

Laravel

Урок 8. Посредники. Сессии в Laravel. Аутентификация

Авторизация и идентификация пользователей.
Разграничение прав доступа к информации.

Оглавление

[Теория](#)

[Аутентификация и авторизация](#)

[Посредники \(middleware\) Laravel](#)

[Сессии в Laravel](#)

[Практика](#)

[Установка Laravel UI \(пользовательского интерфейса\)](#)

[Добавление таблиц для сохранения данных пользователей](#)

[Закрываем доступ к админке сайта](#)

[Разграничение прав пользователей](#)

[Практическое задание](#)

[Дополнительные материалы](#)

[Используемая литература](#)

Теория

Аутентификация и авторизация

Аутентификация — это проверка введенных логина и пароля, а также данных зарегистрированных пользователей в системе. В Laravel предусмотрена своя система аутентификации, которая работает практически из основной сборки. В этом уроке рассмотрим версию фреймворка 6.0.

Авторизация — предоставление лицу или группе лиц прав на выполнение определенных действий, а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

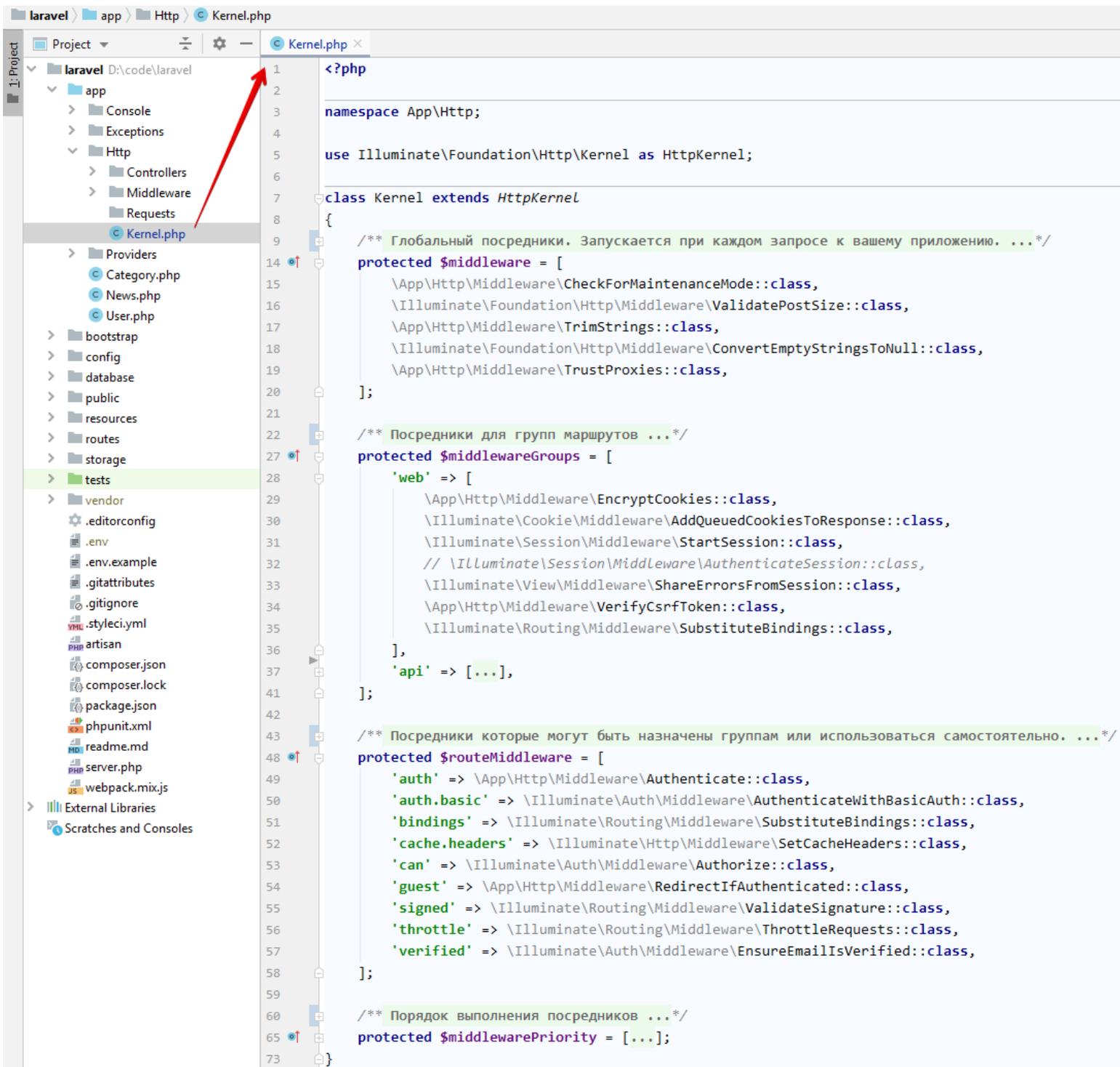
Посредники (middleware) Laravel

Посредники (middleware) — специальные классы Laravel, предназначенные для выполнения пред- и постобработки запроса пользователя. Прежде чем достигнуть контроллера, HTTP-запрос может пройти через ряд посредников. Посредники запускаются по цепочке. Это удобный механизм для фильтрации http-запросов. Посредник может пропустить запрос или в какой-то момент перенаправить пользователя на страницу регистрации.

Для создания посредника можно воспользоваться командой **php artisan make:middleware AnyName**. Она создаст класс **AnyName** в папке **app/Http/Middleware**.

Для использования посредника его необходимо добавить в определенное свойство класса **App\Http\Kernel**.

- **middleware** — свойство содержит глобальные посредники. Они запускаются при каждом запросе к вашему приложению.
- **middlewareGroups** — свойство содержит наборы посредников для групп маршрутов.
- **routeMiddleware** — свойство содержит посредники, которые могут быть назначены группам или использоваться самостоятельно.
- **middlewarePriority** — свойство определяет порядок выполнения посредников.



```
1 <?php
2
3 namespace App\Http;
4
5 use Illuminate\Foundation\Http\Kernel as HttpKernel;
6
7 class Kernel extends HttpKernel
8 {
9     /**
10      * Глобальный посредники. Запускается при каждом запросе к вашему приложению. ...*/
11     protected $middleware = [
12         \App\Http\Middleware\CheckForMaintenanceMode::class,
13         \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
14         \App\Http\Middleware\TrimStrings::class,
15         \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
16         \App\Http\Middleware\TrustProxies::class,
17     ];
18
19     /**
20      * Посредники для групп маршрутов ...*/
21     protected $middlewareGroups = [
22         'web' => [
23             \App\Http\Middleware\EncryptCookies::class,
24             \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
25             \Illuminate\Session\Middleware\StartSession::class,
26             // \Illuminate\Session\Middleware\AuthenticateSession::class,
27             \Illuminate\View\Middleware\ShareErrorsFromSession::class,
28             \App\Http\Middleware\VerifyCsrfToken::class,
29             \Illuminate\Routing\Middleware\SubstituteBindings::class,
30         ],
31         'api' => [...],
32     ];
33
34     /**
35      * Посредники которые могут быть назначены группам или использоваться самостоятельно. ...*/
36     protected $routeMiddleware = [
37         'auth' => \App\Http\Middleware\Authenticate::class,
38         'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
39         'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
40         'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
41         'can' => \Illuminate\Auth\Middleware\Authorize::class,
42         'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
43         'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
44         'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
45         'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
46     ];
47
48     /**
49      * Порядок выполнения посредников ...*/
50     protected $middlewarePriority = [...];
51 }
52 }
```

К примеру, если название класса будет добавлено как элемент массива **web** свойства **middlewareGroups**, то он будет выполняться для всех маршрутов, указанных в роутах файла **web.php**. Если созданный класс посредника должен применяться только к определенному классу контроллера или маршруту, его следует указать в свойстве **routeMiddleware** в качестве значения массива, а в качестве ключа — придумать имя, по которому данный посредник будет вызываться.

Указание вызова самостоятельного посредника может выполняться или в самом классе контроллера, в методе конструктора или в описании маршрута.

```
web.php
1 <?php
2
3 Route::get('/', [
4     'uses' => 'HomeController@index',
5     'as' => 'home',
6     "middleware" => "anyName"
7 ]);
8
9
10

HomeController.php
1 <?php
2 namespace App\Http\Controllers;
3
4 use Illuminate\Http\Request;
5
6 class HomeController extends Controller
7
8
9
10
11 public function __construct()
12 {
13     $this->middleware( middleware: 'anyName');
14 }
15

Kernel.php
41 ];
42
43 /** Посредники которые могут быть назначены группам или использоваться самостоятельно. ... */
44 protected $routeMiddleware = [
45     'auth' => \App\Http\Middleware\Authenticate::class,
46     'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
47     'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
48     'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
49     'can' => \Illuminate\Auth\Middleware\Authorize::class,
50     'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
51     'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
52     'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
53     'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
54     'anyName' => \App\Http\Middleware\AnyName::class,
55 ];
```

Подробнее о посредниках можно прочитать в официальной документации или в русскоязычном переводе:

- <https://laravel.com/docs/6.x/middleware>;
- <http://laravel.su/docs/5.4/middleware>.

Сессии в Laravel

Присмотримся к классу `App\Http\Kernel`, а именно к свойству `middlewareGroups`. Одним из классов-посредников, используемых при выполнении маршрутов `web.php`, является класс `Illuminate\Session\Middleware\StartSession`.

```
namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel
{
    /** Глобальный посредники. Запускается при каждом запросе к вашему приложению. ...*/
    protected $middleware = [...];

    /** Посредники для групп маршрутов ...*/
    protected $middlewareGroups = [
        'web' => [
            \App\Http\Middleware\EncryptCookies::class,
            \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
            \Illuminate\Session\Middleware\StartSession::class,
            // \Illuminate\Session\Middleware\AuthenticateSession::class,
            \Illuminate\View\Middleware\ShareErrorsFromSession::class,
            \App\Http\Middleware\VerifyCsrfToken::class,
            \Illuminate\Routing\Middleware\SubstituteBindings::class,
        ],

        'api' => [
            'throttle:60,1',
            'bindings',
        ],
    ];
};
```

Что говорит название данного класса? Что он создает сессию.

Для настроек работы с сессией используется файл **session.php** в папке **config**. Данный файл возвращает массив, в значениях которого указываются параметры работы сессии. В основном все значения вычисляются проверкой существования их в файле конфигурации приложения (**.env**), и в случае отсутствия подставляется значение по умолчанию. Это вычисление происходит благодаря хелперу **env()**.

```
use Illuminate\Support\Str;

return [
    /*...*/

    'driver' => env( key: 'SESSION_DRIVER', default: 'file'),

    /*...*/

    'lifetime' => env( key: 'SESSION_LIFETIME', default: 120),

    'expire_on_close' => false,

    /*...*/

    'encrypt' => false,

    /*...*/

    'files' => storage_path( path: 'framework/sessions'),

    /*...*/

    'connection' => env( key: 'SESSION_CONNECTION', default: null),
];
```

Подробнее о конфигурации данного файла можно прочитать в документации фреймворка по адресу <https://laravel.com/docs/6.x/session> (<https://laravel.ru/docs/v5/session>).

Рассмотрим наиболее популярные варианты установки данных в сессию, их получения и очистки. Первый — с помощью фасада `Illuminate\Support\Facades\Session`, и второй — с помощью хелпера `session`. Этот фасад и хелпер имеют методы `get`, `put`, `all` и `flash`. В случае с фасадом эти методы являются статическими, а с хелпером — динамическими, но работают одинаково.

Рассмотрим пример:

The image shows a code editor with the following PHP code for `HomeController`:

```
class HomeController extends Controller
{
    public function index()
    {
        session()->put(['key1.key2' => 'Values!']);
        session()->put(['key1.key3' => 'Values!!']);
        session()->put(['key4' => 'Values!!!!']);
        return view( view: 'home');
    }
}
```

Three Evaluate windows are shown:

- Window 1 (Red border):** Shows the code snippet for saving data to the session.
- Window 2 (Orange border):** Shows the expression `session()->all()` and its result, which is an array containing session data like `_previous`, `_flash`, `_token`, and the keys `key1`, `key2`, `key3`, and `key4` with their respective values.
- Window 3 (Green border):** Shows three expressions for retrieving data: `session()->get('key1')`, `session()->get('key1.key2')`, and `session()->get('key4')`, each with its corresponding result.

Здесь показано сохранение данных в сессию и способы получения данных из нее. После выполнения части кода (1) данные будут записаны в сессию. Чтобы получить все данные, хранящиеся в сессии, воспользуемся методом `all` (2). Заметим, что в сессии хранятся не только сохраненные нами данные, но и системные. Обратим также внимание на структуру данных, переданных нами в сессию: в первом и во втором варианте ключ массива, переданного в метод `put`, разделен точкой. В итоге данные в сессии имеют вложенность. В третьем случае сохранения данных в сессию используется ключ без разделения — обратим внимание, как он был записан в сессию.

Способы получения конкретных данных из сессии показаны на скриншоте (3).

Метод `flash` очищает все, что находится в сессии.

Практика

Установка Laravel UI (пользовательского интерфейса)

В шестой версии Laravel для красивого вывода пользовательского интерфейса требуется скачать и установить дополнительный пакет (для Laravel ниже шестой версии этого делать не нужно, поскольку он включен в базовый пакет продукта).

Пакет загружается командой **composer require laravel/ui --dev**

Эта команда подтянет пакет и установит зависимости.

Затем установим пользовательский интерфейс командой **php artisan ui vue --auth**

В нашем приложении уже есть изменения, внесенные в стандартные шаблоны, поэтому в ходе последней команды будет предложено выполнить замену. Согласимся с ней, введя в командной строке **yes**.

```
vagrant@homestead: ~/code/laravel
Package manifest generated successfully.
vagrant@homestead:~/code/laravel$ php artisan ui vue --auth
Vue scaffolding installed successfully.
Please run "npm install && npm run dev" to compile your fresh scaffolding.

The [home.blade.php] view already exists. Do you want to replace it? (yes/no) [
no]:
> yes

Authentication scaffolding generated successfully.
vagrant@homestead:~/code/laravel$ |
```

Добавление таблиц для сохранения данных пользователей

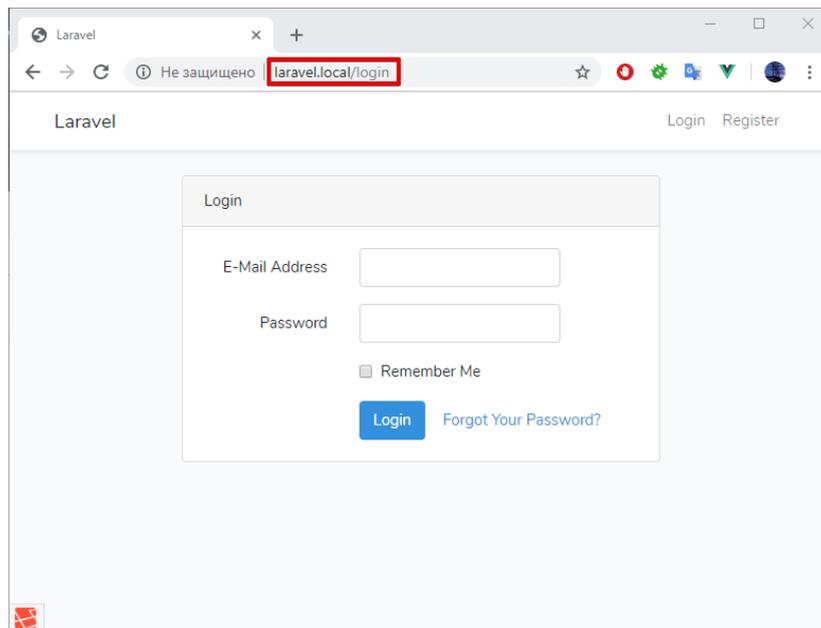
Используя стандартную аутентификацию Laravel, следует выполнить подготовленные миграции, которые были поставлены вместе с фреймворком. Команда для запуска миграций: **php artisan migrate**

Поскольку эти миграции были выполнены в проекте ранее, данная команда не приведет ни к каким действиям: все необходимые таблицы уже созданы.

Базовая аутентификация

Попробуем обратиться к главной странице нашего сайта.

Но доступ к главной странице теперь закрыт. При обращении к ней приложение перенаправляет на страницу, где предлагается ввести логин и пароль зарегистрированного в системе пользователя.

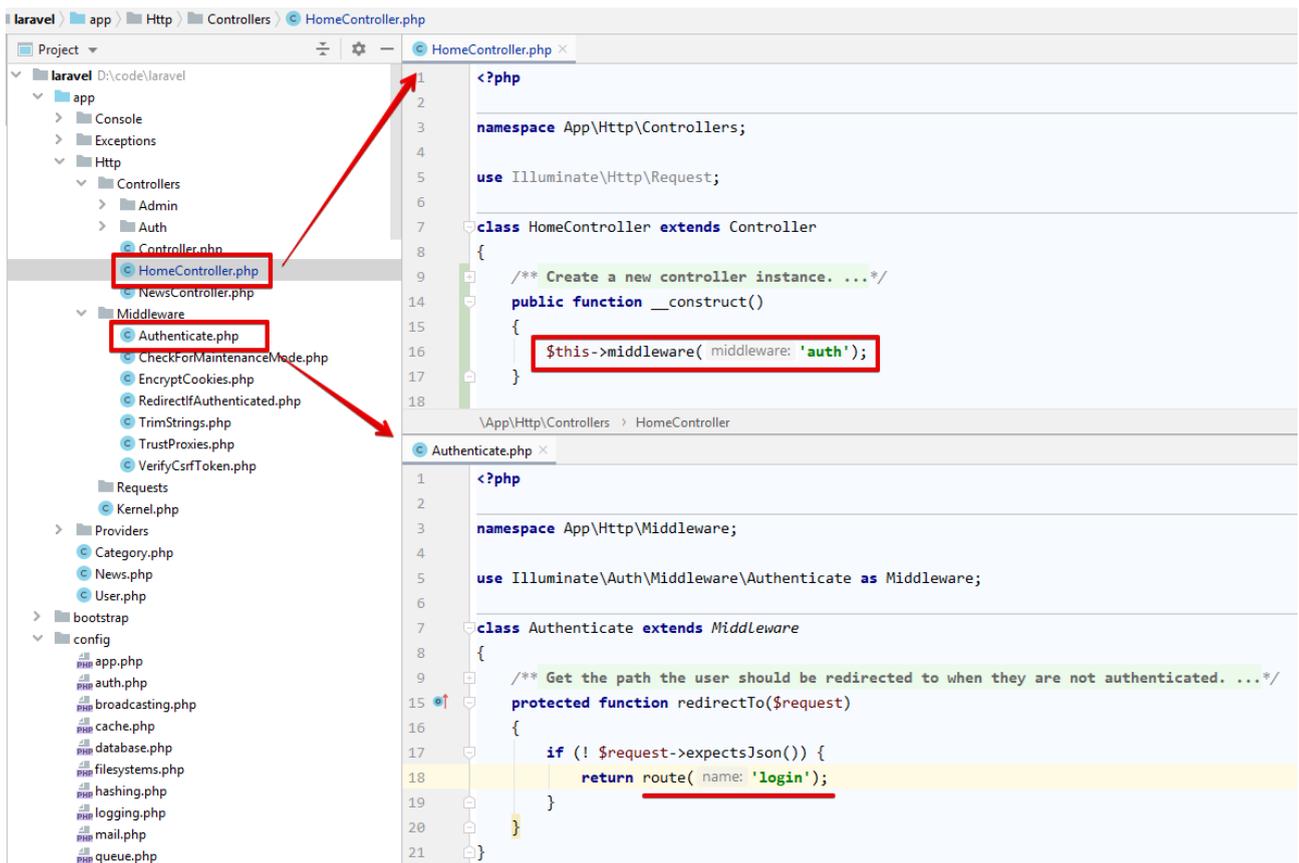


Но почему это так происходит? После успешно выполненных команд в нашем проекте произошли изменения. Рассмотрим их.

В классе **HomeController** появился метод конструктора, в котором вызывается посредник **auth**. Зайдя в класс **App\Http\Kernel**, сможем обнаружить класс, который скрывается за этим именем. Это **App\Http\Middleware\Authenticate**.

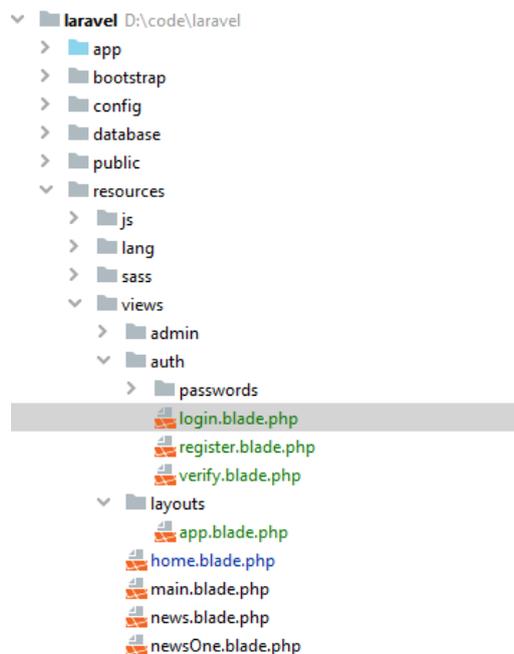
Класс **App\Http\Middleware\Authenticate** является наследником класса **Illuminate\Auth\Middleware\Authenticate** и переопределяет метод **redirectTo**.

redirectTo содержит указание перейти по имени маршрута **login**, если пользователь не авторизован.



С перенаправлением разобрались. Но откуда появились новый роут и форма, которую мы не создавали?

Откроем папку **resources** и заметим, что в ее подпапках появились новые шаблоны.



Так же отметим, что в конце файла **web.php** появились две новые строки. В первой вызывается статический метод **routes** незнакомого класса **Auth** и маршрут, который дублирует роут, ранее использованный нами.

```
<?php
Route::get('/', [
    'uses' => 'HomeController@index',
    'as' => 'home'
]);
Route::group(...);
Route::get('/news', [...]);
Route::get('/news/{news}', [...]);
Auth::routes();
Route::get('/home', 'HomeController@index')->name('home');
```

По аналогии с классом **Route** можем сделать предположение, что и класс **Auth** тоже является фасадом. Проверим это, перейдя в файл **app.php**.

```
'aliases' => [
    'Debugbar' => Barryvdh\Debugbar\Facade::class,
    'App' => Illuminate\Support\Facades\App::class,
    'Arr' => Illuminate\Support\Arr::class,
    'Artisan' => Illuminate\Support\Facades\Artisan::class,
    'Auth' => Illuminate\Support\Facades\Auth::class,
    'Blade' => Illuminate\Support\Facades\Blade::class,
    'Broadcast' => Illuminate\Support\Facades\Broadcast::class,
    'Bus' => Illuminate\Support\Facades\Bus::class,
    'Cache' => Illuminate\Support\Facades\Cache::class,
    'Config' => Illuminate\Support\Facades\Config::class,
```

Предположение оказалось верным. Заглянем в **app.php**.

```

1 <?php
2
3 namespace Illuminate\Support\Facades;
4
5 /** @method static mixed guard(string|null $name = null) ...*/
32 class Auth extends Facade
33 {
34     /** Get the registered name of the component. ...*/
39     protected static function getFacadeAccessor(){...}
43
44     /** Register the typical authentication routes for an application. ...*/
50     public static function routes(array $options = [])
51     {
52         static::$app->make( abstract: 'router')->auth($options);
53     }
54 }
55

```

Создается класс Illuminate\Routing\Router

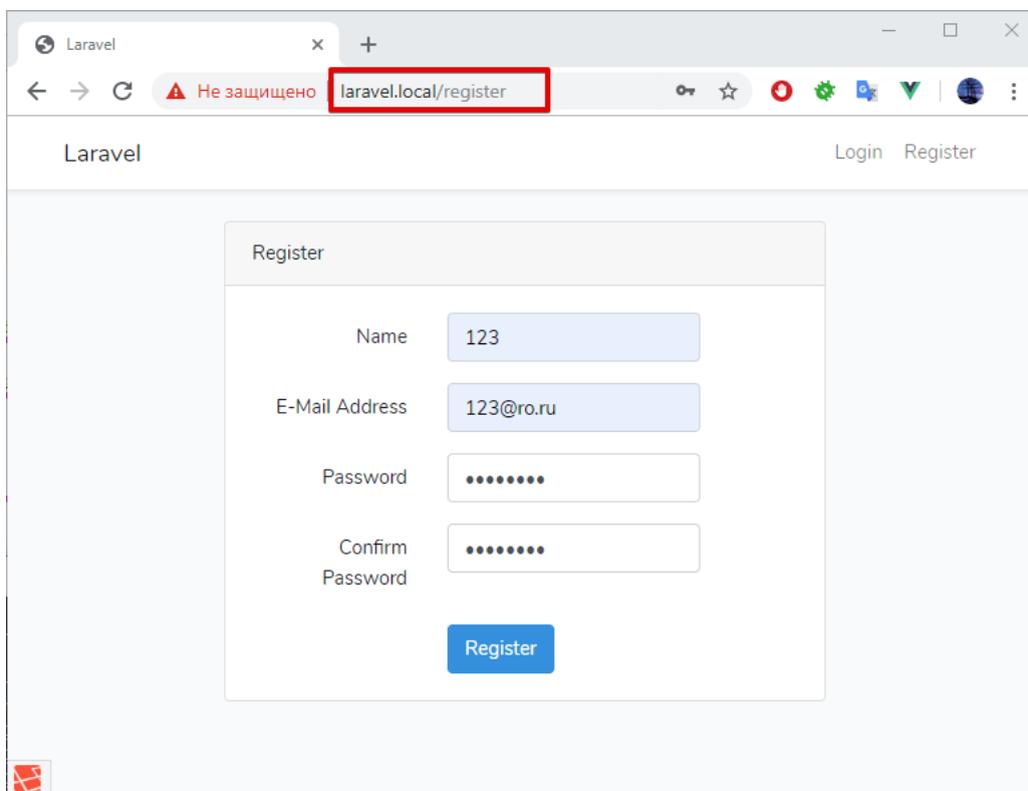
Стало ясно, что в `Auth::routes();` создает экземпляр класса `Illuminate\Routing\Router` и вызывает его метод `auth`. Теперь рассмотрим, что в нем происходит. И увидим, где спрятались новые роуты:

The screenshot shows the `Router.php` file in the `Illuminate\Routing` directory. The `auth` method is highlighted, showing the registration of routes for login and registration. Below the code, a terminal window shows the output of the `php artisan route:list` command, listing all routes in the application.

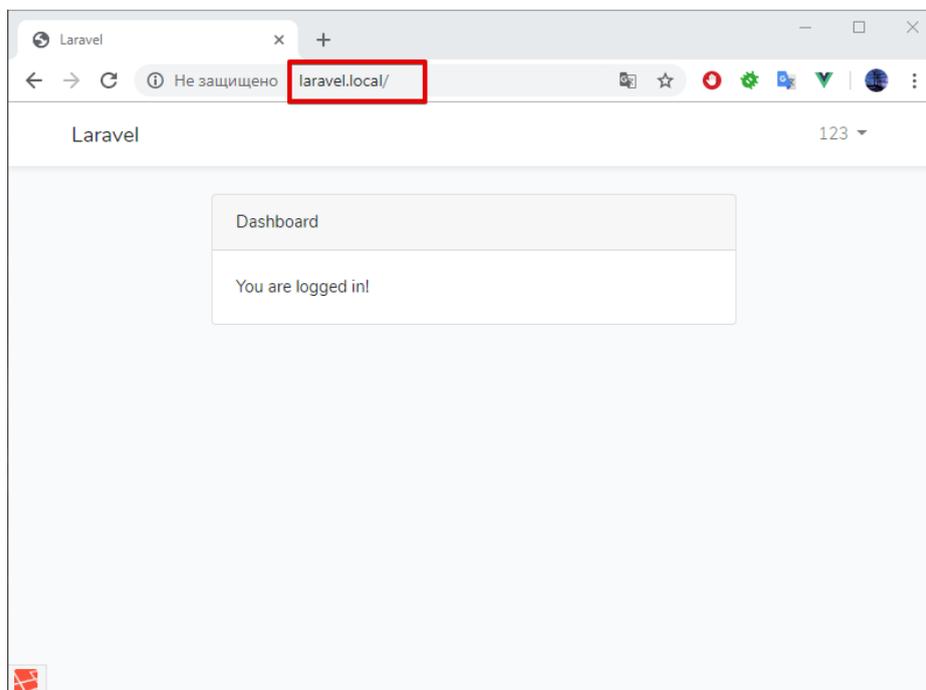
Domain	Method	URI	Name	Action
	GET HEAD	/	home	App\Http\Controllers\HomeController
	GET HEAD	_debugbar/assets/javascript	debugbar.assets.js	Barryvdh\Debugbar\Controllers\Asse
	GET HEAD	_debugbar/assets/stylesheets	debugbar.assets.css	Barryvdh\Debugbar\Controllers\Asse
	DELETE	_debugbar/cache/{key}/{tags?}	debugbar.cache.delete	Barryvdh\Debugbar\Controllers\Cach
	GET HEAD	_debugbar/clockwork/{id}	debugbar.clockwork	Barryvdh\Debugbar\Controllers\Open
	GET HEAD	_debugbar/open	debugbar.openhandler	Barryvdh\Debugbar\Controllers\Open
	GET HEAD	_debugbar/telescope/{id}	debugbar.telescope	Barryvdh\Debugbar\Controllers\Tele
	GET HEAD	admin	admin.index	App\Http\Controllers\Admin\IndexCo
	POST GET HEAD	admin/addNews	admin.addNews	App\Http\Controllers\Admin\NewsCon
	POST GET HEAD	admin/deleteNews/{news}	admin.deleteNews	App\Http\Controllers\Admin\NewsCon
	GET HEAD	admin/news	admin.news	App\Http\Controllers\Admin\NewsCon
	POST GET HEAD	admin/updateNews/{news}	admin.updateNews	App\Http\Controllers\Admin\NewsCon
	GET HEAD	api/user		Closure
	GET HEAD	login	login	App\Http\Controllers\Auth\LoginCon
	POST	login		App\Http\Controllers\Auth\LoginCon
	POST	logout	logout	App\Http\Controllers\Auth\LoginCon
	GET HEAD	news	news	App\Http\Controllers\NewsControlle
	GET HEAD	news/{news}	newsOne	App\Http\Controllers\NewsControlle
	POST	password/email	password.email	App\Http\Controllers\Auth\ForgotPa
	GET HEAD	password/reset	password.request	App\Http\Controllers\Auth\ForgotPa
	POST	password/reset	password.update	App\Http\Controllers\Auth\ResetPas
	GET HEAD	password/reset/{token}	password.reset	App\Http\Controllers\Auth\ResetPas
	GET HEAD	register	register	App\Http\Controllers\Auth\Register
	POST	register		App\Http\Controllers\Auth\Register

Выполнив команду `php artisan route:list`, сможем удостовериться, что такие роуты действительно используются в нашем приложении.

Перейдем по адресу, указанному на скриншоте ниже (для этого достаточно щелкнуть на **Register**), и введем данные нового пользователя.



Теперь у нас есть возможность находиться на главной странице нашего приложения.



Перейдя на главную страницу, отметим, что нет ни главного меню, ни приветственного сообщения. Все это пропало, потому что при применении пользовательского интерфейса мы согласились вернуть

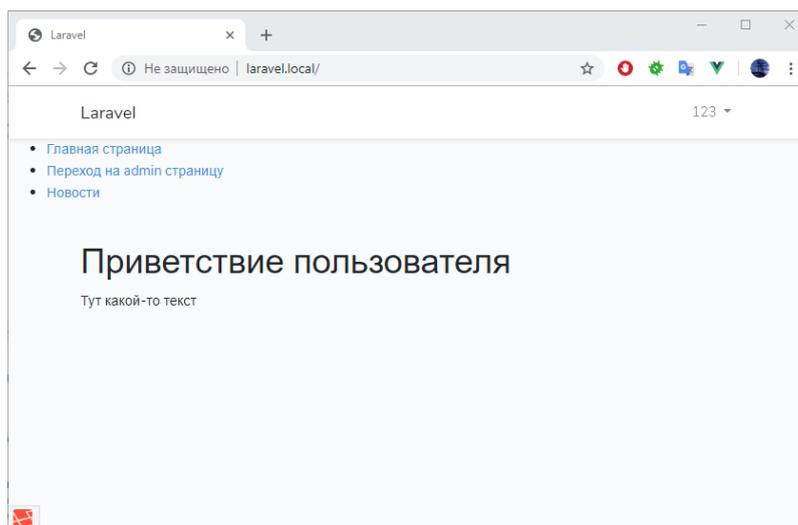
главную страницу к исходному варианту. Внесем изменения, как показано на скриншоте ниже. Наша задача — сохранить стандартную шапку Laravel.

```
home.blade.php
1 @extends('main')
2
3 @section('content')
4     <h1>Приветствие пользователя</h1>
5     Тут какой-то текст<br>
6 @endsection

app.blade.php
19 <!-- Styles -->
20 <link href="{{ asset('css/app.css') }}" rel="stylesheet">
21 </head>
22 <body>
23     <div id="app">
24         <nav class="navbar navbar-expand-md navbar-light bg-white shadow-sm">
25             <div class="container">
26                 <a class="navbar-brand" href="{{ url('/') }}">
27                     {{ config('app.name', 'Laravel') }}
28                 </a>
29                 <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent">
30                     <span class="navbar-toggler-icon"></span>
31                 </button>
32             </div>
33             <div class="collapse navbar-collapse" id="navbarSupportedContent">
34                 <!-- Left Side Of Navbar -->
35                 <ul class="navbar-nav mr-auto">
36
37                 </ul>
38
39                 <!-- Right Side Of Navbar -->
40                 <ul class="navbar-nav ml-auto">
41                     <!-- Authentication Links -->
42
43             </div>
44         </nav>
45     </div>

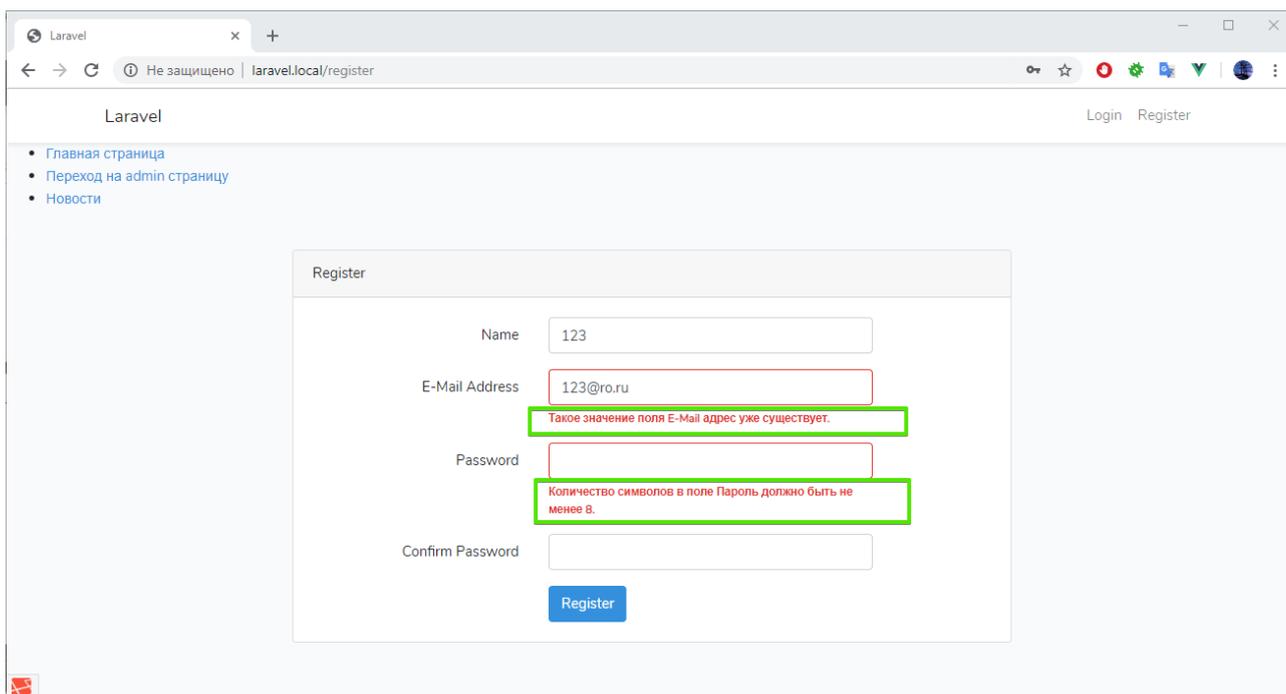
main.blade.php
1 <!doctype html>
2 <html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
3 <head...>
22 <body>
23 <div id="app">
24     <nav class="navbar navbar-expand-md navbar-light bg-white shadow-sm"...>
74     <div class="main-menu">
75         <ul>
76             <li><a href="{{ route('home') }}">Главная страница</a></li>
77             <li><a href="{{ route('admin.index') }}">Переход на admin страницу</a></li>
78             <li><a href="{{ route('news') }}">Новости</a></li>
79         </ul>
80     </div>
81 <main class="py-4 container">
82     @yield('content')
83 </main>
84 </div>
85 </body>
86 </html>
```

Проверим, что все хорошо, перейдя на главную страницу сайта.



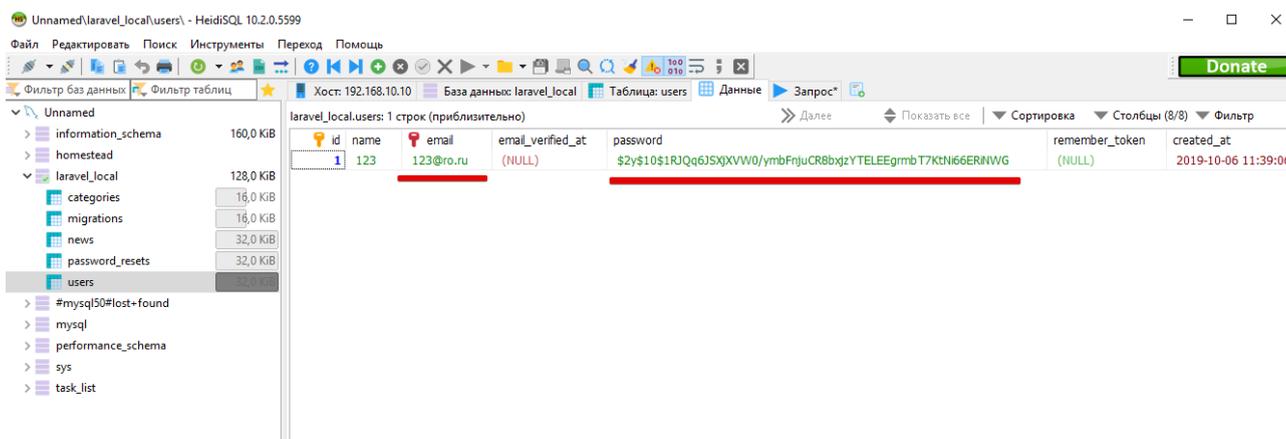
Теперь попробуем проверить встроенные плюшки в базовой аутентификации. Наждем на имя пользователя в правом верхнем углу (в примере — 123) и в выпадающем меню выберем **Logout**.

После этого попробуем повторно зарегистрироваться в приложении, указав тот же email и пароль менее 8 символов. Результат:



Если вы используете новый проект, то сообщения об ошибках будут выводиться на стандартном языке — английском.

Перейдем к базе данных. Зайдем в таблицу **user** и увидим, что в ней есть одна запись, которая соответствует нашему зарегистрированному пользователю. И в базе хранится только хеш пароля.



Закрываем доступ к админке сайта

Поскольку к главной странице доступ должен быть для любого пользователя — удалим автоматически добавленный метод конструктора из **HomeController**.

Теперь укажем, что посредник **auth** должен работать для всех маршрутов в группе, начинающихся с **/admin**. Для этого в соответствующую группу добавим указание на этот посредник.

Чтобы пользователь после аутентификации переходил на главную страницу админки — укажем название данного роута в классе **App\Http\Middleware\RedirectIfAuthenticated**.

```
1 <?php
2
3 namespace App\Http\Middleware;
4
5 use ...
6
7
8 class RedirectIfAuthenticated
9 {
10     /** Handle an incoming request. ...*/
11     public function handle($request, Closure $next, $guard = null)
12     {
13         if (Auth::guard($guard)->check()) {
14             return redirect( to: 'admin.index');
15         }
16
17         return $next($request);
18     }
19 }
20
21
22
23
24
25
26
27
```

И укажем, куда переходить по умолчанию после входа в систему.

```
1 <?php
2
3 namespace App\Http\Controllers\Auth;
4
5 use ...
6
7
8 class LoginController extends Controller
9 {
10     /**
11     *
12     * Where to redirect users after login.
13     *
14     * @var string
15     */
16     protected $redirectTo = '/admin';
17
18     /**
19     * Create a new controller instance.
20     */
21 }
22
23
24
25
26
27
28
29
30
31
```

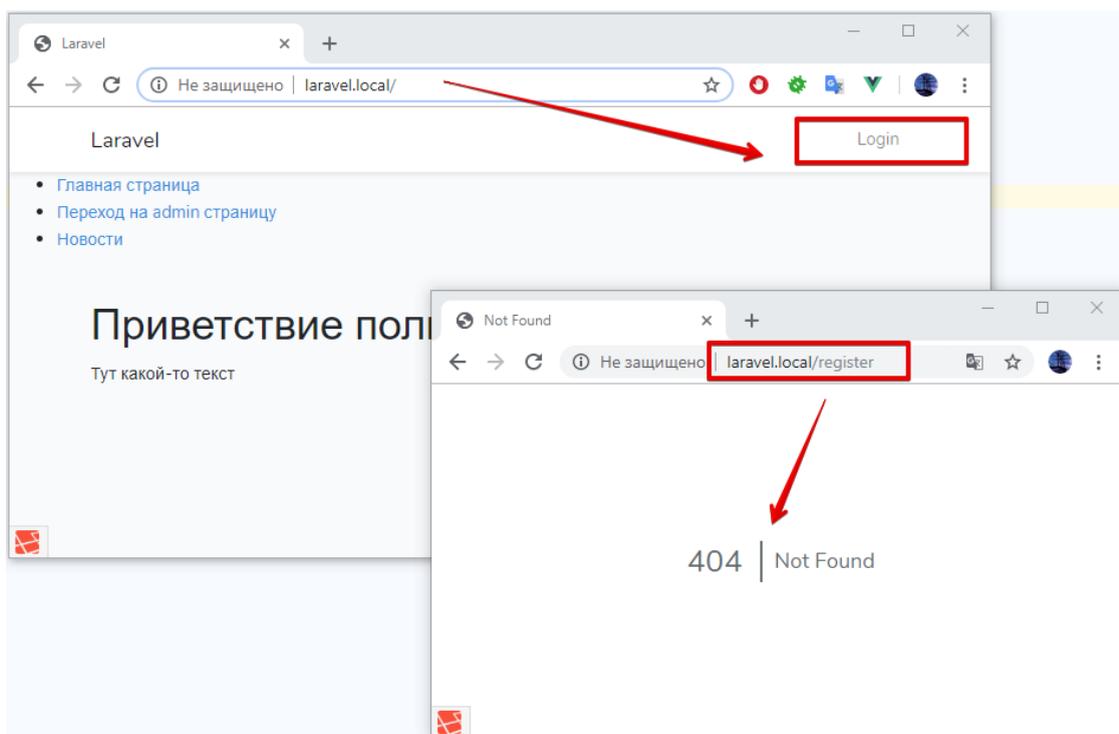
Оптимизация авторизации

Была добавлена аутентификация пользователя. Только у того пользователя, который зарегистрирован в системе, есть права на добавление и редактирование материалов новостей. Но любой пользователь может зарегистрироваться в системе. Это не очень хорошо — получается, что мы закрываем дверь на тяжелый и хитрый замок, а ключ от него оставляем в замочной скважине.

Скроем регистрацию от пользователей. Для этого уберем из наших маршрутов «лишние» роуты. Роуты для авторизации и регистрации система получает из класса `Illuminate\Routing\Router`. Зайдем в него и добавим только необходимые нам маршруты, а `Auth::routes()` закомментируем.

```
1143 /** Register the typical authentication routes for an application. ...*/
1144 public function auth(array $options = [])
1145 {
1150     // Authentication Routes...
1151     $this->get( uri: 'login', action: 'Auth\LoginController@showLoginForm')->name( name: 'login');
1152     $this->post( uri: 'login', action: 'Auth\LoginController@login');
1153     $this->post( uri: 'logout', action: 'Auth\LoginController@logout')->name( name: 'logout');
1154
1155     // Registration Routes...
1156     if ($options['register'] ?? true) {
1157         $this->get( uri: 'register', action: 'Auth\RegisterController@showRegistrationForm')->name( name: 'register');
1158         $this->post( uri: 'register', action: 'Auth\RegisterController@register');
1159     }
1160 }
1161
1162 \Illuminate\Routing\Router > Router > auth()
1163
1164 web.php
1165
1166 //Auth::routes();
1167 Route::get('login', 'Auth\LoginController@showLoginForm')->name('login');
1168 Route::post('login', 'Auth\LoginController@login');
1169 Route::post('logout', 'Auth\LoginController@logout')->name('logout');
```

Теперь в шапке сайта у нас пропала надпись «Регистрация». Если попробовать пройти по адресу, который отвечал за регистрацию, получим сообщение о том, что страница не найдена.



Убрав маршруты для авторизации из списка роутов, вернемся в класс `Illuminate\Routing\Router` и посмотрим, какой класс и какой метод вызывался при стандартной аутентификации.

Изучите рисунок ниже и ответьте:

- что необходимо сделать, чтобы иметь возможность менять данные конкретного пользователя?
- какой класс хранит в себе информацию о пользователях?

```

Router.php
1149 public function auth(array $options = [])
1150 {
1151     // Authentication Routes...
1152     $this->get( uri: 'login', action: 'Auth\LoginController@showLoginForm')->name( name: 'login');
1153     $this->post( uri: 'login', action: 'Auth\LoginController@login');
1154     $this->post( uri: 'logout', action: 'Auth\LoginController@logout')->name( name: 'logout');
1155
1156     // Registration Routes...
1157     if ($options['register'] ?? true) {
1158         $this->get( uri: 'register', action: 'Auth\RegisterController@showRegistrationForm')->name( name: 'register');
1159         $this->post( uri: 'register', action: 'Auth\RegisterController@register');
1160     }
1161 }

RegisterController.php
11 class RegisterController extends Controller
12 {
13     /*...*/
24 use RegistersUsers;
25
26 /** Where to redirect users after registration. ...*/
31 protected $redirectTo = '/home';
32
33 /**
34  * Create a new controller instance.
35  *
36  * @return void
37  */
38 public function __construct(){...}
42
43 /** Get a validator for an incoming registration request. ...*/
49 protected function validator(array $data)
50 {
51     return Validator::make($data, [
52         'name' => ['required', 'string', 'max:255'],
53         'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
54         'password' => ['required', 'string', 'min:8', 'confirmed'],
55     ]);
56 }
57
58 /** Create a new user instance after a valid registration. ...*/
64 protected function create(array $data)
65 {
66     return User::create([
67         'name' => $data['name'],
68         'email' => $data['email'],
69         'password' => Hash::make($data['password']),
70     ]);
71 }
72 }

RegistersUsers.php
2 namespace Illuminate\Foundation\Auth;
3
4 use ...
5
6 trait RegistersUsers
7 {
8     use RedirectsUsers;
9
10 /** Show the application registration form. ...*/
13 public function showRegistrationForm()
14 {
15     return view( view: 'auth.register');
16 }
17
18 /** Handle a registration request for the application. ...*/
23 public function register(Request $request)
24 {
25     $this->validator($request->all())->validate();
26
27     event(new Registered($user = $this->create($request->all())));
28
29     $this->guard()->login($user);
30
31     return $this->registered($request, $user)
32         ?: redirect($this->redirectTo);
33 }
34
35 /**
36  * Get the guard to be used during registration.
37  *
38  * @return \Illuminate\Contracts\Auth\StatefulGuard
39  */
40 }
41
42 /**
43  *
44  */
45 }

```

Выше представлены листинги файлов:

- **vendor/laravel/framework/src/Illuminate/Routing/Router.php**
- **vendor/laravel/framework/src/Illuminate/Foundation/Auth/RegistersUsers.php**
- **app/Http/Controllers/Auth/RegisterController.php**

Нашли ответы? Начнем реализовывать возможность изменения данных авторизованного пользователя. Он у нас пока один в системе :)

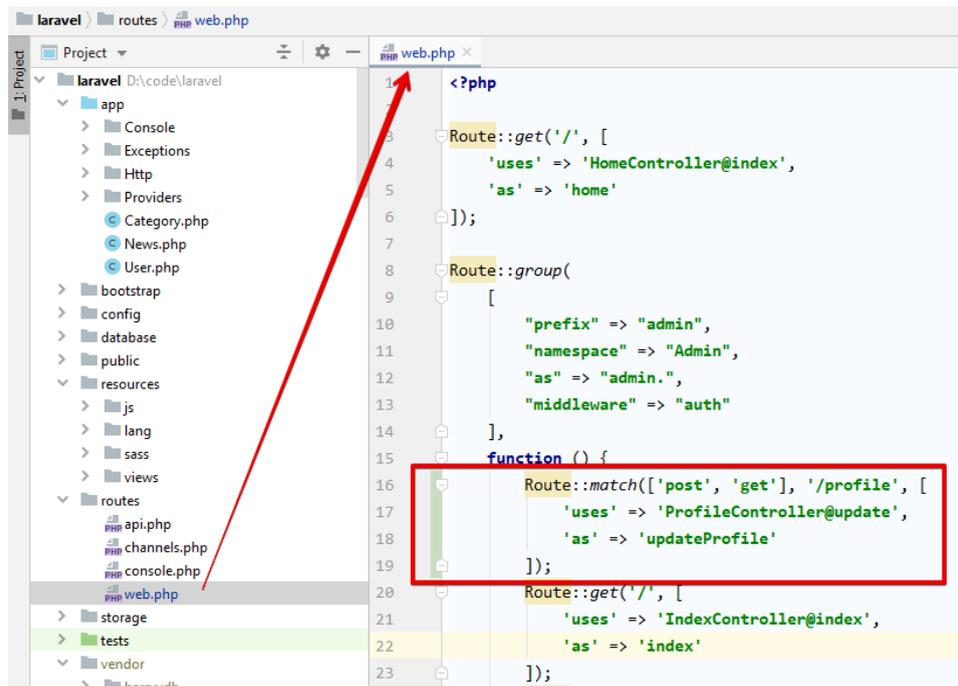
Сначала создадим новый контроллер, доступ к которому будет только у администратора системы. Поэтому создадим его в папке **Admin** следующей командой: **php artisan make:controller Admin/ProfileController**

```

vagrant@homestead: ~/code/laravel
vagrant@homestead:~/code/laravel$ php artisan make:controller Admin/ProfileController
Controller created successfully.
vagrant@homestead:~/code/laravel$ |

```

Далее добавим новый маршрут, который будет выполняться при **get**- и **post**-запросах к адресу **/admin/profile**. Он будет вызывать метод **update** в созданном контроллере:



```
1 <?php
2
3 Route::get('/', [
4     'uses' => 'HomeController@index',
5     'as' => 'home'
6 ]);
7
8 Route::group(
9     [
10        "prefix" => "admin",
11        "namespace" => "Admin",
12        "as" => "admin.",
13        "middleware" => "auth"
14    ],
15    function () {
16        Route::match(['post', 'get'], '/profile', [
17            'uses' => 'ProfileController@update',
18            'as' => 'updateProfile'
19        ]);
20    }
21    Route::get('/', [
22        'uses' => 'IndexController@index',
23        'as' => 'index'
24    ]);
```

Прежде чем перейти в контроллер, создадим новый шаблон, в который поместим форму для изменения данных пользователя.

```
1 @extends('admin.main')
2
3 @section('content')
4     <h1>Изменение учетных данных</h1>
5     <form action="{{route('admin.updateProfile')}}" method="post">
6         @csrf
7
8         @if($errors->has('name'))
9             <div class="alert alert-danger">
10                 @foreach($errors->get('name') as $error)
11                     <p style="...">{{ $error }}</p>
12                 @endforeach
13             </div>
14         @endif
15         <input class="form-control" name="name" placeholder="E-Mail" value="{{ $user->name }}" <br>
16
17         @if($errors->has('email'))
18             <div class="alert alert-danger">
19                 @foreach($errors->get('email') as $error)
20                     <p style="...">{{ $error }}</p>
21                 @endforeach
22             </div>
23         @endif
24         <input class="form-control" name="email" placeholder="E-Mail" value="{{ $user->email }}" <br>
25
26         @if($errors->has('password'))
27             <div class="alert alert-danger">
28                 @foreach($errors->get('password') as $error)
29                     <p style="...">{{ $error }}</p>
30                 @endforeach
31             </div>
32         @endif
33         <input class="form-control" name="password" type="password" placeholder="Текущий пароль" <br>
34
35         @if($errors->has('newPassword'))
36             <div class="alert alert-danger">
37                 @foreach($errors->get('newPassword') as $error)
38                     <p style="...">{{ $error }}</p>
39                 @endforeach
40             </div>
41         @endif
42         <input class="form-control" name="newPassword" type="newPassword" placeholder="Новый пароль" <br>
43
44         <button class="form-control" type="submit">
45             Изменить
46         </button>
47     </form>
48 @endsection
```

И не забудем добавить ссылку на новый маршрут в меню админ-панели.

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport"
6     content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
7 <meta http-equiv="X-UA-Compatible" content="ie=edge">
8 <title>@section('title')Document @show</title>
9 <script src="{{ asset('js/app.js') }}" defer</script>
10 <link href="{{ asset('css/app.css') }}" rel="stylesheet">
11 </head>
12 <body>
13 <ul>
14 <li><a href="{{ route('home') }}">Главная</a></li>
15 <li><a href="{{ route('admin.news') }}">Новости</a></li>
16 <li><a href="{{ route('admin.updateProfile') }}">Изменение данных профиля</a></li>
17 </ul>
18
19 <div class="container">
20     @yield('content')
21 </div>
22 </body>
23 </html>
24
```

Время перейти к контроллеру. Сначала получим данные о пользователе, который у нас авторизован. В этом поможет `фасад Illuminate\Support\Facades\Auth`.

```
1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use ...
6
7
8
9
10 class ProfileController extends Controller
11 {
12     /** @param Request $request ...*/
13     public function update(Request $request)
14     {
15         $user = Auth::user();
16
17         return view( view: 'admin.profile', [
18             'user' => $user,
19         ]);
20     }
21 }
22
23
24
25
```

Задание: посмотреть методы фасада `Auth`. В помощь — документация и (или) соответствующий класс `Illuminate\Support\Facades` в установленной версии фреймворка.

Рассмотрим первый метод этого фасада — `user`. Этот статический метод возвращает экземпляр класса `User`, наполненный данными аутентифицированного пользователя.

Как это работает? Схема:

```
Auth.php
9 * @method static bool guest()
10 * @method static Illuminate\Contracts\Auth\Authenticatable|null user()
11 * @method static int|null id()
12 * @method static bool validate(array $credentials = [])
13 Illuminate\Support\Facades\Auth

Authenticatable.php
1 <?php
2
3 namespace Illuminate\Contracts\Auth;
4
5 interface Authenticatable
6 {

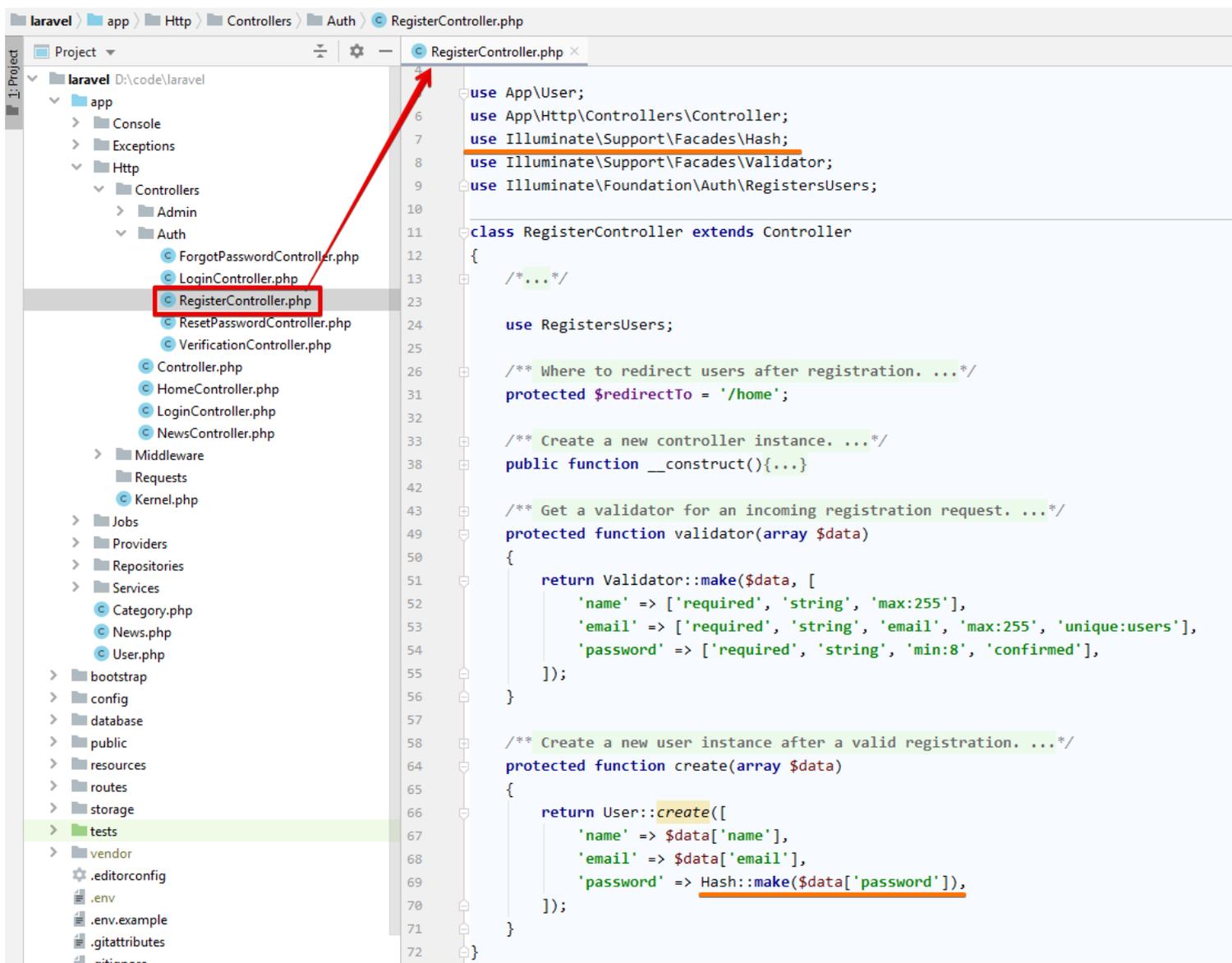
app\User.php
1 <?php
2
3 namespace App;
4
5 use Illuminate\Notifications\Notifiable;
6 use Illuminate\Contracts\Auth\MustVerifyEmail;
7 use Illuminate\Foundation\Auth\User as Authenticatable;
8
9 class User extends Authenticatable
10 {
11     use Notifiable;
12
13     /**
14      * The attributes that are mass assignable.
15      *
16      * @var array
17      */
18     protected $fillable = [
19         'name', 'email', 'password',
20     ];
21 }

Auth\User.php
1 <?php
2
3 namespace Illuminate\Foundation\Auth;
4
5 use Illuminate\Auth\Authenticatable;
6 use Illuminate\Auth\MustVerifyEmail;
7 use Illuminate\Auth\Passwords\CanResetPassword;
8 use Illuminate\Contracts\Auth\Access\Authorizable as AuthorizableContract;
9 use Illuminate\Contracts\Auth\Authenticatable as AuthenticatableContract;
10 use Illuminate\Contracts\Auth\CanResetPassword as CanResetPasswordContract;
11 use Illuminate\Database\Eloquent\Model;
12 use Illuminate\Foundation\Auth\Access\Authorizable;
13
14 class User extends Model implements
15     AuthenticatableContract,
16     AuthorizableContract,
17     CanResetPasswordContract
18 {
19     use Authenticatable, Authorizable, CanResetPassword, MustVerifyEmail;
20 }
```

Поскольку класс **User** является классом-наследником **Illuminate\Database\Eloquent\Model**, работа по изменению информации о текущем пользователе мало чем будет отличаться от редактирования информации модели **News**. Единственное, что еще следует учесть: в базе данных пароль сохраняется в виде хеша.

Как получить этот хеш?

Для этого заглянем в контроллер **App\Http\Controllers\Auth\RegisterController**, который использовался для регистрации новых пользователей.



```
laravel > app > Http > Controllers > Auth > RegisterController.php
Project
  laravel D:\code\laravel
  app
  Console
  Exceptions
  Http
  Controllers
  Admin
  Auth
  ForgotPasswordController.php
  LoginController.php
  RegisterController.php
  ResetPasswordController.php
  VerificationController.php
  Controller.php
  HomeController.php
  LoginController.php
  NewsController.php
  Middleware
  Requests
  Kernel.php
  Jobs
  Providers
  Repositories
  Services
  Category.php
  News.php
  User.php
  bootstrap
  config
  database
  public
  resources
  routes
  storage
  tests
  vendor
  .editorconfig
  .env
  .env.example
  .gitattributes
  .nitianore

use App\User;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;
use Illuminate\Foundation\Auth\RegistersUsers;

class RegisterController extends Controller
{
    /*...*/

    use RegistersUsers;

    /** Where to redirect users after registration. ...*/
    protected $redirectTo = '/home';

    /** Create a new controller instance. ...*/
    public function __construct(){...}

    /** Get a validator for an incoming registration request. ...*/
    protected function validator(array $data)
    {
        return Validator::make($data, [
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
            'password' => ['required', 'string', 'min:8', 'confirmed'],
        ]);
    }

    /** Create a new user instance after a valid registration. ...*/
    protected function create(array $data)
    {
        return User::create([
            'name' => $data['name'],
            'email' => $data['email'],
            'password' => Hash::make($data['password']),
        ]);
    }
}
```

В этом классе и обнаружим, что создание хеша пароля происходило с помощью метода **make** фасада **Hash**. Исследуем данный класс — и обнаружим метод **check**, который и выполняет проверку введенного пароля пользователя с хешем из базы данных.

```
Hash.php
1 <?php
2
3 namespace Illuminate\Support\Facades;
4
5 /**
6  * @method static array info(string $hashedValue)
7  * @method static string make(string $value, array $options = [])
8  * @method static bool check(string $value, string $hashedValue, array $options = [])
9  * @method static bool needsRehash(string $hashedValue, array $options = [])
10  *
11  * @see \Illuminate\Hashing\HashManager
12  */
13 class Hash extends Facade
14 {
15     /**
16      * Get the registered name of the component.
17      *
18      * @return string
19      */
20     protected static function getFacadeAccessor()
21     {
22         return 'hash';
23     }
24 }

HashManager.php
50
51 /** Hash the given value. ...*/
52 public function make($value, array $options = [])
53 {
54     return $this->driver()->make($value, $options);
55 }
56
57 /** Check the given plain value against a hash. ...*/
58 public function check($value, $hashedValue, array $options = [])
59 {
60     return $this->driver()->check($value, $hashedValue, $options);
61 }
62
63
64
65
66
67
68
69
70
71
72
73
74
```

Теперь у нас есть все для того, чтобы менять данные пользователя.

Еще надо провести валидацию данных, передаваемых пользователем, и выполнить вывод ошибок при нарушении правил валидации.

```
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;

class ProfileController extends Controller
{
    /** @param Request $request ...*/
    public function update(Request $request)
    {
        /** @var \Illuminate\Contracts\Auth\Authenticatable|\App\User $user*/
        $user = Auth::user();
        if ($request->isMethod('post')) {
            $this->validate($request, $this->validateRules(), [], $this->attributesName());

            if (Hash::check($request->post('password'), $user->password)) {
                $user->fill([
                    'name' => $request->post('name'),
                    'password' => Hash::make($request->post('newPassword')),
                    'email' => $request->post('email'),
                ]);
                $user->save();
            }

            return redirect()->route('admin.updateProfile');
        }

        return view('admin.profile', [
            'user' => $user,
        ]);
    }

    protected function validateRules()
    {
        return [
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'email', 'max:255', 'unique:users,email,' . Auth::id()],
            'password' => ['required'],
            'newPassword' => ['required', 'string', 'min:8'],
        ];
    }

    protected function attributesName()
    {
        return [
            'newPassword' => 'Новый пароль'
        ];
    }
}
```

Проверяем. Кажется, что-то забыли...

Забыли вывод сообщения при неверно введенном текущем пароле. Метод `route` хелпера `redirect` возвращает экземпляр класса `\Illuminate\Http\RedirectResponse`, который содержит метод `withErrors`. Он позволяет сохранять данные в сессию в качестве флеш-сообщений, которые будут удалены после первого вывода страницы. В эти же сообщения и сохраняются данные об ошибках валидации.

```

/** @param Request $request ...*/
public function update(Request $request)
{
    /**@var \Illuminate\Contracts\Auth\Authenticatable|\App\User $user*/
    $user = Auth::user();
    if ($request->isMethod( method: 'post')) {

        $this->validate($request, $this->validateRules(), [], $this->attributesName() );

        $errors = [];
        if (Hash::check($request->post( key: 'password'), $user->password)) {
            $user->fill([
                'name' => $request->post( key: 'name'),
                'password' => Hash::make($request->post( key: 'newPassword')),
                'email' => $request->post( key: 'email'),
            ]);
            $user->save();
        } else {
            $errors['password'][] = 'Неверно введен текущий пароль';
        }

        return redirect()->route( route: 'admin.updateProfile' )->withErrors($errors);
    }

    return view( view: 'admin.profile', [
        'user' => $user,
    ]);
}

```

Результат работы указанного алгоритма при неверно введенном текущем пароле:

- [Главная](#)
- [Новости](#)
- [Изменение данных профеля](#)

Изменение учетных данных

123

123@ro.ru

Неверно введен текущий пароль

Текущий пароль

Новый пароль

Изменить

Как перенести валидацию этих данных в посредник?

Теперь, когда ошибки выводятся, хорошо бы добавить информацию об успешном выполнении смены данных. Для этого тоже понадобится флеш-сообщение, но оно не должно попадать в ошибки.

Объект класса `Illuminate\Http\Request` тоже содержит доступ к флеш-сообщениям. Вызовем у него метод `session`, который вернет объект класса `Illuminate\Session\Store`, — внутри него и содержится метод `flash`. Добавим в него сообщения об успешном выполнении операции, если все проверки прошли хорошо.

Для вывода этого сообщения воспользуемся фасадом `Session` и его статическим методом `get`.

```
ProfileController.php
20 $user = Auth::user();
21
22 if ($request->isMethod( method: 'post')) {
23
24     $this->validate($request, $this->validateRules(), [], $this->attributesName() );
25
26     $errors['password'][] = 'Неверно введен текущий пароль';
27     if (Hash::check($request->post( key: 'password'), $user->password)) {
28         $user->fill([
29             'name' => $request->post( key: 'name'),
30             'password' => Hash::make($request->post( key: 'newPassword')),
31             'email' => $request->post( key: 'email'),
32         ]);
33         $user->save();
34         $request->session()->flash('MSG', 'Данные сохранены.');
```

```
profile.blade.php
2
3 @section('content')
4 <h1>Изменение учетных данных</h1>
5 @if(Session::has('MSG'))
6     <div class="alert alert-success">
7         {{ Session::get('MSG')}}
8     </div>
9 @endif
10 <form action="{{route('admin.updateProfile')}}" method="post">
11     @csrf
```

Результат выполнения:

- Главная
- Новости
- Изменение данных профеля

Изменение учетных данных

Данные сохранены.

Разграничение прав пользователей

Пока в системе любой пользователь, который пройдет регистрацию и аутентификацию на сайте, становится администратором системы. Но нужно давать пользователям доступ к дополнительным новостям, блокируя возможность их править. Для этого добавим в таблице **users** дополнительное поле, которое будет отвечать за разграничение прав пользователей. Выполним команду для создания новой миграции: **php artisan make:migration AlterTableUsersAddIsAdmin**

Код для добавления нового поля:

```
<?php
use ...

class AlterTableUsersAddIsAdmin extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->boolean('is_admin')
                ->default('false')
                ->comment('Поле определяет является ли пользователь администратором');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn(['is_admin']);
        });
    }
}
```

Выполним миграцию командой **php artisan migrate**

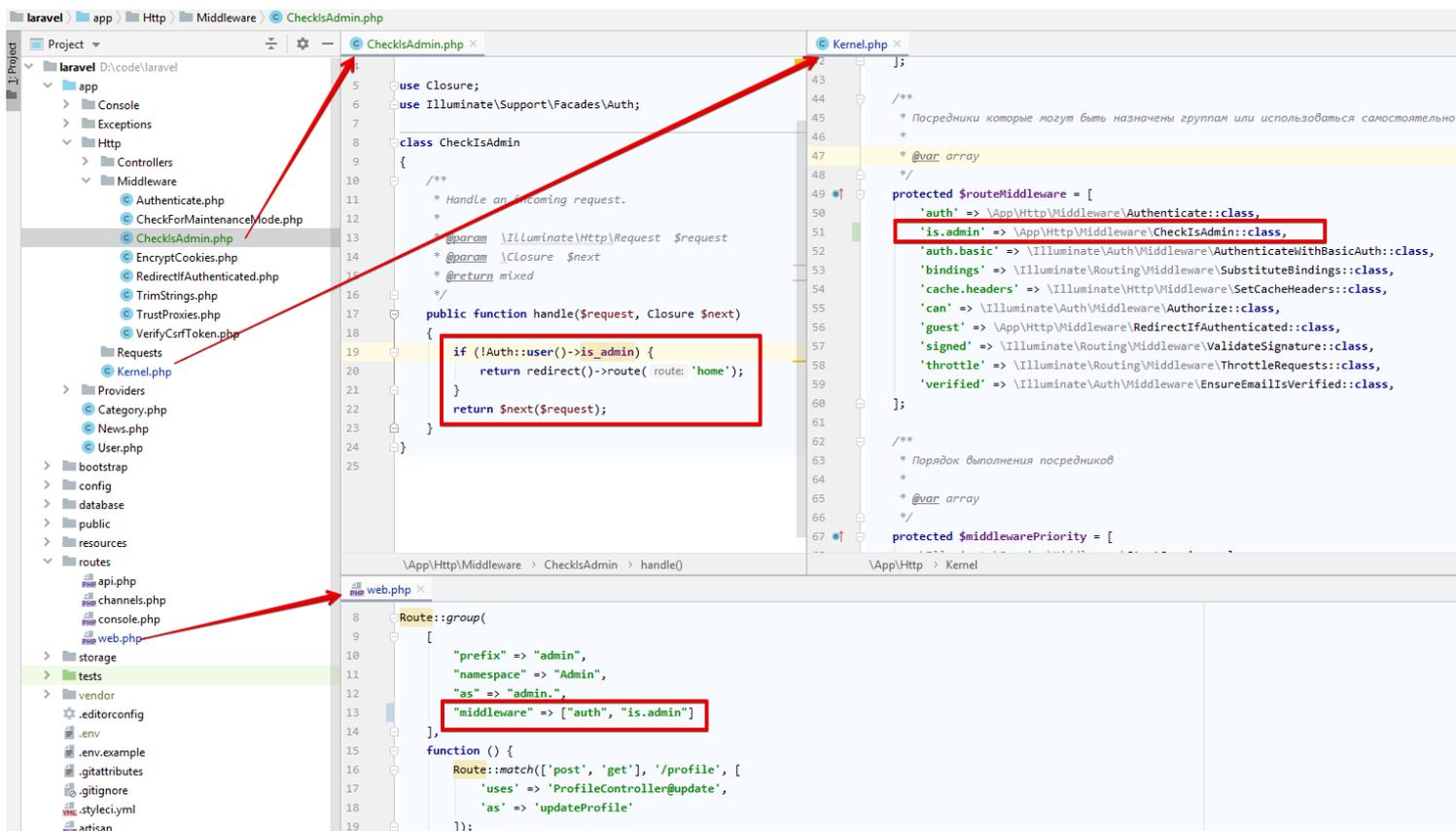
Результат выполнения миграции:

id	name	email	email_verified_at	password	remember_token	created_at	updated_at	is_admin
1	123	123@ro.ru	(NULL)	\$2y\$10\$WwUt7M3DmV7EuvV6LK2CEePW65Ytm...	(NULL)	2019-10-06 11:39:06	2019-10-12 15:18:19	1

Чтобы контролировать наличие единицы в данном поле, создадим посредника:

php artisan make:middleware ChecksAdmin

Этот посредник будет содержать в себе простую проверку: если аутентифицированный пользователь не имеет никакого значения в свойстве **is_admin** — значит, система перенаправит его на домашнюю страницу. Зарегистрируем его и добавим к группе роутов **admin**.



Практическое задание

1. Реализовать механизм регистрации обычных пользователей.
2. В админке агрегатора реализовать редактирование профилей пользователей с возможностью их перевода в администраторы.
3. * Перенести валидации данных в посредники (дополнительное задание).

Дополнительные материалы

1. <https://laravel.com/docs/5.8/authentication>
2. <https://laravel.ru/forum/viewtopic.php?id=881>
3. <http://cccp-blog.com/laravel/laravel-auth>

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <https://laravel.com/docs/6.x>