



LARAVEL: MIDDLEWARE Y AUTENTIFICACIÓN DE USUARIOS

DISEÑO DE SISTEMAS SOFTWARE

Contenido

1. *Middleware* o Filtros
2. Rutas avanzadas
3. Autenticación de usuarios

Laravel: Middleware y autenticación de usuarios

***MIDDLEWARE* O FILTROS**

Middleware o filtros

- Permiten realizar validaciones antes o después de que se ejecute el código asociado a una ruta
- Los filtros se definen como una clase PHP almacenada dentro de la carpeta `app/Http/Middleware`
- Cada middleware aplicará un filtro concreto sobre una petición y podrá permitir su ejecución, dar un error o redireccionar
- Laravel incluye por defecto algunos filtros “auth”, “guest”, “csrf”, ...
- Para crear nuestros propios filtros podemos usar el comando de Artisan:

```
php artisan make:middleware MyMiddleware
```

Middleware o filtros

- Ejemplo de código de un middleware propio:

```
<?php
namespace App\Http\Middleware;
use Closure;

class MyMiddleware {
    public function handle($request, Closure $next) {
        return $next($request);
    }
}
```

- Podemos devolver:
 - Para que continúe la petición: `return $next($request);`
 - Redirección a otra ruta: `return redirect('home');`
 - Lanzar excepción/abortar: `abort(403, 'Unauthorized');`

Uso de *Middleware* o filtros

- Para poder utilizar un middleware propio tendremos que añadirlo al fichero `app/Http/Kernel.php`
- Esta clase define tres arrays:
 - `$middleware`: si lo añadimos a este array se ejecutará en **TODAS** las peticiones
 - `$middlewareGroups`: para que se ejecute solo dentro de un grupo de rutas (web o api)
 - `$routeMiddleware`: si lo añadimos aquí lo podremos usar de forma separada para las rutas que indiquemos

Uso de *Middleware* o filtros

- Ejemplo:

```
protected $routeMiddleware = [  
    'auth' => \Illuminate\Auth\Middleware\Authenticate::class,  
    'can' => \Illuminate\Auth\Middleware\Authorize::class,  
    'myfilter' => \App\Http\Middleware\MyMiddleware::class,  
];
```

Uso de *Middleware* o filtros

- Una vez añadido a `$routeMiddleware` podemos proteger las rutas individualmente.
- Para asociar un filtro a una ruta tenemos que modificar el fichero de rutas `routes/web.php`:

```
Route::get('user', function() {  
    return 'Has superado el filtro!';  
})->middleware('myfilter');  
  
// Podemos indicar varios filtros..  
Route::get('user', 'UserController@showProfile')  
    ->middleware('auth', 'myfilter');
```


Laravel: Middleware y autenticación de usuarios

RUTAS AVANZADAS

Grupos de rutas

- Podemos agrupar varias rutas y aplicar un filtro a todas ellas a la vez:

```
Route::group(['middleware' => 'auth'], function() {  
    Route::get('/', function() {  
        //  
    });  
  
    Route::get('user/profile', function() {  
        //  
    });  
});
```

Grupos de rutas con prefijo

- También podemos usar los grupos de rutas para aplicar un prefijo a todas ellas, por ejemplo:

```
Route::group(['prefix' => 'api'], function() {  
    Route::group(['prefix' => 'v1'], function() {  
        Route::get('recurso', 'ControllerAPIv1@index');  
        Route::post('recurso', 'ControllerAPIv1@store');  
        Route::get('recurso/{id}', 'ControllerAPIv1@show');  
    });  
  
    Route::group(['prefix' => 'v2'], function() {  
        Route::get('recurso', 'ControllerAPIv2@index');  
        Route::post('recurso', 'ControllerAPIv2@store');  
        Route::get('recurso/{id}', 'ControllerAPIv2@show');  
    });  
});
```

Laravel: Middleware y autenticación de usuarios

AUTENTIFICACIÓN DE USUARIOS

Control de usuarios

- Laravel incluye métodos y clases que hacen que la implementación y uso del control de usuarios sea muy sencilla
- La configuración del sistema de autenticación se puede encontrar en el fichero `config/auth.php`, en el cual podremos:
 - Cambiar el sistema de autenticación (“Eloquent” por defecto)
 - Cambiar el modelo de datos (“User” por defecto)
 - Cambiar la tabla de usuarios (“users” por defecto)
- Al crear un nuevo proyecto de Laravel ya se incluye el modelo “User” en la carpeta “app” configurado para utilizar la autenticación

Control de usuarios

- También se incluye la migración de la tabla `users` con el siguiente esquema (función `up()`):

```
Schema::create('users', function($table) {  
    $table->increments('id');  
    $table->string('name');  
    $table->string('email')->unique();  
    $table->string('password');  
    $table->rememberToken();  
    $table->timestamps();  
});
```

- **Importante:**
 - Incluye un `id` único autoincremental para identificar a los usuarios
 - El campo `email` es `unique`
 - El campo `password` estará cifrado mediante el método `bcrypt()`
 - Podemos añadir todos los campos que queramos a esta tabla, por ejemplo apellidos, dirección, teléfono, etc.

Controladores

- Laravel también incorpora por defecto los controladores para la gestión de usuarios:
 - `LoginController` y `RegisterController`:
Incluyen métodos para ayudarnos en el proceso de autenticación (o *login*), registro y cierre de sesión
 - `ResetPasswordController` y `ForgotPasswordController`:
Contienen la lógica para ayudarnos en el proceso de restaurar una contraseña
- Los podemos encontrar en la carpeta (y espacio de nombres):
`App\Http\Controllers\Auth`

Generar rutas y vistas

- Lo único que falta son las rutas y las vistas, estas no vienen por defecto pero las podemos generar mediante el comando:

```
php artisan make:auth
```

- Al ejecutar este comando se añadirán todas las vistas necesarias a la carpeta `resources/views/auth` y las rutas al fichero `routes/web.php`
- Si editamos el fichero `routes/web.php` podremos ver que únicamente nos ha añadido las siguientes dos líneas:

```
Auth::routes();  
Route::get('/home', 'HomeController@index');
```


Rutas

Con `php artisan route:list` podremos ver las nuevas rutas:

Método	Ruta	Acción	Vista	Filtros
GET	login	LoginController@showLoginForm	login.blade	web,guest
POST	login	LoginController@login		web,guest
GET	logout	LoginController@logout		web
GET	register	RegisterController@showRegistrationForm	register.blade	web,guest
POST	register	RegisterController@register		web,guest
GET	password/reset	ForgotPasswordController@showLinkRequestForm	email.blade	web,guest
POST	password/email	ForgotPasswordController@sendResetLinkEmail		web,guest
GET	password/reset/{token}	ResetPasswordController@showResetForm	reset.blade	web,guest
POST	password/reset	ResetPasswordController@reset		web,guest
GET	home	HomeController@index		web, auth

Vistas

- Al ejecutar `php artisan make:auth` se **generan** también las 4 vistas necesarias para: login, registro y recuperar la contraseña
- Estas vistas las podemos encontrar en `resources/views/auth`
- Es importante que **no cambiemos** ni el nombre ni la ruta de las vistas pues los controladores ya están preparados para acceder con esos datos
- Sin embargo sí que podemos **modificar** el contenido y apariencia de las vistas, con la única precaución de respetar la URL a la que se envía el formulario y los nombres de los inputs
- Las vistas heredan del **layout** `layouts/app.blade.php`, el cual lo podemos modificar o cambiar por otro

Login

- Si accedemos a la ruta `login` nos aparecerá el formulario de login para iniciar sesión mediante nuestro email y contraseña
- En caso de que el login se realice **correctamente**:
 - Por defecto se redirigirá a la ruta `/home`
 - Para cambiar la ruta tenemos que modificar el controlador `LoginController` y establecer la propiedad:

```
protected $redirectTo = '/dashboard';
```

- Además podemos definir esta propiedad en `RegisterController` y `ResetPasswordController` para cambiar la URL de redirección después del registro y después de restablecer la contraseña, respectivamente

Registro

- Si accedemos a la ruta `register` nos aparecerá el formulario de registro para crear nuevos usuarios
- Si además de los campos nombre, email y contraseña queremos almacenar otros tenemos que modificar:
 - La migración de la tabla de usuarios con las modificaciones
 - Las siguientes funciones de `RegisterController`:
 - `validator`: realiza la validación de los datos
 - `create`: se encarga de crear el nuevo registro

Registro

- Ejemplo de método create:

```
protected function create(array $data) {  
    return User::create([  
        'name' => $data['name'],  
        'email' => $data['email'],  
        'phone' => $data['phone'],           // Campo añadido  
        'password' => bcrypt($data['password']),  
    ]);  
}
```

Añadir un usuario manualmente

- Lo podemos crear usando Eloquent de forma normal
- La única precaución que tenemos que llevar es cifrar el password manualmente:

```
$password_cifrado = bcrypt( 'mi-super-password' );
```

- Por ejemplo, para recoger los datos de un formulario y crear un nuevo registro:

```
public function store(Request $request) {  
    $user = new User;  
    $user->name = $request->input('name');  
    $user->email = $request->input('email');  
    $user->password = bcrypt($request->input('password'));  
    $user->save();  
}
```

Acceder a los datos del usuario

- Una vez que el usuario está autenticado podemos acceder a los datos del mismo a través del método `Auth::user()`, por ejemplo:

```
$email = Auth::user()->email;
```

- El método `Auth::user()` devolverá `null` si el usuario no está autenticado
- **¡Importante!** Para utilizar la clase `Auth` tenemos que añadir el espacio de nombres:

```
use Illuminate\Support\Facades\Auth;
```

Comprobar usuario autenticado

- Para comprobar si el usuario actual se ha autenticado en la aplicación podemos utilizar el método `Auth::check()` de la forma:

```
if (Auth::check()) {  
    // El usuario está correctamente autenticado  
}
```

- Por ejemplo, en una vista con Blade podríamos hacer:

```
@if (Auth::check())  
    Usuario autenticado: {{ Auth::user()->name }}  
@endif
```

- También podéis usar el método `Auth::guest()` para comprobar si es un usuario invitado