Practical 4: Working with Forms and Validation

In this lab, we will explore the concept forms and validations in Laravel. Previously, we learned that data can be passed from a form using "**POST**" method into a HTTP request. With the existing knowledge of form creation and passing of data in HTTP request, let's explore the concept in the following practical session.

1 Forms

Forms are inevitable in a web application and in Laravel, **CSRF** token must be inserted at the initial part of a form to avoid cross-site request.
In the previous lab, forms for creating a new user's record, editing an existing user's record and user's sign up were explored. With the knowledge and understandings, perform the following exercise.

Exercise: As shown in Figure 1, create:

1. A user login form
2. A route to the login form
3. A controller to process the data in login form
4. A route to controller that process the login data

Hint: similar to exercise of user sign up

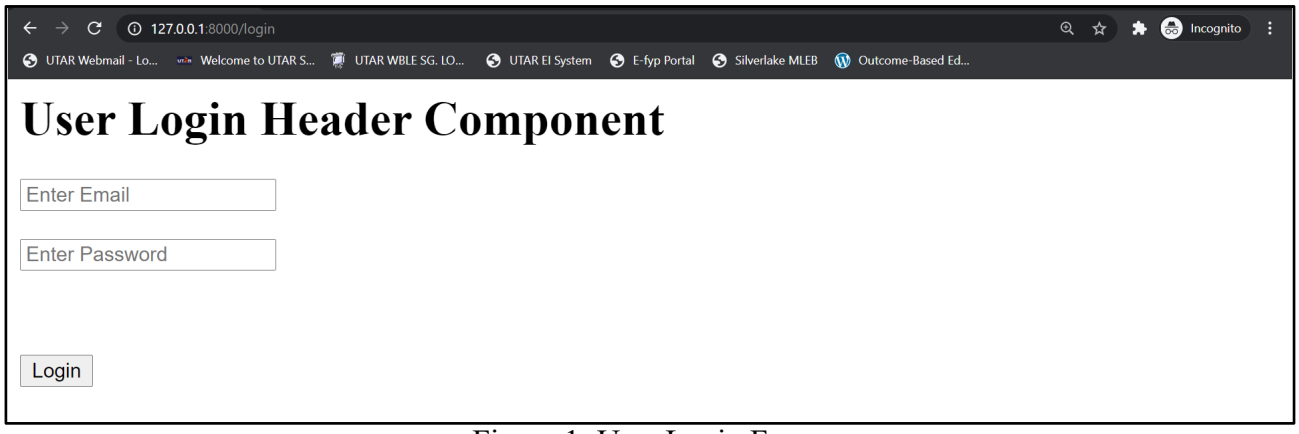


Figure 1: User Login Form.

1.1 Validation: validating form input fields with error messages

It is inevitable that a database table might have columns that are required to be non-null. In order to ensure users do not submit empty input into a non-null database table field, "**validate**" method could be used in processing the input of the form. Validation can be put in backend within controller instead of view, to avoid malicious users from changing the web application scripts. For the purpose of this practical, let's put validation to ensure "**email**" within login form is not empty as shown in Figure 2.

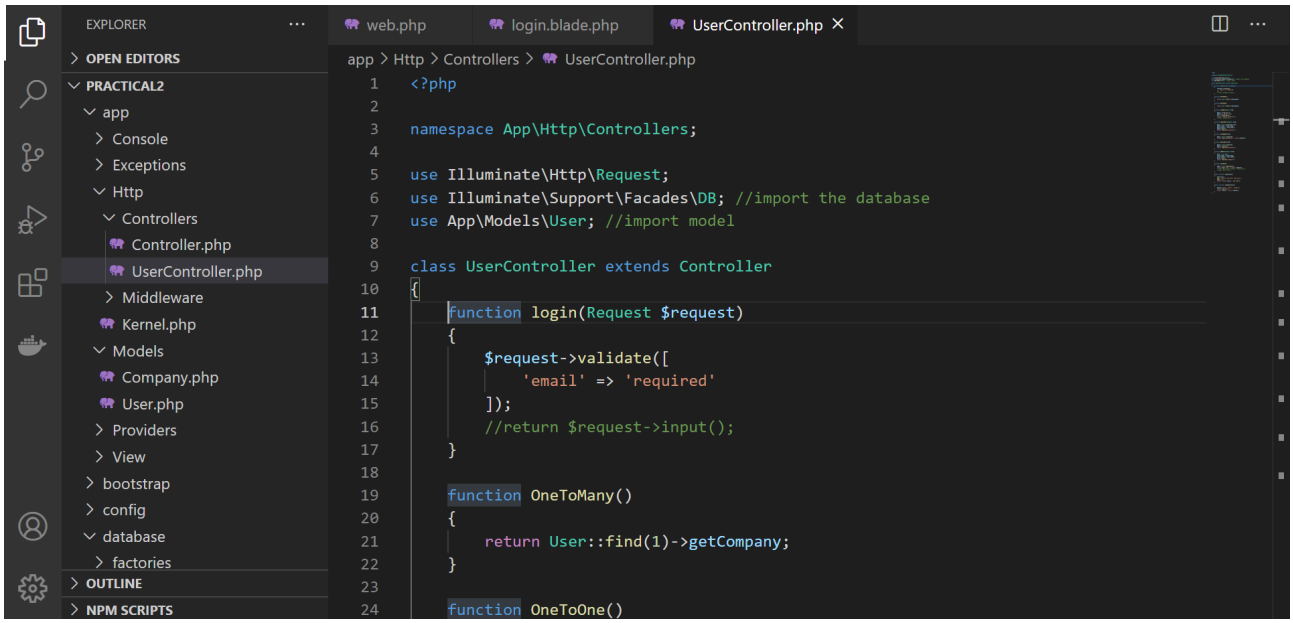


Figure 2: Laravel validate form in controller.

Then, in order to see the error messaged in view, if user insisted to submit with an empty "**email**" input, login view can be inserted with "**errors**" command as shown in Figure 3.

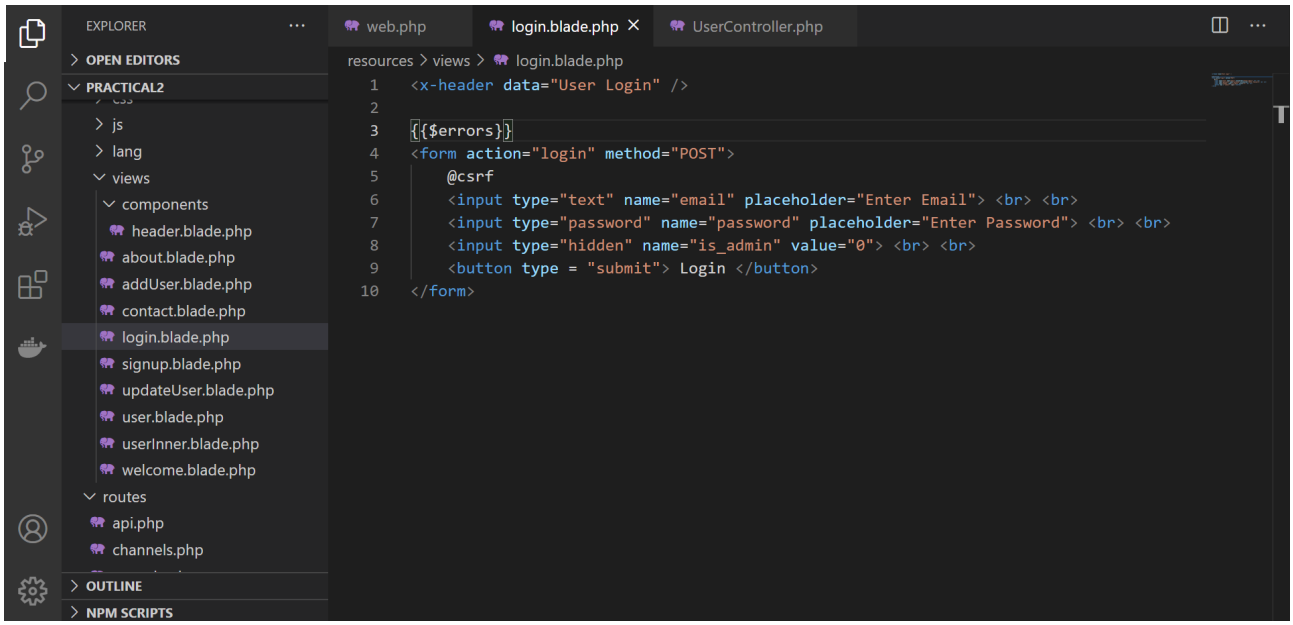


Figure 3: Laravel error messages in view.

The output as the result of validation is as illustrated in Figure 4.

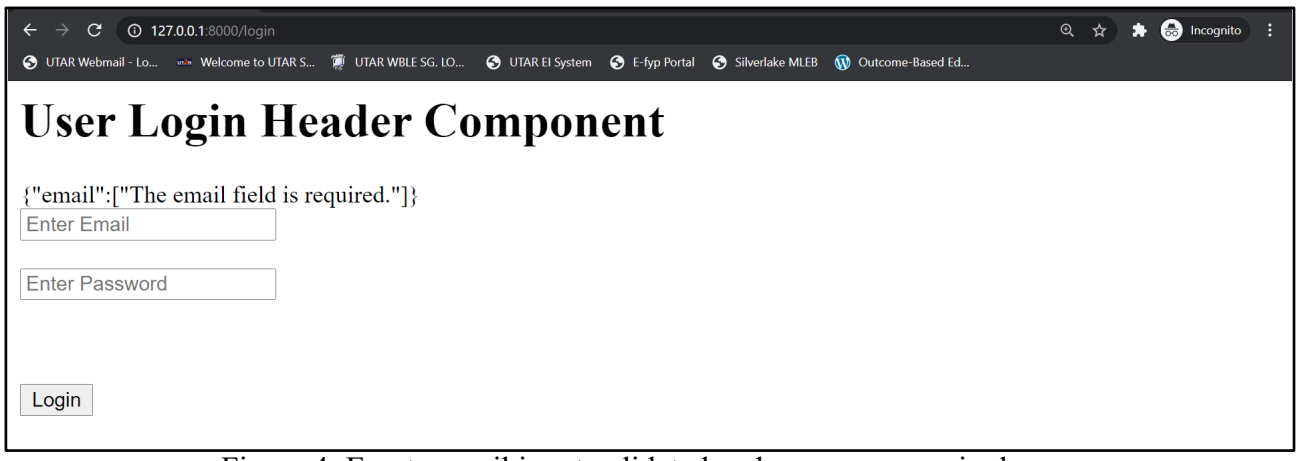


Figure 4: Empty email input validated and error message is shown.

Now let's create validation for password input as well and see whether there will be two errors because of two empty input fields as shown in Figure 5.

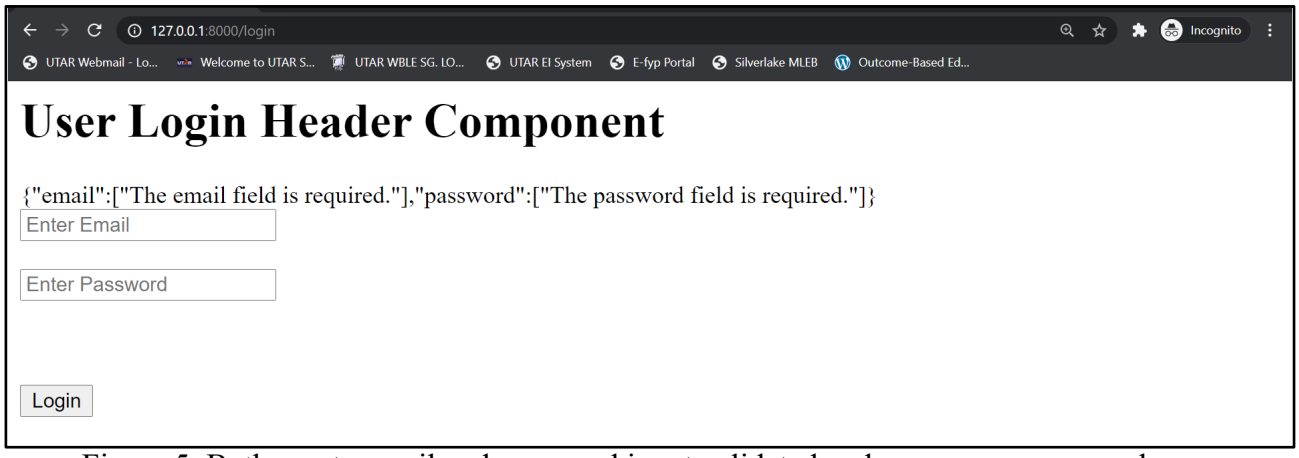


Figure 5: Both empty email and password input validated and error messages are shown.

In order to list out the error messages instead of having the error messages jumbled up, errors parameter in login view can be scripted as shown in Figure 6.

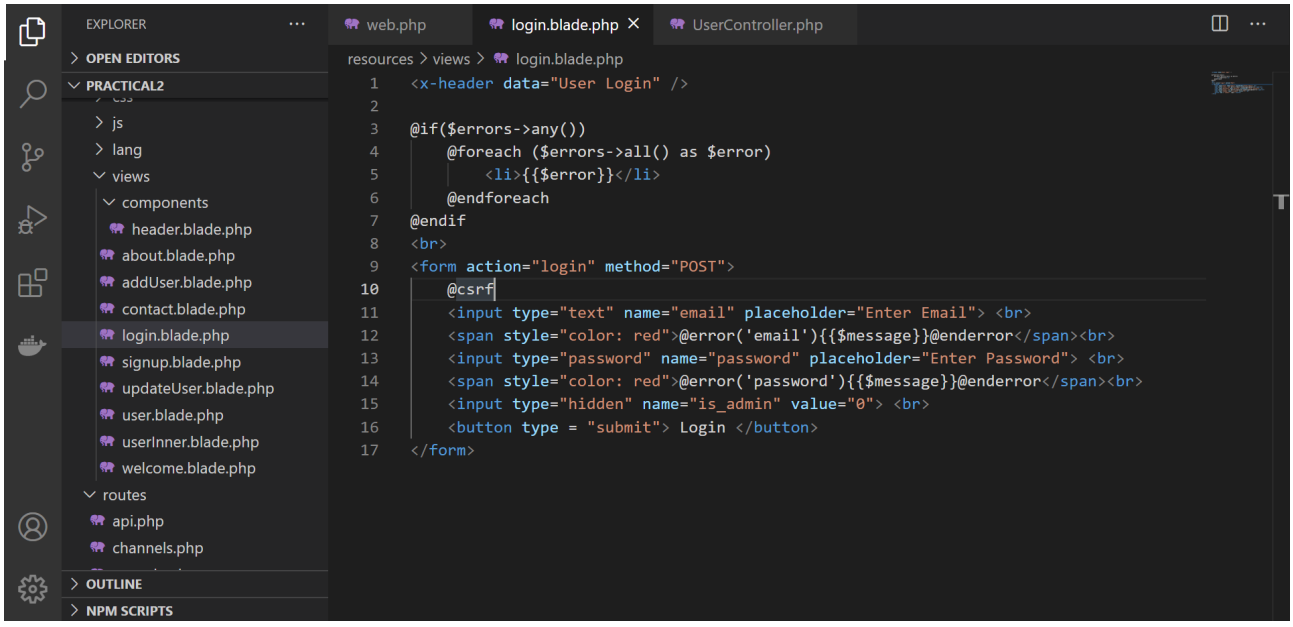


Figure 6: Two ways of formulating error message.

Now let's explore usage of additional requirements to validation of input fields to ensure the users input according to the minimum or maximum character entered as shown in Figure 7.

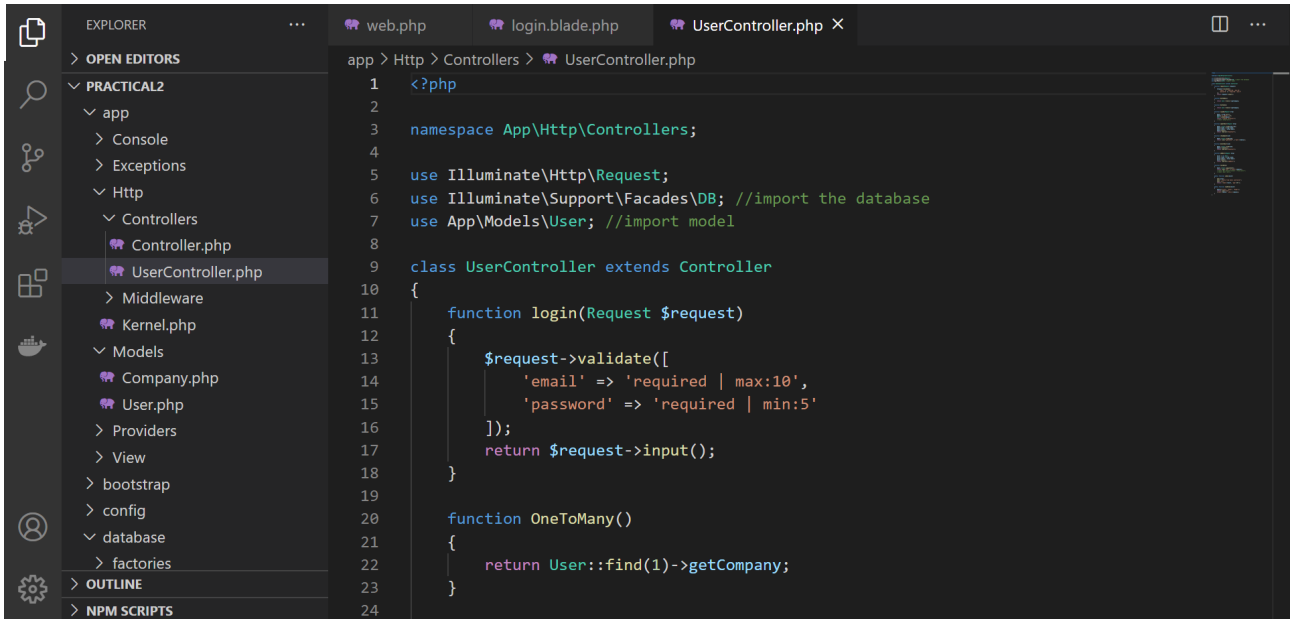


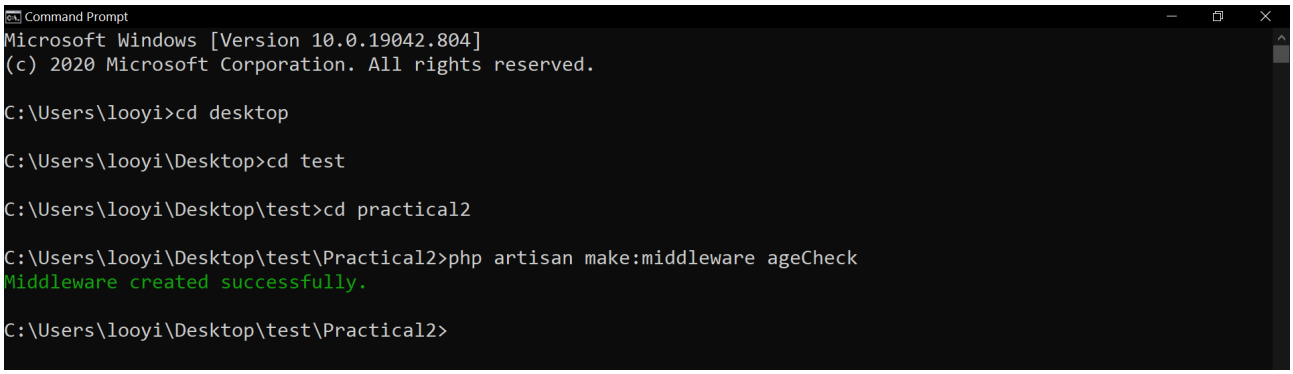
Figure 7: Validating length of inputs.

2 Middleware: filtering user request

Middleware can be used to filter user's request of web application access. For example, a web application might have specific group of users who can access specific pages in the web application; age restrictions, function restrictions, page restrictions, etc. Thus middleware plays an important role to restrict the access of specific group of users. There are three types of middleware; global, group and route middleware. Global middleware is applied to the whole web application, while group middleware is applied only to specific pages and route middleware is applied to a single route at a time. In order to explore the concept, user login that was previously created will be used to assist.

2.1 Global Middleware.

Firstly, create a global middleware using Artisan CLI "**php artisan make:middleware**" in order to explore the concept, as shown in Figure 8.



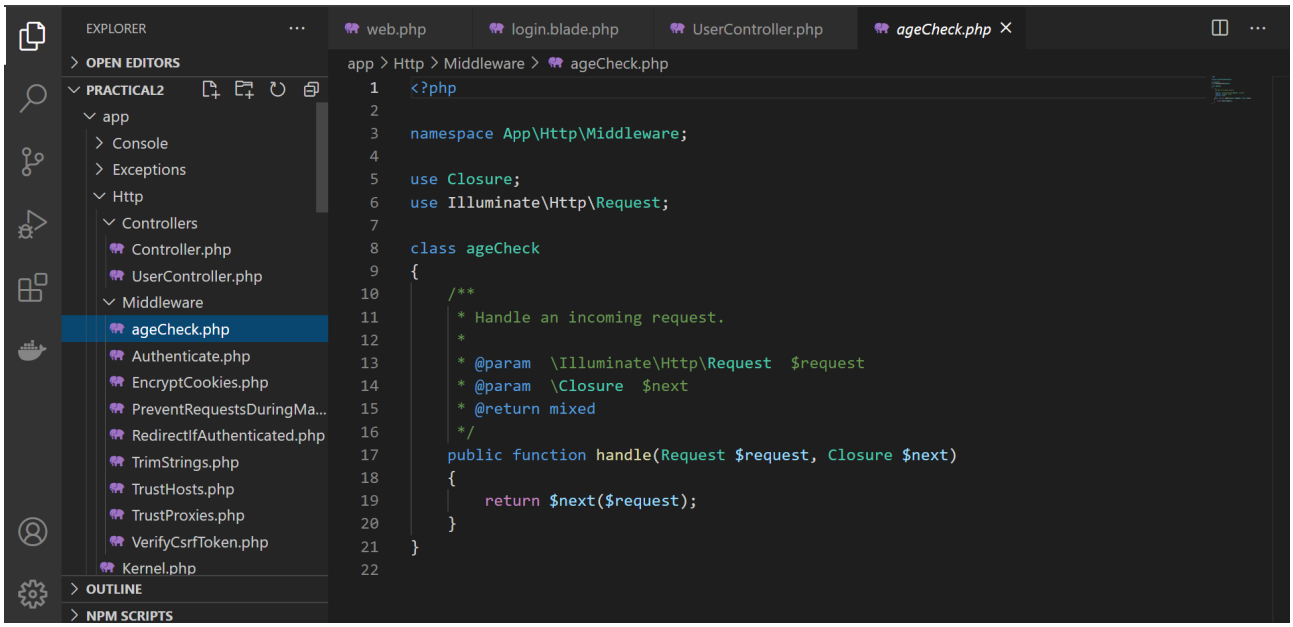


Figure 8: Global ageCheck middleware in Laravel web application.

After creating the middleware, it has to be registered in the application's **kernel** (**app\http\Kernel**) as shown in Figure 9.

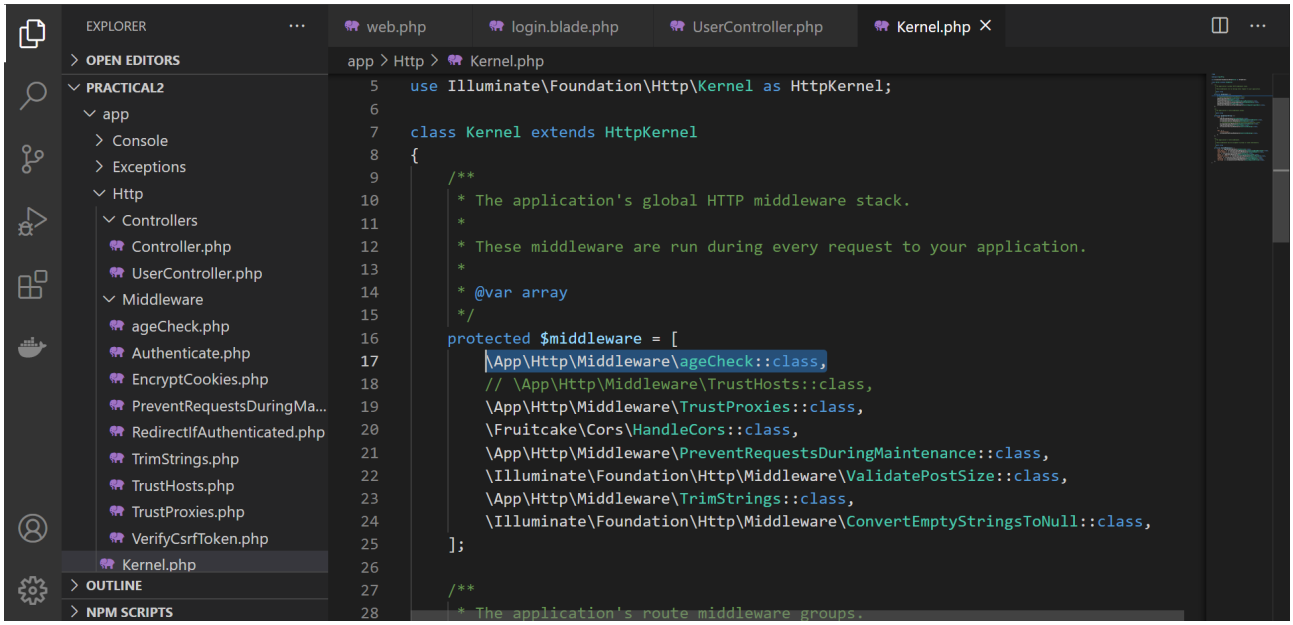


Figure 9: Registering global ageCheck middleware in Laravel web application.

Let's modify the **ageCheck** middleware to check for user's age as shown in Figure 10.

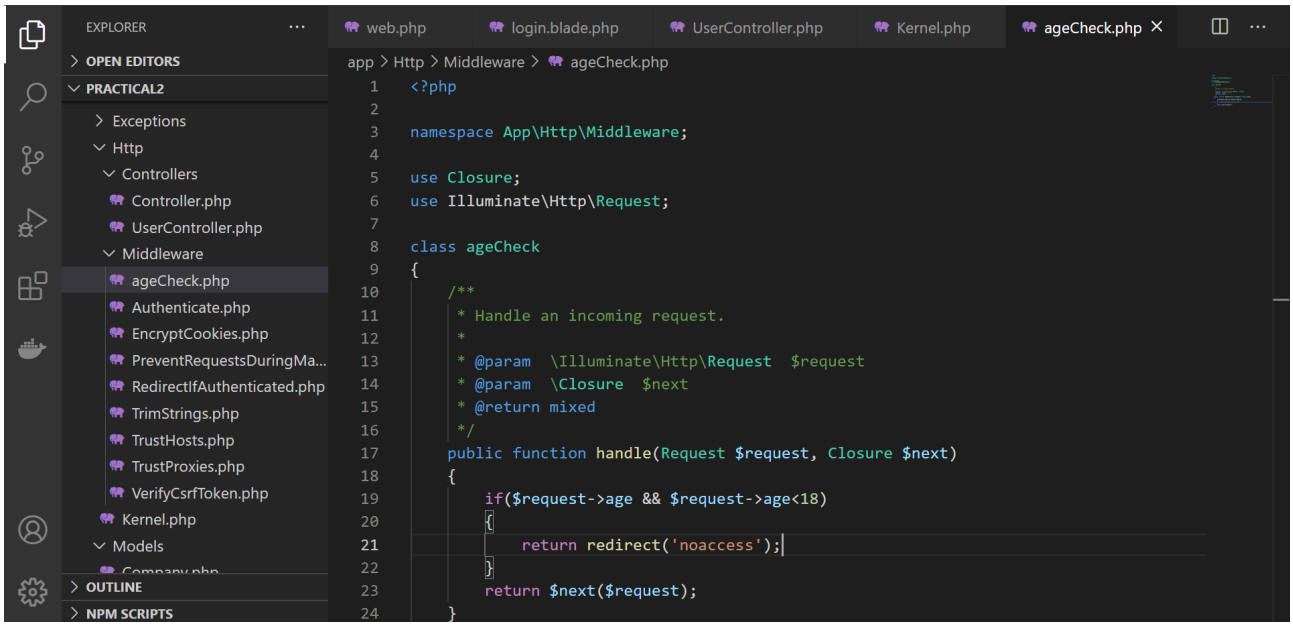


Figure 10: Filter under-aged users from accessing the web application.

Now, create a simple view to notify under-aged users that they are not allowed to access the web application. Create the route to the view so that under-aged users will be redirected to the page. Now, let's check whether the global middleware does work as it should as shown in Figure 11.

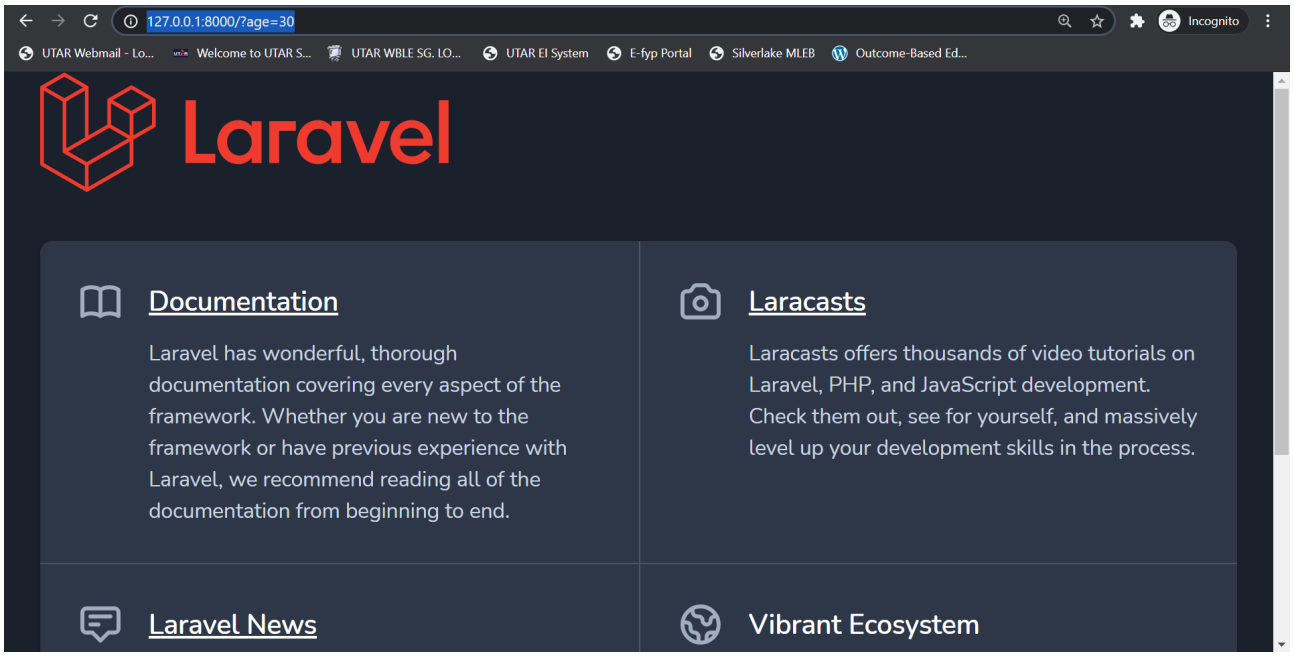


Figure 11: Test out the global ageCheck middleware.

2.2 Group Middleware

Group middleware will enable web developer to apply HTTP request filter on specific group of routes. Let's modify global **ageCheck** middleware into a group middleware by registering the middleware in route middleware group within **app\http\Kernel** file as shown in Figure 12.

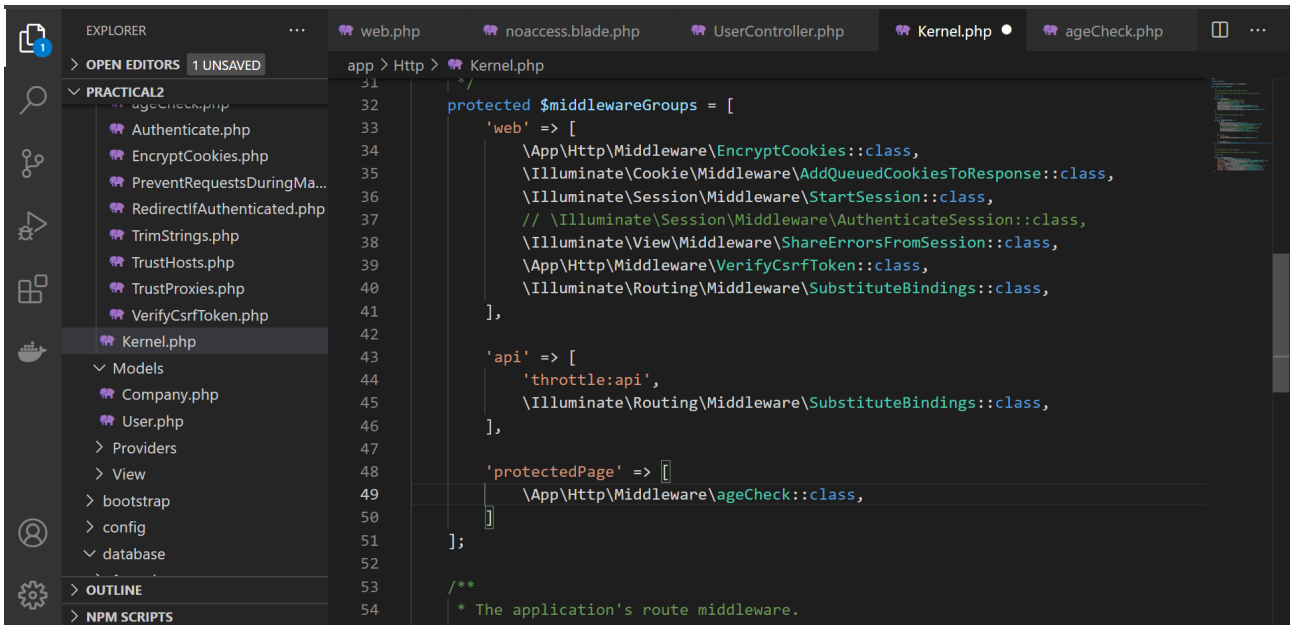


Figure 12: Register group ageCheck middleware.

In order to apply the group middleware in specific pages, declare the group middleware in route as shown in Figure 13.

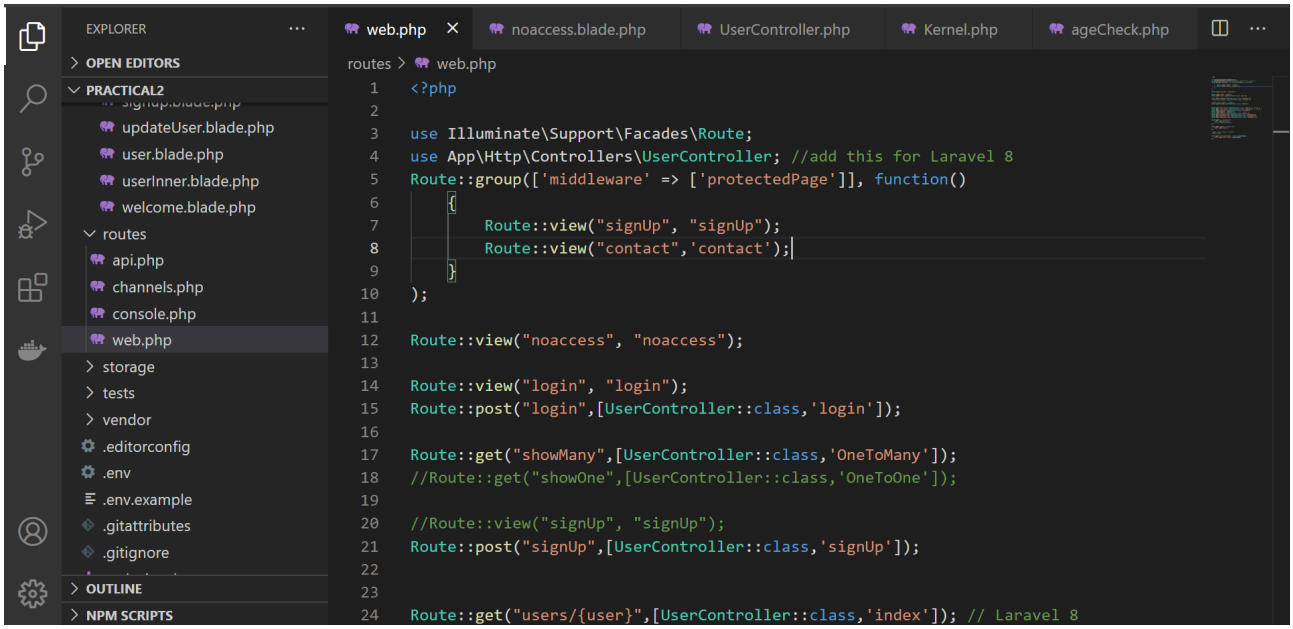


Figure 13: Declare group ageCheck middleware for a group of routes.

Now, let's check whether the group middleware does work as it should as shown in Figure 14.

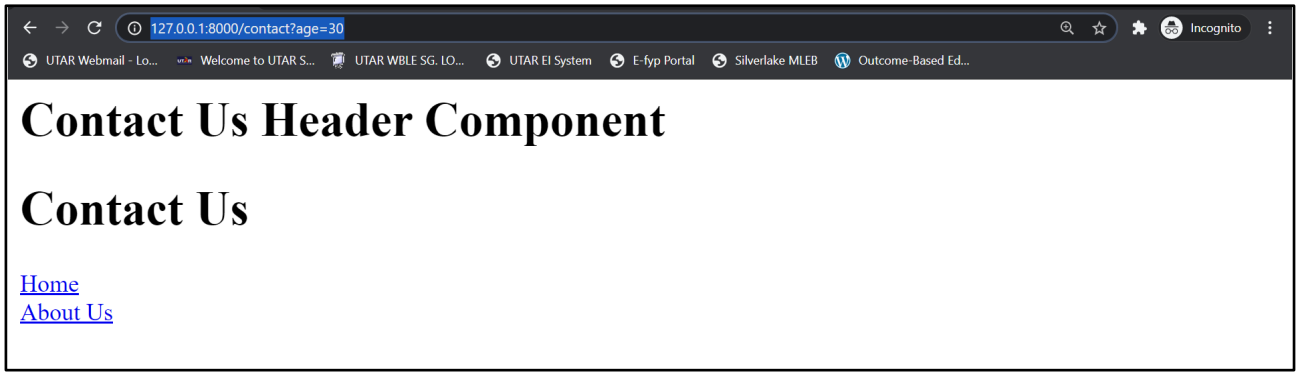


Figure 14: Test out the group ageCheck middleware.

2.3 Route Middleware

Route middleware enables web developer to apply HTTP request filter on specific route. Let's modify group **ageCheck** middleware into a route middleware by registering the middleware in route middleware within **app\http\Kernel** file as shown in Figure 15.

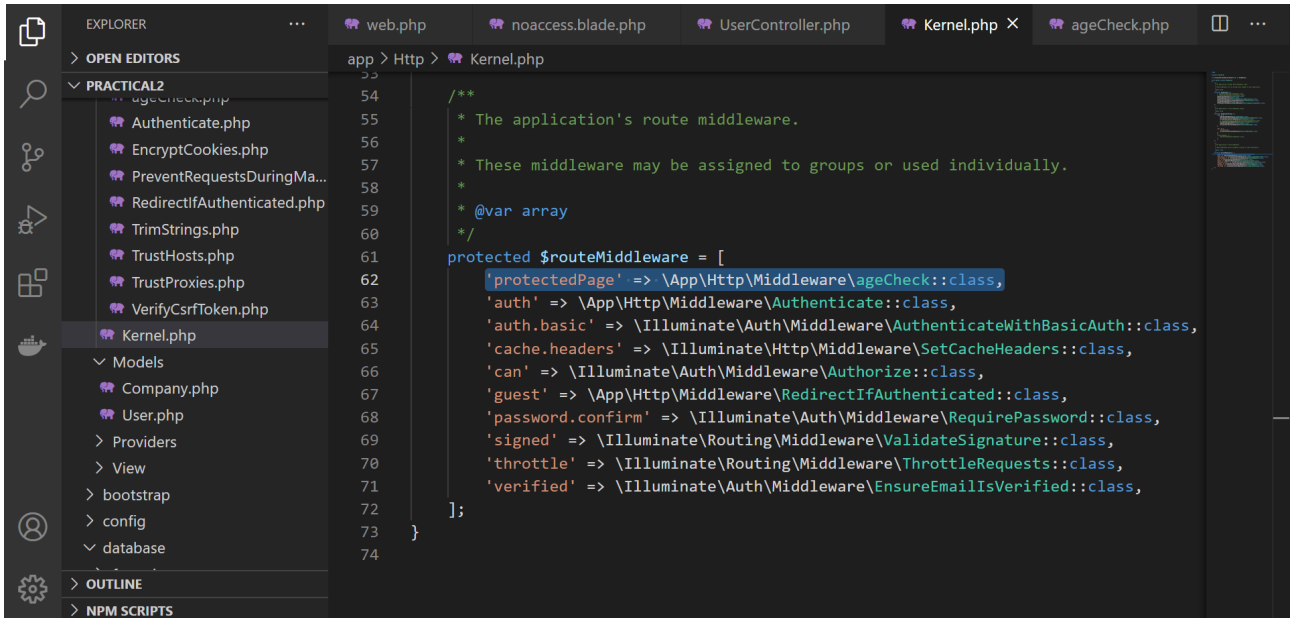


Figure 15: Register route ageCheck middleware.

For instance, if web developer would like to restrict age groups who may access the web application's login page, then the route middleware may be declared in the route as shown in Figure 16.

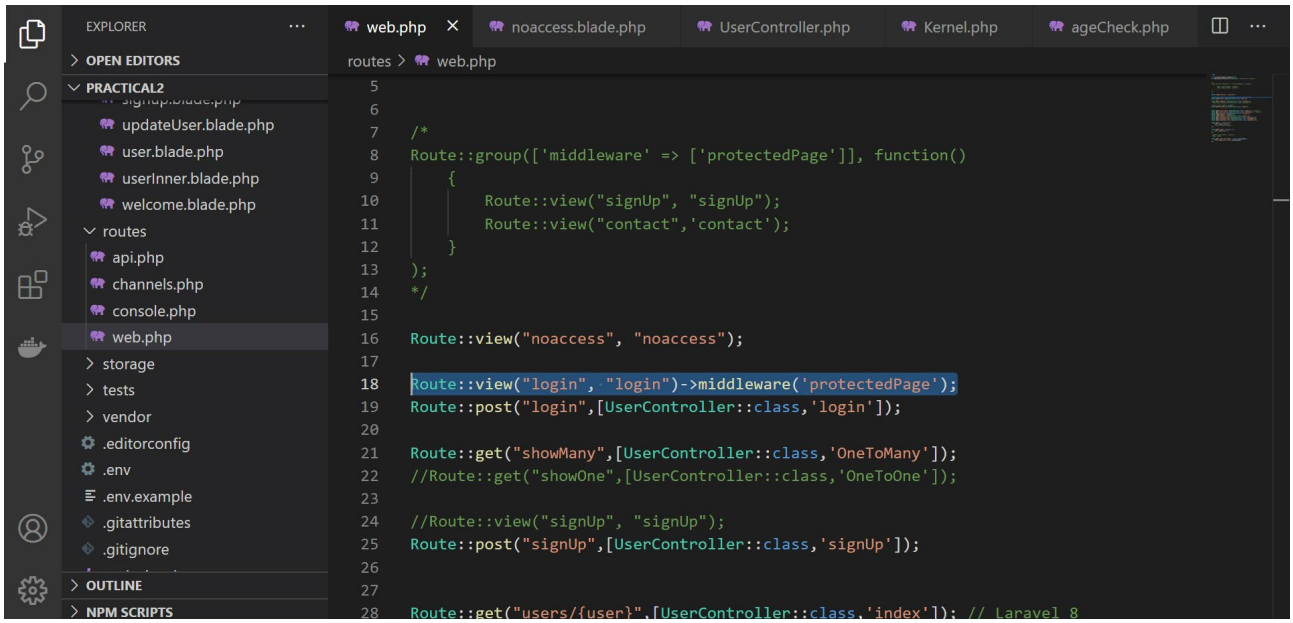


Figure 16: Declare route ageCheck middleware for a specific route.

Now, let's check whether the route middleware does work as it should as shown in Figure 17.

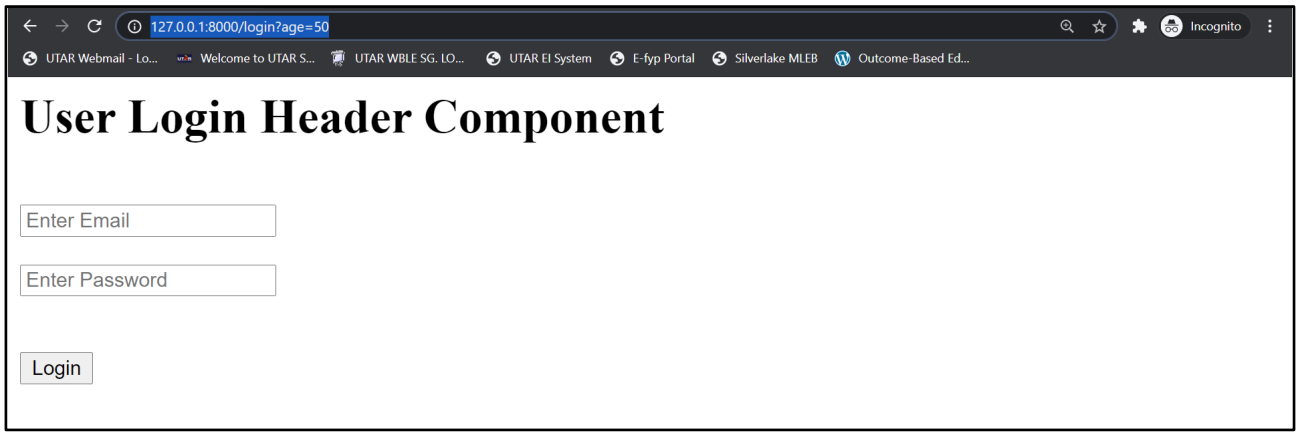


Figure 17: Test out the route ageCheck middleware.