BICOL UNIVERSITY COLLEGE OF SCIENCE

LEGAZPI CITY, ALBAY

IT Elect 1 – WEB DEVELOPMENT

DOCUMENTATION

(GROUP 5, LABORATORY 4)

MIDDLEWARE

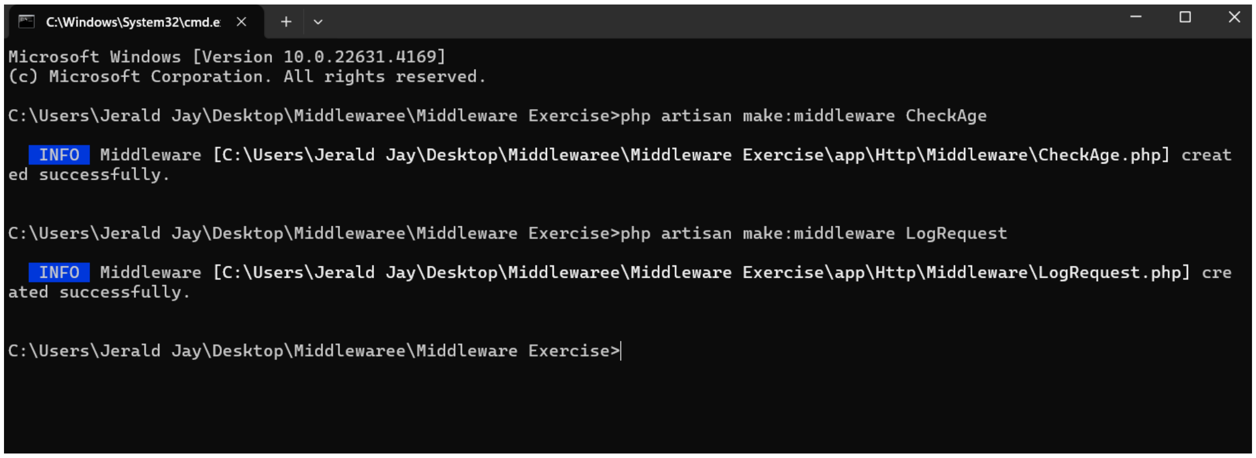
SUBMITTED BY:

JERALD JAY G. BUBAN

BSIT – 3C

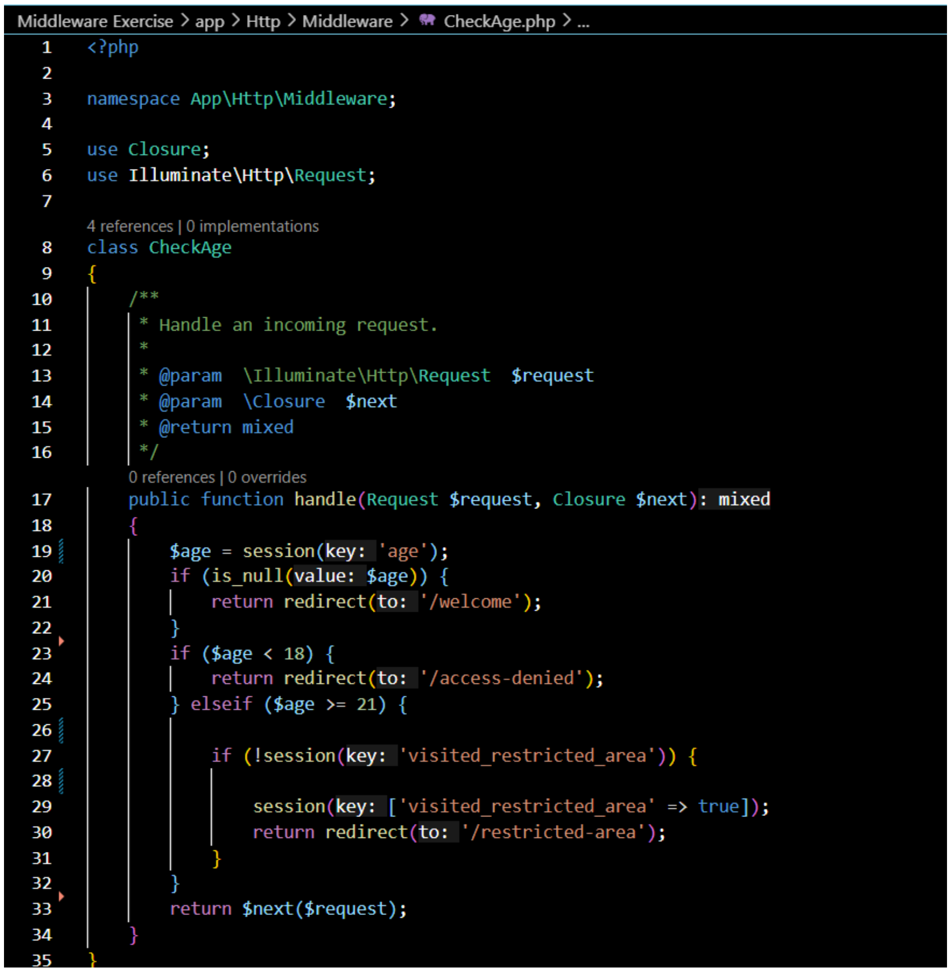
Part 1: Create and Register New Middleware:

Using the command line, create new middleware named CheckAge and LogRequests.



```
Microsoft Windows [Version 10.0.22631.4169]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Jerald Jay\Desktop\Middlewaree\Middleware Exercise>php artisan make:middleware CheckAge

   INFO  Middleware [C:\Users\Jerald Jay\Desktop\Middlewaree\Middleware Exercise\app\Http\Middleware\CheckAge.php] creat
ed successfully.


C:\Users\Jerald Jay\Desktop\Middlewaree\Middleware Exercise>php artisan make:middleware LogRequest

   INFO  Middleware [C:\Users\Jerald Jay\Desktop\Middlewaree\Middleware Exercise\app\Http\Middleware\LogRequest.php] cre
ated successfully.


C:\Users\Jerald Jay\Desktop\Middlewaree\Middleware Exercise>
```

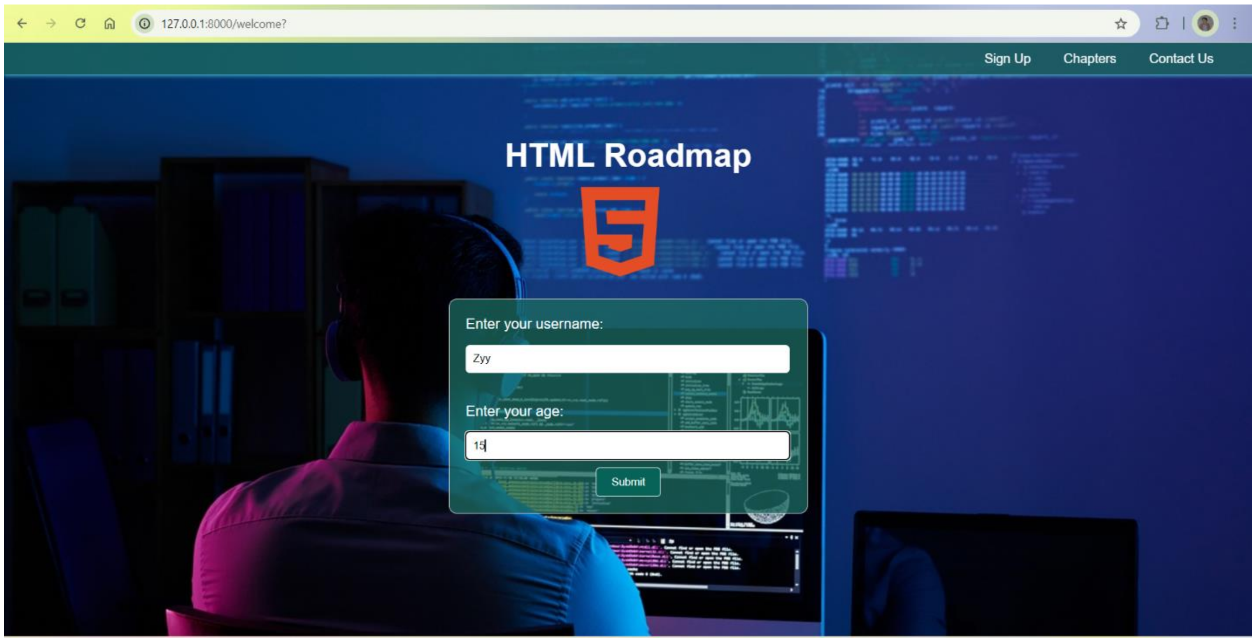
The CheckAge middleware should check if a user's age is greater than or equal to 18. If the age does not meet the condition, redirect the user to an “Access Denied" page.



Middleware Exercise > app > Http > Middleware > CheckAge.php > ...

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

class CheckAge
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle(Request $request, Closure $next): mixed
    {
        $age = session(key: 'age');
        if (is_null(value: $age)) {
            return redirect(to: '/welcome');
        }
        if ($age < 18) {
            return redirect(to: '/access-denied');
        } elseif ($age >= 21) {

            if (!session(key: 'visited_restricted_area')) {

                session(key: ['visited_restricted_area' => true]);
                return redirect(to: '/restricted-area');
            }
        }
        return $next($request);
    }
}
```

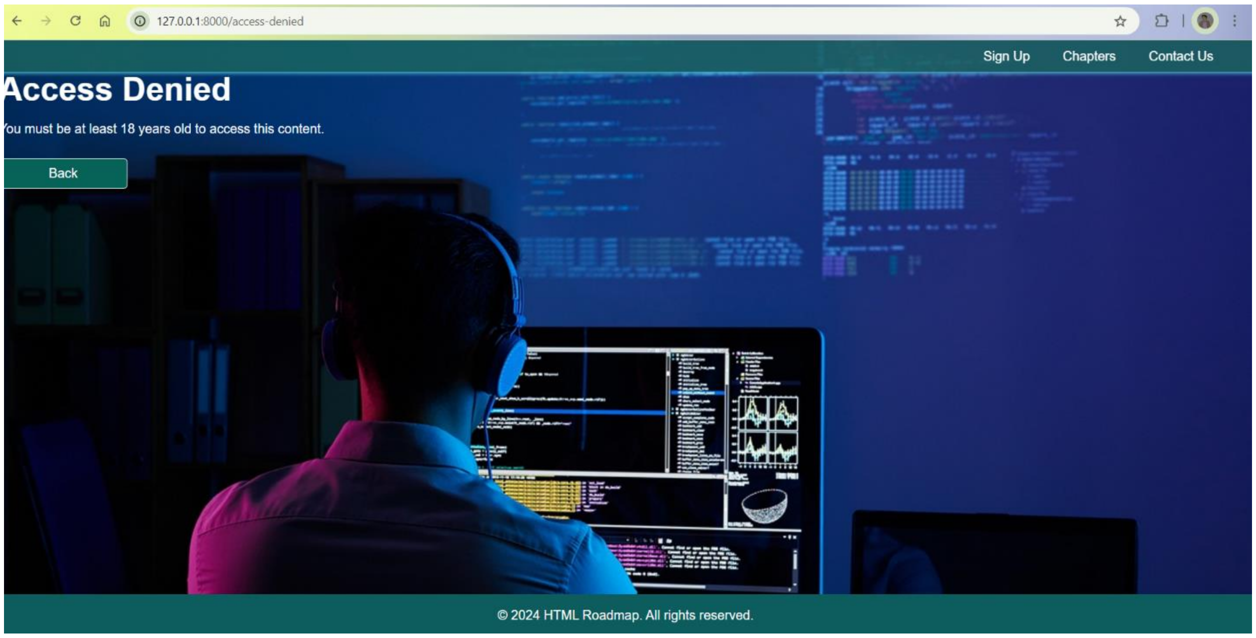
- In this code, you can see in the logic that if the user's age is less than 17, they will be redirected to the access-denied page. And they cannot access the remaining pages.
- It enforces an age restriction by checking if a user is under 18 and, if so, redirects them to an "access denied" page. This is often used to limit access to certain areas of an application or website for users who do not meet the age requirement, ensuring compliance with age-related regulations or guidelines.



Enter your username: Zyy

Enter your age: 15

- Here's the example of entering an age less than 18.



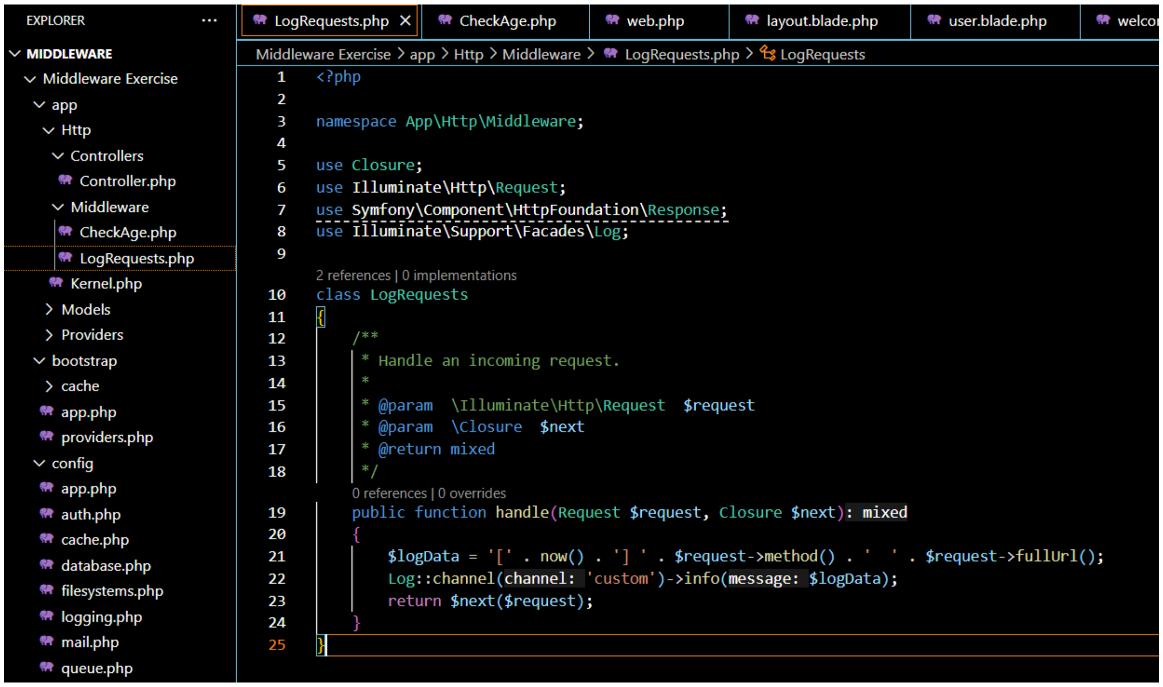
Access Denied

You must be at least 18 years old to access this content.

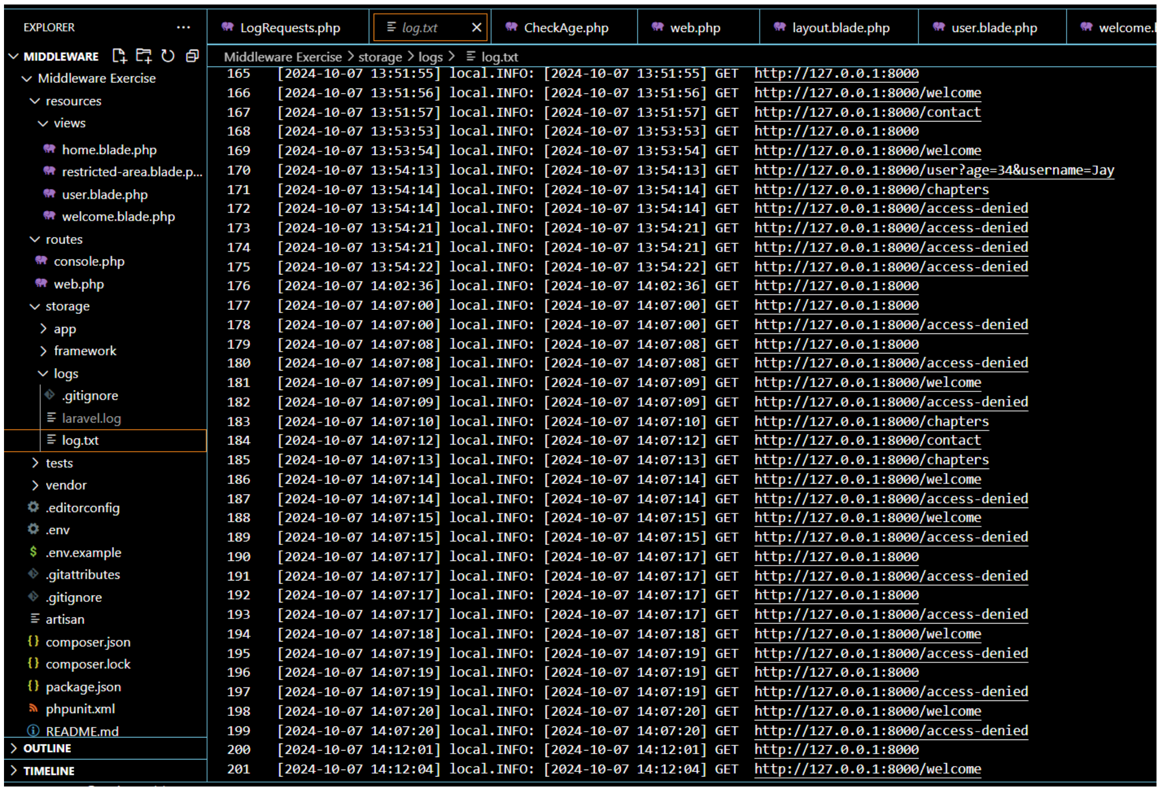
© 2024 HTML Roadmap. All rights reserved.

- After submitting, it automatically directs the user here in this access-denied page because he didn't met the age requirement.

LogRequests should log the details of all HTTP requests to a file called log.txt, include the URL method, and timestamp



```php
<?php

namespace App\Http\Middleware;

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

2 references | 0 implementations
class LogRequests
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    0 references | 0 overrides
    public function handle(Request $request, Closure $next): mixed
    {
        $logData = '[' . now() . '] ' . $request->method() . ' ' . $request->fullUrl();
        Log::channel(channel: 'custom')->info(message: $logData);
        return $next($request);
    }
}
```



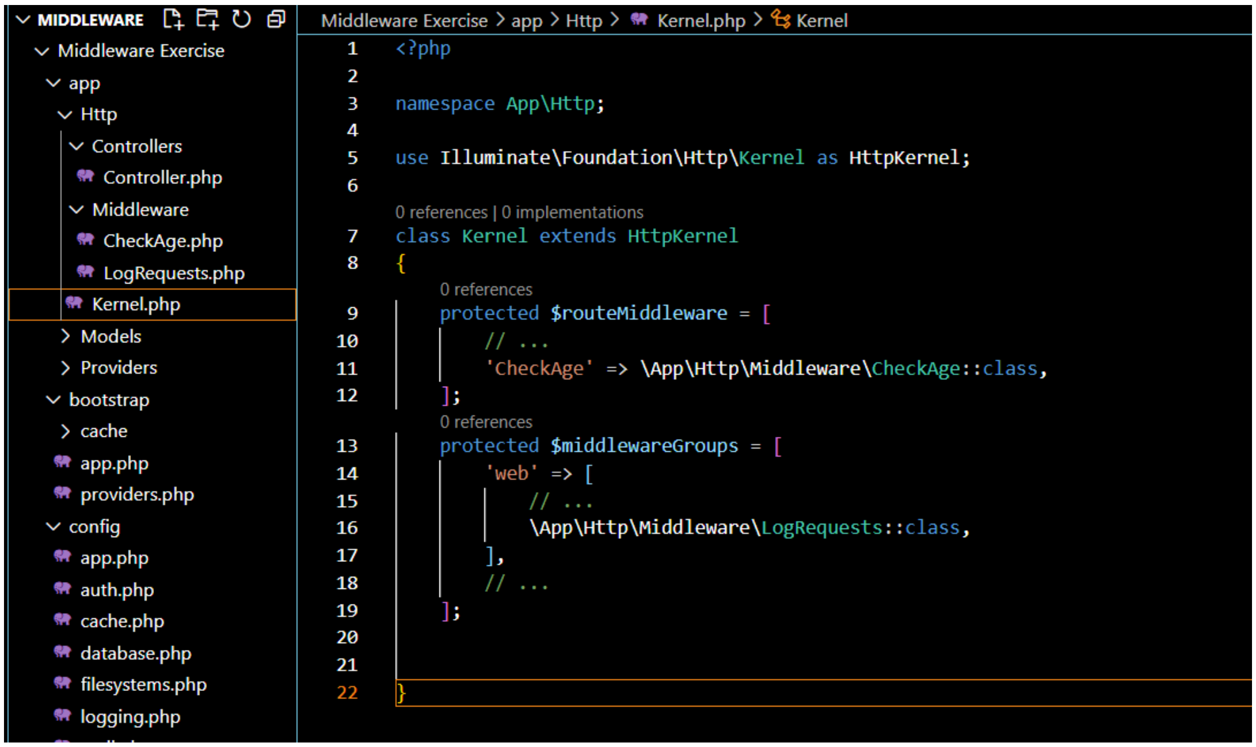
```
[2024-10-07 13:51:55] local.INFO: [2024-10-07 13:51:55] GET http://127.0.0.1:8000
[2024-10-07 13:51:56] local.INFO: [2024-10-07 13:51:56] GET http://127.0.0.1:8000/welcome
[2024-10-07 13:51:57] local.INFO: [2024-10-07 13:51:57] GET http://127.0.0.1:8000/contact
[2024-10-07 13:53:53] local.INFO: [2024-10-07 13:53:53] GET http://127.0.0.1:8000
[2024-10-07 13:53:54] local.INFO: [2024-10-07 13:53:54] GET http://127.0.0.1:8000/welcome
[2024-10-07 13:54:13] local.INFO: [2024-10-07 13:54:13] GET http://127.0.0.1:8000/user?age=34&username=Jay
[2024-10-07 13:54:14] local.INFO: [2024-10-07 13:54:14] GET http://127.0.0.1:8000/chapters
[2024-10-07 13:54:14] local.INFO: [2024-10-07 13:54:14] GET http://127.0.0.1:8000/access-denied
[2024-10-07 13:54:21] local.INFO: [2024-10-07 13:54:21] GET http://127.0.0.1:8000/access-denied
[2024-10-07 13:54:21] local.INFO: [2024-10-07 13:54:21] GET http://127.0.0.1:8000/access-denied
[2024-10-07 13:54:22] local.INFO: [2024-10-07 13:54:22] GET http://127.0.0.1:8000/access-denied
[2024-10-07 14:02:36] local.INFO: [2024-10-07 14:02:36] GET http://127.0.0.1:8000
[2024-10-07 14:07:00] local.INFO: [2024-10-07 14:07:00] GET http://127.0.0.1:8000
[2024-10-07 14:07:00] local.INFO: [2024-10-07 14:07:00] GET http://127.0.0.1:8000/access-denied
[2024-10-07 14:07:08] local.INFO: [2024-10-07 14:07:08] GET http://127.0.0.1:8000
[2024-10-07 14:07:08] local.INFO: [2024-10-07 14:07:08] GET http://127.0.0.1:8000/access-denied
[2024-10-07 14:07:09] local.INFO: [2024-10-07 14:07:09] GET http://127.0.0.1:8000/welcome
[2024-10-07 14:07:09] local.INFO: [2024-10-07 14:07:09] GET http://127.0.0.1:8000/access-denied
[2024-10-07 14:07:10] local.INFO: [2024-10-07 14:07:10] GET http://127.0.0.1:8000/chapters
[2024-10-07 14:07:12] local.INFO: [2024-10-07 14:07:12] GET http://127.0.0.1:8000/contact
[2024-10-07 14:07:13] local.INFO: [2024-10-07 14:07:13] GET http://127.0.0.1:8000/chapters
[2024-10-07 14:07:14] local.INFO: [2024-10-07 14:07:14] GET http://127.0.0.1:8000/welcome
[2024-10-07 14:07:14] local.INFO: [2024-10-07 14:07:14] GET http://127.0.0.1:8000/access-denied
[2024-10-07 14:07:15] local.INFO: [2024-10-07 14:07:15] GET http://127.0.0.1:8000/welcome
[2024-10-07 14:07:15] local.INFO: [2024-10-07 14:07:15] GET http://127.0.0.1:8000/access-denied
[2024-10-07 14:07:17] local.INFO: [2024-10-07 14:07:17] GET http://127.0.0.1:8000
[2024-10-07 14:07:17] local.INFO: [2024-10-07 14:07:17] GET http://127.0.0.1:8000/access-denied
[2024-10-07 14:07:17] local.INFO: [2024-10-07 14:07:17] GET http://127.0.0.1:8000
[2024-10-07 14:07:17] local.INFO: [2024-10-07 14:07:17] GET http://127.0.0.1:8000/access-denied
[2024-10-07 14:07:18] local.INFO: [2024-10-07 14:07:18] GET http://127.0.0.1:8000/welcome
[2024-10-07 14:07:19] local.INFO: [2024-10-07 14:07:19] GET http://127.0.0.1:8000/access-denied
[2024-10-07 14:07:19] local.INFO: [2024-10-07 14:07:19] GET http://127.0.0.1:8000
[2024-10-07 14:07:19] local.INFO: [2024-10-07 14:07:19] GET http://127.0.0.1:8000/access-denied
[2024-10-07 14:07:20] local.INFO: [2024-10-07 14:07:20] GET http://127.0.0.1:8000/welcome
[2024-10-07 14:07:20] local.INFO: [2024-10-07 14:07:20] GET http://127.0.0.1:8000/access-denied
[2024-10-07 14:12:01] local.INFO: [2024-10-07 14:12:01] GET http://127.0.0.1:8000
[2024-10-07 14:12:04] local.INFO: [2024-10-07 14:12:04] GET http://127.0.0.1:8000/welcome
```

- This `LogRequests` middleware logs each HTTP request by capturing its timestamp, method, and full URL. The `Log::channel('custom')->info($logData);` command records this information to a custom logging channel for monitoring requests. Finally, `return

$next($request);` allows the request to proceed to the next middleware or controller action.

- In the LogRequest, it will take note of all the pages that the user go to or accessed, with an accurate date and timestamp.

Register the middleware in the app/Http/Kernel.php file under the appropriate section.

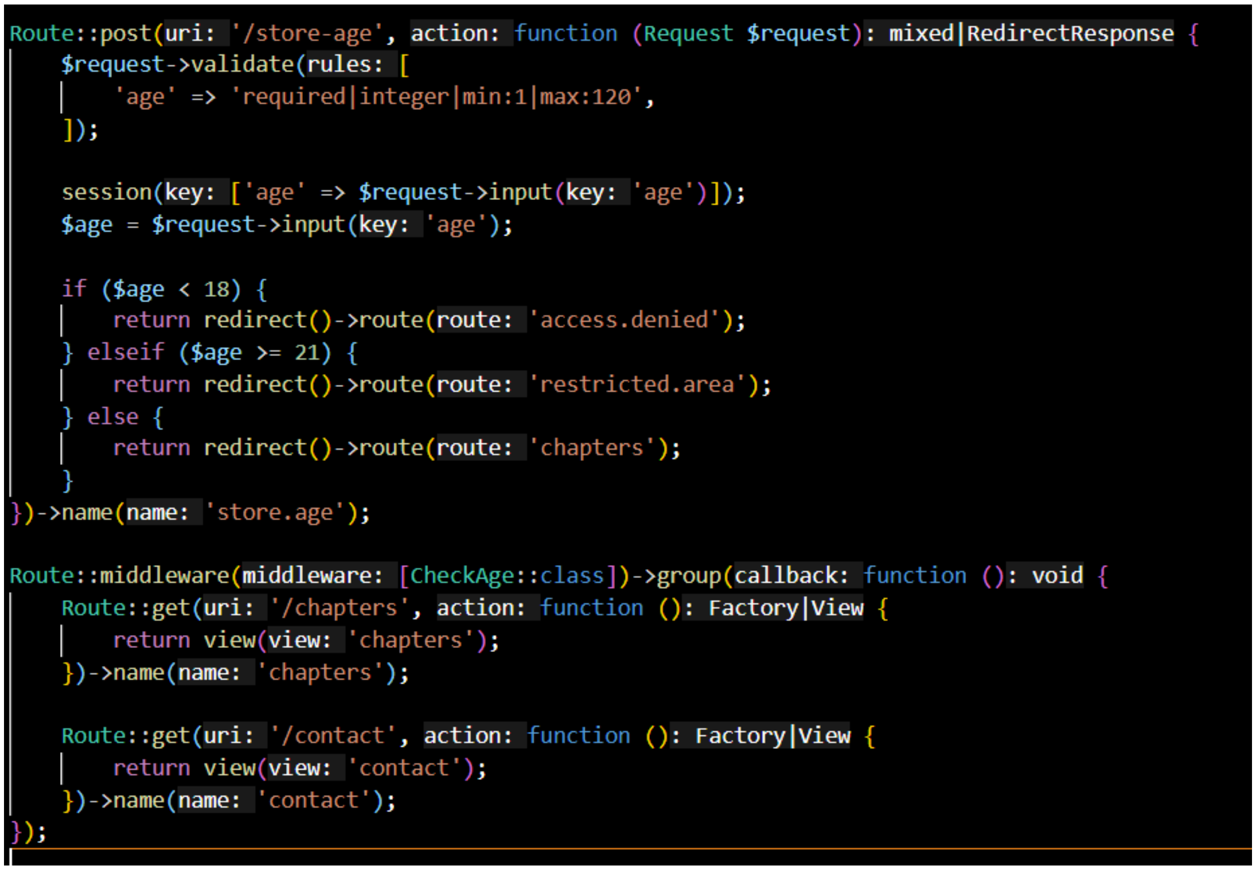


```php
<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel
{
    protected $routeMiddleware = [
        // ...
        'CheckAge' => \App\Http\Middleware\CheckAge::class,
    ];
    protected $middlewareGroups = [
        'web' => [
            // ...
            \App\Http\Middleware\LogRequests::class,
        ],
        // ...
    ];


}
```

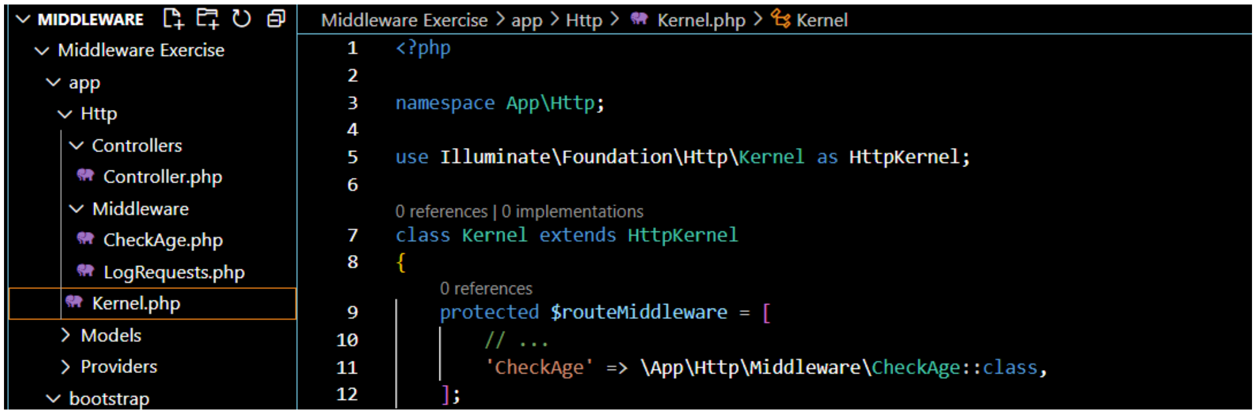
- **GlobalMiddleware(LogRequests):**
  - This middleware logs every request made to the application, tracking user activity and diagnose issues.
  - The LogRequests middleware is added under the 'web' group. To ensure that the middleware is globally applied to all routes under the 'web' middleware group, logging every incoming request made to the application Route-Specific.
- **Route-Specific Middleware(CheckAge):**
  - The CheckAge middleware is registered in the routeMiddleware array. This allows it to be applied to specific routes by using the 'CheckAge' key. This middleware controls access to specific routes based on the user's age.
  - If the user inputs an age that didn't met the age requirement which is 18, the user is redirected to the access-denied page and can't access the remaining page.

Part 2: Assign Middleware to Routes:

Create a route group that assigns the CheckAge middleware to a specific route.



```php
Route::post(uri: '/store-age', action: function (Request $request): mixed|RedirectResponse {
    $request->validate(rules: [
        'age' => 'required|integer|min:1|max:120',
    ]);

    session(key: ['age' => $request->input(key: 'age')]);
    $age = $request->input(key: 'age');

    if ($age < 18) {
        return redirect()->route(route: 'access.denied');
    } elseif ($age >= 21) {
        return redirect()->route(route: 'restricted.area');
    } else {
        return redirect()->route(route: 'chapters');
    }
})->name(name: 'store.age');

Route::middleware(middleware: [CheckAge::class])->group(callback: function (): void {
    Route::get(uri: '/chapters', action: function (): Factory|View {
        return view(view: 'chapters');
    })->name(name: 'chapters');

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

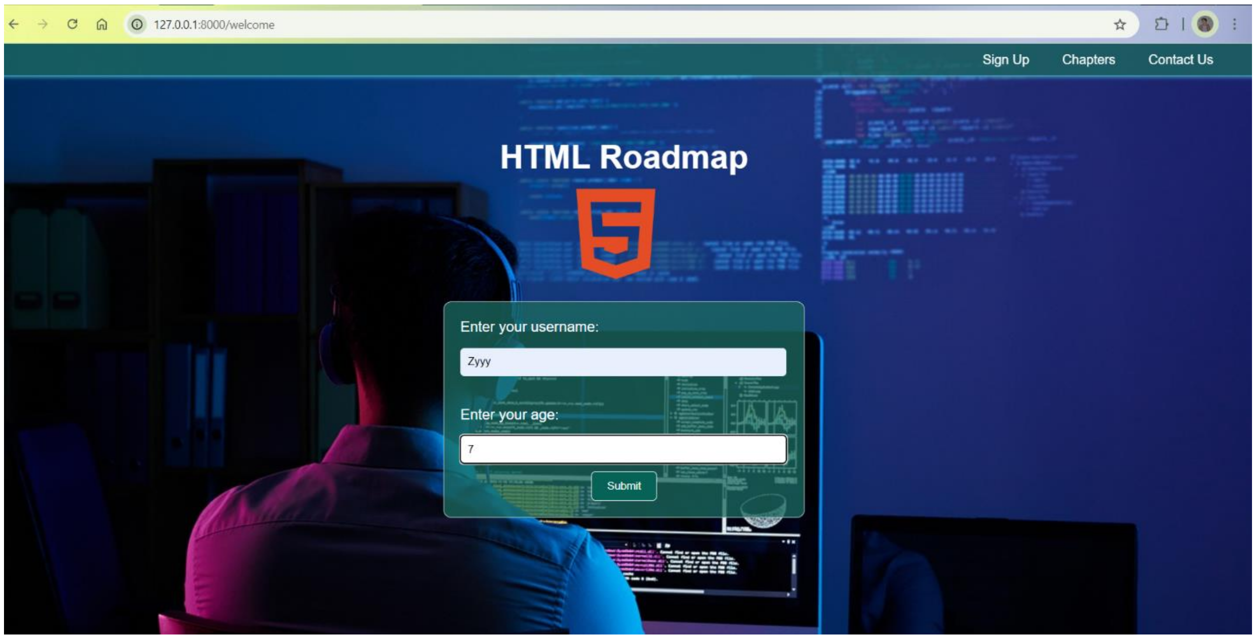


```php
<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

0 references | 0 implementations
class Kernel extends HttpKernel
{
    0 references
    protected $routeMiddleware = [
        // ...
        'CheckAge' => \App\Http\Middleware\CheckAge::class,
    ];
```

- **Storing and Validating Age**: The /store-age route handles a POST request, validating the age input, storing it in the session, and redirecting users based on age—under 18 to access.denied, 21+ to restricted.area, and others to chapters.
- **Age-Restricted Routes**: The CheckAge middleware is applied to a group of routes, including chapters and contact, restricting access based on age validation to enforce controlled access to certain views.

- And the code in the Kernel.php registers a route-specific middleware called CheckAge by adding it to the $routeMiddleware array with the alias 'CheckAge', linking it to the CheckAge middleware class. This allows the CheckAge middleware to be applied to specific routes, enabling conditional access based on user age.

Test the middleware by simulating different age values in the request (Test various scenarios where the middleware passes or fails the request).

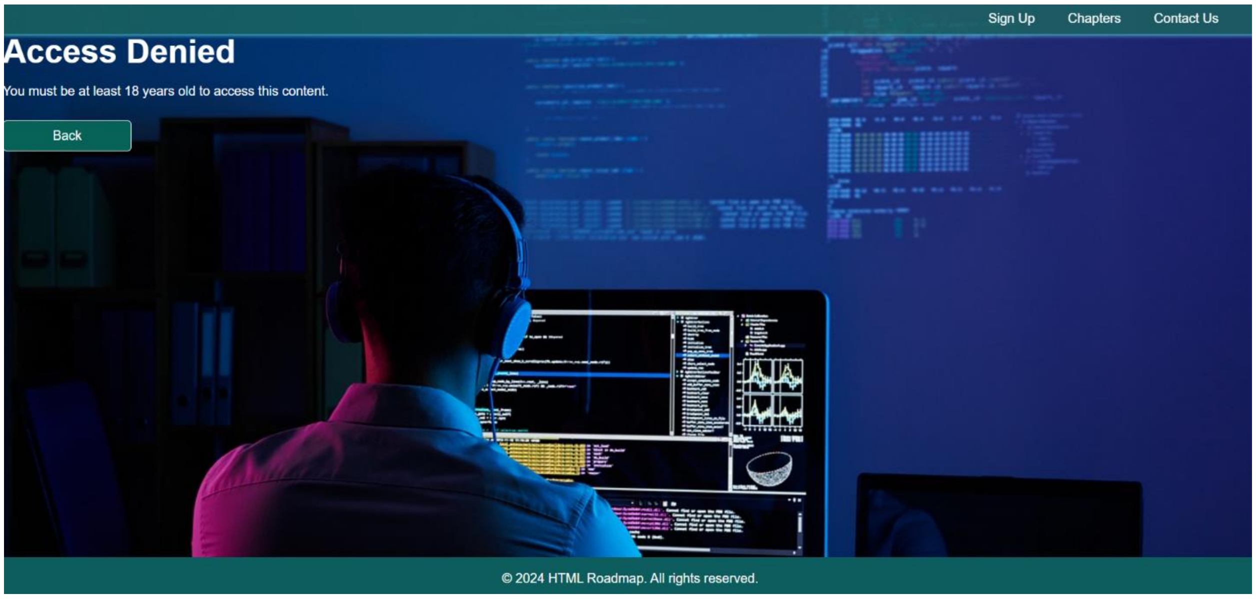


Enter your username:

Zyyy

Enter your age:

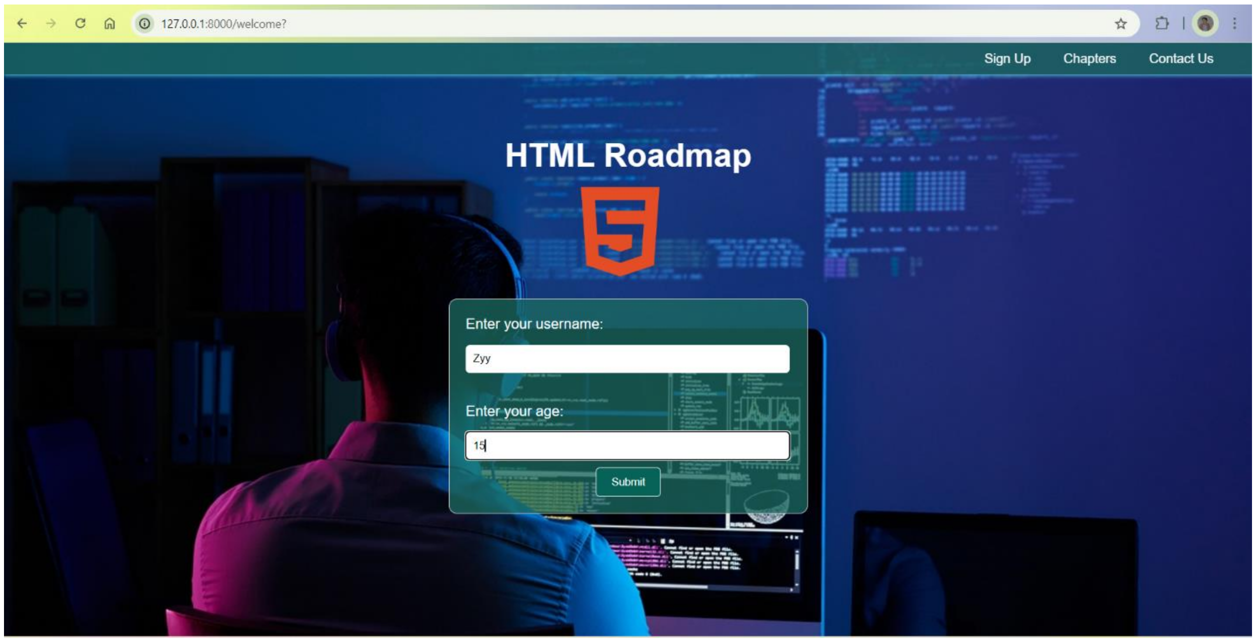
7



Access Denied

You must be at least 18 years old to access this content.

© 2024 HTML Roadmap. All rights reserved.


127.0.0.1:8000/welcome?

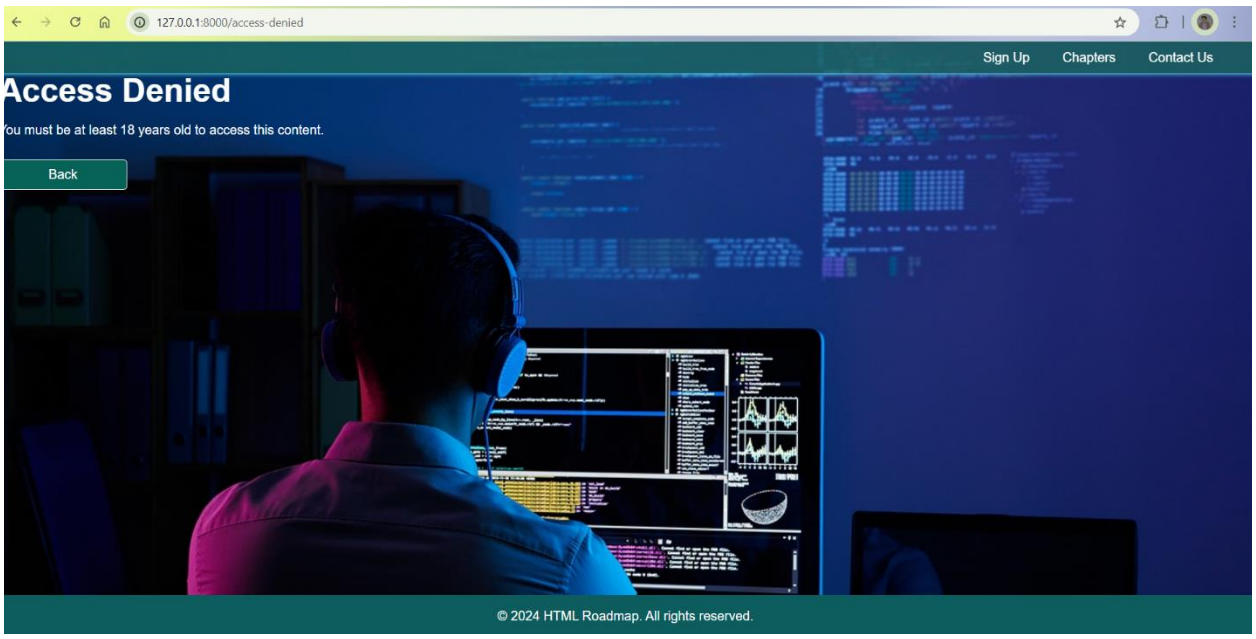
HTML Roadmap

Enter your username:

Zyy

Enter your age:

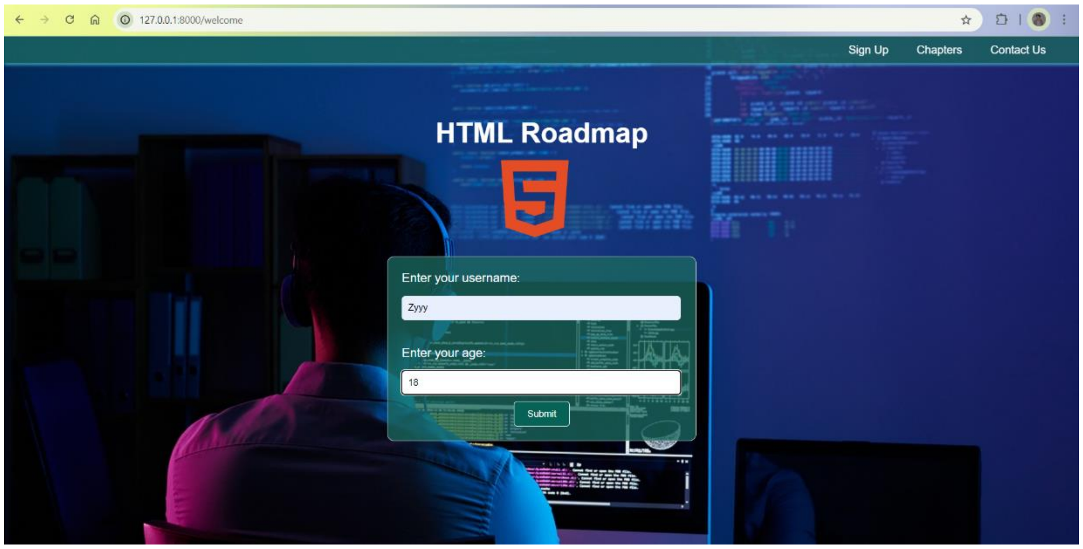
15


127.0.0.1:8000/access-denied

Access Denied

You must be at least 18 years old to access this content.

© 2024 HTML Roadmap. All rights reserved.



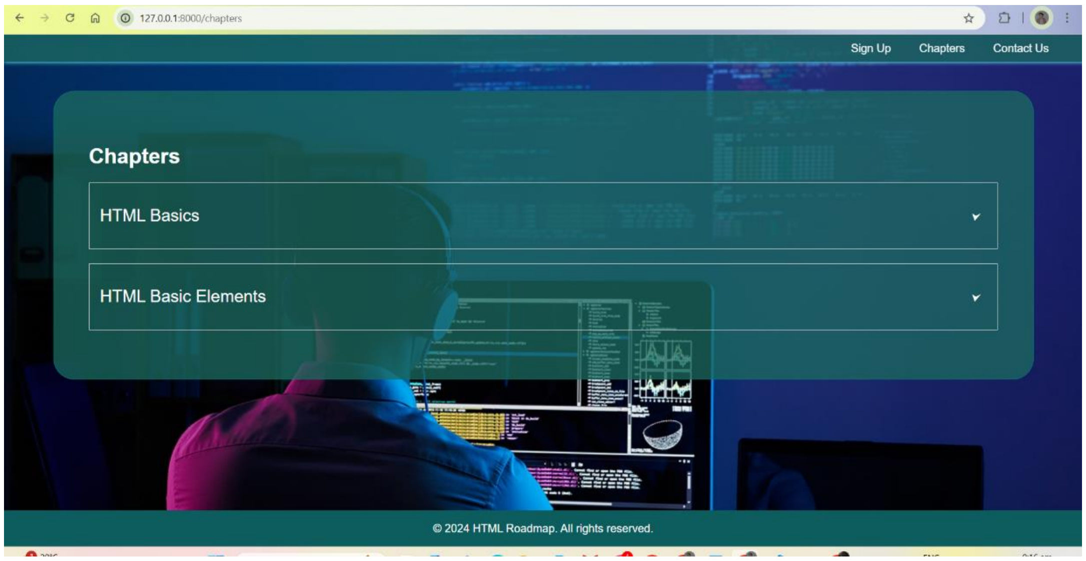
HTML Roadmap

Enter your username:

Zyyy

Enter your age:

18

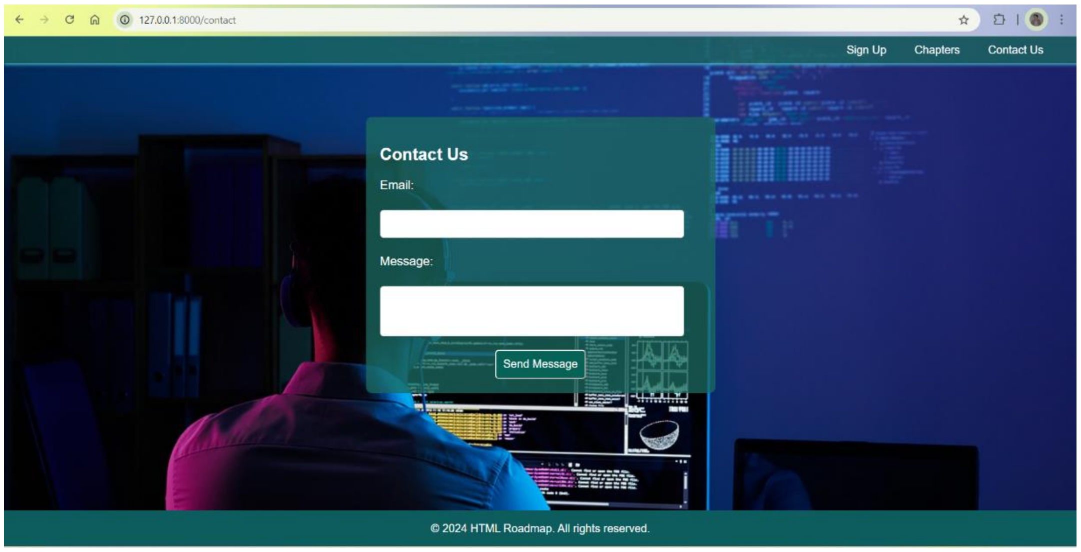


Chapters

HTML Basics

HTML Basic Elements

© 2024 HTML Roadmap. All rights reserved.



Contact Us

Email:

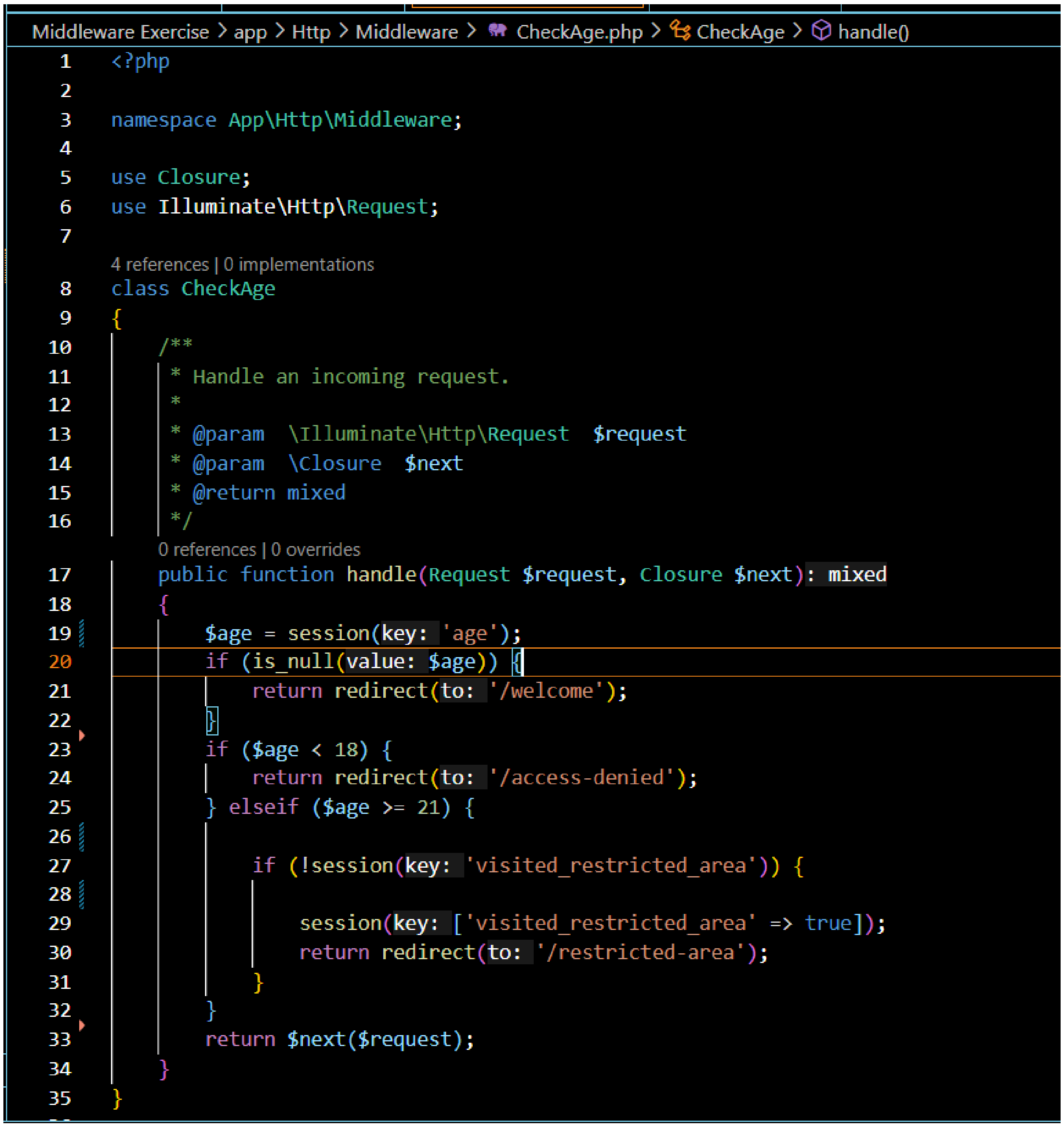
Message:

© 2024 HTML Roadmap. All rights reserved.

- Here are some of the examples of entering an age that doesn't met the age requirement and an age that meets it.

Part 3: Create Middleware with Parameters:

Modify the CheckAge middleware to accept a parameter (e.g., the minimum age requirement).

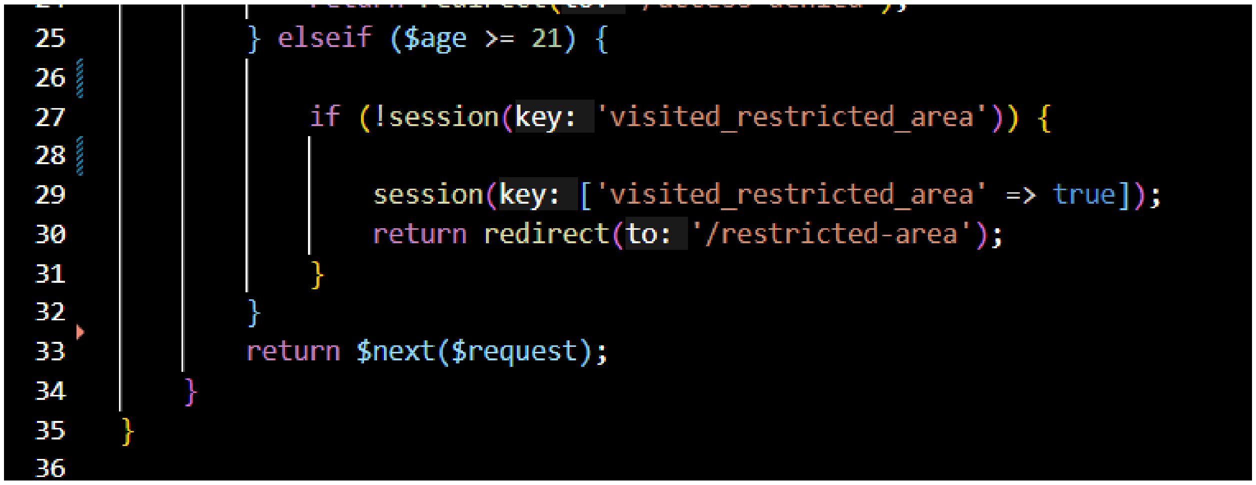


Middleware Exercise > app > Http > Middleware > CheckAge.php > CheckAge > handle()

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

4 references | 0 implementations
class CheckAge
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    0 references | 0 overrides
    public function handle(Request $request, Closure $next): mixed
    {
        $age = session(key: 'age');
        if (is_null(value: $age)) {
            return redirect(to: '/welcome');
        }
        if ($age < 18) {
            return redirect(to: '/access-denied');
        } elseif ($age >= 21) {

            if (!session(key: 'visited_restricted_area')) {

                session(key: ['visited_restricted_area' => true]);
                return redirect(to: '/restricted-area');
            }
        }
        return $next($request);
    }
}
```

- The `CheckAge` middleware controls access based on the user's age stored in the session. It retrieves the `age` from the session, and if `age` is `null`, it redirects the user to `/welcome`, ensuring they have set their age before accessing age-restricted areas. If the age is under 18, it redirects them to `/access-denied`; if the age is 21 or over and they haven't visited the restricted area, it marks it in the session and redirects to `/restricted-area`. If none of these conditions apply, `return $next($request);` lets the request continue, allowing users within age criteria to access the intended route.

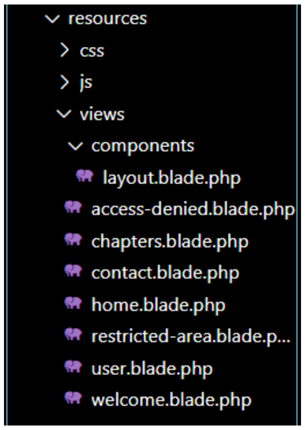
Create a new route that assigns the middleware with a parameter to enforce a different age restriction (e.g., 21 years old).



```
            } elseif ($age >= 21) {

                if (!session(key: 'visited_restricted_area')) {

                    session(key: ['visited_restricted_area' => true]);
                    return redirect(to: '/restricted-area');
                }
            }
            return $next($request);
        }
    }
```

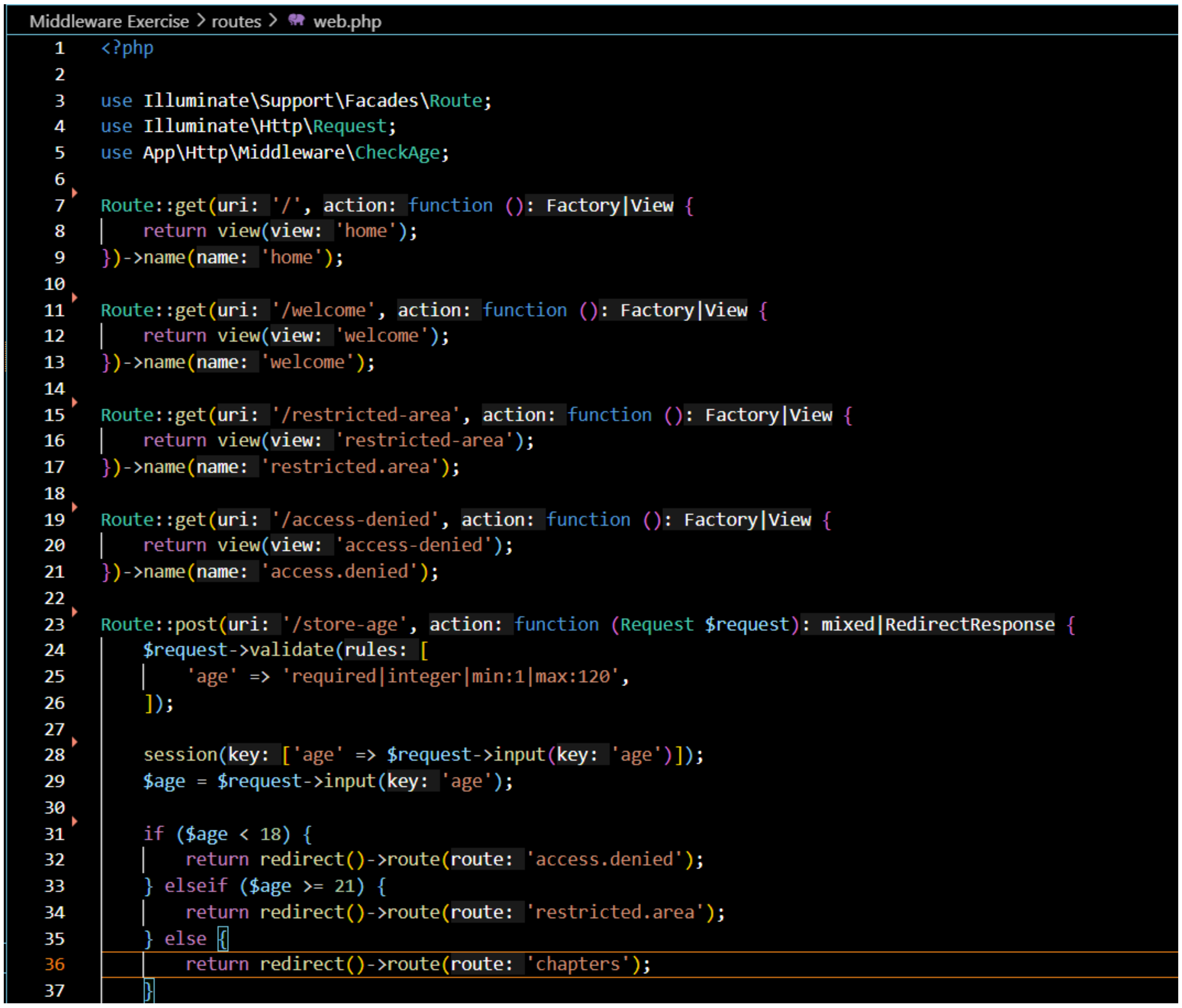
- This checks if the user's age is 21 or older, and if they haven't yet visited the restricted area (`session('visited_restricted_area')` is false), it sets a session flag (`visited_restricted_area`) to true and redirects them to `/restricted-area`. This ensures that users of the appropriate age are given access to restricted content but only redirected once to the restricted area for initial access. By setting this session flag, it avoids repeated redirections, streamlining user experience.

Blade Template Files



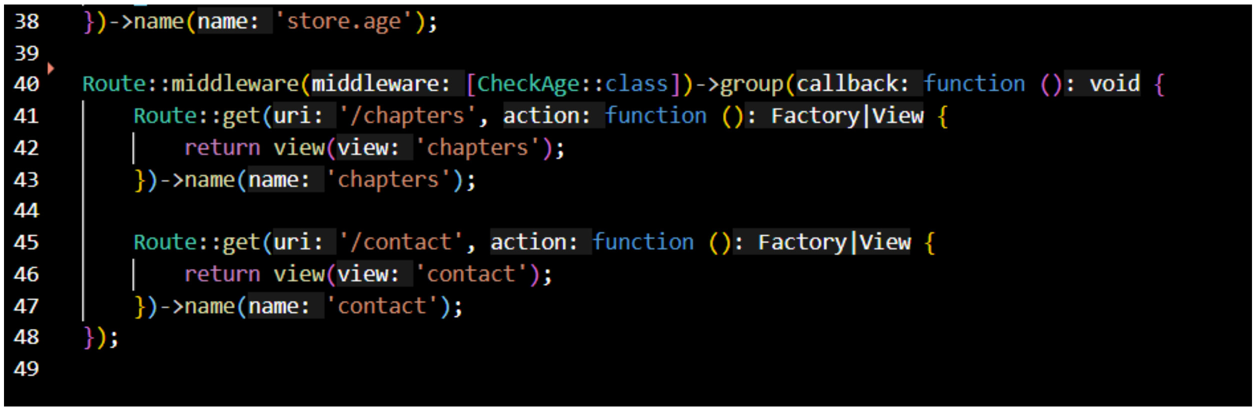
```
resources
  > css
  > js
  v views
    v components
        layout.blade.php
      access-denied.blade.php
      chapters.blade.php
      contact.blade.php
      home.blade.php
      restricted-area.blade.p...
      user.blade.php
      welcome.blade.php
```

Routing Configuration



Middleware Exercise > routes > web.php

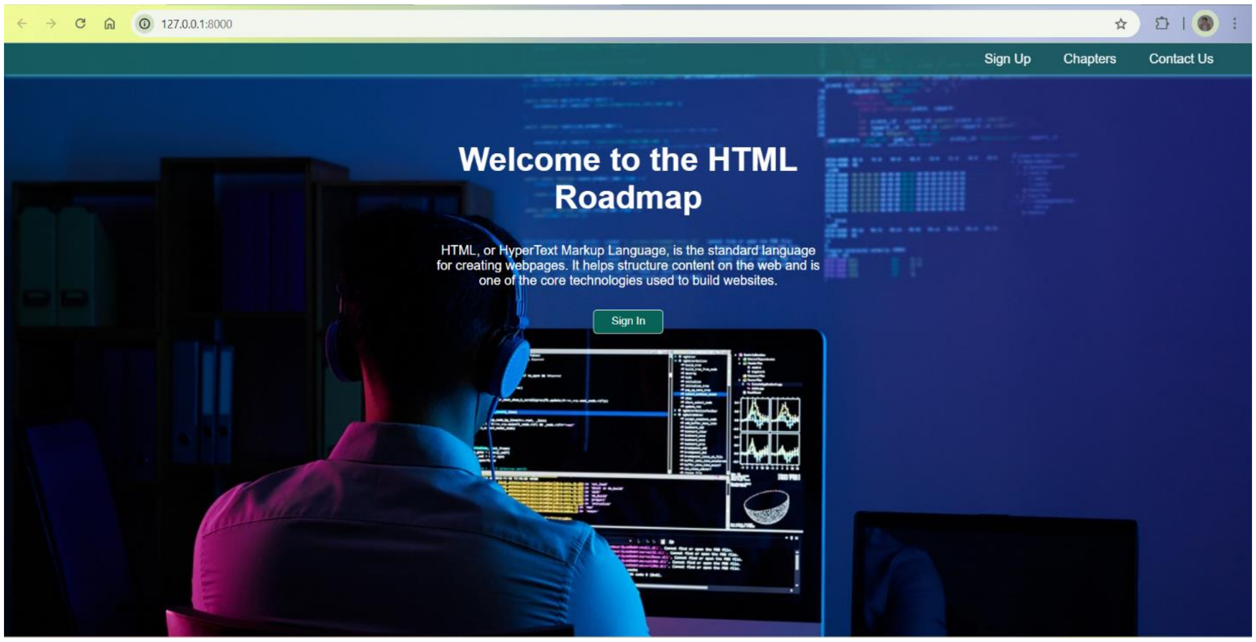
```php
<?php

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

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

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

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

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

Route::post(uri: '/store-age', action: function (Request $request): mixed|RedirectResponse {
    $request->validate(rules: [
        'age' => 'required|integer|min:1|max:120',
    ]);

    session(key: ['age' => $request->input(key: 'age')]);
    $age = $request->input(key: 'age');

    if ($age < 18) {
        return redirect()->route(route: 'access.denied');
    } elseif ($age >= 21) {
        return redirect()->route(route: 'restricted.area');
    } else {
        return redirect()->route(route: 'chapters');
    }
```



```php
})->name(name: 'store.age');

Route::middleware(middleware: [CheckAge::class])->group(callback: function (): void {
    Route::get(uri: '/chapters', action: function (): Factory|View {
        return view(view: 'chapters');
    })->name(name: 'chapters');

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

```

Rendered Web Pages

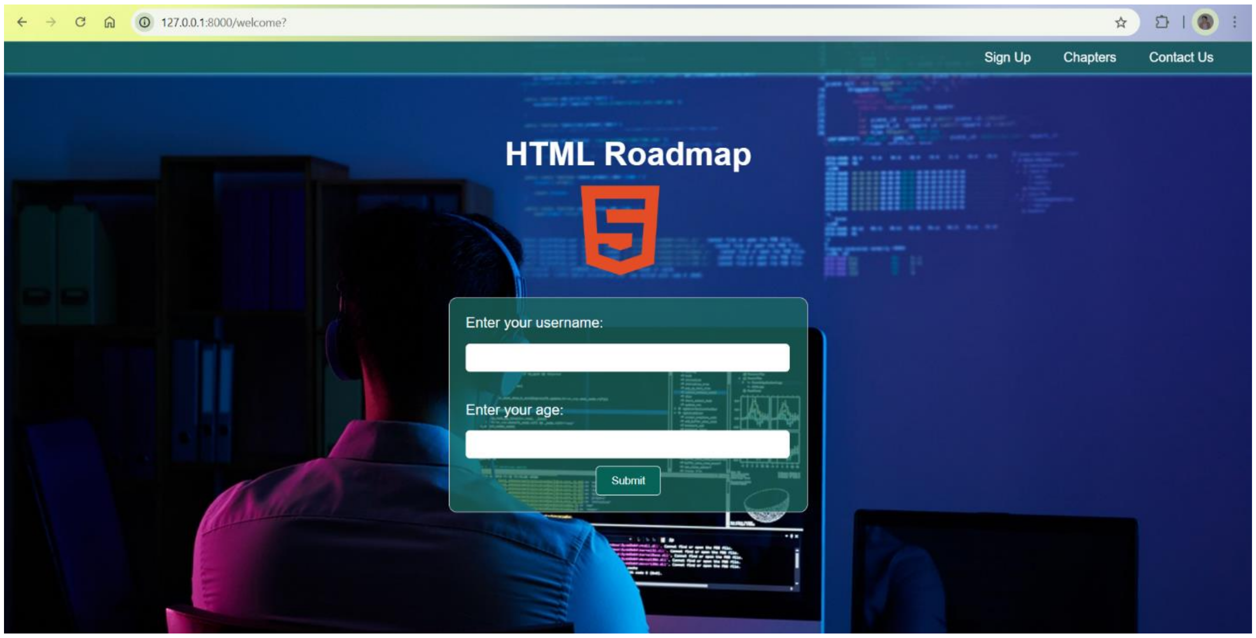
Home Page



Welcome to the HTML Roadmap

HTML, or HyperText Markup Language, is the standard language for creating webpages. It helps structure content on the web and is one of the core technologies used to build websites.

Sign Up Page

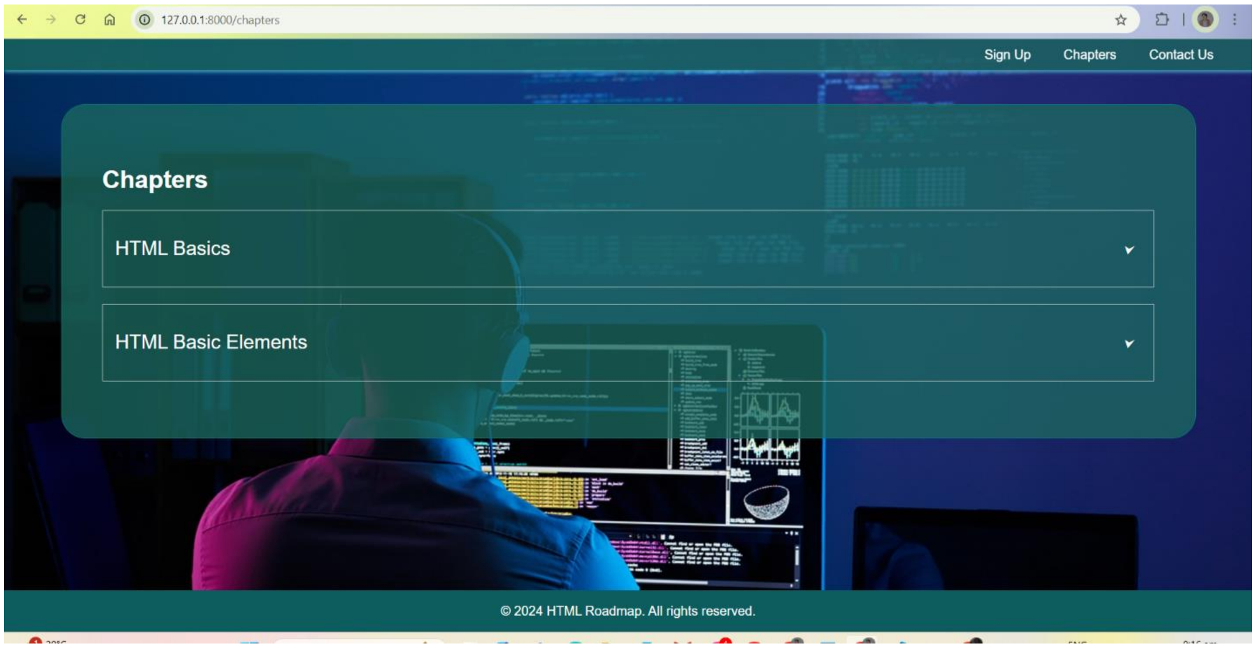


HTML Roadmap

Enter your username:

Enter your age:

Chapters Page


127.0.0.1:8000/chapters

Sign Up Chapters Contact Us

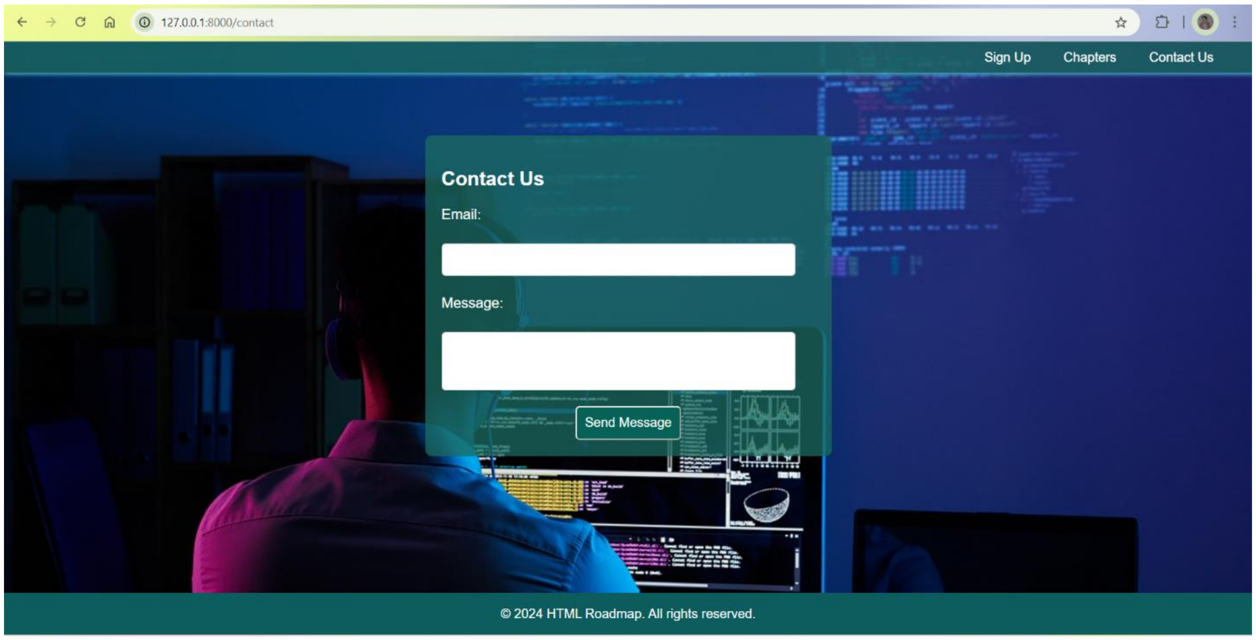
Chapters

HTML Basics

HTML Basic Elements

© 2024 HTML Roadmap. All rights reserved.

Contact Us Page


127.0.0.1:8000/contact

Sign Up Chapters Contact Us

Contact Us

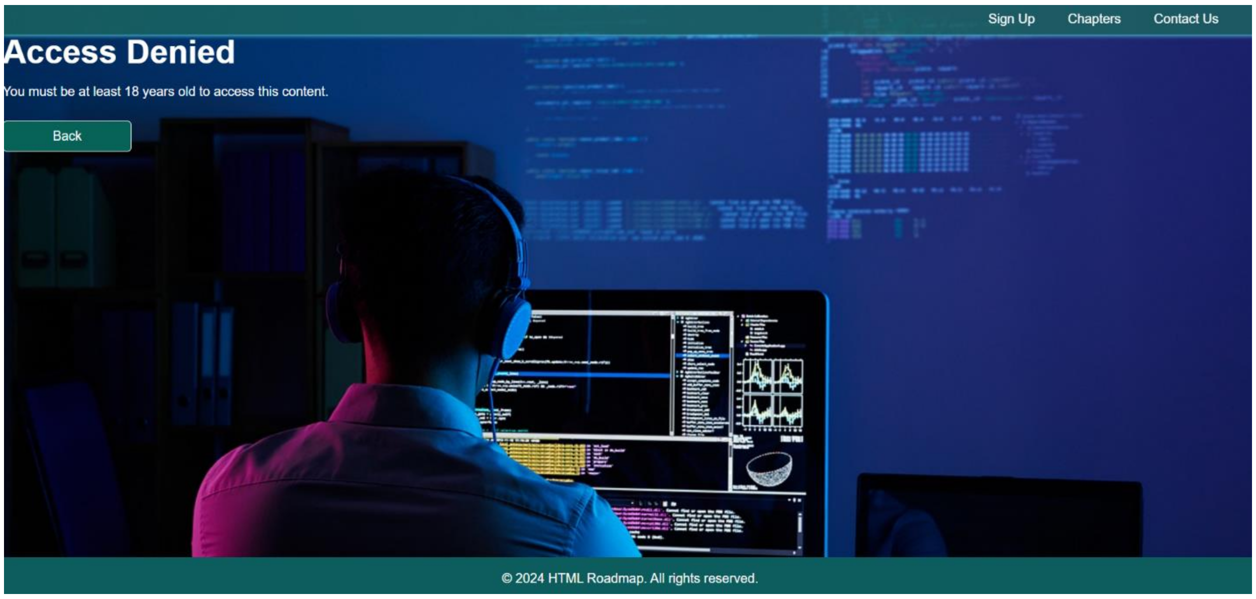
Email:

Message:

Send Message

© 2024 HTML Roadmap. All rights reserved.

Access Denied Page

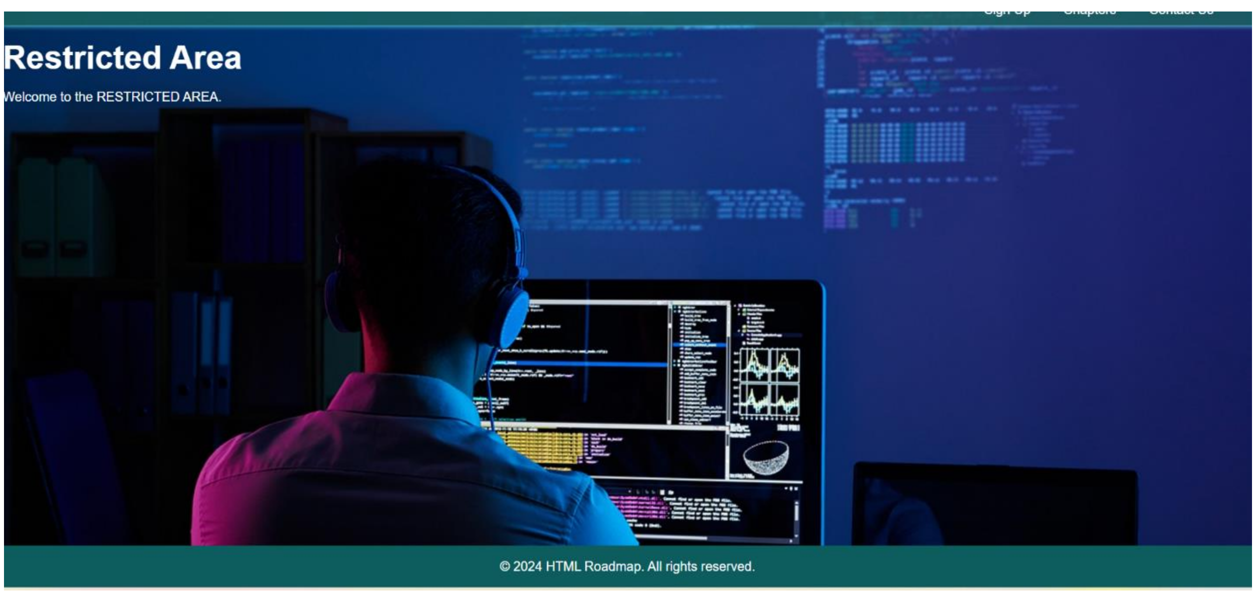


Access Denied

You must be at least 18 years old to access this content.

© 2024 HTML Roadmap. All rights reserved.

Restricted Area Page



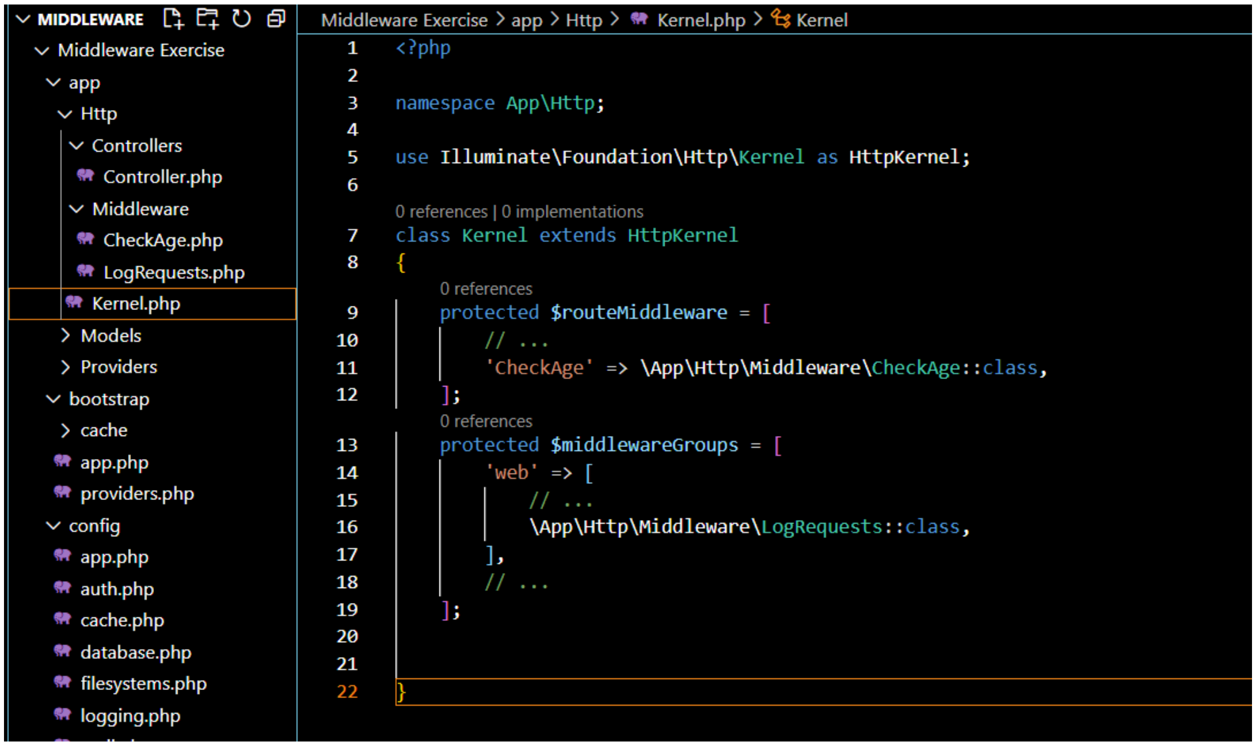
Restricted Area

Welcome to the RESTRICTED AREA.

© 2024 HTML Roadmap. All rights reserved.

Registration and use of global middleware

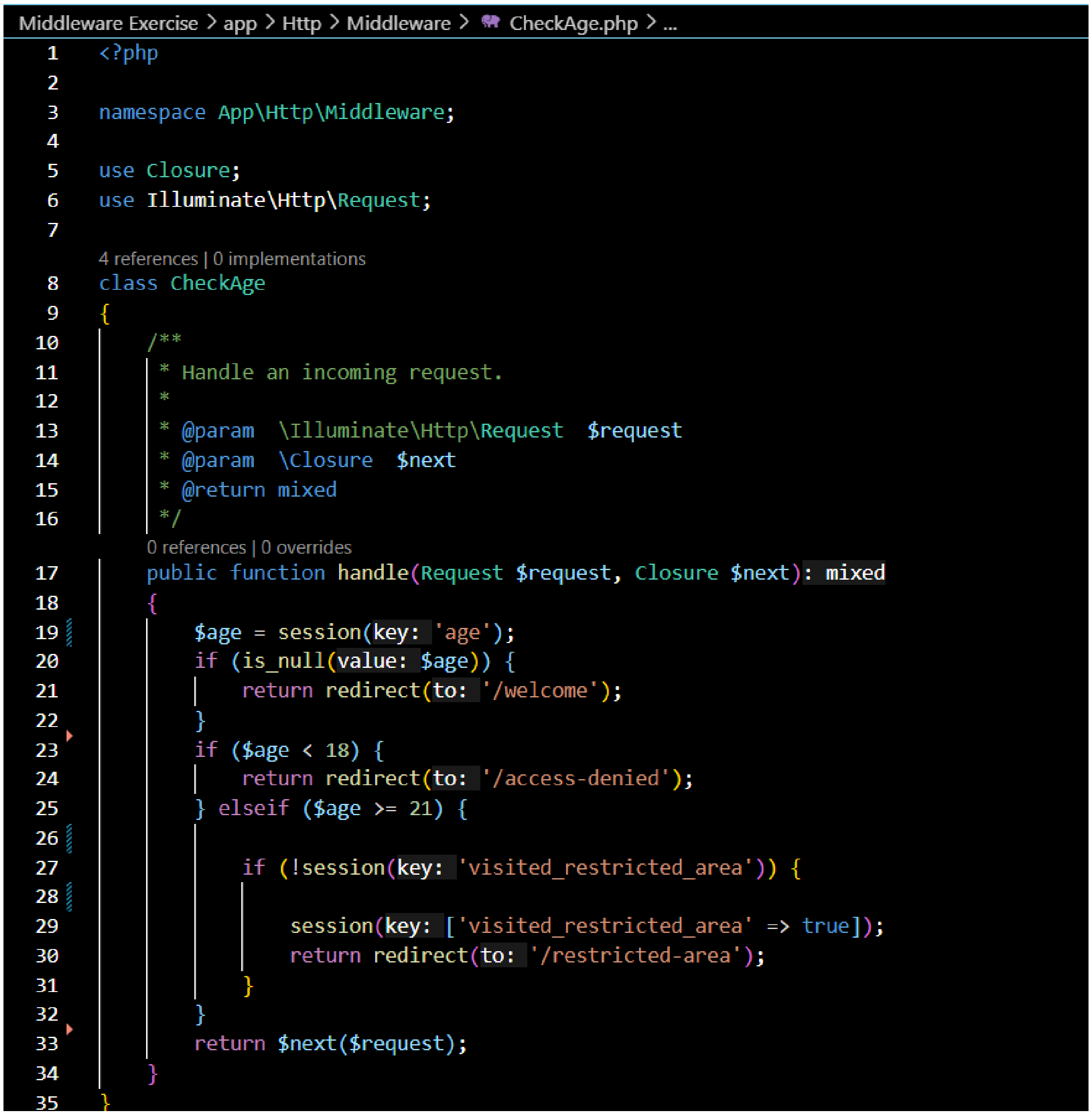
In the app/Http/Kernel.php file



```php
<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

0 references | 0 implementations
class Kernel extends HttpKernel
{
    0 references
    protected $routeMiddleware = [
        // ...
        'CheckAge' => \App\Http\Middleware\CheckAge::class,
    ];
    0 references
    protected $middlewareGroups = [
        'web' => [
            // ...
            \App\Http\Middleware\LogRequests::class,
        ],
        // ...
    ];


}
```

- **GlobalMiddleware(LogRequests):**
  - This middleware logs every request made to the application, tracking user activity and diagnose issues.
  - The LogRequests middleware is added under the 'web' group. To ensure that the middleware is globally applied to all routes under the 'web' middleware group, logging every incoming request made to the application Route-Specific.
- **Route-Specific Middleware(CheckAge):**
  - The CheckAge middleware is registered in the routeMiddleware array. This allows it to be applied to specific routes by using the 'CheckAge' key. This middleware controls access to specific routes based on the user's age.
  - If the user inputs an age that didn’t met the age requirement which is 18, the user is redirected to the access-denied page and can’t access the remaining page.

Middleware Parameters


Middleware Exercise > app > Http > Middleware > CheckAge.php > ...

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

4 references | 0 implementations
class CheckAge
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    0 references | 0 overrides
    public function handle(Request $request, Closure $next): mixed
    {
        $age = session(key: 'age');
        if (is_null(value: $age)) {
            return redirect(to: '/welcome');
        }
        if ($age < 18) {
            return redirect(to: '/access-denied');
        } elseif ($age >= 21) {

            if (!session(key: 'visited_restricted_area')) {

                session(key: ['visited_restricted_area' => true]);
                return redirect(to: '/restricted-area');
            }
        }
        return $next($request);
    }
}
```

- In this middleware CheckAge, the parameters are crucial for processing and managing HTTP requests. The handle method receives two key parameters: **Request $request**, which represents the incoming HTTP request, and **Closure $next**, a function that passes the request to the next middleware.
- **Request $request** holds the incoming request data, like user details and session info.
- **Closure $next** is a function that moves the request to the next step in the app if all conditions are met.
- The middleware first checks if the user's age is saved in the session. If no age is found, it sends them to a /welcome page.

- If the user is under 18, they are sent to an "access denied" page. If they are 21 or older, it checks if they’ve visited a restricted area before, then tracks this visit in the session.
- The middleware either redirects the user based on these checks or allows them to continue to the requested page.

Terminable Middleware

- A terminable middleware in Laravel is a type of middleware that performs additional actions after the HTTP response has been sent to the user's browser. While most middleware handles requests before they reach the controller or during the request, terminable middleware runs after the response is returned, making it unique.

Purpose of Terminable Middleware:

The main purpose of a terminable middleware is to handle tasks that can or should happen **after** the response has been delivered to the user. This could include actions like:

1. **Logging**: Saving logs of the request and response.
2. **Cleaning up resources**: Closing database connections or clearing caches.
3. **Queueing background jobs**: Triggering background tasks that don't affect the user's experience.
4. **Modifying response data**: In some cases, you may adjust headers or other response data right before it's finalized.

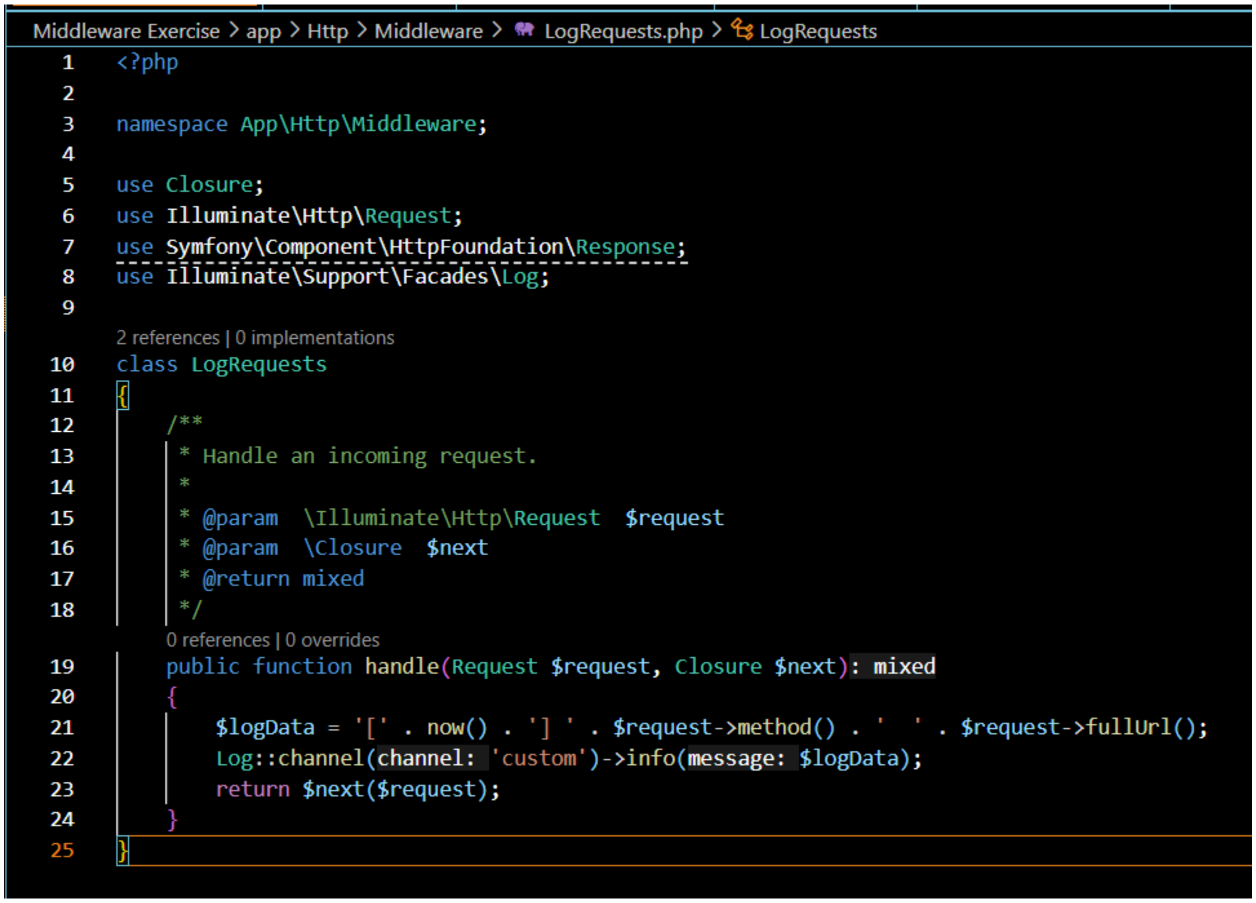
How It Works:

- A middleware becomes "terminable" by implementing the terminate method, in addition to the handle method. Laravel automatically calls this method after the response is sent.

Importance:

1. **Efficiency**: Since the user doesn't have to wait for post-processing tasks, it improves user experience by reducing the time it takes to get a response.
2. **Separation of concerns**: Terminable middleware allows you to separate pre-response actions from post-response actions, keeping your code cleaner and more organized.
3. **Background tasks**: It lets you handle heavy tasks (like sending emails, logging, or processing data) without delaying the user's response.

In our web page, we don't have the terminable middleware but our LogRequests middleware can be turned into a terminable middleware. To turn it into a terminable middleware, we would need to add a terminate method that performs actions after the response is sent.


Middleware Exercise > app > Http > Middleware > LogRequests.php > LogRequests
2 references | 0 implementations
0 references | 0 overrides

```php
<?php

namespace App\Http\Middleware;

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

class LogRequests
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle(Request $request, Closure $next): mixed
    {
        $logData = '[' . now() . '] ' . $request->method() . '  ' . $request->fullUrl();
        Log::channel(channel: 'custom')->info(message: $logData);
        return $next($request);
    }
}
```

How to Make This Middleware Terminable:

- By adding a terminate method to log the details after the response has been processed. This could be useful if we want to log both the request and the response, or perform some additional logging after the request cycle is complete.