Приветствую вас, друзья!

Сегодня вас ожидает очередная статья, посвящённая созданию сайтов на Laravel PHP framework, из цикла, который продолжается уже больше года.

Если вы не следите за публикациями (чтобы это исправить, достаточно [подписаться на уведомления](http://eepurl.com/duAlVH)), то напомню, что я поставил себе цель — создание корпоративного сайта на Laravel с формой обратной связи, блогом, админкой для создания статей и наполнения сайта другой информацией, мультиязычностью витрины и т.д.

На данный момент натянут шаблон витрины, готова форма обратной связи, работает [отправка email сообщений](http://cccp-blog.com/laravel/laravel-mail) и в планах самое вкусное — прикручивание [Laravel админки](http://cccp-blog.com/laravel/laravel-admin-panel) и долгожданная работа с моделями и миграциями для создания БД, наполнения её информацией и работой с ней.

В предыдущей статье я говорил, что в следующей, т.е. сегодняшней, укажу, на каком варианте Laravel админки я остановился.

Однако, я решил немного продлить интригу и сообщить данную новость немного позже, т.к., начав натягивать приглянувшийся мне вариант на сайт, понял, что логичнее сначала будет настроить механизм аутентификации для входа в панель администрирования под определённой учёткой.

Поэтому данная статья будет посвящена Laravel auth (сокращённо от «authentication» — аутентификация), а точнее, её установке и настройке. Также я расскажу об основных методах фасада Laravel Auth и покажу, как их можно кастомизировать.

Поехали!

СОДЕРЖАНИЕ

* [Работа с Laravel auth: начальные условия](http://cccp-blog.com/laravel/laravel-auth#rabota-s-laravel-auth-nachalnye-usloviya)
* [Laravel регистрация пользователей и их аутентификация: установка](http://cccp-blog.com/laravel/laravel-auth#laravel-registratsiya-polzovatelej-i-ih-autentifikatsiya-ustanovka)
* [Laravel аутентификация: настройка](http://cccp-blog.com/laravel/laravel-auth#laravel-autentifikatsiya-nastrojka)
* [Кастомизация Laravel auth методов](http://cccp-blog.com/laravel/laravel-auth#kastomizatsiya-laravel-auth-metodov)
* [Laravel аутентификация: практикум](http://cccp-blog.com/laravel/laravel-auth#laravel-autentifikatsiya-praktikum)
* [Методы фасада Laravel Auth и хэлперы аутентификации](http://cccp-blog.com/laravel/laravel-auth#metody-fasada-laravel-auth-i-helpery-autentifikatsii)

**Работа с Laravel auth: начальные условия**

Перед тем, как мы начнём, хочу напомнить, что все описанные в статье действия актуальны для Laravel 5.6, т.к. на данной версии фреймворка сейчас и работает разрабатываемый в рамках данного курса тестовый сайт.

Однако, думаю, что для проектов на более ранних версиях Laravel алгоритм установки и настройки механизма Laravel регистрации и аутентификации останется таким же за исключением каких-то отдельных методов и фич.

Поэтому, хотя это и не критично, но всё же рекомендую вам сделать [обновление версии Laravel](http://cccp-blog.com/laravel/laravel-update), тем более, что процесс этот не обладает какой-то особой сложностью.

Инструкции по Laravel version update вы можете найти в статье по ссылке выше.

И ещё одно уточнение. Описанный в сегодняшней статье механизм называется именно аутентификацией, а не авторизацией, которые все постоянно путают, включая меня самого  Как [говорит всемогущая Википедия](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), аутентификация — это процедура проверки легальности данных или пользователя путём сверки предоставляемой информации с неким оригиналом (записи в БД или проверки цифровой подписи по ключу шифрования).

А авторизация — это проверка, может ли пользователь, прошедший аутентификацию (легальный), выполнять определённые действия, например, переходить на определённые страницы и выполнять определённые запросы.

Как по мне, грань достаточно тонкая, но Laravel чётко разделяет эти понятия путём вынесения методов аутентификации в фасад Laravel Auth, который будет рассмотрен сегодня, а функции авторизации доступны через Laravel Gate facade.

[](https://www.templatemonster.com/ru/?aff=cccpblog&utm_campaign=blog_site_cccp-blog&utm_source=test_banner&utm_medium=web)

**Laravel регистрация пользователей и их аутентификация: установка**

Laravel authentication — это механизм, встроенный в сам фреймворк по умолчанию, поэтому для реализации и настройки аутентификации не нужно устанавливать дополнительно никаких пакетов.

Итак, чтобы добавить в свой проект файлы, необходимые для организации Laravel аутентификации пользователей (формы логина, регистрации, восстановления пароля), а также правила маршрутизации для произведения соответствующих действий), первым делом, открываем консоль сервера, переходим в каталог Laravel проекта и выполняем следующую artisan команду:

|  |  |
| --- | --- |
| 1 | php artisan make:auth |

Если всё прошло успешно, то вы увидите в консоли сообщение **Authentication scaffolding generated successfully.**

Если с установкой Laravel аутентификации на данном этапе возникли какие-то проблемы, то пишите ваши сообщения об ошибках в комментариях под статьёй — будем разбираться вместе

Следующим этапом, если вы запускаете Laravel проект с нуля и ещё не создали таблицу БД для хранения пользователей, данную таблицу нужно сделать, чтобы было где хранить наших юзеров и их учётные данные.

Для этого в Laravel по умолчанию есть базовые миграции для создания таблицы пользователей (по умолчанию называется **users**) и таблицы для хранения попыток сброса паролей (**password\_resets**), которые расположены в каталоге **корень\_сайта\database\migrations** и носят следующие названия:

* **2014\_10\_12\_000000\_create\_users\_table.php**
* **2014\_10\_12\_100000\_create\_password\_resets\_table.php**

Я на своём сайте изначально планировал организовать Laravel аутентификацию для логина администратора в админке, т.е. предполагалось, что пользователь будет один, и никакой сброс пароля и регистрация новых юзеров на сайте нужны не будут. Поэтому я смело удаляю миграцию для создания таблицы password\_resets.

Ещё один нюанс. Я хочу, чтобы имя моего пользователя хранилось в поле не name, как по умолчанию, а login, которое, как и email, должно быть уникальным. Поэтому я внёс необходимые правки в файл миграции, который стал выглядеть так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | <?php    use Illuminate\Support\Facades\Schema;  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Database\Migrations\Migration;    class CreateUsersTable extends Migration  {      /\*\*       \* Run the migrations.       \*       \* @return void       \*/      public function up()      {          Schema::create('users', function (Blueprint $table) {              $table->increments('id');              $table->string('login')->unique();              $table->string('email')->unique();              $table->string('password');              $table->rememberToken();              $table->timestamps();          });      }        /\*\*       \* Reverse the migrations.       \*       \* @return void       \*/      public function down()      {          Schema::drop('users');      }  } |

К числу дополнительных приготовлений также относится подготовка Laravel seeder для добавления в БД данных администратора. С этой целью я откорректировал базовый файл по пути **корень\_сайта\database\seeds\DatabaseSeeder.php** следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <?php    use Illuminate\Database\Seeder;    class DatabaseSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*       \* @return void       \*/      public function run()      {          DB::table('users')->insert([              'login' => 'admin',              'email' => 'admin@gmail.com',              'password' => bcrypt('admin'),              'created\_at' => date('Y-m-d H:i:s'),              'updated\_at' => date('Y-m-d H:i:s'),          ]);      }  } |

Думаю, не стоит говорить, что не нужно вслепую копировать и выполнять мой код  Создавайте администратора с реальными данными, т.к. в дальнейшем вам придётся с ними работать, поэтому поменяйте email, логин и пароль на что-то более вменяемое, чем у меня в примере.

В этом же файле вы, кстати, можете поменять и название всей таблицы в целом, использовав вместо стандартного users то, что посчитаете нужным.

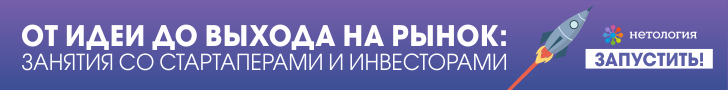
После этого всё, что осталось для того, чтобы создать таблицу в базе данных и наполнить её данными, — это запустить следующую Laravel artisan команду:

|  |  |
| --- | --- |
| 1 | php artisan migrate --seed |

Я не буду сейчас углубляться в описание особенностей миграций, сидеров и операций с ними, т.к. это тема для отдельной статьи.

Описанные выше действия были актуальны для запуска Laravel проекта с нуля, когда у вас нет ещё ни БД, ни пользователей.

Если же вы производили установку файлов форм Laravel аутентификации на уже существующий проект, то добавление authentication функционала в него ограничится лишь корректировкой существующей таблицы БД, хранящей ваших пользователей, и указанием её или соответствующей модели в качестве источника данных пользователей для их входа в систему.

[](http://netology.ru/kursy?pid=up6076434&utm_banner=728x90)

Как это сделать, мы поговорим в следующем блоке статьи, посвящённом настройке Laravel authentication.

По поводу структуры таблицы БД с данными пользователей, сами разработчики Laravel в [официальной документации](https://laravel.com/docs/5.6/authentication) дают рекомендации использовать для хранения пароля строковый тип поля длиной не менее 60 символов и добавить в таблицу поле **remember\_token** длиной в 100 символов (тип, естественно, строковый) и со значением **NULL** по умолчанию.

Последнее необходимо будет для длительного хранения сессии пользователя и восстановлении сеанса при повторном входе на сайт.

**Laravel аутентификация: настройка**

Итак, после того, как мы добавили необходимые для Laravel auth файлы в наш проект и создали таблицу для хранения пользователей, осталось только всё это дело связать, т.е. указать фреймворку, в какой именно таблице следует искать Laravel auth user данные при его аутентификации на сайте.

Чтобы это сделать, нужно заглянуть в соответствующий Laravel config файл (или конфиг, по-русски), в котором описаны все необходимые настройки, необходимые при настройке authentication — **корень\_сайта\config\auth.php**. Данный файл, как и контроллеры, доступен в Laravel приложении также по умолчанию.

Он состоит из секций, описывающих **guards (гвард)** и **providers (провайдер)** для произведения Laravel user auth. Первые определяют действия, которые происходят в процессе аутентификации, а вторые — где хранятся данные пользователей.

По умолчанию в Laravel auth provider один — **users**, который выглядит так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | ...    'providers' => [      'users' => [          'driver' => 'eloquent',          'model' => App\User::class,      ],        // 'users' => [      //     'driver' => 'database',      //     'table' => 'users',      // ],  ],    ... |

Как видите, здесь представлены параметры, с помощью которых можно указать в качестве источника данных как имя таблицы БД, так и модель. По умолчанию указана дефолтная модель **User**, расположенная в файле **корень\_сайта\app\User.php** и использующая в качестве источника таблицу БД **users**.

Если же таблица с данными пользователей у вас называется по-другому, то вам необходимо в данной секции конфига прописать свою модель или таблицу БД, выбрав необходимый драйвер — **eloquent** для использования модели или **database**для таблиц БД.

Также Laravel позволяет создавать свои кастомные провайдеры. Как это сделать, очень подробно описано [здесь](https://laravel.com/docs/5.6/authentication#adding-custom-user-providers).

Секция настроек для конфигурирования guards выглядит же следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | 'guards' => [      'web' => [          'driver' => 'session',          'provider' => 'users',      ],        'api' => [          'driver' => 'token',          'provider' => 'users',      ],  ], |

Laravel auth guards, доступные по умолчанию, имеют названия созвучные их областям применения. Web подходит для большинства стандартных сайтов, т.к. для аутентификации пользователей используются данные, хранящиеся в сессии на сервере (физически они могут быть оформлены как в виде файлов, так и таблицы в БД, но Laravel session будет посвящен отдельный разговор).

Api же отлично подходит для запросов, отправляемых через API на сервер. Аутентификация в этом случае происходит с помощью специальных токенов — уникальных наборов символов, которые генерируются при успешном входе в систему под определёнными учётками.

Драйвера для guards, задающие способ аутентификации, бывают двух типов: session и token, принцип работы которых был описан выше. Также для Laravel auth guards можно указывать providers, чтобы разграничивать хранилища данных для каждого гварда. В качестве провайдера указываются структуры, описанные в секции providers этого же конфига auth.php.

Как и в случае с провайдерами, Laravel позволяет создавать свои кастомные гварды. Пример реализации описан [здесь](https://laravel.com/docs/5.6/authentication#adding-custom-guards).

И ещё одной секцией рассматриваемого нами конфига Laravel аутентификации является блок настроек сброса пароля пользователей, который выглядит так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 'passwords' => [      'users' => [          'provider' => 'users',          'table' => 'password\_resets',          'expire' => 60,      ],  ], |

Для каждого блока данной секции нужно указывать провайдер, чтобы фреймворк понимал, с какими данными оперировать при сбросе пароля, и таблицу, в которую будут записываться попытки сброса.

Параметр expire позволяет указать время существования токена сброса. Другими словами, определяет время, в течении которого пользователь может сбросить свой пароль после оставления им заявки, при которой reset token и генерируется.

Настройки Laravel аутентификации и регистрации, действующие по умолчанию, вынесены в отдельный блок в самом верху файла:

|  |  |
| --- | --- |
| 1  2  3  4 | 'defaults' => [      'guard' => 'web',      'passwords' => 'users',  ], |

Здесь указан дефолтный гвард и блок настроек для сброса пароля.

Как вы могли заметить, все настройки взаимосвязаны: providers используются в guards и passwords, guards — в настройках по умолчанию.

Также ещё одной особенностью настроек является возможность создания нескольких провайдеров и, соответственно, гвардов и секций сброса пароля в зависимости от типов пользователей, имеющихся в вашем проекте.

Эта фича Laravel может быть полезна, как я уже и говорил ранее, при запуске marketplace и CRM, где нужно разграничивать роли пользователей, их права и доступные роуты.

В таком случае один из Laravel auth guards будет использоваться по умолчанию при указании его в секции defaults конфига, а остальные по необходимости.

**Кастомизация Laravel auth методов**

После запуска Laravel artisan команды **make:auth** ваш проект был расширен следующими файлами (спасибо Git, что он есть и позволяет определять данные изменения), которые были скопированы из недр Laravel framework:

* **app\Http\Controllers\HomeController.php**
* **resources\views\auth\login.blade.php**
* **resources\views\auth\passwords\email.blade.php**
* **resources\views\auth\passwords\reset.blade.php**
* **resources\views\auth\register.blade.php**
* **resources\views\home.blade.php**
* **resources\views\layouts\app.blade.php**

А также был обновлён файл маршрутизации (со списком Laravel routes, они же — правила маршрутизации или маршруты) **routes\web.php**, в который добавились следующие строчки:

|  |  |
| --- | --- |
| 1  2  3 | Auth::routes();    Route::get('/home', 'HomeController@index')->name('home'); |

Первая строка представляет собой список стандартных правил маршрутизации Laravel, связанных с аутентификацией. Для общего развития приведу их полный список, чтобы вы знали, какие они есть в принципе:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | /\*\*   \* Роуты аутентификации...   \*/    //отображение формы аутентификации  Route::get('login', 'Auth\LoginController@showLoginForm')->name('login');  //POST запрос аутентификации на сайте  Route::post('login', 'Auth\LoginController@login');  //POST запрос на выход из системы (логаут)  Route::post('logout', 'Auth\LoginController@logout')->name('logout');    /\*\*   \* Маршруты регистрации...   \*/    //страница с формой Laravel регистрации пользователей  Route::get('register', 'Auth\RegisterController@showRegistrationForm')->name('register');  //POST запрос регистрации на сайте  Route::post('register', 'Auth\RegisterController@register');    /\*\*   \* URL для сброса пароля...   \*/    //POST запрос для отправки email письма пользователю для сброса пароля  Route::post('password/email', 'Auth\ForgotPasswordController@sendResetLinkEmail')->name('password.email');  //ссылка для сброса пароля (можно размещать в письме)  Route::get('password/reset', 'Auth\ForgotPasswordController@showLinkRequestForm')->name('password.request');  //страница с формой для сброса пароля  Route::get('password/reset/{token}', 'Auth\ResetPasswordController@showResetForm')->name('password.reset');  //POST запрос для сброса старого и установки нового пароля  Route::post('password/reset', 'Auth\ResetPasswordController@reset'); |

HTTP POST запросы отправляются, как правило, через HTML формы, поэтому, если по какой-то причине вы не знаете, как это делать, то [эта статья](http://cccp-blog.com/koding/kak-peredat-peremennuyu-php-v-html-i-obratno) как раз для вас.

Вышеприведенный список будет вам полезен ещё и в тех случаях, когда вы захотите убрать некоторые из данных Laravel auth routes из своего проекта из-за банальной ненадобности.

В таком случае вам нужно будет просто удалить **Auth::routes();** из **routes\web.php**, а вместо этого добавить в файл только те роуты, которые вам будут необходимы.

Таким же образом вы сможете изменить название самого Laravel auth url (маршрут) и его обработчик.

Если присмотреться к Laravel auth routes, то заметно, что для данных стандартных маршрутов аутентификации уже существуют и указаны не менее стандартные обработчики, которыми являются контроллеры из директории **корень\_сайта\app\Http\Controllers\Auth**.

Данный каталог же находится в Laravel приложении по указанному пути сразу после установки фреймворка на ПК. Честно говоря не понимаю, почему разработчики Laravel не сделают его создание после выполнения консольной команды **make:auth**, которая переносит в приложение роуты Laravel аутентификации и другие необходимые файлы.

Как по мне, это было бы логично. А так начинающие пользователи, которые копаются в файлах фреймворка после его установки, начинают ломать голову, что это за каталог и зачем он нужен (если он им вообще будет нужен).

Возвращаясь к стандартным Laravel auth routes, хотелось бы сказать, что, несмотря на их кажущуюся полноценность, они всё же составлены немного нелепо, т.к. в качестве обработчиков у них указаны несуществующие методы.

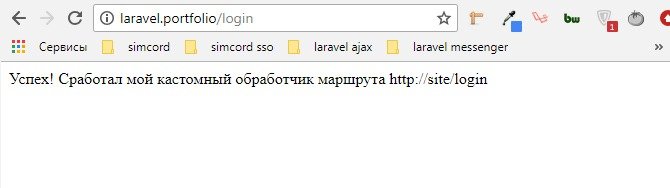
Например, судя по роутам, в файле **app\Http\Controllers\Auth\LoginController.php** должно быть три метода:

* **showLoginForm();**
* **login();**
* **logout();**

Однако, если вы загляните в данный файл, то не найдёте в нём ни одного из вышеперечисленных. Но, если вы добавите следующую функцию в самый конец контроллера, то при вводе соответствующего URL в браузере увидите, что обработчик сработал.

|  |  |
| --- | --- |
| 1  2  3  4 | public function showLoginForm()  {      return 'Успех! Сработал мой кастомный обработчик маршрута http://site.com/login';  } |

И, собственно говоря, результат в браузере:

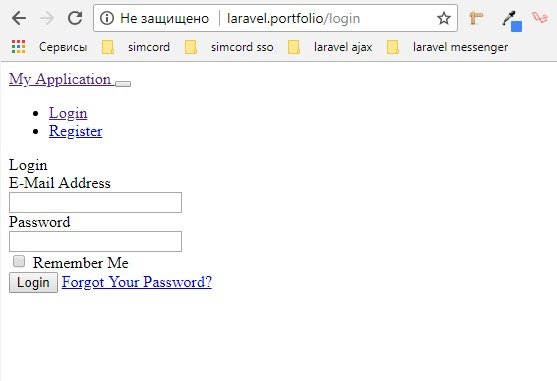


Если же всё оставить на своих местах, т.е. не писать кастомный обработчик, то при вводе URL в формате **http://site.com/login** на экране браузера вы увидите стандартную форму входа, файл которой был добавлены в приложение при запуске Artisan команды **make:auth**.

Данным файлом является **resources\views\auth\login.blade.php**. Решил уточнить на случай, если вы захотите её использовать в дальнейшем либо вам могут понадобиться какие-то её элементы для конструирования собственной.

У меня, правда, она выглядит совсем неказисто за счёт того, что использует стили и скрипты из стандартных статических файлов Laravel **public\css\app.css** и **public\js\app.js** соответственно, которые я уже давно и благополучно удалил

В итоге, выглядела она у меня так:



Поэтому, если у вас форма логина выглядит похожим образом, держу в курсе, что это проблемы со стилями, точнее с их отсутствием по указанным в шаблоне путям. Так что, если вы действительно дорожите данной стандартной формой логина, то укажите правильные пути в файле **resources\views\layouts\app.blade.php.**

Я лично не стал этим заниматься, т.к. стандартную форму я всё равно буду удалять и использовать ту, которая содержится в используемом мною шаблоне Laravel админки, процесс прикручивания которой я подробно опишу в следующей статье.

Итак, возвращаясь к обработчикам стандартных Laravel auth routes, давайте разберёмся, что это за магия приключилась с ними, что в качестве обработчиков по умолчанию им указаны несуществующие методы, вместо которых используются совершенно другие.

Для примера возьмём метод, который мы только что переопределяли — **http://site.com/login**.

При отсутствующем кастомном обработчике по умолчанию при переходе на данный URL в браузере выполняется метод **showLoginForm()** из файла по пути **директория\_сайта\vendor\laravel\framework\src\Illuminate\Foundation\Auth\AuthenticatesUsers.php**, который выглядит следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | /\*\*   \* Show the application's login form.   \*   \* @return \Illuminate\Http\Response   \*/  public function showLoginForm()  {     return view('auth.login');  } |

Как видите, он как раз и ответственен за вывод на экран страницы с auth формой, код которой и хранится в уже упоминаемом нами файле **resources\views\auth\login.blade.php**.

Почему же при создании кастомного обработчика в контроллере LoginController.php он начинает вызываться вместо того, что указан выше? Всё просто. Приведённый выше метод является оригинальным методом core-класса фреймворка Laravel.

А **LoginController** является классом, наследующим оригинальный. Таким образом он позволяет создавать врапперы (от «wrapper» — оболочка, обёртка) стандартных методов с пользовательским кодом внутри.

Нужно сказать, что данный механизм, который встречается в Laravel сплошь и рядом, очень удобный и практичный, т.к. позволяет легко и быстро переопределять базовые конструкции, не шарясь и, не дай Бог, не изменяя код движка, изменения в котором будут потеряны при первом же обновлении версии Laravel.

Если заглянуть в код файла **Auth\LoginController.php**, то в самом верху можно увидеть следующую строку, благодаря которой становится возможным использование кастомных врапперов:

|  |  |
| --- | --- |
| 1 | use Illuminate\Foundation\Auth\AuthenticatesUsers; |

Такая же строка присутствует во всех стандартных контроллерах из каталога **app\Http\Controllers\Auth**, которые являются наследниками базовых классов Laravel регистрации и аутентификации пользователей, благодаря чему в них можно описывать wrappers для всех public методов стандартных классов (найти их можно поиском по папке vendor, указывая имена методов из числа тех, что были приведены в начале статьи).

И по этому поводу держите ещё один лайфхак, связанный с кастомизацией Laravel auth методов

Если вы захотите переопределить базовые методы либо продублировать их, то достаточно создать контроллер и использовать в нём trait **AuthenticatesUsers** тем же способом, что был описан выше. Этот приём может быть полезен, если вам захочется сделать отдельные формы и механизмы Laravel аутентификации и регистрации для разных типов пользователей, которые будут храниться в разных таблицах БД.

Данный функционал актуален для marketplaces, социальных сетей и других [типов сайтов](http://cccp-blog.com/sozdanie-saytov/klassifikaciya-sajtov), обеспечивающих взаимодействие пользователей с разными атрибутами.

**Laravel аутентификация: практикум**

О базовой установке, настройке и кастомизации Laravel регистрации и аутентификации я вам рассказал. Теперь пришло время поведать, как пользоваться Laravel auth.

Рассказывать об этом я буду, естественно, на примере данного сайта.

Итак, что у нас есть на текущий момент?

Во-первых, форма логина, которая доступна по url **http://site.com/login**, и о которой я уже рассказывал и даже показывал

Кроме этого в мой проект добавилась форма Laravel регистрации, сброса пароля и тестовая страница, на которую редиректит после удачной аутентификации пользователя в системе и которая доступна по адресу **http://site.com/home**.

Поскольку следующим этапом создания моего тестового сайта будет установка админки, в которой будет работать только администратор, то я буду отключать регистрацию в Laravel приложении.

А также мне не нужен будет стандартный для Laravel сброс пароля.

Кроме того, мне не нужна тестовая страница, поэтому вместо неё я сделаю страницу-заглушку, которая будет доступна по адресу **http://site.com/admin** и на которую будет происходить редирект после удачной аутентификации. Также я хочу сделать редирект на страницу логина Laravel при выходе пользователя из системы (logout).

Ну, и ещё, как я уже говорил, я хочу сделать аутентификацию не по значению поля email, а по полю login, для чего мне понадобятся определённые настройки.

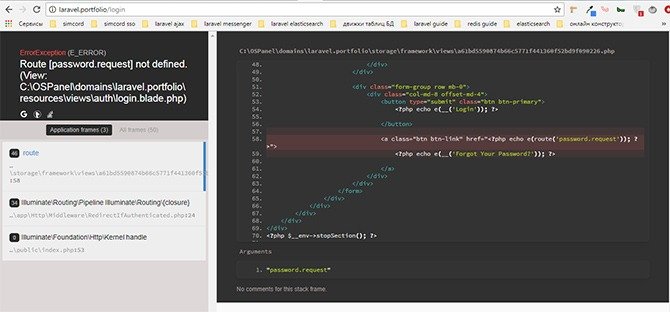
План действий составлен — за дело

Первым делом я удаляю Laravel Blade шаблоны страниц сброса пароля и регистрации, которые находятся в каталоге **корень\_сайта\resources\views\auth\passwords** и файле **корень\_сайта\resources\views\auth\register.blade.php**соответственно.

Теперь нужно удалить их роуты. Для этого я убираю в файле **routes\web.php** строку с обозначением стандартных Laravel роутов маршрутизации **Auth::routes();** и заменяю её на одинарные правила маршрутизации для отображения формы логина, самого запроса на логин в системе и удаления сессии пользователя при logout:

|  |  |
| --- | --- |
| 1  2  3  4 | //Кастомные правила маршрутизации  Route::get('login', 'Auth\LoginController@showLoginForm')->name('login');  Route::post('login', 'Auth\LoginController@login');  Route::post('logout', 'Auth\LoginController@logout'); |

После этого при переходе на страницу логина я получил следующую ошибку:



Причина её заключается в том, что роут и шаблон формы регистрации я удалил, а вызов роута в коде приложения остался, поэтому для устранения ошибки необходимо удалить из файла **корень\_сайта\resources\views\auth\login.blade.php**следующую строку:

|  |  |
| --- | --- |
| 1  2  3 | <a class="btn btn-link" href="{{ route('password.request') }}">      {{ \_\_('Forgot Your Password?') }}  </a> |

После этого подобная ошибка возникла снова, теперь уже из-за вызова в шаблоне Laravel auth форм**корень\_сайта\resources\views\layouts\app.blade.php** роута регистрации, ссылку на который я тоже удалил:

|  |  |
| --- | --- |
| 1 | <li><a class="nav-link" href="{{ route('register') }}">{{ \_\_('Register') }}</a></li> |

*Вывод: если вы используете вызов именованных роутов в коде Laravel сайтов, то при удалении их из файла маршрутизации во избежание описанных фейлов не забывайте удалять ссылки на них в коде, особенно из blade-шаблонов, которые на этапе компиляции в обычный HTML код будут сыпать errors*

После произведения вышеописанных действий при обращении к url **http://site.com/register** сайт стал отдавать 404 ошибку HTTP запроса, чего я и добивался.

Также попутно я удалил все контроллеры из каталога **корень\_сайта\app\Http\Controllers\Auth**, оставив только **LoginController.php**, который по умолчанию содержит все необходимые для моего сайта роуты.

Следующим действием с Laravel auth стало изменения данных для аутентификации на сайте. Вместо **email** я решил логинить пользователя по его **login**.

Для этого я в контроллере **корень\_сайта\app\Http\Controllers\Auth\LoginController.php** добавил следующую функцию, которая переопределяет стандартный метод username(), возвращающий название поля таблицы БД с пользователями, по которому нужно проверять юзеров на сайте:

|  |  |
| --- | --- |
| 1  2  3  4 | public function username()  {      return 'login';  } |

Также вносим коррективы в файл формы входа, чтобы настроить поле email HTML формы для ввода в него логина. Для этого в login.blade.php меняем данный код:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <div class="form-group row">      <label for="email" class="col-sm-4 col-form-label text-md-right">{{ \_\_('E-Mail Address') }}</label>        <div class="col-md-6">          <input id="email" type="email" class="form-control{{ $errors->has('email') ? ' is-invalid' : '' }}" name="email" value="{{ old('email') }}" required autofocus>            @if ($errors->has('email'))          <span class="invalid-feedback">              <strong>{{ $errors->first('email') }}</strong>          </span>          @endif      </div>  </div> |

На следующий:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <div class="form-group row">      <label for="login" class="col-sm-4 col-form-label text-md-right">{{ \_\_('Login') }}</label>        <div class="col-md-6">          <input id="login" class="form-control{{ $errors->has('login') ? ' is-invalid' : '' }}" name="login" value="{{ old('login') }}" required autofocus>            @if ($errors->has('login'))          <span class="invalid-feedback">              <strong>{{ $errors->first('login') }}</strong>          </span>          @endif      </div>  </div> |

После этого я спокойно зашёл на сайт под логином admin, что мне и нужно было.

Кстати, в Laravel из коробки реализован механизм блокирования попыток входа при слишком частом вводе некорректных данных.

На английском языке этот механизм называется **Login Throttling** (дословно по-русски это звучит примерно как «подавление попыток аутентификации»), который в Laravel реализован кодом трейта **Illuminate\Foundation\Auth\ThrottlesLogins**, используемым по умолчанию в дефолтном контроллере Laravel аутентификации **app\Http\Controllers\Auth\LoginController**через используемый в нём trait **Illuminate\Foundation\Auth\AuthenticatesUsers**.

По умолчанию попытки входа блокируются на 1 минуту при вводе неверных кредов 5 раз подряд. Думаю, не стоит говорить, зачем Laravel Throttling нужен? Если всё таки не догадались — для блокирования попыток автоматического ввода данных пользователей с целью подбора реальных комбинаций для взлома сайта.

Блокировка уникальная для набора данных пользователя, состоящего из логина/email адреса и IP адреса, с которого происходит вход.

Механизм мощный и проверенный, но лучше всё-таки усилить форму входа старой-доброй [капчей](http://cccp-blog.com/sozdanie-saytov/chto-takoe-kapcha), например, Google Recaptcha, которую я обязательно прикручу в будущем и к своему тестовому сайту.

Идём дальше.

Следующим этапом является создание заглушки для админки, которую будут видеть прошедшие аутентификацию пользователи. Сделать я её решил на базе шаблона **корень\_сайта\resources\views\home.blade.php**, который переименовал в **admin.blade.php**, внутри которого разместил следующий код:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @extends('layouts.app')    @section('content')  Добро пожаловать в админку, {{ Auth::user()->login }}  @endsection |

Кроме приветственной надписи я решил ещё и вывести логин аутентифицированного пользователя, чтобы мельком продемонстрировать методы фасада Laravel Auth, с помощью которых мы можем манипулировать информацией залогиненных юзеров. Более полный их список, а также Laravel helpers для работы с данной информацией я приведу немного позже.

Итак, вместе с переименованием шаблона, я также изменил соответствующее правило маршрутизации, которое выглядело так:

|  |  |
| --- | --- |
| 1 | Route::get('/home', 'HomeController@index')->name('home'); |

А стало таким:

|  |  |
| --- | --- |
| 1 | Route::get('/admin', 'AdminController@index')->name('admin'); |

Также, вслед за роутом и шаблоном, я переименовал и контроллер с HomeController на AdminController. Код его я приводить не буду, т.к. он, за исключением имени класса и названия вызываемого в методе **index()** шаблона, не изменился.

И ещё, единственное, я решил удалить использование [Laravel Middleware](http://cccp-blog.com/laravel/laravel-middleware) в контроллере и вынести его в файл маршрутизации. Для этого я удалил метод **\_\_construct()** из контроллера AdminController, а для роутов админки создал группу в web.php, включив в неё пока что единственный маршрут для главной страницы админки:

|  |  |
| --- | --- |
| 1  2  3 | Route::group(['middleware' => 'auth'], function(){      Route::get('/admin', 'AdminController@index')->name('admin');  }); |

В дальнейшем все маршруты админки, которые будут добавляться в приложение, будут размещаться в данной группе. Таким образом, перечисленные запросы смогут отправлять только аутентифицированные пользователи.

Здесь же, кстати, можно применять и создаваемые Laravel auth guards, указывая их следующим образом:

|  |  |
| --- | --- |
| 1  2  3 | Route::group(['middleware' => 'auth:guard'], function(){      ...  }); |

Или при указании Laravel auth middleware в контроллере можете использовать аналогичный вариант:

|  |  |
| --- | --- |
| 1  2  3  4 | public function \_\_construct()  {      $this->middleware('auth:api');  } |

Если описанное выше действие было субъективным и, в принципе, необязательным (я говорю о создании группы роутов), то следующий шаг необходим для редиректа на созданный «админский» роут после успешного входа в систему, который заключается в изменении стандартного Laravel Middleware **RedirectIfAuthenticated**.

Описан он в файле **корень\_сайта\app\Htpp\Middleware\RedirectIfAuthenticated.php**.

В нём необходимо подкорректировать главный метод **handle()**, указав в нём роут **admin** вместо **home**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public function handle($request, Closure $next, $guard = null)  {      if (Auth::guard($guard)->check()) {          return redirect('/admin');      }        return $next($request);  } |

Также для уверенности указываем наш новый роут свойству redirectTo класса LoginController (хотя на практике достаточно всего одного из указанных способов):

|  |  |
| --- | --- |
| 1 | protected $redirectTo = '/admin'; |

Тестим: всё шик  То, что я хотел, получилось — после аутентификации на сайте автоматически происходит перенаправление на url **http://site.com/admin**.

Единственный шаг моего плана кастомизации стандартного механизма Laravel аутентификации на моём сайте, который остался нереализованным — это аналогичный редирект при логауте пользователя только его нужно сделать на страницу логина, а не на главную страницу админки.

В принципе, такой вариант всё равно приведёт к желаемому результату, но вместе с этим у нас добавится один бессмысленный редирект со страницы админки на экран входа, т.к. пользователь на этот момент не будет аутентифицирован.

Итак, для задуманного мне нужно написать ещё один кастомный обработчик стандартного метода Laravel **logout()** и разместить его в том же LoginController, что и предыдующие. Метод имеет следующий вид:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | //Перенаправление на страницу логина после выхода пользователя из системы  public function logout()  {      Auth::logout();      return redirect('/login');  } |

Смысл его кода примерно таков: делаем выход пользователя из системы и после этого производим перенаправление на страницу с формой входа на сайт.

Также у Laravel имеется ещё один хак, связанный с редиректами и authentication. Он заключается в том, что в приложении можно указывать адрес (или действие), на который будет перенаправлен незалогиненный пользователь при попытке перейти на url, требующий аутентификации юзера.

По умолчанию Laravel возвращает ответ в формате JSON с 401 кодом при AJAX запросах и редиректит на страницу с url **http://site.com/login**.

Чтобы изменить данное поведение, нужно будет добавить функцию unauthenticated в файл **корень\_сайта/app/Exceptions/Handler.php** и прописать в ней необходимые сценарии поведения примерно таким образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?php    use Illuminate\Auth\AuthenticationException;    protected function unauthenticated($request, AuthenticationException $exception)  {      return $request->expectsJson()                  ? response()->json(['message' => $exception->getMessage()], 401)                  : redirect()->guest(route('custom\_url'));  } |

Все описанные выше изменения вы традиционно можете найти в [репозитории проекта](https://github.com/Pashaster12/Laravel5Portfolio), разрабатываемого в рамках данного цикла статей о Laravel, в коммите с хэшом **84e3efbf05000953db017a6f0c48e8e43b66b995**.

И вот на этой ноте я плавно подошёл к завершающему блоку сегодняшней статьи — к обзору методов фасада Laravel Auth и соответствующих им хэлперов (Laravel helpers), которые пригодятся при работе с данными аутентифицированных пользователей в коде приложения.

**Методы фасада Laravel Auth и хэлперы аутентификации**

Фасад Laravel Auth, как и все остальные facades, — это классы, которые предоставляют статические интерфейсы для методов других классов. Следовательно, для корректного вызова методов фасада Auth в коде пользовательского класса необходимо в самом начале класса добавить следующий код:

|  |  |
| --- | --- |
| 1 | use Illuminate\Support\Facades\Auth; |

Использовать перечисленные далее методы можно по единому сценарию:

|  |  |
| --- | --- |
| 1 | Auth::method\_name(); |

Методов у фасада Auth нереально много, на любой вкус и цвет, как говорится. Поэтому далее будут очень кратко перечислены названия методов, входные/выходные параметры, их тип и краткое описание, для чего текущий метод предназначен. Останавливаться и приводить примеры буду лишь для некоторых особо непонятных

И, естественно, что все перечисленные методы будут public, а не protected или private, т.к. при таких условиях использовать их за пределами класса не получится. Описание методов будет приведено в формате **тип\_возвращаемых\_данных имя\_метода(тип входных параметров имя\_параметра [, …])**.

Как уже говорилось ранее, сценарий работы приложения при аутентификации пользователя задаётся с помощью Laravel гвардов, у которых по умолчанию имеется два драйвера: **session** и **api**. Они реализованы в виде отдельных классов, каждый из которых содержит набор собственных методов.

Если вам будет интересно, можете самостоятельно поизучать их код, который доступен в следующих файлах:

* **директория\_сайта\vendor\laravel\framework\src\Illuminate\Auth\TokenGuard.php**
* **директория\_сайта\vendor\laravel\framework\src\Illuminate\Auth\SessionGuard.php**

А также при использовании каждого из гвардов доступны базовые методы, описанные в трейте **Illuminate\Auth\GuardHelpers**, который используют перечисленные классы Laravel guards.

Описание методов я решил начать с общих, которые будут доступны при вызове их через фасад, независимо от выбора драйвера гварда. Среди них как методы из общего трейта, так и просто те, которые присутствуют в обоих Laravel guards (реализация их, естественно, различается, детали которой вы найдёте в их исходном коде).

|  |  |
| --- | --- |
| 1 | Authenticatable|null user() |

Возвращает объект с данными текущего аутентифицированного пользователя. Если пользователь не залогинен, то происходит его аутентификация. В session Laravel Auth guard фреймворк сначала пытается аутентифицировать пользователя по его идентификатору сессии, а если таковой не будет найден, то по его зашифрованному идентификатору из cookies браузера.

При использовании token гварда же authentication сначала происходит по токену. Дальнейший сценарий идентичен Laravel auth session guard.

На базе данного метода работает большая часть перечисленных далее, в которых нужно производить проверку аутентификации в Laravel приложении.

|  |  |
| --- | --- |
| 1 | bool validate(array $credentials = []) |

Метод проверяет корректность учётных данных (креды, credentials) пользователей.

|  |  |
| --- | --- |
| 1 | Authenticatable authenticate() |

Аналог **user()**.

|  |  |
| --- | --- |
| 1 | bool check() |

Нужен для проверки аутентификации в Laravel приложении текущего пользователя. Функция Laravel Auth check() возвращает true или false в зависимости от возвращаемого функцией **user()** значения.

|  |  |
| --- | --- |
| 1 | bool guest() |

Аналог **user()**.

|  |  |
| --- | --- |
| 1 | int|null id() |

Возвращает ID текущего аутентифицированного пользователя.

|  |  |
| --- | --- |
| 1 | $this setUser(Authenticatable $user) |

Устанавливает текущего пользователя, данные которого можно будет потом получить через **user()** и **authenticate()**. **Не является аналогом Laravel Auth login()!**

|  |  |
| --- | --- |
| 1 | UserProvider getProvider() |

Возвращает объект Laravel Auth provider для текущего гварда с целью использования его в коде.

|  |  |
| --- | --- |
| 1 | void setProvider(UserProvider $provider) |

Метод позволяет задавать Auth провайдер текущему гварду из кода.

|  |  |
| --- | --- |
| 1 | $this setRequest(Request $request) |

Инициализирует текущий запрос данными переданного в метод объекта. Также для использования в коде.

В случае использования **token** гварда к данному списку добавится ещё следующий:

|  |  |
| --- | --- |
| 1 | string getTokenForRequest() |

Возвращает токен для текущего запроса.

Ну, а при использовании session Auth гварда помимо перечисленных выше методов вам также будут доступны следующие методы:

|  |  |
| --- | --- |
| 1 | Authenticatable|null getUser() |

Возвращает текущего закэшированного пользователя. Упрощённая версия user().

|  |  |
| --- | --- |
| 1 | bool viaRemember() |

Функция позволяет определить, был ли пользователь аутентифицирован с установленной галочкой «запомнить меня», при которой данные пользователя помимо переменной сессии на сервере записываются ещё и в cookie «remember me» браузера.

|  |  |
| --- | --- |
| 1 | string getRecallerName() |

Возвращает имя переменной cookie, используемой для хранения «recaller».

|  |  |
| --- | --- |
| 1 | string getName() |

Получает уникальный идентификатор переменной сессии, используемой для хранения данных аутентифицированного пользователя.

|  |  |
| --- | --- |
| 1 | Authenticatable getLastAttempted() |

Возвращает объект с данными последнего пользователя, который пытался зайти на сайт.

|  |  |
| --- | --- |
| 1 | void attempting(mixed $callback) |

Используется для регистрации слушателя (listener) события с попыткой аутентификации.

|  |  |
| --- | --- |
| 1 | $this logoutOtherDevices(string $password, string $attribute = 'password') |

Сбрасывает все переменные сессий, которые создавались для хранения его данных при входе с других устройств.

Для использования данного метода ваше Laravel приложение должно использовать **AuthenticateSession** middleware. Добавлять его нужно в группу **web** массива **$middlewareGroups** файла **корень\_сайта\app\Http\Kernel.php** следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5 | 'web' => [      // ...      \Illuminate\Session\Middleware\AuthenticateSession::class,      // ...  ], |

Про себя скажу, что на практике использование Laravel logoutOtherDevices() сильно упрощает жизнь, когда нужно изменить пароль существующему пользователю, но при этом под ним могут быть залогинены люди на других устройствах. У меня такое было с публичным аккаунтом администратора, который одновременно использовался контентщиками, менеджером и мною, естественно.

Поэтому меня вызов logoutOtherDevices() через консоль сильно выручил, предотвратив ручное удаление всех файлов сессий с сервера.

|  |  |
| --- | --- |
| 1 | void logout() |

Разлогинивает пользователя в вашем Laravel приложении.

|  |  |
| --- | --- |
| 1 | void login(Authenticatable $user, bool $remember = false) |

Аутентифицирует (логинит) пользователя на Laravel сайте. При указании значения второму параметру true для пользователя будет генерироваться remember\_token, который в дальнейшем будет записываться в cookie и использоваться для автоматической аутентификации до тех пор, пока пользователь не сбросит cookie браузера или не разлогинится самостоятельно.

|  |  |
| --- | --- |
| 1 | Authenticatable loginUsingId(mixed $id, bool $remember = false) |

Аутентифицирует пользователя в Laravel приложении по его id из таблицы БД с пользователями.

|  |  |
| --- | --- |
| 1 | bool attempt(array $credentials = [], bool $remember = false) |

Делается попытка аутентификации пользователя по его учётным данным (credentials). Данный метод можно использовать при реализации ручной аутентификации в Laravel без использования стандартного метода фасада Laravel Auth login(), если он вам по каким-то причинам не будет подходить либо если вы захотите заменить стандартный LoginController на полностью свой кастомный.

В последнем случае метод аутентификациивашего контроллера может выглядеть примерно следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?php    namespace App\Http\Controllers;    use Illuminate\Http\Request;  use Illuminate\Support\Facades\Auth;    class LoginController extends Controller  {      public function authenticate(Request $request)      {          $credentials = $request->only('login', 'password');            if (Auth::attempt($credentials)) {              // Authentication passed...              return redirect()->intended('dashboard');          }      }  } |

Смысл его кода прост: при вызове метода выделяем из объекта запроса $request значения полей, по которым происходит Laravel аутентификация пользователя в системе, и передаём их в качестве входных параметров методу Laravel attempt().

Если аутентификация данных удалась (attempt() вернул true, а не false), то происходит редирект на url, на который пытался перейти пользователь, но был перенаправлен на страницу входа с помощью Auth middlewarte, посредством метода intended().

Данному методу, кстати, можно указывать fallback url, на который будет происходить редирект в случае, если целевой url по каким-то причинам будет недоступен (например, при нехватке прав).

В метод Laravel Auth attempt(), кстати, можно передавать не только непосредственно значения полей, по которым происходит аутентификация , но и другие поля таблицы БД пользователей для формирования дополнительных условий входа.

Например, если у вас в таблице БД users есть поле is\_deleted типа boolean (true/false) для «мягкого удаления» (soft delete) пользователей, то при аутентификации через Laravel Auth attempt() вы сможете проверять, был ли удалён юзер или нет следующим образом:

|  |  |
| --- | --- |
| 1  2  3 | if (Auth::attempt(['email' => $login, 'password' => $password, 'is\_deleted' => false])) {      // действия, происходящие при успешной аутентификации  } |

Следовательно, даже если у пользователя введённые им креды будут соответствовать содержащимся в БД, но при этом он будет удалённым (is\_deleted = true), то он не будет аутентифицирован на сайте.

Поэтому Laravel Auth attempt() кроме реализации кастомных сценариев аутентификации ещё и отлично подходит для случаев обработки специфичных условий, как, например, приведённое выше, когда нужно формировать кастомные сообщения об ошибках в зависимости от введённых пользователем данных.

|  |  |
| --- | --- |
| 1 | bool once(array $credentials = []) |

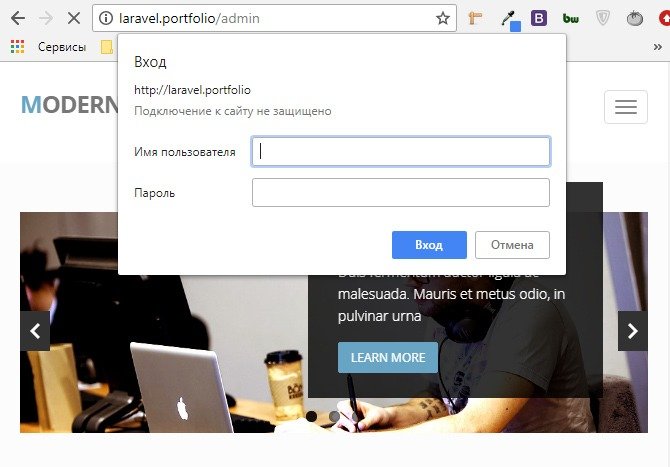
Аутентифицирует пользователя в Laravel приложении без записи данных в cookies и session. Подходит для выполнения разовых запросов, требующих аутентификации, если постоянная проверка в данном случае не нужна.

|  |  |
| --- | --- |
| 1 | bool onceUsingId(mixed $id) |

Логинит пользователя на Laravel сайте по его id из таблицы БД с пользователями без использования cookies и сессий. Как понятно из названия, — симбиоз **loginUsingId()** и **once()**.

|  |  |
| --- | --- |
| 1 | Response|null basic(string $field = 'email', array $extraConditions = []) |

Попытка входа на сайт с использованием базовой HTTP аутентификации (**HTTP Basic Auth**). Она заключается в том, что для Laravel аутентификации пользователя на сайте используется не кастомная форма входа, имеющая отдельный url, а стандартная браузерная форма, которая выглядит вот так:



Для того, чтобы настроить вход пользователя на сайт посредством данного типа аутентификации, в Laravel уже есть всё необходимое. Достаточно применить Laravel middleware auth.basic к роуту или их группе в файле маршрутизации. По умолчанию для базовой HTTP authentication в Laravel используется поле **email**.

Если вы используете PHP FastCGI, то HTTP Basic аутентификация может неправильно работать по умолчанию. В таком случае вам нужно будет добавить в файл .htaccess следующие строки:

|  |  |
| --- | --- |
| 1  2 | RewriteCond %{HTTP:Authorization} ^(.+)$  RewriteRule .\* - [E=HTTP\_AUTHORIZATION:%{HTTP:Authorization}] |

Главное, добавить код в нужный .htaccess, т.к. после [установки Laravel](http://cccp-blog.com/laravel/laravel-ustanovka) в приложении их целых два. Нам нужен тот, что находится в директории public, а точнее, на одном уровне с index.php (если вы вдруг изменили свою файловую структуру проекта).

|  |  |
| --- | --- |
| 1 | Response|null onceBasic(string $field = 'email', array $extraConditions = []) |

Производит базовую HTTP аутентификацию пользователя на сайте без сохранения его состояния (разово). Является симбиозом методов **basic()** и **once()**, о чём несложно догадаться из названия

Ну что, думаю, что выбрать вам будет из чего  В крайнем случае, если среди перечисленных методов вы не найдёте инструмента для решения вашей задачи, можете написать собственный Laravel Auth guard и добавить свой кастомный метод в него для дальнейшего использования при аутентификации и регистрации пользователей на Laravel сайте.

В завершение обзора фасада Laravel Auth хотел поделиться с вами ссылкой на описание [Laravel API](https://laravel.com/api/5.6/), а также призвать заглядывать в код движка самостоятельно для изучения принципов работы фреймворка.

По крайней мере, [при поиске работы](http://cccp-blog.com/razrabotchiku/rabota-php-programmist) в следующий раз вы сможете гордо добавить в своё резюме навык «умение разбираться в чужом коде», да и вообще способность нужная по жизни, особенно, если предстоит иметь дело с саппортом чужого кода

Ну, и последнее, что я хотел сегодня вам рассказать, это — познакомить с Laravel helper auth(), который позволяет получать экземпляр фасада Laravel Auth без прямого его подключения. Это очень удобно особенно в Blade шаблонах.

Использовать его очень просто. Следующая конструкция позволит вам получить данные аутентифицированного пользователя:

|  |  |
| --- | --- |
| 1 | auth()->user(); |

А следующая позволит вам узнать, аутентифицирован ли пользователь в системе с использованием Laravel Auth гварда **guard**:

|  |  |
| --- | --- |
| 1 | auth('guard')->check(); |

Также хотелось бы добавить, что данные аутентифицированного в Laravel приложении пользователя вы всегда можете получить через объект класса Illuminate\Http\Request в своём коде, например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?php    namespace App\Http\Controllers;  use Illuminate\Http\Request;    class UserController extends Controller  {      public function update(Request $request)      {          $request->user(); //всё равно, что вызов auth()->user, к примеру      }  } |

Вот мы и рассмотрели все public методы фасада Laravel Auth и Laravel helpers, которые позволяют манипулировать данными аутентифицированных пользователей, а также производить принудительную аутентификацию и выход из системы юзеров в коде приложения.

В качестве резюме по сегодняшнему материалу хотелось бы сказать, и вы наверняка заметили это при обзоре функций, аутентификация в Laravel бывает трёх типов:

* Кратковременная через переменные сессии на сервере (доступны через суперглобальный массив **PHP $\_SESSION**), которая по умолчанию в PHP длится 1440 секунд или 24 минуты (задаётся в переменной **session.gc\_maxlifetime**главного PHP конфига **php.ini**);
* Долгосрочная через cookies браузеров клиентов, в которые записывается значение remember\_token, генерируемого при аутентификации клиента на сайте с определёнными настройками;
* Разовая без записи cookies и session по кредам пользователя, которая годится только для выполнения отдельных запросов, требующих аутентификации.

Также хочется напомнить, что сценарии аутентификации и регистрации пользователя на сайте задаются кодом Laravel гвардов и Laravel Auth providers, которые можно создавать самостоятельно и использовать их набор функций для формирования логики приложения.

На этом сегодня всё, [подписывайтесь на обновления](http://eepurl.com/duAlVH), ждите следующих статей и оставляйте свои отзывы о том, что понравилось/не понравилось или можно улучшить в комментариях.

Ну, и самое главное, — развивайтесь и давайте такую возможность мне своими комментариями, где кроме перечисленного выше вы всегда можете поделиться своим опытом и подсказать идеи мне для новых статей.

Удачи и до новый встреч

**P.S.**: если вам нужен сайт либо необходимо внести правки на существующий, но для этого нет времени и желания, могу предложить свои услуги. **Более 5 лет опыта** профессиональной разработки сайтов. Работа с **PHP**, **OpenCart**, **WordPress**, **Laravel**, **Yii**, **MySQL**, **PostgreSQL**, **JavaScript**, **React**, **Angular** и другими технологиями web-разработки. Опыт разработки проектов различного уровня: **лендинги**, **корпоративные сайты**, **Интернет-магазины**, **CRM**, **порталы**. В том числе поддержка и разработка **HighLoad проектов**. Единой ценовой политики нет, стоимость услуг устанавливается для каждого клиента индивидуально. Присылайте ваши заявки на email [cccpblogcom@gmail.com](mailto:cccpblogcom@gmail.com) - договоримся  И с друзьями не забудьте поделиться