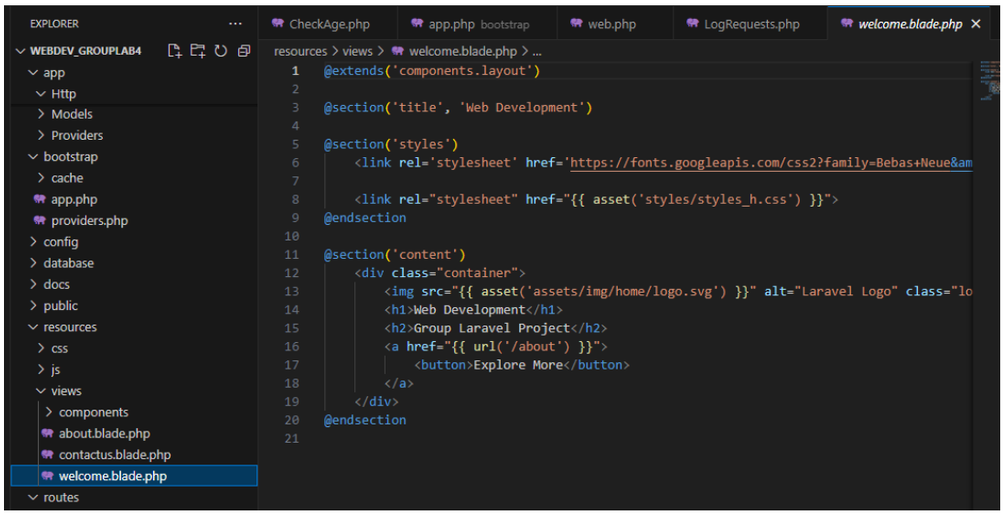
- **Screenshots: Take screenshots of your Blade template files, routing configuration, and the rendered web pages.**

BLADE TEMPLATE FILES

Wecome.blade.php

(only welcome.blade.php was included because the remaining 2 views are just the same and there aren't any changes in there)



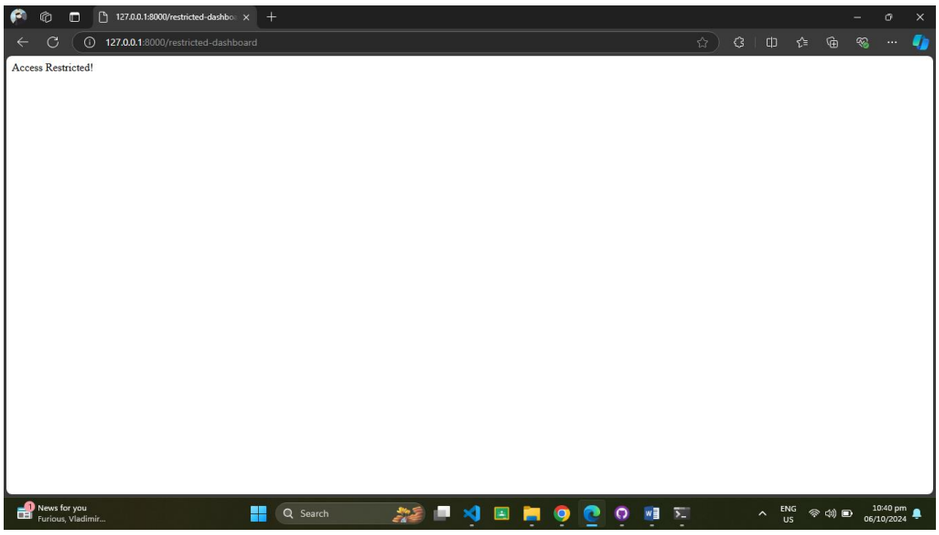
ROUTING CONFIGURATION

1. **Middleware Application**
   - The file uses two middleware: `LogRequests` and `CheckAge`.
   - **LogRequests Middleware** is applied to a group of routes to log HTTP requests when users access these routes.
   - **CheckAge Middleware** is applied with a parameter of `18` to check the user's age, restricting access to certain routes for users under 18.

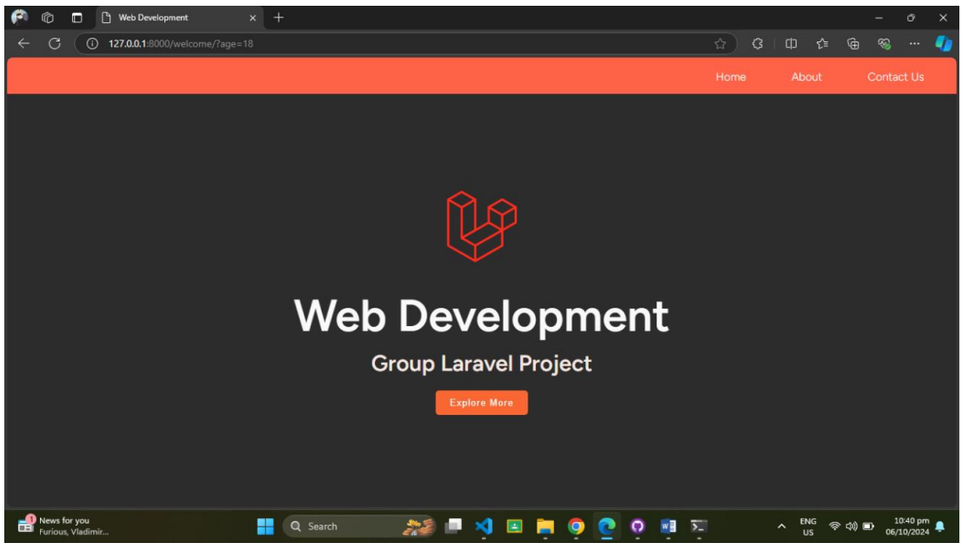
2. **Routes Grouped with LogRequests Middleware**
   - Routes within this group are:
   - `'/'`: Displays the `welcome` view (home page).
   - `/contactus`: Displays the `contactus` view (contact page).
   - `/about`: Displays the `about` view (about page).
   - `/restricted-dashboard`: Returns a message saying "Access Restricted!" indicating that access is restricted.
   - `/access-denied`: Returns a message "Access Denied!" indicating that access is denied, likely for minors.

3. **Routes Grouped with CheckAge Middleware**
   - This group applies age validation where users must be 18 or older to access:
   - `/welcome`: Displays the `welcome` view (likely a different page than the home route).
   - `/dashboard`: Displays the `dashboard` view (likely a restricted dashboard).

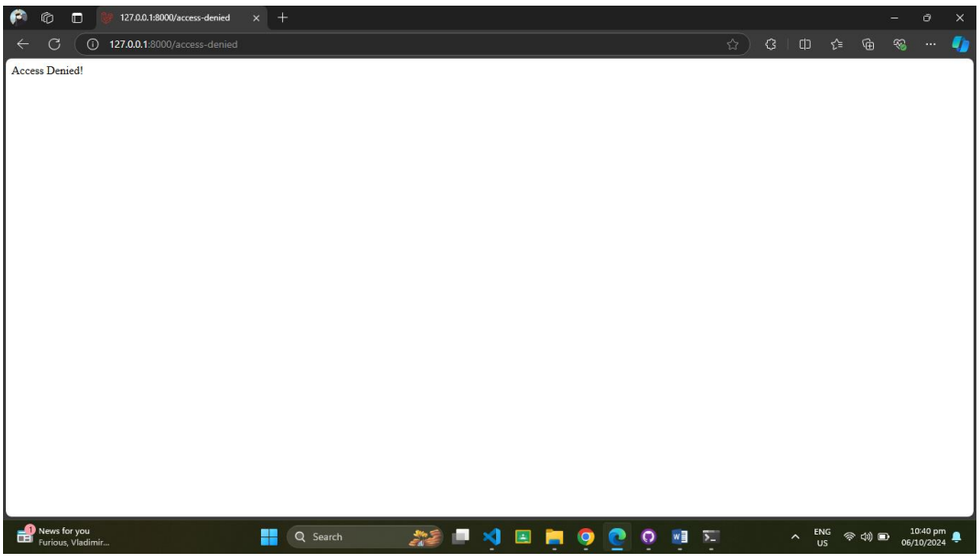
RENDERED WEBPAGES



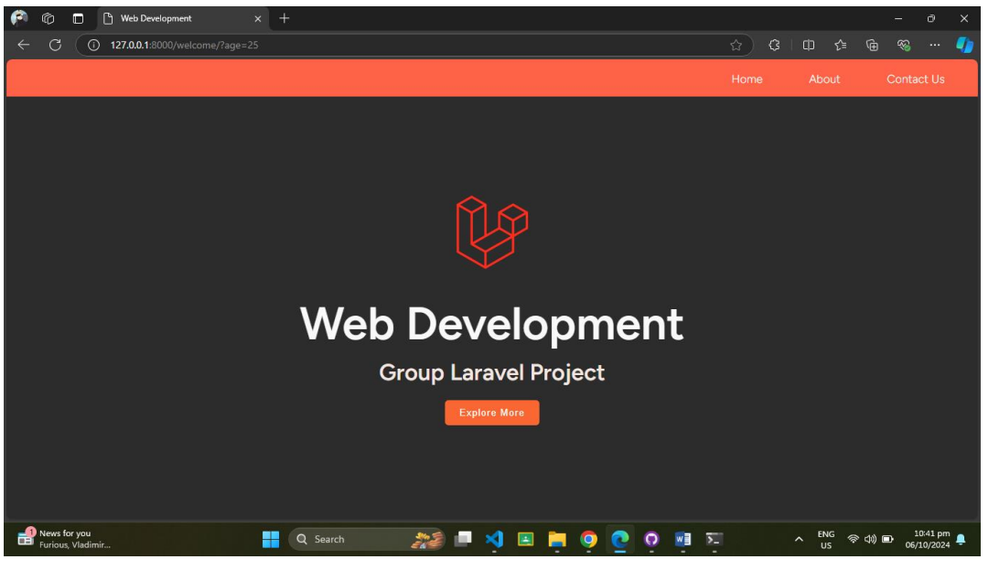
On this page, the age entered was 21. Due to the restrictions applied in the logic, when the user enters an age of 21 in the URL, they will be directed to a restricted dashboard.



On this page, the welcome page was accessed freely because the user met the age requirement.



The page was redirected to "Access Denied" because the user entered an age below 18. The logic dictates that when a user enters an age below 18, access to the page will be denied.



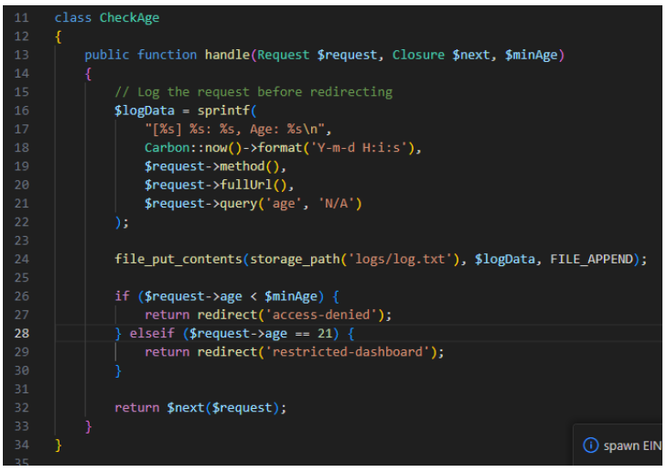
The age entered in the URL was 25, which allowed access to the homepage freely because it was not restricted by the age logic. It was above and not specifically restricted at 21.

- Explanations: Write a detailed explanation of your work:
  - Registration and use of global middleware, middleware parameters, terminable middleware.

Global Middleware:

- The $app->middleware() function registers middleware that will run on every HTTP request in the application.
- CheckAge Middleware (App\Http\Middleware\CheckAge::class) checks the user's age and applies restrictions based on the specified age limit.
- LogRequests Middleware (App\Http\Middleware\LogRequests::class) logs details of every incoming HTTP request.

```
$app->middleware([
    App\Http\Middleware\CheckAge::class,
    App\Http\Middleware\LogRequests::class,
]);
```



```php
class CheckAge
{
    public function handle(Request $request, Closure $next, $minAge)
    {
        // Log the request before redirecting
        $logData = sprintf(
            "[%s] %s: %s, Age: %s\n",
            Carbon::now()->format('Y-m-d H:i:s'),
            $request->method(),
            $request->fullUrl(),
            $request->query('age', 'N/A')
        );

        file_put_contents(storage_path('logs/log.txt'), $logData, FILE_APPEND);

        if ($request->age < $minAge) {
            return redirect('access-denied');
        } elseif ($request->age == 21) {
            return redirect('restricted-dashboard');
        }

        return $next($request);
    }
}
```

In our `CheckAge` middleware, I pass a `$minAge` parameter to validate the user's age.

- If their age is **less than** the specified minimum (e.g., 18), I redirect them to the "access-denied" page.

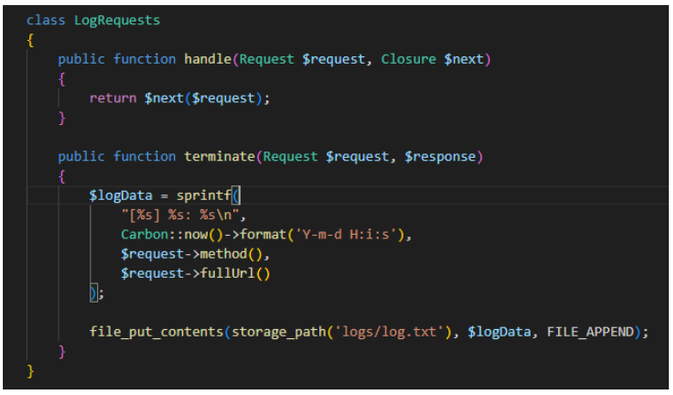
- If their age is **exactly 21**, I send them to the "restricted-dashboard."

- Otherwise, I allow the request to proceed.

Additionally, I log the request details (method, URL, and age) before making these checks.

```php
// Apply CheckAge middleware with a minimum age parameter
Route::middleware([CheckAge::class.':18'])->group(function () {
    Route::get('/welcome', function () {
        return view('welcome');
    })->name('welcome');

    Route::get('/dashboard', function () {
        return view('dashboard');
    })->name('dashboard');
});
```



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

    public function terminate(Request $request, $response)
    {
        $logData = sprintf(
            "[%s] %s: %s\n",
            Carbon::now()->format('Y-m-d H:i:s'),
            $request->method(),
            $request->fullUrl()
        );

        file_put_contents(storage_path('logs/log.txt'), $logData, FILE_APPEND);
    }
}
```

In this `LogRequests` middleware, we used terminable middleware to log requests **after** the response is sent to the client.

How it works:

1. handle() method: This method allows the request to pass through the application by calling `$next($request)` without doing anything before the response is generated.

2. terminate() method*: This method is executed **after** the response has been sent to the client. It logs the request details (timestamp, HTTP method, and full URL) into a log file (`log.txt`).

This is useful for post-response operations like logging, where the request and response cycle is already completed.