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:

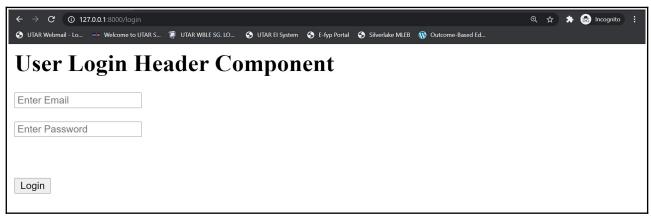


Figure 1: User Login Form.

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

2. 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.

^{**}Hint: similar to exercise of user sign up

```
end.php
                                                                    ♥ UserController.php X
> OPEN EDITORS
∨ PRACTICAL2
                                      namespace App\Http\Controllers;
  > Console
  > Exceptions
                                      use Illuminate\Http\Request;
                                      use Illuminate\Support\Facades\DB; //import the database

✓ Controllers

                                      use App\Models\User; //import model
    Controller.php
   > Middleware
                                           function login(Request $request)
   Kernel.php
                                               $request->validate([
   Company.php
   👫 User.php
  > Providers
  > View
                                           function OneToMany()
  > confia
  ∨ database
                                               return User::find(1)->getCompany;
  > factories
> OUTLINE
> NPM SCRIPTS
                                           function OneToOne()
```

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.

```
♣ login.blade.php ×
♣ UserController.php
                                                                                                                                        □ ...
(L)
     > OPEN EDITORS
                                     resources > views > * login.blade.php
     ∨ PRACTICAL2
                                            {{$errors}}
                                            <form action="login" method="POST">
        > lang
                                                <input type="text" name="email" placeholder="Enter Email"> <br> <br>

✓ components

          neader.blade.php
                                                <input type="hidden" name="is_admin" value="0"> <br>
         💏 about.blade.php
                                                <button type = "submit"> Login </button>
         addUser.blade.php
        contact.blade.php
         🕶 signup.blade.php
         updateUser.blade.php
         💏 user.blade.php
        m userInner.blade.php
         welcome.blade.php
        end.iga 🐄
        e channels.php
     > OUTLINE
      NPM SCRIPTS
```

Figure 3: Laravel error messages in view.

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

← → C ③ 127.0.0.1:8000/login	⊕ ☆ 🖈	lncognito :
🔇 UTAR Webmail - Lo 💀 Welcome to UTAR S 🌹 UTAR WBLE SG. LO 🔇 UTAR EI System 🔇 E-fyp Portal 🔇 Silverlake MLEB 🐧 Outcome-Based Ed		
User Login Header Component		
{"email":["The email field is required."]} Enter Email		
Enter Password		
Login		

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.

← → C ③ 127.0.0.1:8000/login	⊕ ☆ 🖈	lncognito :
🔇 UTAR Webmail - Lo 💀 Welcome to UTAR S 🌹 UTAR WBLE SG. LO 🔇 UTAR EI System 🔇 E-fyp Portal 🔇 Silverlake MLEB 🐧 Outcome-Based Ed		
User Login Header Component		
{"email":["The email field is required."],"password":["The password field is required."]} Enter Email		
Enter Password		
Login		

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.

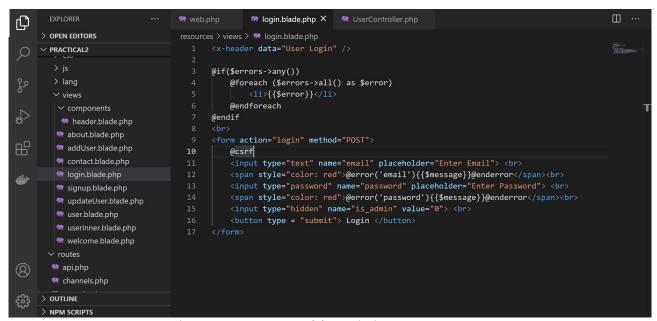


Figure 6: Two ways of formulating error message.

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.

```
♥ UserController.php ×
凸
     > OPEN EDITORS
                                     app > Http > Controllers > ♥ UserController.php
     ∨ PRACTICAL2
       ∨ app
                                           namespace App\Http\Controllers;
        > Exceptions
                                           use Illuminate\Http\Request;
        ∨ Http
                                            use Illuminate\Support\Facades\DB; //import the database

∨ Controllers

                                            use App\Models\User; //import model
         Controller.php
                                           class UserController extends Controller
         > Middleware
                                                function login(Request $request)
        Kernel.php
        ∨ Models
                                                    $request->validate([
                                                       'password' => 'required | min:5'
        💏 User.php
                                                     return $request->input();
        > View
       > config
                                                function OneToMany()

✓ database

        > factories
                                                     return User::find(1)->getCompany;
     > OUTLINE
```

Figure 7: Validating length of inputs.

3. 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.

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.

```
e web php
                                                                                                🦬 ageCheck.php 🗙
> OPEN EDITORS
                                 app > Http > Middleware > 💝 ageCheck.php
               日
日
日
日
日
∨ PRACTICAL2
 ∨ app
  > Exceptions
                                        use Illuminate\Http\Request;
     Controller.php
    W UserController.php

✓ Middleware

    💏 ageCheck.php
    Authenticate.php
                                             * @param \Illuminate\Http\Request $request
* @param \Closure $next
    encryptCookies.php
    PreventRequestsDuringMa...
    RedirectIfAuthenticated.php
                                             public function handle(Request $request, Closure $next)
    TrimStrings.php
     TrustHosts.php
                                                 return $next($request);
    TrustProxies.php
     VerifyCsrfToken.php
   Kernel.php
> OUTLINE
```

Figure 8: Global ageCheck middleware in Laravel web application.

After creating the middleware, it has to be registered in the application's kernel as shown in Figure 9.

```
    ≪ Kernel.php ×

凸
     > OPEN EDITORS
                                       use Illuminate\Foundation\Http\Kernel as HttpKernel:

∨ PRACTICAL2

                                        class Kernel extends HttpKernel
       > Exceptions

✓ Controllers

         Controller.php
                                             * These middleware are run during every request to your application.
         W UserController.php
         ageCheck.php
                                            protected $middleware = [
         Authenticate.php
                                               \App\Http\Middleware\ageCheck::class,
         m EncryptCookies.php
                                                   \App\Http\Middleware\TrustHosts::class,
                                               \App\Http\Middleware\TrustProxies::class,
         PreventRequestsDuringMa...
                                               \Fruitcake\Cors\HandleCors::class,
         RedirectIfAuthenticated.php
                                               \App\Http\Middleware\PreventRequestsDuringMaintenance::class,
                                               \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
         TrustHosts.php
                                                \App\Http\Middleware\TrimStrings::class,
         TrustProxies.php
                                                VerifyCsrfToken.php
        Kernel.php
    > OUTLINE
```

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

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

```
end.php
                                                     😭 login.blade.php
                                                                          # UserController.php
                                                                                                  M Kernel.php
                                                                                                                  * ageCheck.php X
凸
     > OPEN EDITORS
                                     app > Http > Middleware > 🦛 ageCheck.php
     ∨ PRACTICAL2
                                           namespace App\Http\Middleware;
        ∨ Http
                                           use Closure:
          Controller.php
                                           use Illuminate\Http\Request:
         WaterController.php

∨ Middleware

          💏 ageCheck.php
          Authenticate.php
          m EncryptCookies.php
          PreventRequestsDuringMa...
          RedirectIfAuthenticated.php
          TrimStrings.php
          TrustHosts.php
                                                public function handle(Request $request, Closure $next)
          TrustProxies.php
          VerifyCsrfToken.php
                                                     if($request->age && $request->age<18)
         Kernel.php
(8)
                                                         return redirect('noaccess');

∨ Models

     > OUTLINE
                                                     return $next($request);
```

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.

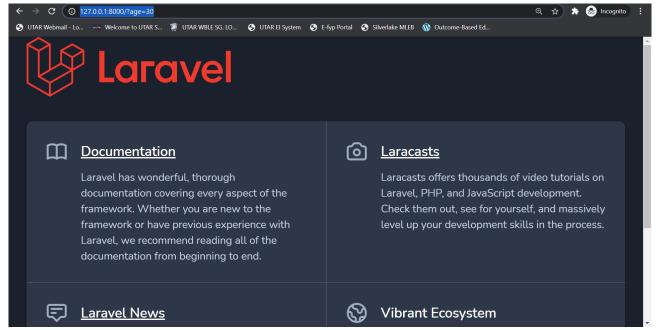


Figure 11: Test out the global ageCheck middleware.

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 HTTP Kernel file as shown in Figure 12.

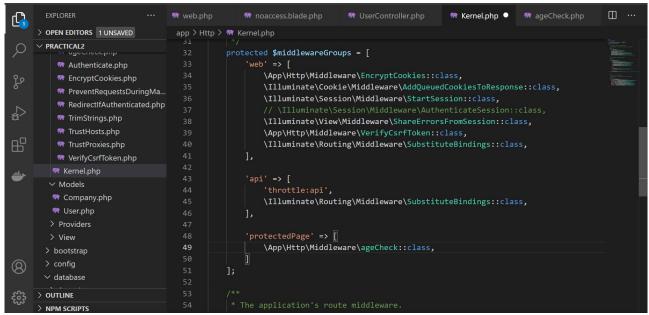


Figure 12: Register group ageCheck middleware.

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

```
W UserController.php
> OPEN EDITORS
                                routes > 💝 web.php

✓ PRACTICAL2

   npdateUser.blade.php
                                      use Illuminate\Support\Facades\Route;
   🕶 user.blade.php
                                      use App\Http\Controllers\UserController; //add this for Laravel 8
                                      Route::group(['middleware' => ['protectedPage']], function()
   userInner.blade.php
   m welcome.blade.php
                                               Route::view("signUp", "signUp");
Route::view("contact",'contact');

✓ routes

  💝 api.php
  en channels.php
  ensole.php
  web.php
                                       Route::view("noaccess", "noaccess");
 > storage
                                      Route::view("login", "login");
Route::post("login",[UserController::class,'login']);
 > vendor
                                      Route::get("showMany",[UserController::class,'OneToMany']);
 .env
 gitattributes
                                      Route::post("signUp",[UserController::class,'signUp']);
  .gitignore
> OUTLINE
                                      Route::get("users/{user}",[UserController::class,'index']);
NPM SCRIPTS
```

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.

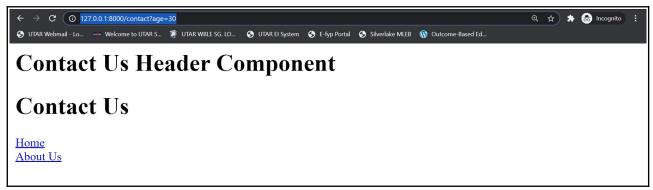


Figure 14: Test out the group ageCheck middleware.

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 HTTP Kernel file as shown in Figure 15.

```
e web.php
                                                                             W UserController.php

    ★ Kernel.php ×  
    ★ ageCheck.php

D
     > OPEN EDITORS
                                     app > Http > 🤫 Kernel.php
     ∨ PRACTICAL2
          Authenticate.php
          m EncryptCookies.php
          PreventRequestsDuringMa...
          RedirectIfAuthenticated.php
          TrimStrings.php
          TrustHosts.php
                                                protected $routeMiddleware = [
                                                    'protectedPage' => \App\Http\Middleware\ageCheck::class,
          TrustProxies.php
                                                     'auth' => \App\Http\Middleware\Authenticate::class,
          ₩ VerifyCsrfToken.php
                                                     'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
         ** Kernel.php
                                                    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,

✓ Models

                                                     'can' => \Illuminate\Auth\Middleware\Authorize::class,
         Company.php
                                                     'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
        👭 User.php
                                                     'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
                                                     'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
                                                     'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
        > View
                                                     'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
       > bootstrap
       > config

✓ database

     > OUTLINE
```

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.

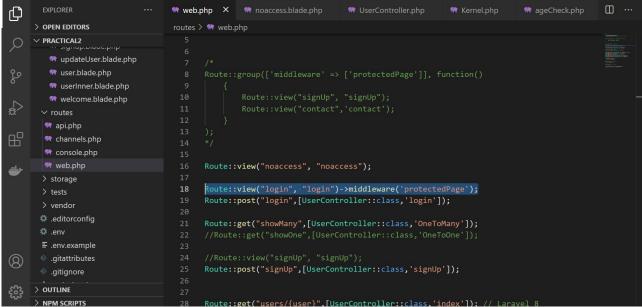


Figure 16: Declare route ageCheck middleware for a specific route.

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

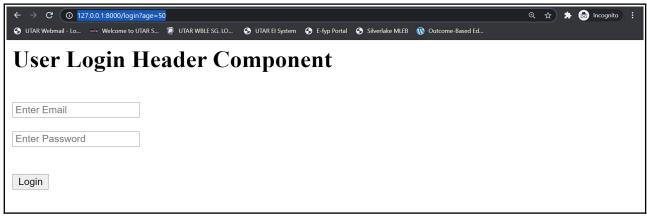


Figure 17: Test out the route ageCheck middleware.