Laravel Middleware

Introduction

Middleware provide a convenient mechanism for inspecting and filtering HTTP requests entering your application. For example, Laravel includes a middleware that verifies the user of your application is authenticated. If the user is authenticated, the middleware will allow the request to proceed further into the application.

A variety of middleware are included in Laravel, including middleware for authentication and CSRF protection; however, all user-defined middleware are typically located in your application's app/Http/Middleware directory.

Defining Middleware

To create a new middleware, use the make:middleware Artisan command:
php artisan make:middleware EnsureTokenIsValid

This command will place a new EnsureTokenIsValid class within your app/Http/Middleware directory. In this middleware, we will only allow access to the route if the supplied token input matches a specified value. Otherwise, we will redirect the users back to the /home URI.
if the given token does not match our secret token, the middleware will return an HTTP redirect to the client; otherwise, the request will be passed further into the application. To pass the request deeper into the application (allowing the middleware to "pass"), you should call the $next callback with the $request.

It's best to envision middleware as a series of "layers" HTTP requests must pass through before they hit your application. Each layer can examine the request and even reject it entirely.

Registering Middleware

Global Middleware

If you want a middleware to run during every HTTP request to your application, you may append it to the global middleware stack in your application's bootstrap/app.php file.

The $middleware object provided to the withMiddleware closure is an instance of Illuminate\Foundation\Configuration\Middleware and is responsible for managing the middleware assigned to your application's routes. The append method adds the middleware to the end of the list of global middleware. If you would like to add a middleware to the beginning of the list, you should use the prepend method.

Manually Managing Laravel's Default Global Middleware
If you would like to manage Laravel's global middleware stack manually, you may provide Laravel's default stack of global middleware to the use method.

Assigning Middleware to Routes

If you would like to assign middleware to specific routes, you may invoke the middleware method when defining the route.

Excluding Middleware

When assigning middleware to a group of routes, you may occasionally need to prevent the middleware from being applied to an individual route within the group. You may accomplish this using the withoutMiddleware method.
You may also exclude a given set of middleware from an entire group of route definitions.

Middleware Groups

Sometimes you may want to group several middleware under a single key to make them easier to assign to routes. You may accomplish this using the appendToGroup method within your application's bootstrap/app.php file.
Middleware groups may be assigned to routes and controller actions using the same syntax as individual middleware.

Laravel's Default Middleware Groups

Laravel includes predefined web and api middleware groups that contain common middleware you may want to apply to your web and API routes. Laravel automatically applies these middleware groups to the corresponding routes/web.php and routes/api.php files.

Manually Managing Laravel's Default Middleware Groups

If you would like to manually manage all of the middleware within Laravel's default web and api middleware groups, you may redefine the groups entirely.

By default, the web and api middleware groups are automatically applied to your application's corresponding routes/web.php and routes/api.php files by the bootstrap/app.php file.

Middleware Aliases

You may assign aliases to middleware in your application's bootstrap/app.php file. Middleware aliases allow you to define a short alias for a given middleware class, which can be especially useful for middleware with long class names:

```
use App\Http\Middleware\EnsureUserIsSubscribed;

->withMiddleware(function (Middleware $middleware) {
    $middleware->alias([
        'subscribed' => EnsureUserIsSubscribed::class
    ]);
})
```

Once the middleware alias has been defined in your application's bootstrap/app.php file, you may use the alias when assigning the middleware to routes:

```
Route::get('/profile', function () {
    // ...
})->middleware('subscribed');
```

For convenience, some of Laravel's built-in middleware are aliased by default. For example, the auth middleware is an alias for the Illuminate\Auth\Middleware\Authenticate middleware.

Sorting Middleware

Rarely, you may need your middleware to execute in a specific order but not have control over their order when they are assigned to the route. In these situations, you may specify your middleware priority using the priority method in your application's bootstrap/app.php file.

Middleware Parameters

- If your application needs to verify that the authenticated user has a given "role" before performing a given action, you could create an EnsureUserHasRole middleware that receives a role name as an additional argument.
- Additional middleware parameters will be passed to the middleware after the $next argument.
- Middleware parameters may be specified when defining the route by separating the middleware name and parameters with a ::
- Multiple parameters may be delimited by commas.

Terminable Middleware

Sometimes a middleware may need to do some work after the HTTP response has been sent to the browser. If you define a terminate method on your middleware and your web server is using FastCGI, the terminate method will automatically be called after the response is sent to the browser.

The terminate method should receive both the request and the response. Once you have defined a terminable middleware, you should add it to the list of routes or global middleware in your application's bootstrap/app.php file.

When calling the terminate method on your middleware, Laravel will resolve a fresh instance of the middleware from the service container. If you would like to use the same middleware instance when the handle and terminate methods are called, register the middleware with the container using the container's singleton method.

Custom Middleware in Laravel 11

1. To create middleware use the following command:

php artisan make:middleware IsAdmin

app/Http/Middleware/IsAdmin.php

```
<?php
namespace App\Http\Middleware;
use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class IsAdmin
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request):
(\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle(Request $request, Closure $next): Response
    {
        if (\Auth::user()->role_id != 1) {
            return response()->json('Opps! You do not have permission to access.');
        }
return $next($request);
    }
}
```

2. Register Middleware:

In this step, we will register our custom middleware in the app.php file, as illustrated in the code snippet below:

bootstrap/app.php

```
<?php
use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware) {
        $middleware->alias([
            'isAdmin' => \App\Http\Middleware\IsAdmin::class,
        ]);
    })
    ->withExceptions(function (Exceptions $exceptions) {
        //
    })->create();
```

3. Apply Middleware:

We'll create two routes and apply the "isAdmin" middleware. Let's proceed by updating the code accordingly.

routes/web.php

```
<?php
use Illuminate\Support\Facades\Route;
Route::middleware(['isAdmin'])->group(function () {
    Route::get('/dashboard', function () {
        return 'Dashboard';
    });
    Route::get('/users', function () {
        return 'Users';
    });
});
```