 'admin'])
              ->default('user')
              ->after('password');
    });
}
```

```php
public function down()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('username');
        $table->dropColumn('rol');
    });
}
```

4.- Definir el modelo para usuarios

Se puede usar el modelo **app\Http\Models\User.php** que Laravel ofrece por defecto.

Si se han añadido campos se deben añadir también en el modelo.

5.- Definir vistas para el registro y para el login

Para mantener una buena organización, las vistas relacionadas con el registro y acceso se pueden crear dentro de una carpeta llamada **auth**

- resources\views\auth**register**.blade.php
  Contendrá el formulario con los campos para el registro.

- resources\views\auth**login**.blade.php
  Contendrá el formulario con los campos para el login.

5.- Definir vistas para el registro y para el login

Formulario de registro

resources\views\auth**register**.blade.php

```
<form action="{{ route('registro') }}" method="POST">
    @csrf

    <label for="username">Nombre de usuario:</label> <br>
    <input type="text" name="username" id="username" value="{{old('username')}}"> <br>

    <label for="name">Nombre completo:</label> <br>
    <input type="text" name="name" id="name" value="{{old('name')}}"> <br>

    <label for="email">Email:</label> <br>
    <input type="email" name="email" id="email" value="{{old('email')}}"> <br>

    <label for="password">Contraseña:</label> <br>
    <input type="password" name="password" id="password"> <br>

    <label for="password_confirmation">Repite Contraseña:</label> <br>
    <input type="password" name="password_confirmation" id="password_confirmation"> <br>

    <input type="submit" name="enviar" value="Enviar">
</form>

@if ($errors->any())
    <ul>
        @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
        @endforeach
    </ul>
@endif
```

5.- Definir vistas para el registro y para el login

Para crear las reglas de validación para los campos del formulario se tiene que crear un objeto del tipo **Request**:

```
php artisan make:request RegisterRequest
```

Se debe añadir arriba del todo la instrucción:

```
use Illuminate\Validation\Rules;
```

```php
class RegisterRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, mixed>
     */
    public function rules()
    {
        return [
            'username' => ['required', 'string', 'min:5', 'max:20', 'unique:users'],
            'name' => ['required', 'string', 'min:2', 'max:255'],
            'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
            'password' => ['required', 'confirmed', Rules\Password::defaults()],
        ];
    }

    public function messages()
    {
        return [
            'username.required' => 'El nombre de usuario es obligatorio.',
            'username.min' => 'El nombre de usuario debe tener como mínimo 5 caracteres.',
            'username.max' => 'El nombre de usuario debe tener como máximo 20 caracteres.',
            'username.unique' => 'El nombre de usuario ya existe.',
            'name.required' => 'El nombre completo es obligatorio.',
            'name.unique' => 'El nombre completo es obligatorio.',
            'name.max' => 'El nombre completo debe tener como máximo 255 caracteres.',
            'email.required' => 'El email de usuario es obligatorio.',
            'email.max' => 'El email debe tener como máximo 255 caracteres.',
            'email.unique' => 'El email ya existe.',
            'password.required' => 'La contraseña es obligatoria.',
            'password.confirmed' => 'Las contraseñas no coinciden.',
            'password.min' => 'La contraseña debe tener al menos 8 caracteres.',
        ];
    }
}
```

5.- Definir vistas para el registro y para el login

Formulario de login

resources\views\auth**login**.blade.php

```
<form action="{{ route('login') }}" method="POST">
    @csrf
    <label for="username">Nombre de usuario:</label> <br>
    <input type="text" name="username" id="username"> <br>

    <label for="password">Contraseña:</label> <br>
    <input type="password" name="password" id="password"> <br>

    <input type="submit" name="enviar" value="Enviar">
</form>

@if ($errors->any())
    <ul>
        @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
        @endforeach
    </ul>
@endif
```

6.- Definir un controlador para registro/login

El controlador se encargará de gestionar las rutas de registro, login y logout.
Se crea LoginController para separar las acciones de registro/login de las propias de los usuarios (ver cuenta, editar datos propios…).

```
php artisan make:controller LoginController
```

```php
use App\Http\Requests\RegisterRequest;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;

You, 4 minutes ago | 1 author (You)
class LoginController extends Controller
{
    public function registerForm()
    {
        return view('auth.register');
    }

    public function register(RegisterRequest $request)
    {
        $user = new User();
        $user->username = $request->get('username');
        $user->name = $request->get('name');
        $user->email = $request->get('email');
        $user->password = Hash::make($request->get('password'));
        $user->save();

        Auth::login($user);

        return redirect()->route('users.account');
    }
```

```php
    public function loginForm()
    {
        if (Auth::viaRemember()) {
            return 'bienvenido de nuevo';
        }else
                if(Auth::check())
            return redirect()->route('users.account');
        else
            return view('auth.login');
    }

    public function login(Request $request)
    {
        $credenciales = $request->only('username', 'password');

        if (Auth::guard('web')->attempt($credenciales))
        {   // Autenticación exitosa
            $request->session()->regenerate();
            return redirect()->route('users.account');
        } else {
            $error = 'Error al acceder a la aplicación';
            return view('auth.login', compact('error'));
        }
    }

    public function logout(Request $request)
    {
        Auth::guard('web')->logout();
        $request->session()->invalidate();
        $request->session()->regenerateToken();
        return redirect('/');
    }
}
```

7.- Añadir rutas para registro/login

```php
Route::get('registro', [LoginController::class, 'registerForm']);
Route::post('registro', [LoginController::class, 'register'])->name('registro');
Route::get('login', [LoginController::class, 'loginForm']);
Route::post('login', [LoginController::class, 'login'])->name('login');
Route::get('logout', [LoginController::class, 'logout'])->name('logout');

Route::get('cuenta', function() {
    return view('auth.account');
})->name('users.account')
  ->middleware('auth');
```

La ruta **cuenta** en el ejemplo devuelve una vista directamente pero podría diseñarse un controlador UserController que tuviera los diferentes métodos para las gestiones que puede tener un usuario desde su panel.

8.- Recordar login

Si se quiere implementar el recordar el login, se debe añadir un checkbox en el formulario de login.

Si el checkbox está marcado cuando se comprueban las credenciales con el método **attempt** se debe pasar como segundo el valor **true**.

Para que funcione, tiene que existir el campo **remember_token** en la tabla Users.

```php
public function login(Request $request)
{
    $credenciales = $request->only('username', 'password');
    $recordar = (request()->remember) ? true : false;
    if (Auth::guard('web')->attempt($credenciales, $recordar))
    {   // Autenticación exitosa
        $request->session()->regenerate();
        return redirect()->route('users.account');
    } else {
        $error = 'Error al acceder a la aplicación';
        return view('auth.login', compact('error'));
    }
}
```

https://laravel.com/docs/9.x/authentication#remembering-users

8.- Recordar login

Para cambiar el tiempo durante el cuál se recuerda el login se debe añadir la siguiente línea al archivo **config/auth.php**.

```
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'users',
        'remember' => 1440,
    ],
],
```

Se debe indicar el número de minutos que durará la cookie que permite recordar el login. (1440 es un día: 24horas*60minutos).