

REQUEST MIDDLEWARE

■ Understanding over the middleware structure

●●● CheckRequestDetails.php

```
1  <?php
2  namespace App\Http\Middleware;
3  use Closure;
4  use Illuminate\Http\Request;
5  use Symfony\Component\HttpFoundation\Response;
6  class CheckRequestDetails
7  {
8      /**
9       * Handle an incoming request.
10      *
11      * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
12      */
13     public function handle(Request $request, Closure $next): Response
14     {
15         return $next($request);
16     }
17 }
18
```

REQUEST MIDDLEWARE

■ Check Request Header Inside Middleware

web.php

```
2 use App\Http\Controllers\DemoController;
3 use App\Http\Middleware\CheckRequestDetails;
4 use Illuminate\Support\Facades\Route;
5 Route::post('/DemoAction', [DemoController::class, 'DemoAction'])->middleware([CheckRequestDetails::class]);
```

CheckRequestDetails.php

```
13 public function handle(Request $request, Closure $next): Response
14 {
15     $key= $request->header('key');
16     if($key=="123"){
17         return $next($request);
18     }
19     else{
20         return response()->json("unauthorized",401);
21     }
22 }
```

REQUEST MIDDLEWARE

Redirect Request From Middleware

```
●●● CheckRequestDetails.php

13     public function handle(Request $request, Closure $next): Response
14     {
15         if($request->header('key')== "123"){
16             return $next($request);
17         }
18         else{
19             return redirect('/');
20         }
21     }
```

REQUEST MIDDLEWARE

Apply For Specific Route

```
Route::get('/DemoAction', [DemoController::class, 'DemoAction'])->middleware([CheckRequestDetails::class]);
```

Apply For Route Group

web.php

```
6 Route::middleware(['CheckRequestDetails'])->group(function () {
7     Route::get('/DemoAction1', [DemoController::class, 'DemoAction']);
8     Route::get('/DemoAction2', [DemoController::class, 'DemoAction']);
9     Route::get('/DemoAction3', [DemoController::class, 'DemoAction']);
10 });
11 Route::get('/DemoAction4', [DemoController::class, 'DemoAction']);
```

Kernel.php

```
55 protected $middlewareAliases = [
56     'CheckRequestDetails' => \App\Http\Middleware\CheckRequestDetails::class,
```


REQUEST MIDDLEWARE

Apply For Whole Application

```
Kernel.php

31     protected $middlewareGroups = [
32         'web' => [
33             \App\Http\Middleware\CheckRequestDetails::class,
34             \App\Http\Middleware\EncryptCookies::class,
35             \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
36             \Illuminate\Session\Middleware\StartSession::class,
37             \Illuminate\View\Middleware\ShareErrorsFromSession::class,
38             // \App\Http\Middleware\VerifyCsrfToken::class,
39             \Illuminate\Routing\Middleware\SubstituteBindings::class,
40         ],
41
42         'api' => [
43             \App\Http\Middleware\CheckRequestDetails::class,
44             // \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,
45             \Illuminate\Routing\Middleware\ThrottleRequests::class.':api',
46             \Illuminate\Routing\Middleware\SubstituteBindings::class,
47         ],
48     ];
```

REQUEST MIDDLEWARE

- Manipulate (ADD) Request Details Inside Middleware

```
$request->headers->set('key3', 'variable3');
```

- Manipulate (Remove) Request Details Inside Middleware

```
$request->request->remove('name');
```

- Manipulate (Replace) Request Details Inside Middleware

```
$request->headers->replace(['name'=>'XXX', 'age'=>100]);
```

REQUEST RATE LIMITING

Inside Kernel file middlewaregroup

Kernel.php

```
31     protected $middlewareGroups = [  
32         'web' => [  
33             'throttle:60,1',  
34             \App\Http\Middleware\EncryptCookies::class,  
35             \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,  
36             \Illuminate\Session\Middleware\StartSession::class,  
37             \Illuminate\View\Middleware\ShareErrorsFromSession::class,  
38             // \App\Http\Middleware\VerifyCsrfToken::class,  
39             \Illuminate\Routing\Middleware\SubstituteBindings::class,  
40         ],  
41         'api' => [  
42             'throttle:60,1',  
43             // \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,  
44             \Illuminate\Routing\Middleware\ThrottleRequests::class.':api',  
45             \Illuminate\Routing\Middleware\SubstituteBindings::class,  
46         ],  
47     ];
```

REQUEST RATE LIMITING

■ For Specific Routing End Point

● ● ● web.php

```
1 <?php
2 use App\Http\Controllers\DemoController;
3 use Illuminate\Support\Facades\Route;
4 Route::post('/DemoAction', [DemoController::class, 'DemoAction'])->middleware('throttle:5,1');
5
```


BASIC CONTROLLERS

- To quickly generate a new controller, you may run the `make:controller` Artisan command.
- A controller may have any number of public methods which will respond to incoming HTTP requests.

● ● ● DemoController.php

```
9      function DemoAction():string
10     {
11         return "Basic Controller";
12     }
```

● ● ● web.php

```
17  Route::get('/DemoAction', [DemoController::class, 'DemoAction']);
```

SINGLE ACTION CONTROLLERS

- Dedicate an entire controller class for single action.
- To accomplish this, you may define a single `__invoke` method within the controller
- `php artisan make:controller SingleActionControllers --invokable`

● ● ● SingleActionControllers.php

```
12     public function __invoke(Request $request)
13     {
14         return "I am Single Action Controllers";
15     }
```

```
Route::get(uri: "/SingleAction", action: SingleActionControllers::class);
```

RESOURCE CONTROLLERS

- Laravel resource routing assigns the typical create, read, update, and delete ("CRUD")
- `php artisan make:controller PhotoController --resource`

PhotoController.php

```

12     public function index()
13     {
14         return "index";
15     }
16     public function create()
17     {
18         return "create";
19     }
20     public function store(Request $request)
21     {
22         return "store";
23     }
24     public function show(string $id)
25     {
26         return "show";
27     }
28     public function edit(string $id)
29     {
30         return "edit";
31     }
32     public function update(Request $request, string $id)
33     {
34         return "update";
35     }
36     public function destroy(string $id)
37     {
38         return "destroy";
39     }

```

web.php

```

10 Route::resource('photos', PhotoController::class);
11 /*
12 GET()      INDEX    http://127.0.0.1:8000/photos
13 GET()      CREATE   http://127.0.0.1:8000/photos/create
14 POST()     STORE    http://127.0.0.1:8000/photos
15 GET()      SHOW     http://127.0.0.1:8000/photos/{photo}
16 GET()      EDIT     http://127.0.0.1:8000/photos/{photo}/edit
17 PUT/PATCH() UPDATE   http://127.0.0.1:8000/photos/{photo}
18 DELETE()   DESTROY  http://127.0.0.1:8000/photos/{photo}
19 */

```

CONTROLLER MIDDLEWARE

Using the middleware method within your controller's constructor, you can assign middleware to the controller's actions

PhotoController.php

```
9      public function __construct()
10     {
11         $this->middleware('auth');
12         $this->middleware('log')->only('index');
13         $this->middleware('subscribed')->except('store');
14     }
```


Laravel Blade

- Laravel Blade is a templating engine that comes built-in with the Laravel PHP framework. It allows developers to write clean and structured HTML templates with dynamic content and reusable components.
- Blade templates use a combination of plain HTML and special Blade syntax, such as double curly braces `{{ }}` for variable output and at signs `@` for control structures like loops and conditionals.
- Blade templates can be extended to create a base template that can be reused across multiple pages with different content.
- Blade also provides several built-in directives that simplify common tasks like including subviews, injecting content into sections, and rendering JSON data.



PASS AND DISPLAYING DATA

- Display data that is passed to your Blade views by wrapping the variable in curly braces
- Blade's {{ }} echo statements are automatically sent through PHP's htmlspecialchars function to prevent XSS attacks.

```
● ● ● DemoController.php

7      function DemoAction(Request $request)
8      {
9          $sum=$request->num1+$request->num2;
10         return view('Home', ['sum' => $sum]);
11     }
```

```
● ● ● Home.blade.php

8      <body>
9          <h1>Sum is={{ $sum }}</h1>
10     </body>
```

BLADE IF STATEMENT

You may construct if statements using the @if, @elseif, @else, and @endif directives.

● ● ● Home.blade.php

```
8    <body>
9    @if ($sum === 1)
10        <h6>I have 1 record!</h6>
11    @elseif ($sum === 2)
12        <h6>I have 2 record!</h6>
13    @else
14        <h6>I have many record!</h6>
15    @endif
16 </body>
```

BLADE SWITCH CASE

Switch statements can be constructed using the @switch, @case, @break, @default and @endswitch directives

```
● ● ● Home.blade.php

8   <body>
9   @switch($sum)
10      @case(1)
11          <h3>First case...</h3>
12          @break
13      @case(2)
14          <h3>Second case...</h3>
15          @break
16      @default
17          Default case...
18  @endswitch
19  </body>
```


BLADE FOR LOOP

● ● ● Home.blade.php

```
8   <body>
9   @for ($i = 0; $i < 10; $i++)
10      The current value is {{ $i }}
11   @endfor
12 </body>
```

BLADE FOREACH LOOP

● ● ● DemoController.php

```
7     function DemoAction(Request $request)
8     {
9         $users=[
10             ["fname" => "Zoe", "lname" => "Upfield"],
11             ["fname" => "Bella", "lname" => "Cooke"],
12             ["fname" => "Liam", "lname" => "Ridley"]
13         ];
14         return view('Home', ['users' =>$users]);
15     }
```

● ● ● Home.blade.php

```
8     <body>
9     @foreach ($users as $user)
10         <p>This is user {{ $user['fname'] }}</p>
11     @endforeach
12 </body>
```

INCLUDING ASSET

●●● Home.blade.php

```
1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <meta charset="utf-8">
5          <meta name="viewport" content="width=device-width, initial-scale=1">
6          <title>Laravel</title>
7          <link href="{{asset('/css/bootstrap.min.css')}}" rel="stylesheet">
8      </head>
9      <body>
10     
11 </body>
12 </html>
```

INCLUDING SUBVIEWS

Blade's `@include` directive allows you to include a Blade view from within another view.
All variables that are available to the parent view will be made available to the included view:

```
● ● ● Home.blade.php

10  <div>
11      @include('shared.errors')
12
13      <form>
14          <!-- Form Contents -->
15      </form>
16  </div>
```


MASTER LAYOUT CONCEPT

Layout.blade.php

```
5 <body>
6     @include('component.menu')
7     @yield('content')
8     @include('component.footer')
9 </body>
```

Home.blade.php

```
1 @extends('Layout')
2
3 @section('content')
4
5     @include('Component.HomeBanner')
6     @include('Component.HomeService')
7     @include('Component.HomeCourse')
8     @include('Component.HomeProjects')
9     @include('Component.HomeContact')
10    @include('Component.HomeReview')
11
12 @endsection
```