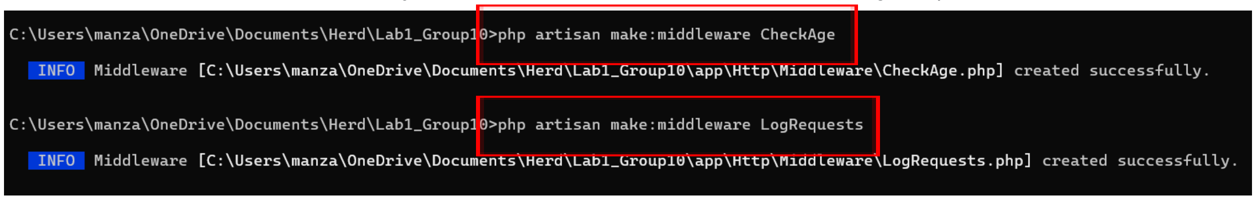
Jethro J. Manzanillo
BSIT - 3C
Web Development 2

Part 1: **Create and Register New Middleware:**

- **Create Middleware**

First, we used the command line to generate two new middleware classes that we named **`CheckAge`** and **`LogRequests`.** We ran the following commands:

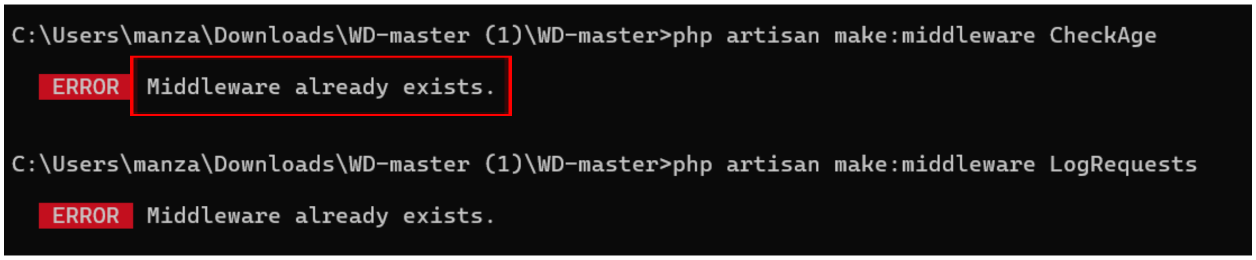
(Note: The path is only an example for command execution. Our backend developer, Janine Ishe Matamorosa, has already created the middleware in the project.)



```
C:\Users\manza\OneDrive\Documents\Herd\Lab1_Group10>php artisan make:middleware CheckAge

  INFO  Middleware [C:\Users\manza\OneDrive\Documents\Herd\Lab1_Group10\app\Http\Middleware\CheckAge.php] created successfully.

C:\Users\manza\OneDrive\Documents\Herd\Lab1_Group10>php artisan make:middleware LogRequests

  INFO  Middleware [C:\Users\manza\OneDrive\Documents\Herd\Lab1_Group10\app\Http\Middleware\LogRequests.php] created successfully.
```

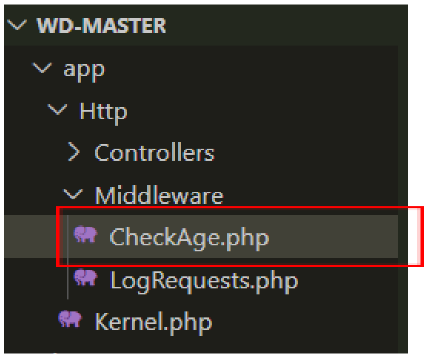
This message should appear. This created the necessary files for our middleware in the **`app/Http/Middleware`** directory.

If the message "middleware already exists" appears, it means that the middleware file has already been created.

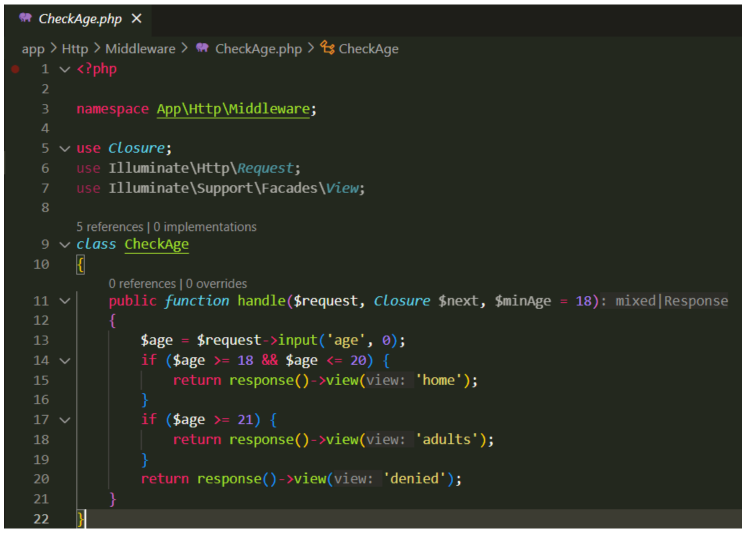


```
C:\Users\manza\Downloads\WD-master (1)\WD-master>php artisan make:middleware CheckAge

  ERROR  Middleware already exists.

C:\Users\manza\Downloads\WD-master (1)\WD-master>php artisan make:middleware LogRequests

  ERROR  Middleware already exists.
```

- **CheckAge Middleware Logic**



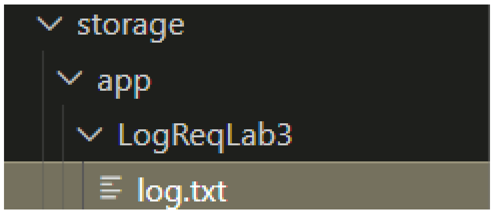
Next, we focused on the **`CheckAge`** middleware. In **`app/Http/Middleware/CheckAge.php`**.



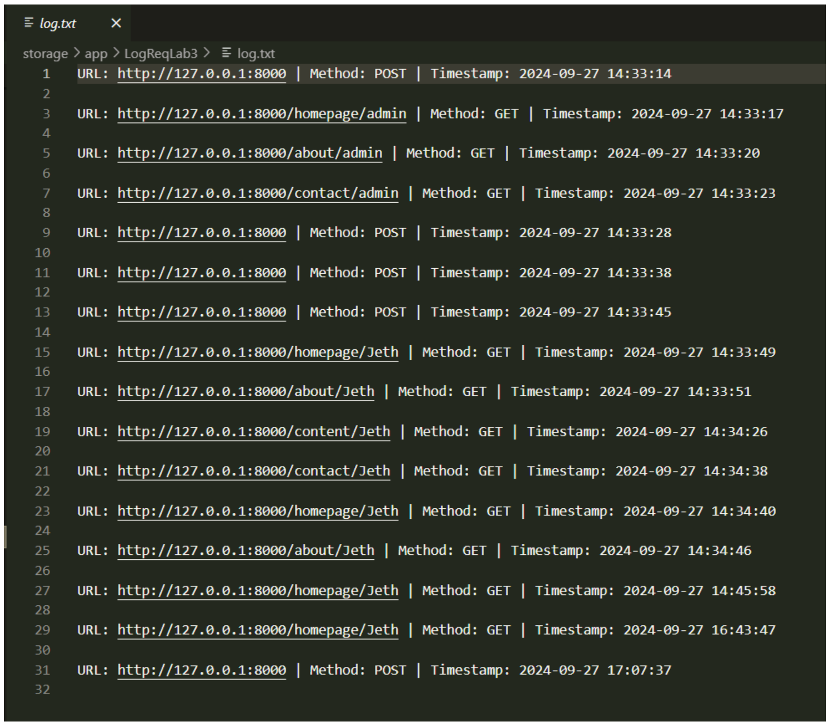
```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\View;

5 references | 0 implementations
class CheckAge
{
    0 references | 0 overrides
    public function handle($request, Closure $next, $minAge = 18): mixed|Response
    {
        $age = $request->input('age', 0);
        if ($age >= 18 && $age <= 20) {
            return response()->view(view: 'home');
        }
        if ($age >= 21) {
            return response()->view(view: 'adults');
        }
        return response()->view(view: 'denied');
    }
}
```

We implemented logic to determine the appropriate response based on the user's age.

- **LogRequests Middleware Logic**



For the **`LogRequests`** middleware, our goal was to log every incoming HTTP request. In **`app/Http/Middleware/LogRequests.php`**, located in the **`LogReqLab3`** directory,

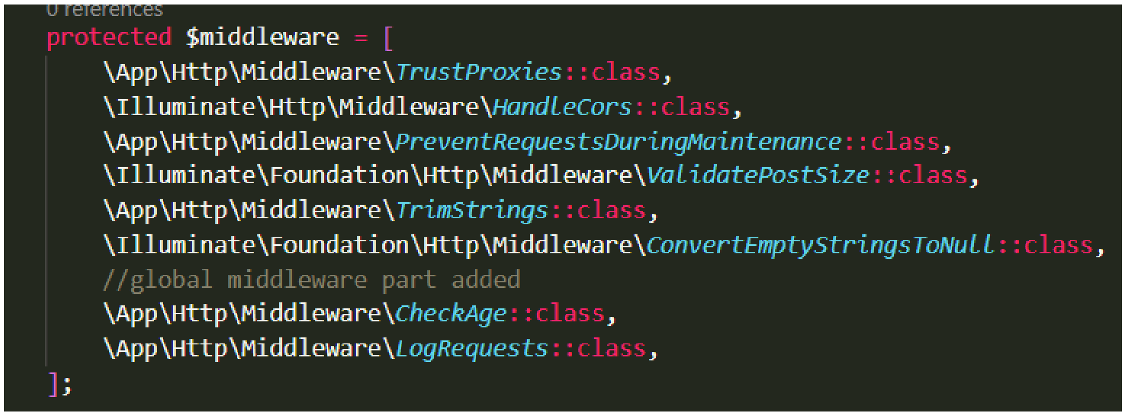


```
URL: http://127.0.0.1:8000 | Method: POST | Timestamp: 2024-09-27 14:33:14

URL: http://127.0.0.1:8000/homepage/admin | Method: GET | Timestamp: 2024-09-27 14:33:17

URL: http://127.0.0.1:8000/about/admin | Method: GET | Timestamp: 2024-09-27 14:33:20

URL: http://127.0.0.1:8000/contact/admin | Method: GET | Timestamp: 2024-09-27 14:33:23

URL: http://127.0.0.1:8000 | Method: POST | Timestamp: 2024-09-27 14:33:28

URL: http://127.0.0.1:8000 | Method: POST | Timestamp: 2024-09-27 14:33:38

URL: http://127.0.0.1:8000 | Method: POST | Timestamp: 2024-09-27 14:33:45

URL: http://127.0.0.1:8000/homepage/Jeth | Method: GET | Timestamp: 2024-09-27 14:33:49

URL: http://127.0.0.1:8000/about/Jeth | Method: GET | Timestamp: 2024-09-27 14:33:51

URL: http://127.0.0.1:8000/content/Jeth | Method: GET | Timestamp: 2024-09-27 14:34:26

URL: http://127.0.0.1:8000/contact/Jeth | Method: GET | Timestamp: 2024-09-27 14:34:38

URL: http://127.0.0.1:8000/homepage/Jeth | Method: GET | Timestamp: 2024-09-27 14:34:40

URL: http://127.0.0.1:8000/about/Jeth | Method: GET | Timestamp: 2024-09-27 14:34:46

URL: http://127.0.0.1:8000/homepage/Jeth | Method: GET | Timestamp: 2024-09-27 14:45:58

URL: http://127.0.0.1:8000/homepage/Jeth | Method: GET | Timestamp: 2024-09-27 16:43:47

URL: http://127.0.0.1:8000 | Method: POST | Timestamp: 2024-09-27 17:07:37

```

We implemented logic to capture the URL, HTTP method, and timestamp of each request. The log data is then saved to a file named **`log.txt`**, allowing us to keep a record of all requests for monitoring and analysis.

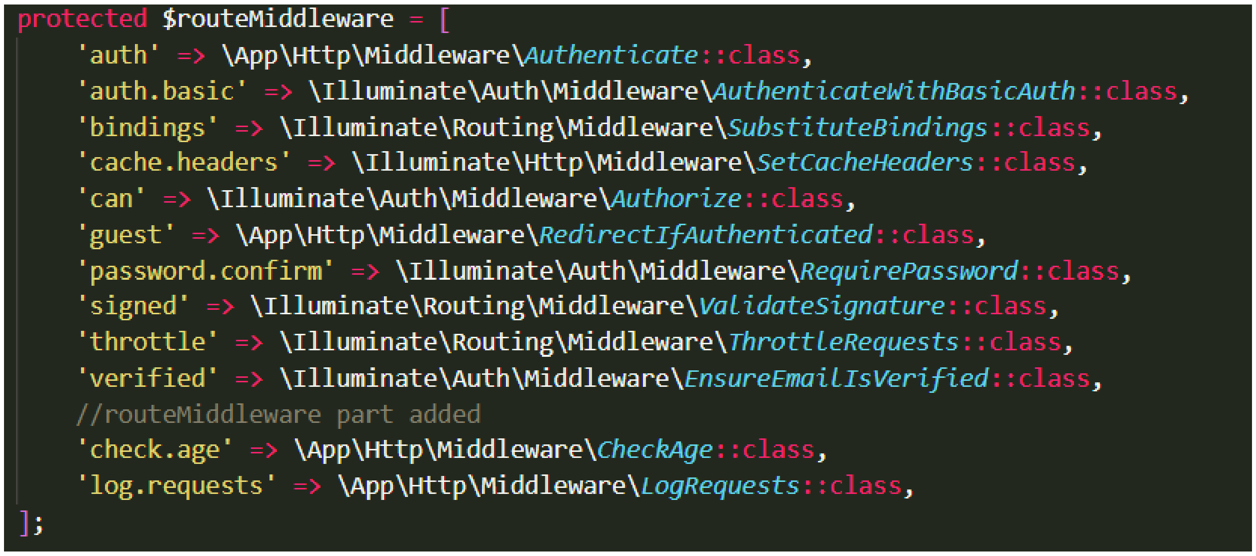
- **Register Middleware in Kernel**

Next, we registered our middleware in the `app/Http/Kernel.php` file. For global middleware, we added `CheckAge` and `LogRequests` to the `$middleware` array to ensure they are applied to every incoming request. Our updated `$middleware` array looks like this:



```
0 references
protected $middleware = [
    \App\Http\Middleware\TrustProxies::class,
    \Illuminate\Http\Middleware\HandleCors::class,
    \App\Http\Middleware\PreventRequestsDuringMaintenance::class,
    \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
    \App\Http\Middleware\TrimStrings::class,
    \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
    //global middleware part added
    \App\Http\Middleware\CheckAge::class,
    \App\Http\Middleware\LogRequests::class,
];
```

Additionally, we registered the middleware in the **`$routeMiddleware`** array to allow for route-specific assignments. This means that we can use **`CheckAge`** and **`LogRequests`** as middleware for individual routes as needed:



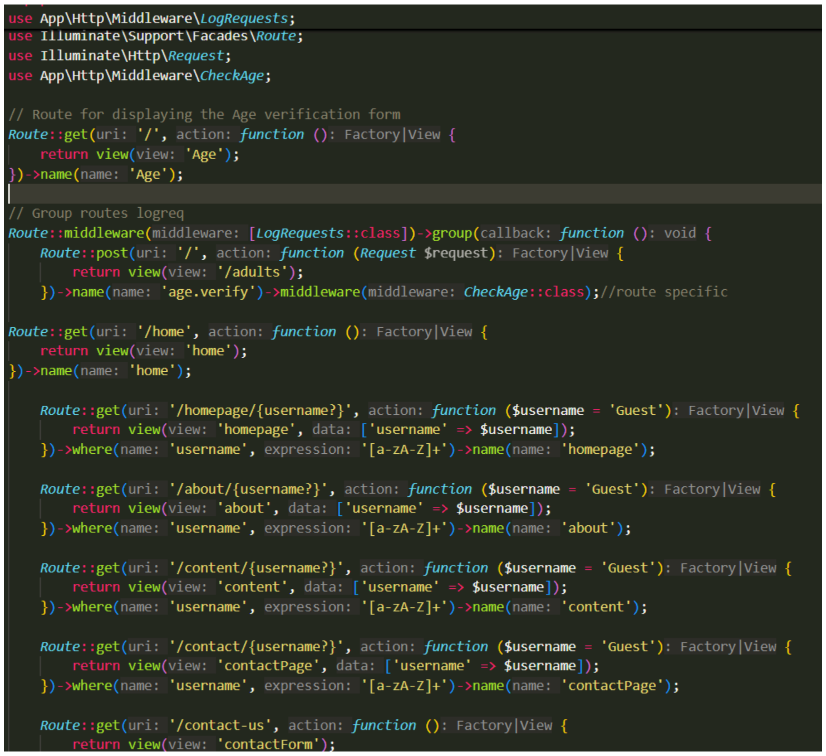
```
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    //routeMiddleware part added
    'check.age' => \App\Http\Middleware\CheckAge::class,
    'log.requests' => \App\Http\Middleware\LogRequests::class,
];
```

This setup ensures that our **`CheckAge`** and **`LogRequests`** middleware are effectively integrated into the application, allowing us to manage user access based on age and log requests seamlessly.

Part 2: **Assign Middleware to Routes:**

In this part, we created a route group that assigns the **`CheckAge`** middleware to specific routes to ensure that only users who meet the age requirement could access them.

- Creating a Route Group with **`CheckAge`** Middleware

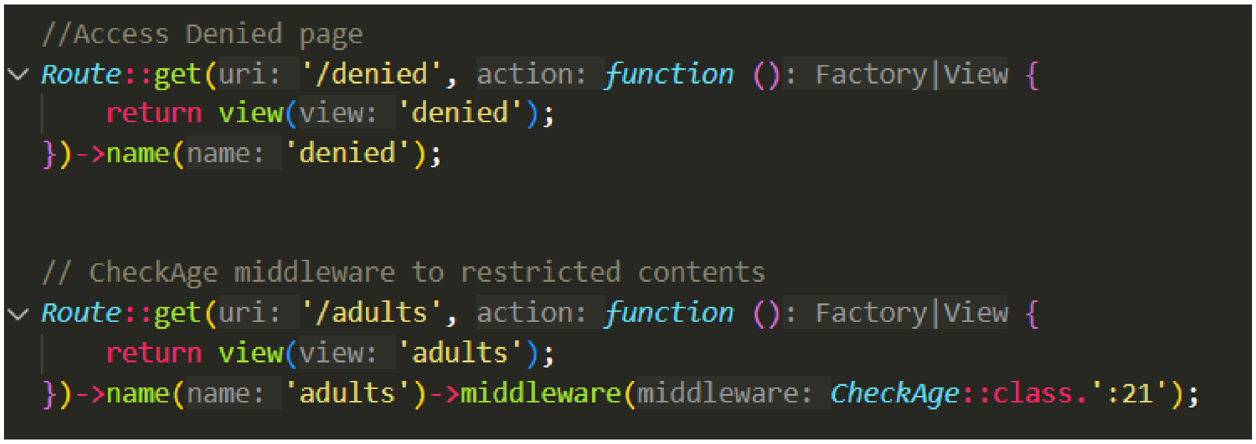


```php
use App\Http\Middleware\LogRequests;
use Illuminate\Support\Facades\Route;
use Illuminate\Http\Request;
use App\Http\Middleware\CheckAge;

// Route for displaying the Age verification form
Route::get(uri: '/', action: function (): Factory|View {
    return view(view: 'Age');
})->name(name: 'Age');

// Group routes logreq
Route::middleware(middleware: [LogRequests::class])->group(callback: function (): void {
    Route::post(uri: '/', action: function (Request $request): Factory|View {
        return view(view: '/adults');
    })->name(name: 'age.verify')->middleware(middleware: CheckAge::class);//route specific

Route::get(uri: '/home', action: function (): Factory|View {
    return view(view: 'home');
})->name(name: 'home');

    Route::get(uri: '/homepage/{username?}', action: function ($username = 'Guest'): Factory|View {
        return view(view: 'homepage', data: ['username' => $username]);
    })->where(name: 'username', expression: '[a-zA-Z]+')->name(name: 'homepage');

    Route::get(uri: '/about/{username?}', action: function ($username = 'Guest'): Factory|View {
        return view(view: 'about', data: ['username' => $username]);
    })->where(name: 'username', expression: '[a-zA-Z]+')->name(name: 'about');

    Route::get(uri: '/content/{username?}', action: function ($username = 'Guest'): Factory|View {
        return view(view: 'content', data: ['username' => $username]);
    })->where(name: 'username', expression: '[a-zA-Z]+')->name(name: 'content');

    Route::get(uri: '/contact/{username?}', action: function ($username = 'Guest'): Factory|View {
        return view(view: 'contactPage', data: ['username' => $username]);
    })->where(name: 'username', expression: '[a-zA-Z]+')->name(name: 'contactPage');

    Route::get(uri: '/contact-us', action: function (): Factory|View {
        return view(view: 'contactForm');
```

We started by defining a route for displaying the age verification form and then grouping routes that utilize the **`LogRequests`** middleware while also applying **`CheckAge`** to the necessary routes.

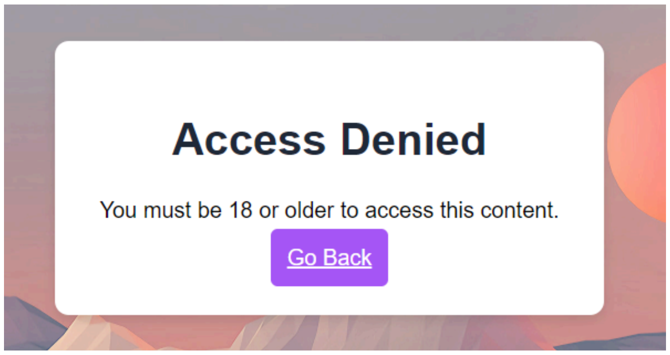
- **Testing the Middleware**

To ensure our middleware functions correctly, we will simulate different age values when users submit the form.

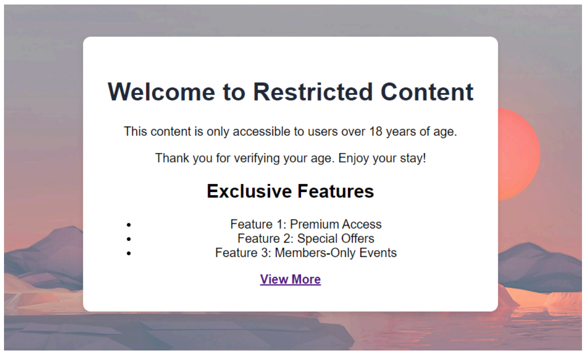


```php
//Access Denied page
Route::get(uri: '/denied', action: function (): Factory|View {
    return view(view: 'denied');
})->name(name: 'denied');


// CheckAge middleware to restricted contents
Route::get(uri: '/adults', action: function (): Factory|View {
    return view(view: 'adults');
})->name(name: 'adults')->middleware(middleware: CheckAge::class.':21');
```

Which are these views:


Access Denied

You must be 18 or older to access this content.


Welcome to Restricted Content

This content is only accessible to users over 18 years of age.

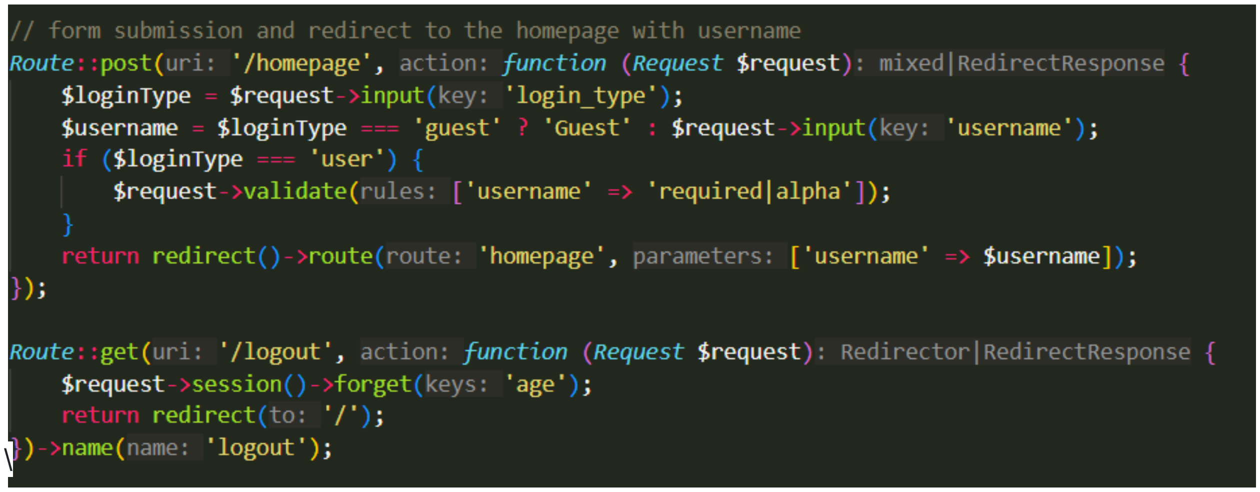
Thank you for verifying your age. Enjoy your stay!

Exclusive Features

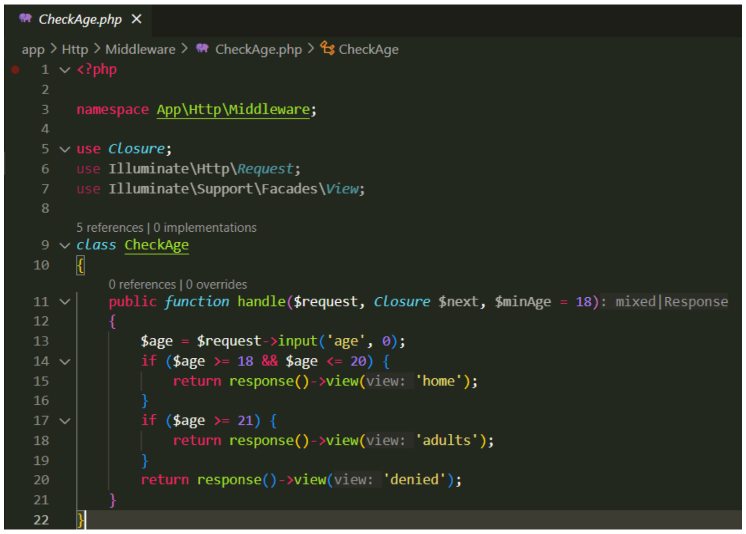
- Feature 1: Premium Access
- Feature 2: Special Offers
- Feature 3: Members-Only Events

- **Handling User Input and Redirects**

We also set up a form submission route that validates user input and redirects to the appropriate page based on the age.


```php
// form submission and redirect to the homepage with username
Route::post(uri: '/homepage', action: function (Request $request): mixed|RedirectResponse {
    $loginType = $request->input(key: 'login_type');
    $username = $loginType === 'guest' ? 'Guest' : $request->input(key: 'username');
    if ($loginType === 'user') {
        $request->validate(rules: ['username' => 'required|alpha']);
    }
    return redirect()->route(route: 'homepage', parameters: ['username' => $username]);
});

Route::get(uri: '/logout', action: function (Request $request): Redirector|RedirectResponse {
    $request->session()->forget(keys: 'age');
    return redirect(to: '/');
})->name(name: 'logout');
```

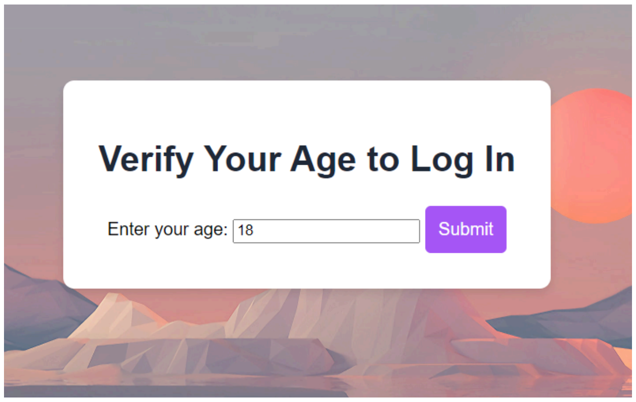
Part 3: **Create Middleware with Parameters**



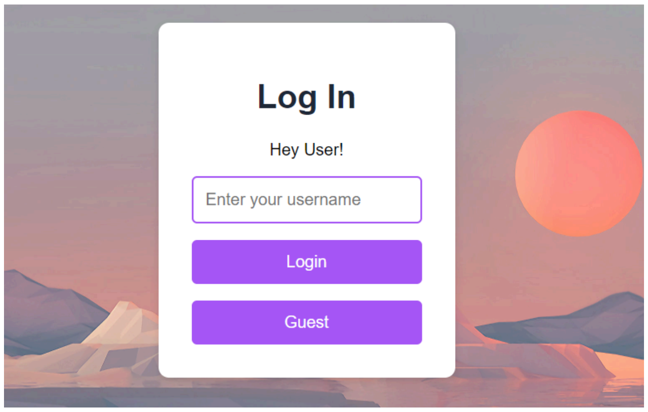
```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\View;

5 references | 0 implementations
class CheckAge
{
    0 references | 0 overrides
    public function handle($request, Closure $next, $minAge = 18): mixed|Response
    {
        $age = $request->input('age', 0);
        if ($age >= 18 && $age <= 20) {
            return response()->view(view: 'home');
        }
        if ($age >= 21) {
            return response()->view(view: 'adults');
        }
        return response()->view(view: 'denied');
    }
}
```

We implemented logic to determine the appropriate response based on the user’s age.

If a user's age is between 18 and 20, they are redirected to the "home" view.


Verify Your Age to Log In

Enter your age: 18 Submit


Log In

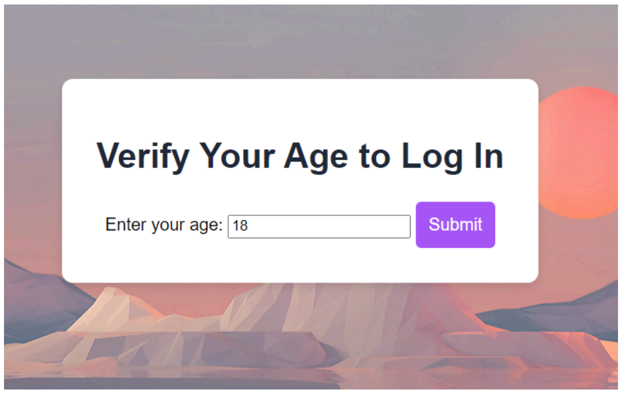
Hey User!

Enter your username

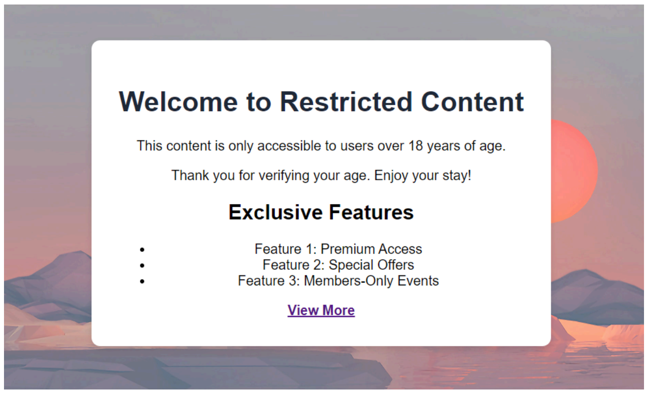
Login

Guest

If they are 21 or older, they are redirected to the "adults" view.


Verify Your Age to Log In

Enter your age: 18 Submit


Welcome to Restricted Content

This content is only accessible to users over 18 years of age.

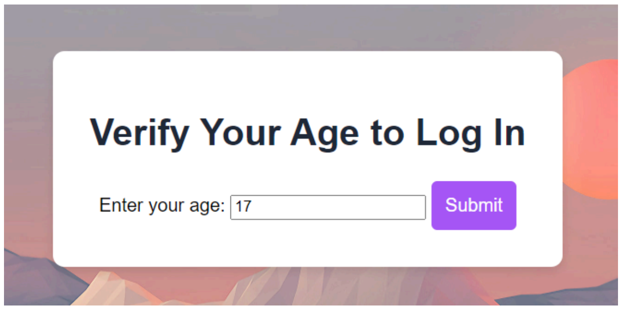
Thank you for verifying your age. Enjoy your stay!

Exclusive Features

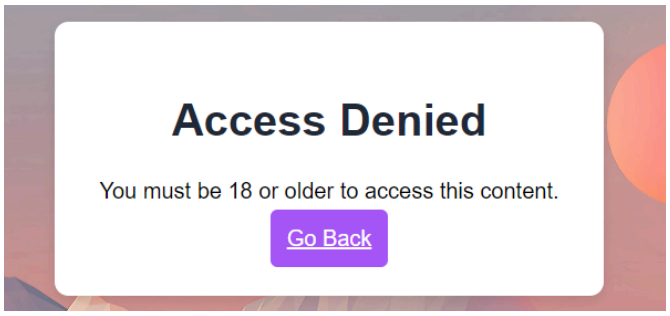
- Feature 1: Premium Access
- Feature 2: Special Offers
- Feature 3: Members-Only Events

View More

However, if a user is under 18, they are sent to the "denied" view instead.


Verify Your Age to Log In

Enter your age: 17 Submit

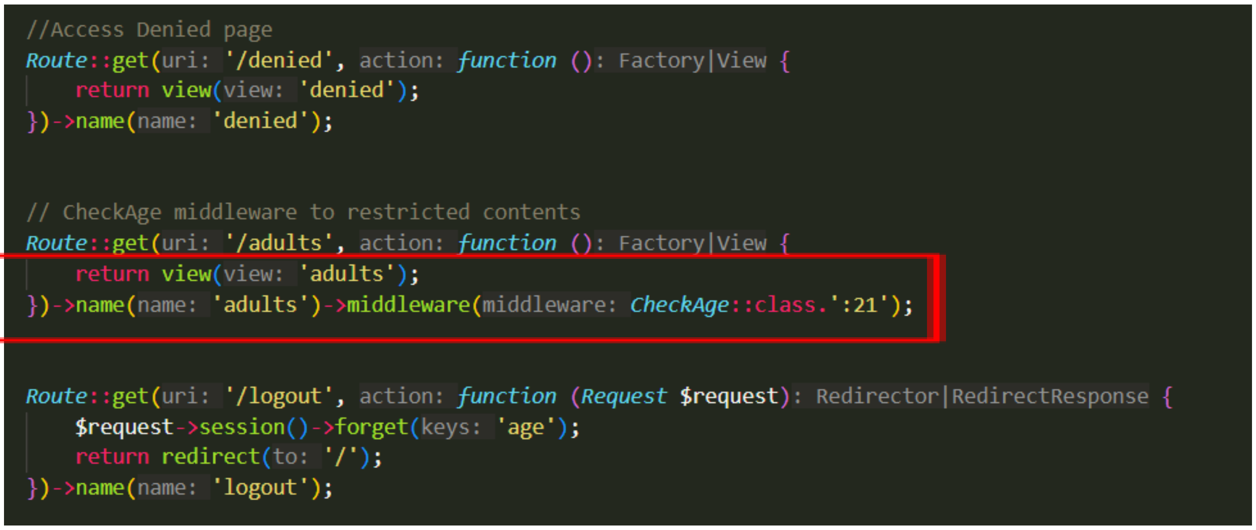

Access Denied

You must be 18 or older to access this content.

Go Back

3.2 Create a Route with a Different Age Restriction

Next, we created a new route that enforces a stricter age restriction (21 years old). We applied the modified CheckAge middleware to this route with the age parameter set to 21.



```
//Access Denied page
Route::get(uri: '/denied', action: function (): Factory|View {
    return view(view: 'denied');
})->name(name: 'denied');


// CheckAge middleware to restricted contents
Route::get(uri: '/adults', action: function (): Factory|View {
    return view(view: 'adults');
})->name(name: 'adults')->middleware(middleware: CheckAge::class.':21');


Route::get(uri: '/logout', action: function (Request $request): Redirector|RedirectResponse {
    $request->session()->forget(keys: 'age');
    return redirect(to: '/');
})->name(name: 'logout');
```

Part 4: **Explanation**

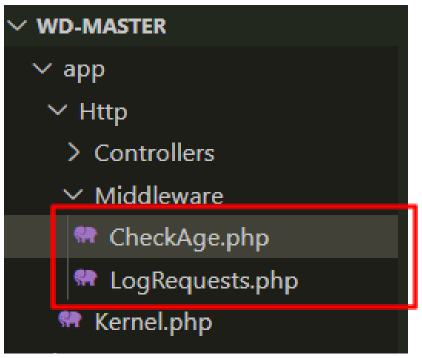
- **What is Global Middleware?**

Global middleware runs during every HTTP request, regardless of the specific route. This is useful for tasks that should be applied universally across the application, such as logging requests, checking authentication, or setting default security policies.

- **How We Registered Global Middleware**

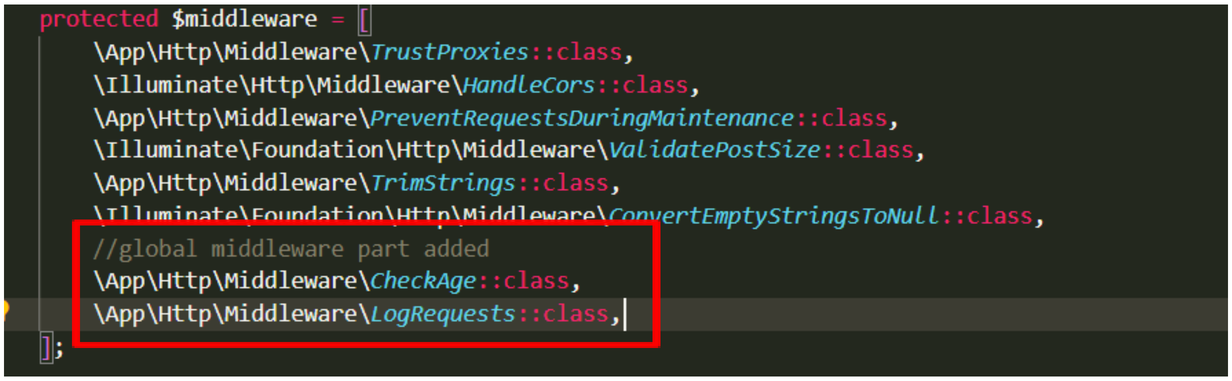
In our project, we created two middleware classes:

1. **`CheckAge`** - This middleware is used to restrict access based on a user's age.
2. **`LogRequests`** - This middleware logs every HTTP request made to the application.



These middleware were registered globally in the **`app/Http/Kernel.php`** file. The **`Kernel`** class is responsible for maintaining a list of middleware for the entire application.

Here's how the middleware was registered:

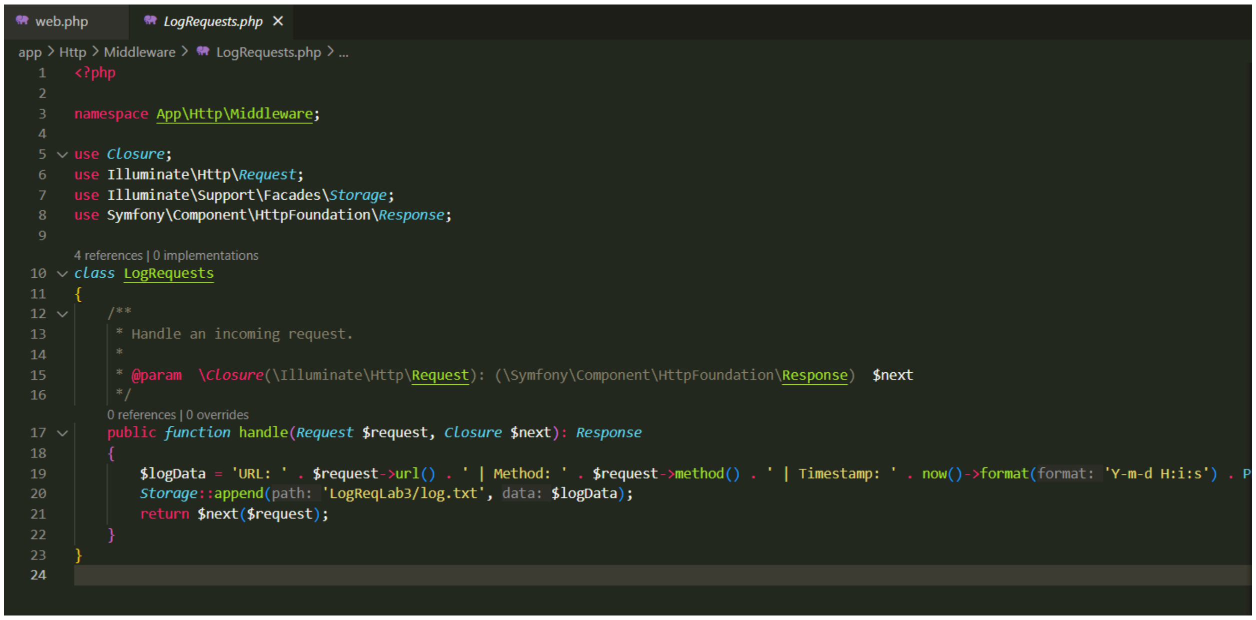


```php
protected $middleware = [
    \App\Http\Middleware\TrustProxies::class,
    \Illuminate\Http\Middleware\HandleCors::class,
    \App\Http\Middleware\PreventRequestsDuringMaintenance::class,
    \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
    \App\Http\Middleware\TrimStrings::class,
    \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
    //global middleware part added
    \App\Http\Middleware\CheckAge::class,
    \App\Http\Middleware\LogRequests::class,
];
```

With this configuration, both `CheckAge` and `LogRequests` middleware are applied to every incoming request, unless specifically excluded in route definitions.

- **Why Global Middleware?**

Global middleware is particularly useful when the functionality it provides (e.g., logging requests, age restrictions) should be enforced across all routes. For instance, the `LogRequests` middleware ensures that all requests are logged, regardless of the specific route, providing a consistent log for debugging or analytics purposes.

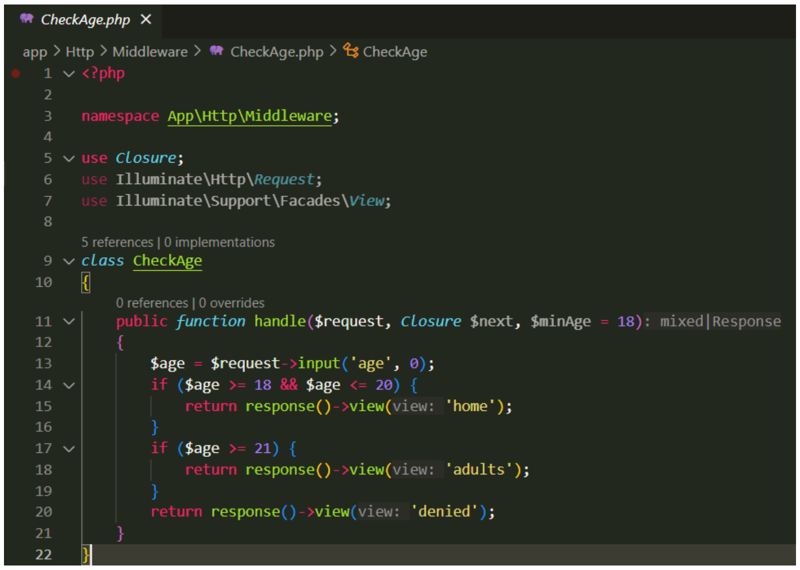


```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Storage;
use Symfony\Component\HttpFoundation\Response;

4 references | 0 implementations
class LogRequests
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
     */
    0 references | 0 overrides
    public function handle(Request $request, Closure $next): Response
    {
        $logData = 'URL: ' . $request->url() . ' | Method: ' . $request->method() . ' | Timestamp: ' . now()->format(format: 'Y-m-d H:i:s') . P
        Storage::append(path: 'LogReqLab3/log.txt', data: $logData);
        return $next($request);
    }
}

```

- **What Are Middleware Parameters?**

Middleware parameters allow you to pass additional arguments to middleware at runtime. These parameters can modify the behavior of the middleware dynamically, which is particularly useful for cases where you need flexible logic, such as restricting access based on user roles, locations, or in our case—age.

`CheckAge` Middleware with Parameters



CheckAge.php ×

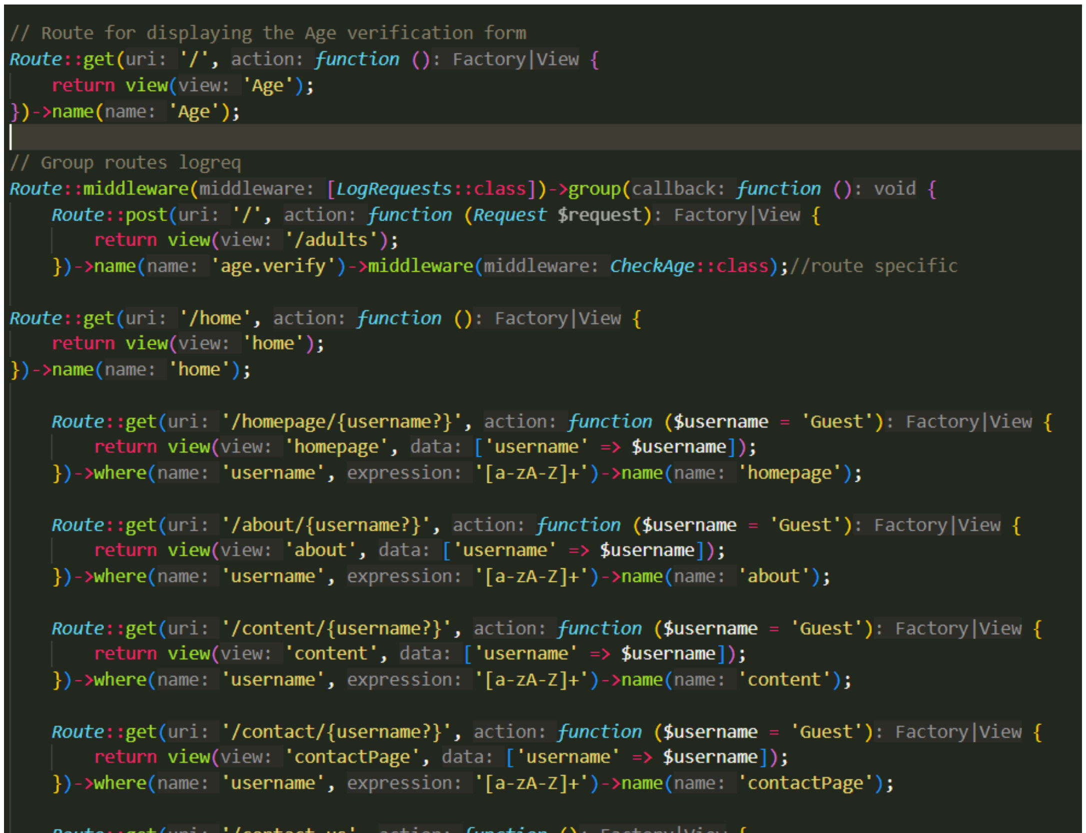
app > Http > Middleware > CheckAge.php > CheckAge

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\View;

5 references | 0 implementations
class CheckAge
{
    0 references | 0 overrides
    public function handle($request, Closure $next, $minAge = 18): mixed|Response
    {
        $age = $request->input('age', 0);
        if ($age >= 18 && $age <= 20) {
            return response()->view(view: 'home');
        }
        if ($age >= 21) {
            return response()->view(view: 'adults');
        }
        return response()->view(view: 'denied');
    }
}
```

We modified the **`CheckAge`** middleware to accept a parameter that defines the minimum age required to access certain routes. By passing a parameter to the middleware, we allow different age restrictions for different parts of the application.

- **Registering Routes with Middleware Parameters**

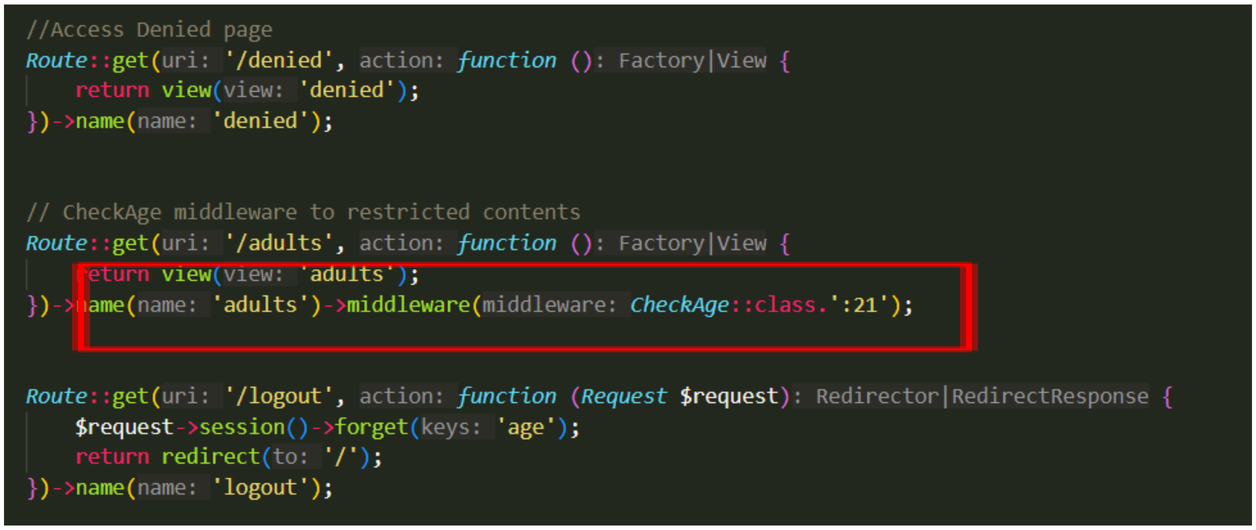
We used middleware parameters in route definitions to enforce age restrictions.



```
// Route for displaying the Age verification form
Route::get(uri: '/', action: function (): Factory|View {
    return view(view: 'Age');
})->name(name: 'Age');

// Group routes logreq
Route::middleware(middleware: [LogRequests::class])->group(callback: function (): void {
    Route::post(uri: '/', action: function (Request $request): Factory|View {
        return view(view: '/adults');
    })->name(name: 'age.verify')->middleware(middleware: CheckAge::class);//route specific

Route::get(uri: '/home', action: function (): Factory|View {
    return view(view: 'home');
})->name(name: 'home');

    Route::get(uri: '/homepage/{username?}', action: function ($username = 'Guest'): Factory|View {
        return view(view: 'homepage', data: ['username' => $username]);
    })->where(name: 'username', expression: '[a-zA-Z]+')->name(name: 'homepage');

    Route::get(uri: '/about/{username?}', action: function ($username = 'Guest'): Factory|View {
        return view(view: 'about', data: ['username' => $username]);
    })->where(name: 'username', expression: '[a-zA-Z]+')->name(name: 'about');

    Route::get(uri: '/content/{username?}', action: function ($username = 'Guest'): Factory|View {
        return view(view: 'content', data: ['username' => $username]);
    })->where(name: 'username', expression: '[a-zA-Z]+')->name(name: 'content');

    Route::get(uri: '/contact/{username?}', action: function ($username = 'Guest'): Factory|View {
        return view(view: 'contactPage', data: ['username' => $username]);
    })->where(name: 'username', expression: '[a-zA-Z]+')->name(name: 'contactPage');

    Route::get(uri: '/contact-us', action: function (): Factory|View {
```

We applied the middleware with parameters directly in the route definitions by using:

This passes the value 21 as a parameter to the CheckAge middleware, enforcing a minimum age of 21 for that route.



```
//Access Denied page
Route::get(uri: '/denied', action: function (): Factory|View {
    return view(view: 'denied');
})->name(name: 'denied');


// CheckAge middleware to restricted contents
Route::get(uri: '/adults', action: function (): Factory|View {
    return view(view: 'adults');
})->name(name: 'adults')->middleware(middleware: CheckAge::class.':21');


Route::get(uri: '/logout', action: function (Request $request): Redirector|RedirectResponse {
    $request->session()->forget(keys: 'age');
    return redirect(to: '/');
})->name(name: 'logout');
```

- The `/home` route uses the `CheckAge` middleware with a parameter of `18`, allowing users aged 18 and above to access the page.
- The `/restricted` route uses the `CheckAge` middleware with a parameter of `21`, restricting access to users aged 21 and above.

Benefits of Middleware Parameters

- **Flexibility**: Middleware parameters make the code reusable across multiple routes, with slight changes in behavior. For example, by passing different age restrictions to the `CheckAge` middleware, we can enforce access rules for multiple sections of the application without duplicating logic.
- **Cleaner Code**: Middleware parameters help avoid hardcoding values into the middleware, keeping the codebase clean and maintainable.