GeekBrains

Laravel

Урок 8. Посредники. Сессии в Laravel. Аутентификация

Авторизация и идентификация пользователей. Разграничение прав доступа к информации.

Оглавление

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.

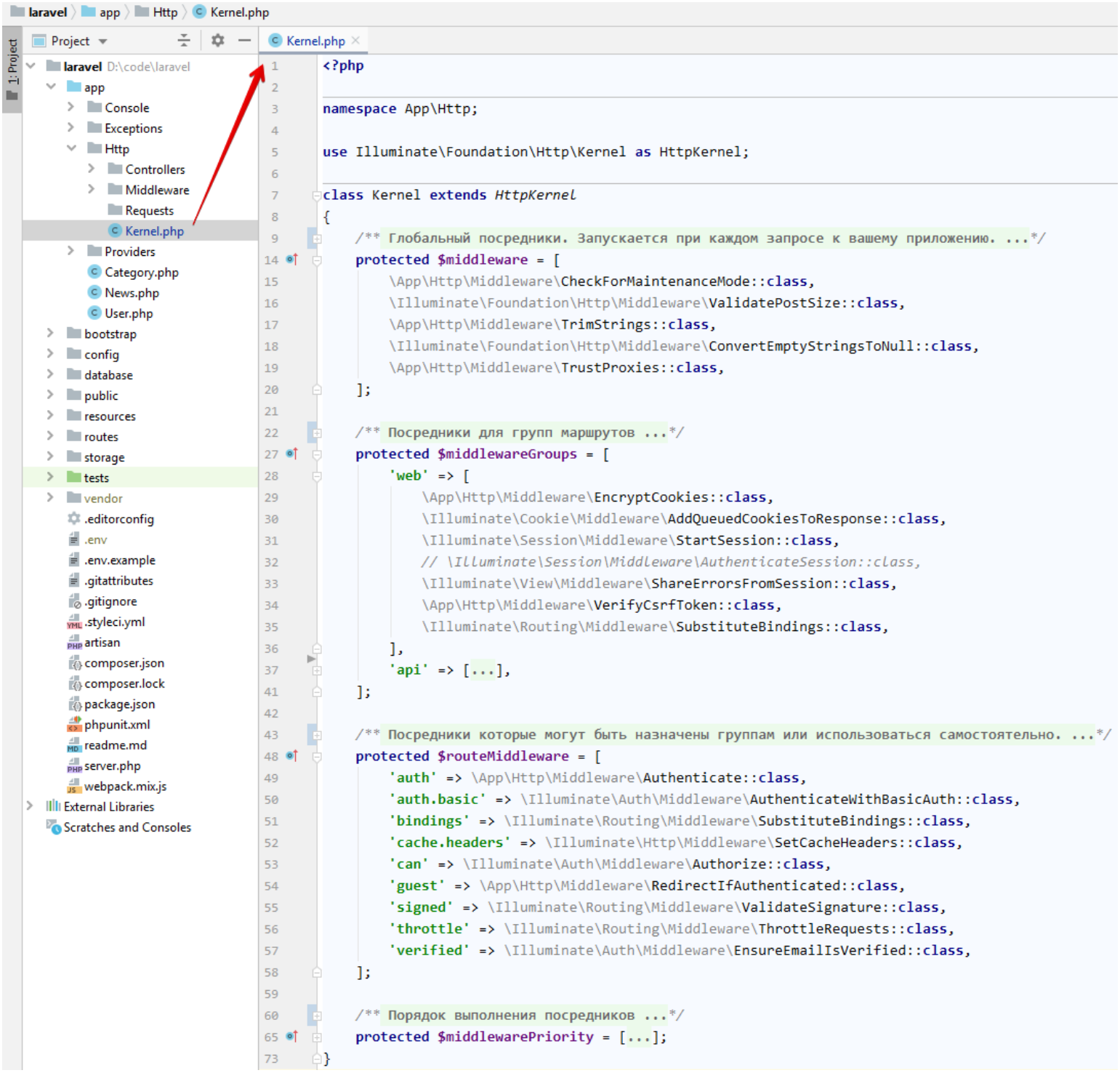
Посредники (middleware) Laravel

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**.

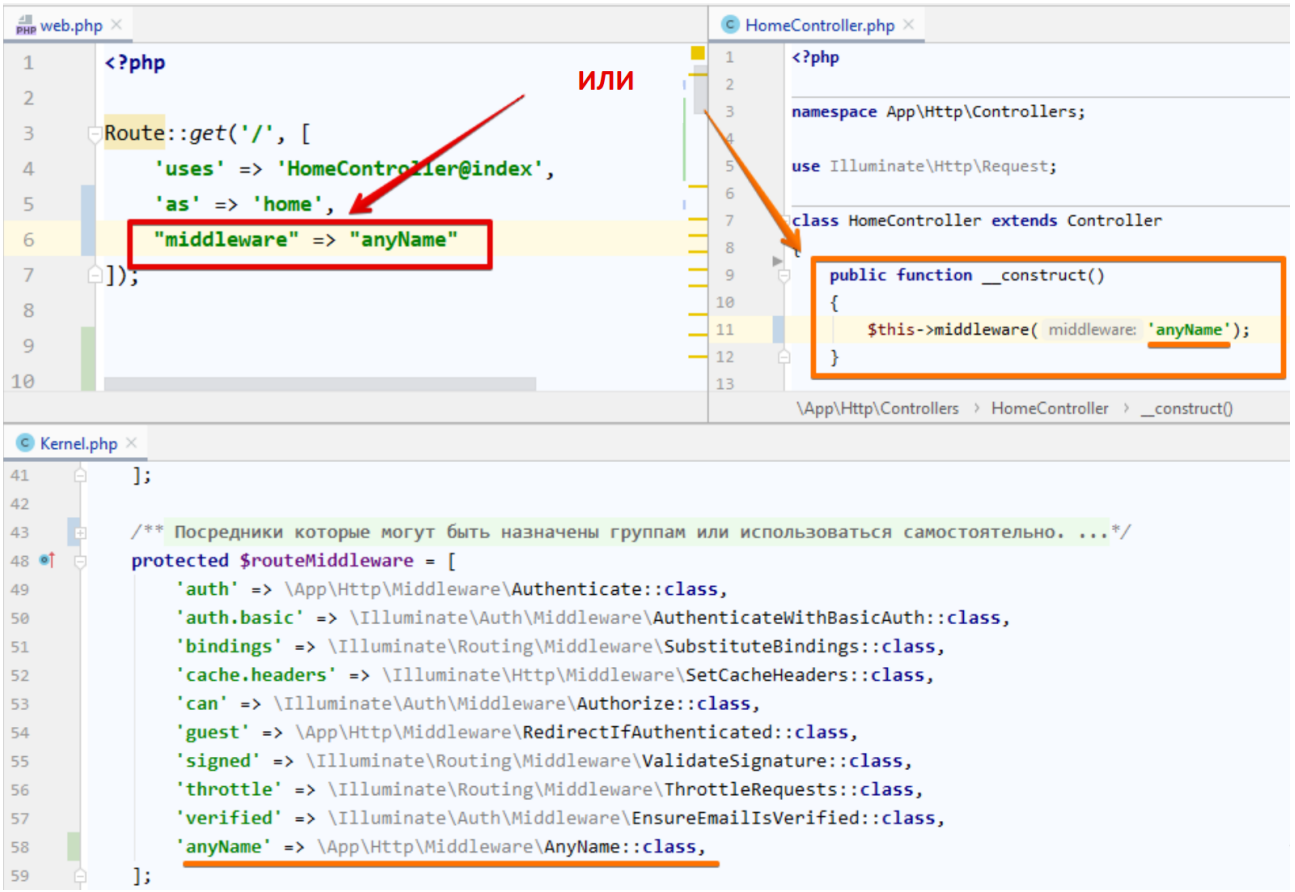
Для использования посредника его необходимо добавить в определенное свойство класса **App\Http\Kernel**.

- **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.



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.

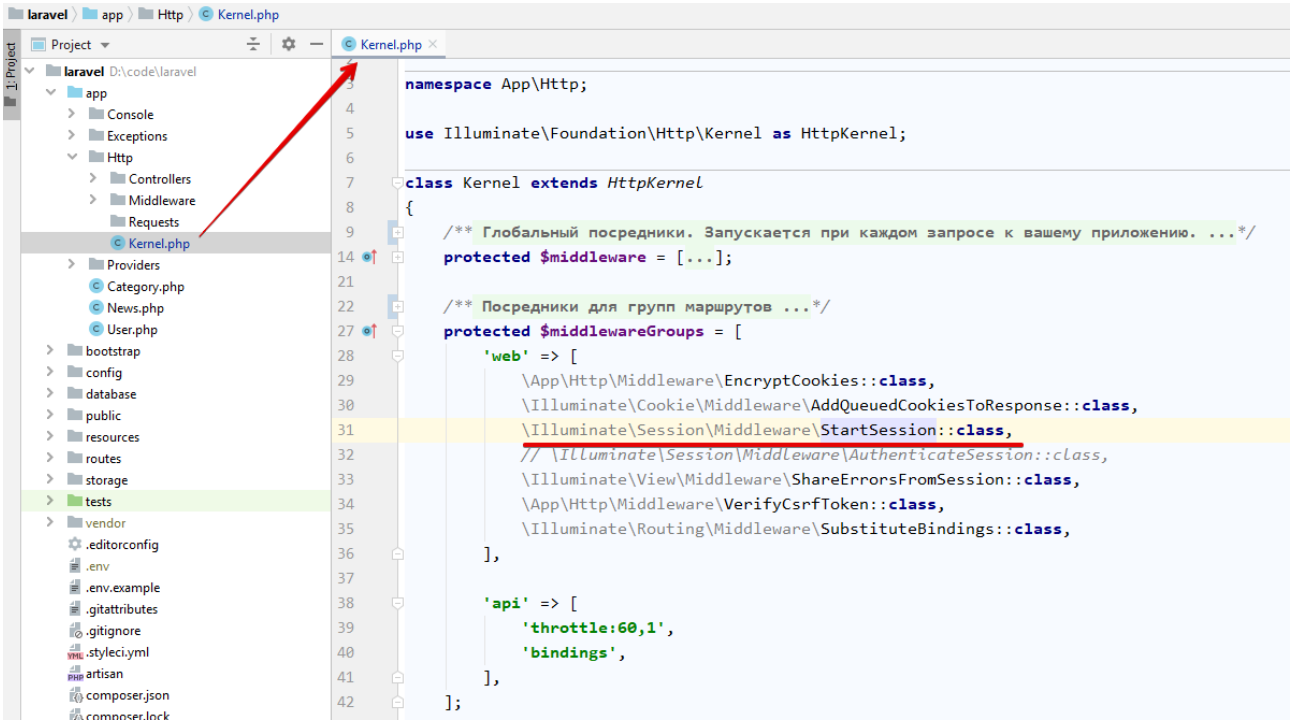


Подробнее о посредниках можно прочитать в официальной документации или в русскоязычном переводе:

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

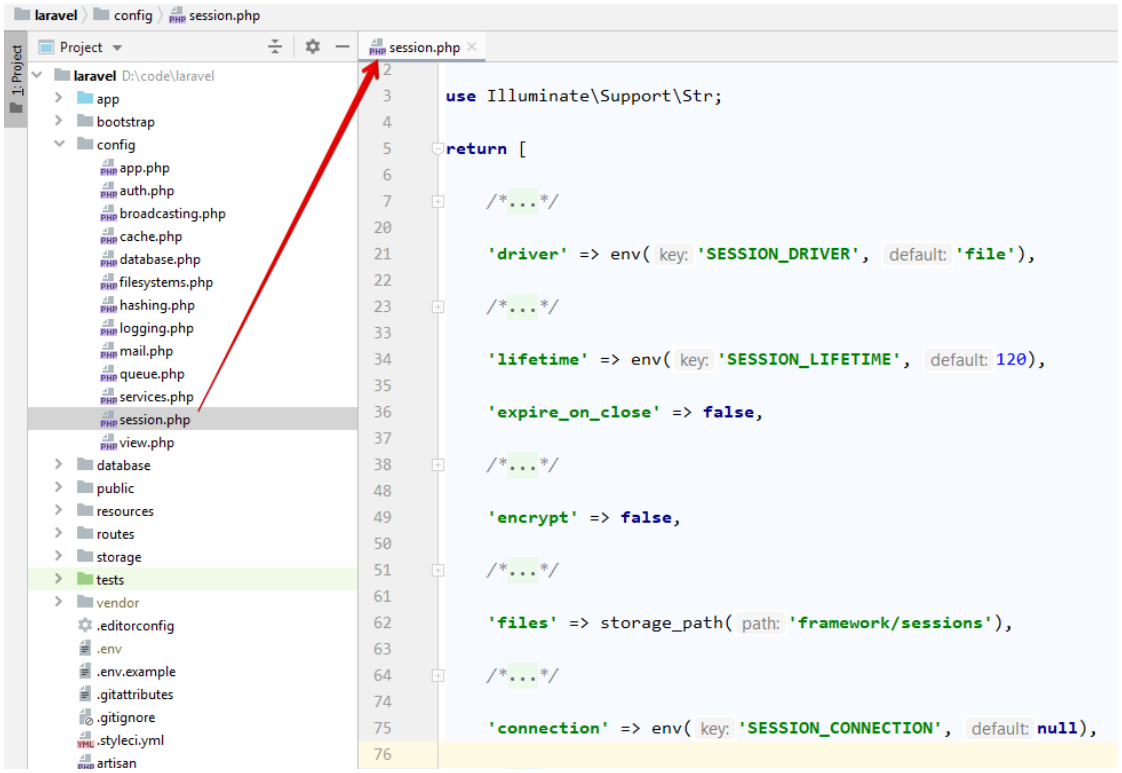
Сессии в Laravel

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**.



Что говорит название данного класса? Что он создает сессию.

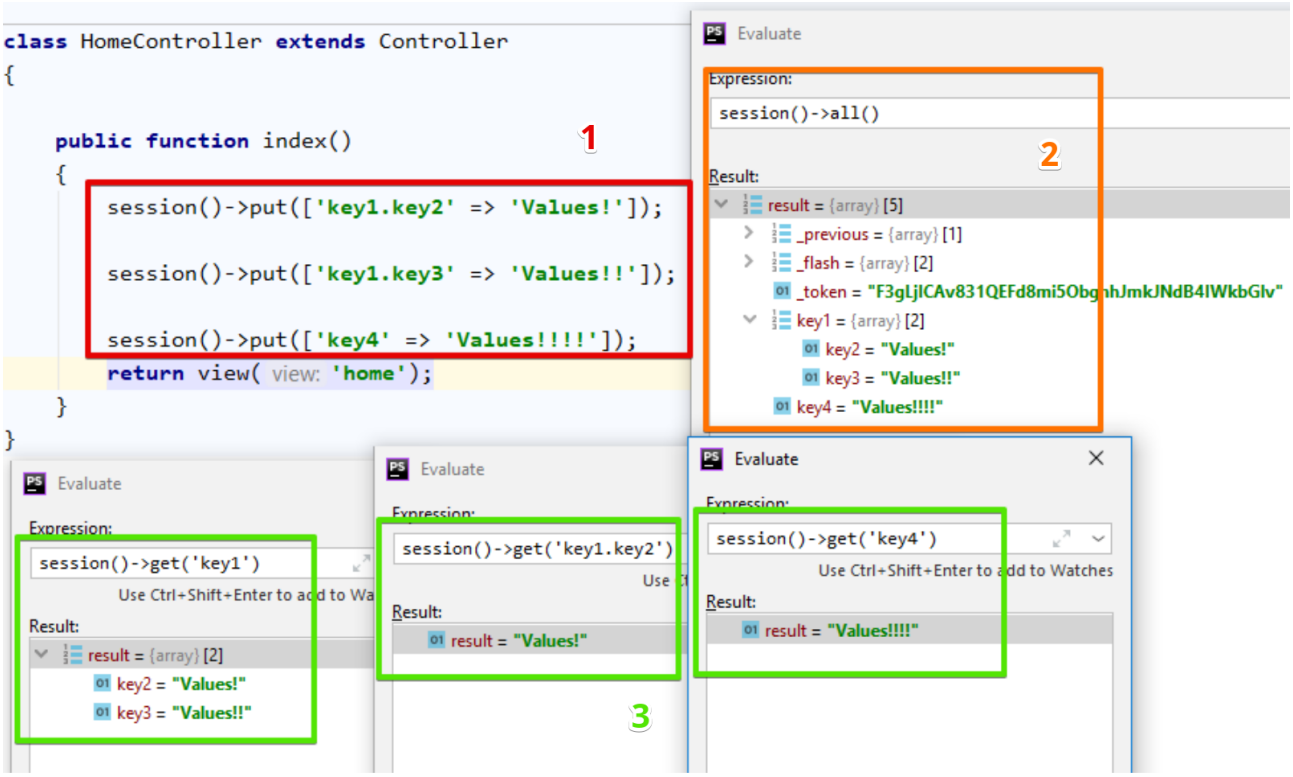
Для настроек работы с сессией используется файл **session.php** в папке **config**. Данный файл возвращает массив, в значениях которого указываются параметры работы сессии. В основном все значения вычисляются проверкой существования их в файле конфигурации приложения (**.env**), и в случае отсутствия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



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 статичными, а с хелпером — динамическими, но работают одинаково.

Рассмотрим пример:



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)

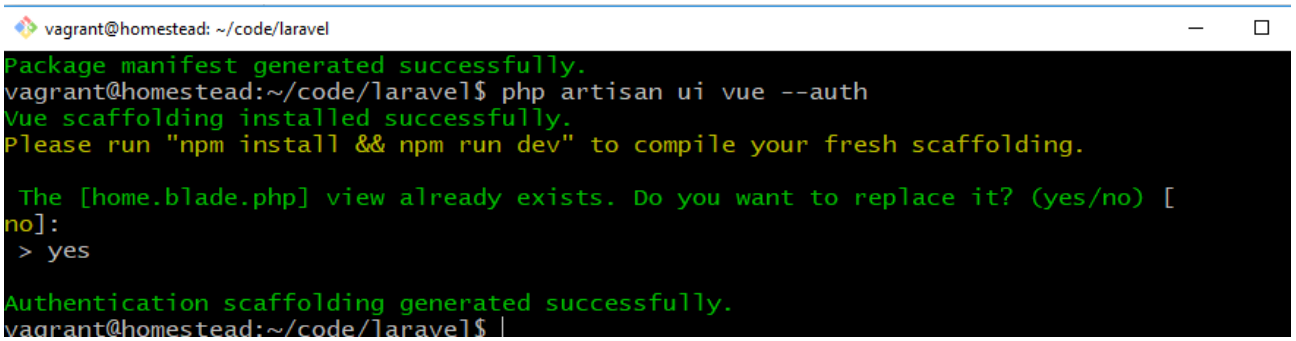
В шестой версии Laravel для красивого вывода пользовательского интерфейса требуется скачать и установить дополнительный пакет (для Laravel ниже шестой версии этого делать не нужно, поскольку он включен в базовый пакет продукта).

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**.



Добавление таблиц для сохранения данных пользователей

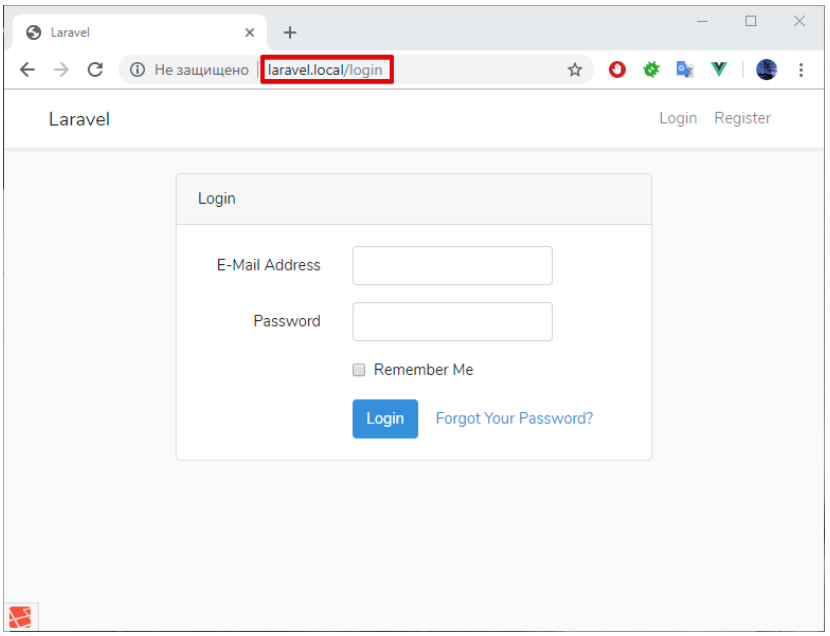
Используя стандартную аутентификацию Laravel, следует выполнить подготовленные миграции, которые были поставлены вместе с фреймворком. Команда для запуска миграций: **php artisan migrate**

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.

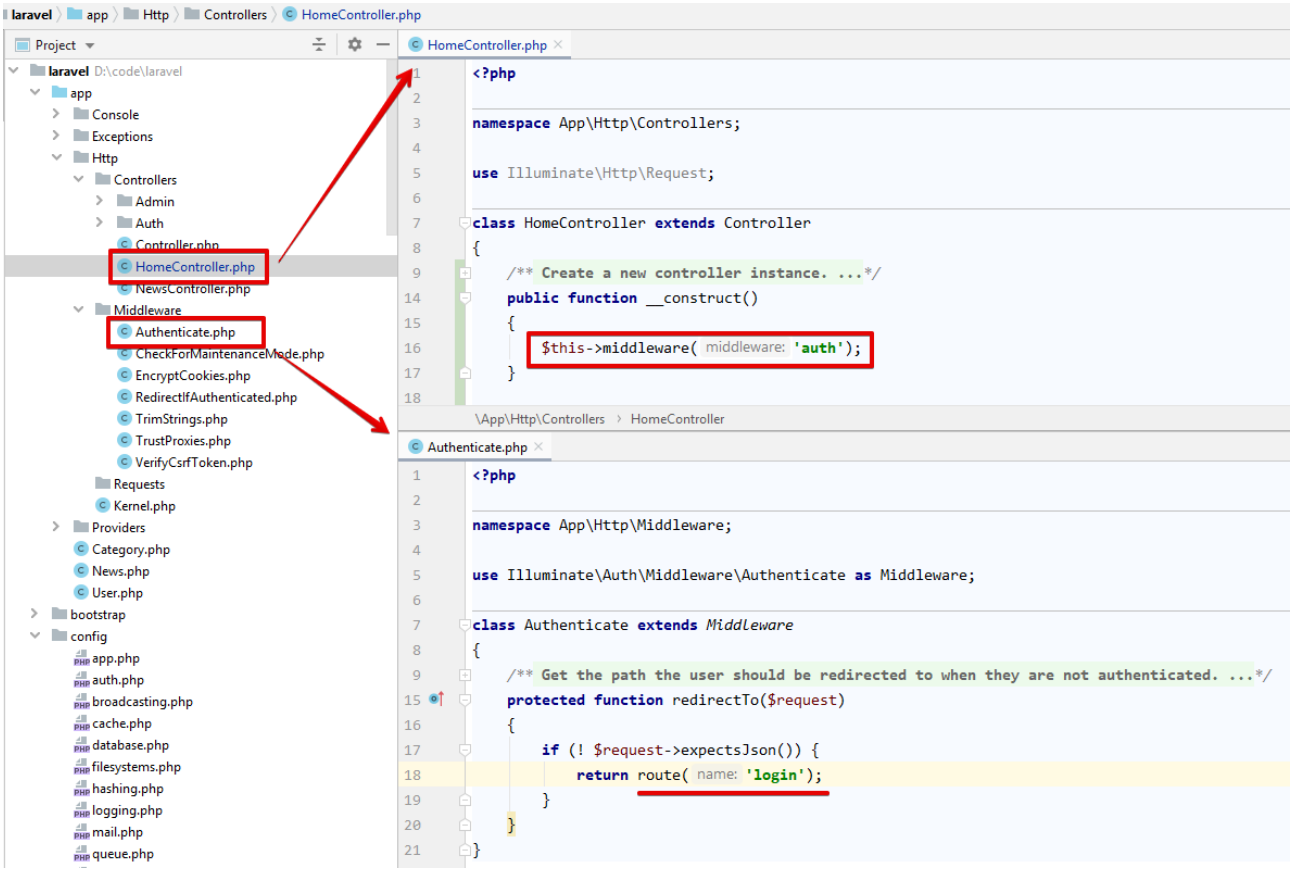


Но почему это так происходит? После успешно выполненных команд в нашем проекте произошли изменения. Рассмотрим их.

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**.

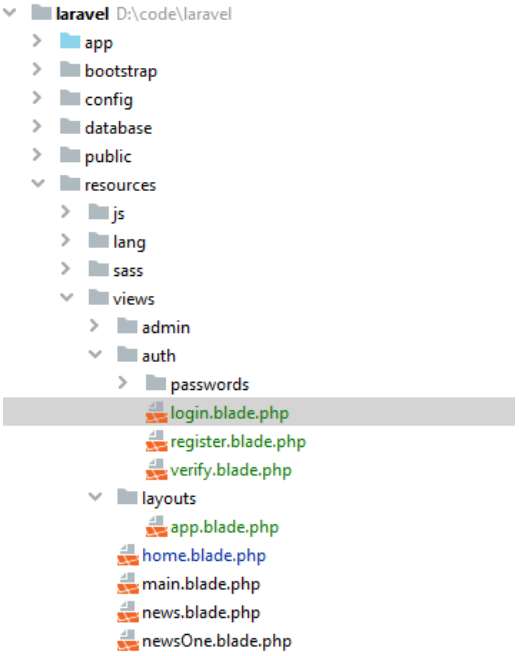
Класс **\App\Http\Middleware\Authenticate** является наследником класса **Illuminate\Auth\Middleware\Authenticate** и переопределяет метод **redirectTo**.

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

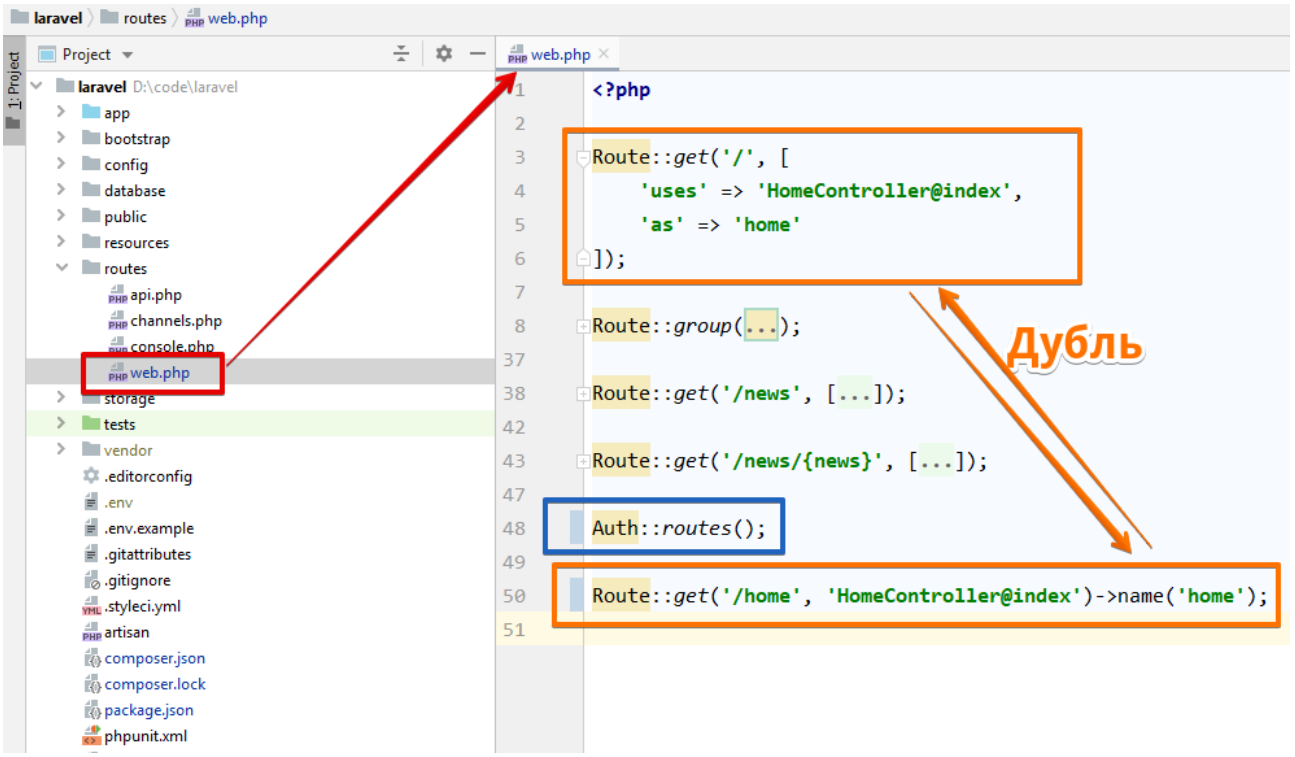


С перенаправлением разобрались. **Но откуда появились новый роут и форма, которую мы не создавали?**

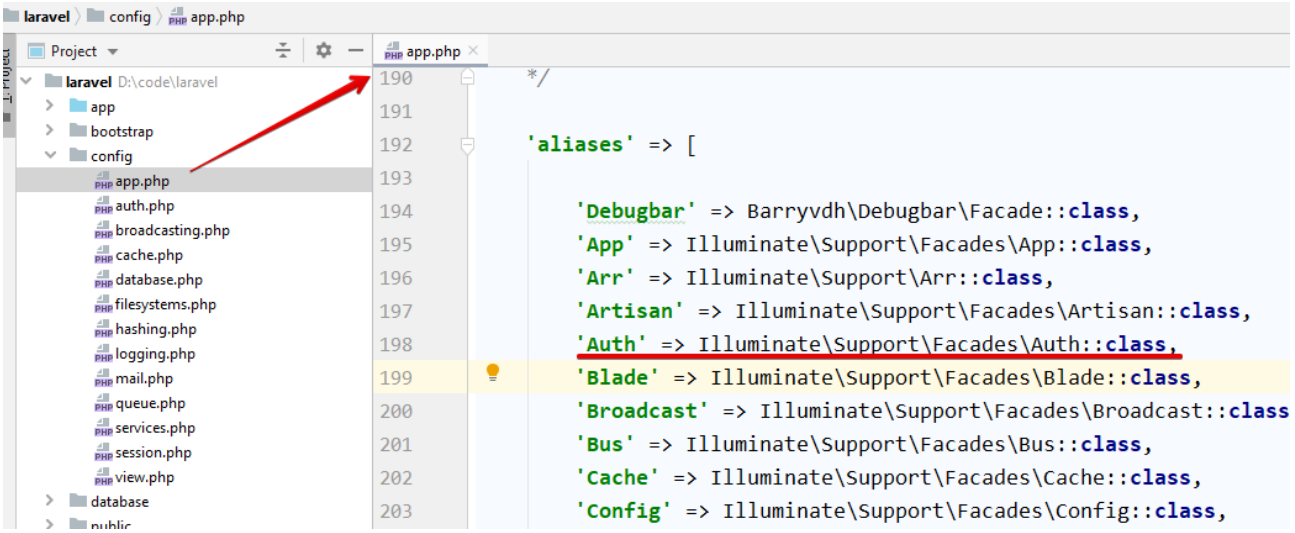
Откроем папку **resources** и заметим, что в ее подпапках появились новые шаблоны.



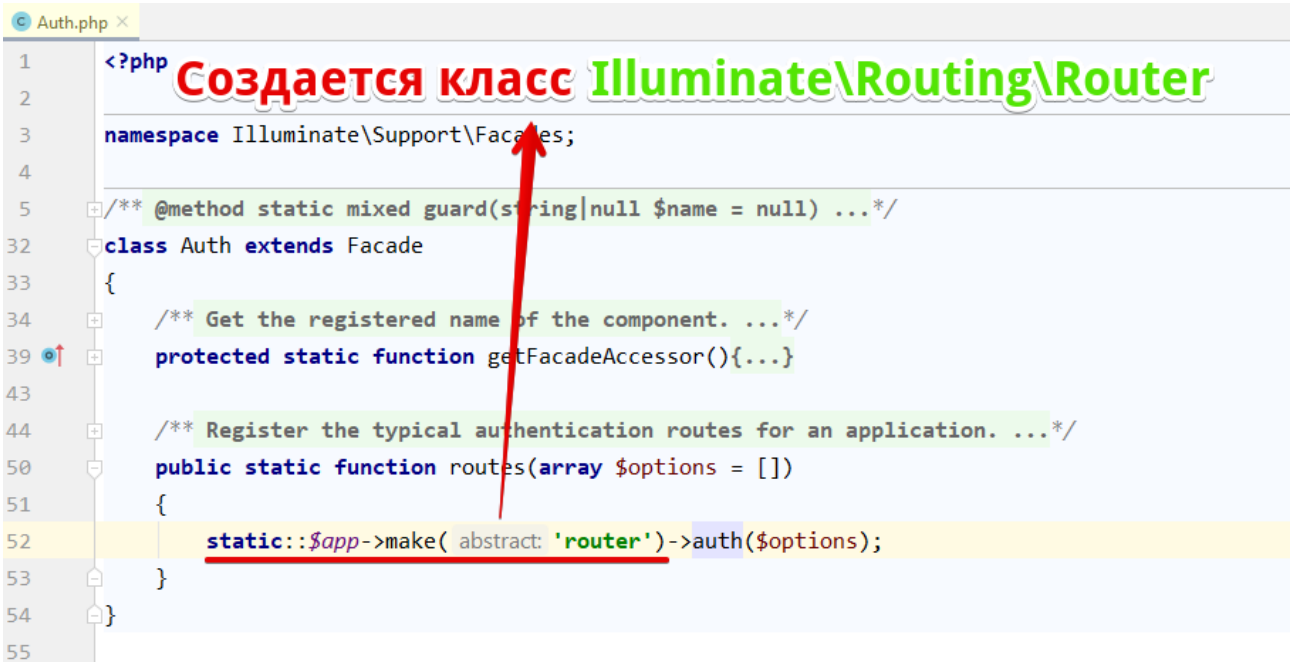
Так же отметим, что в конце файла **web.php** появились две новые строки. В первой вызывается статический метод **routes** незнакомого класса **Auth** и маршрут, который дублирует роут, ранее использованный нами.



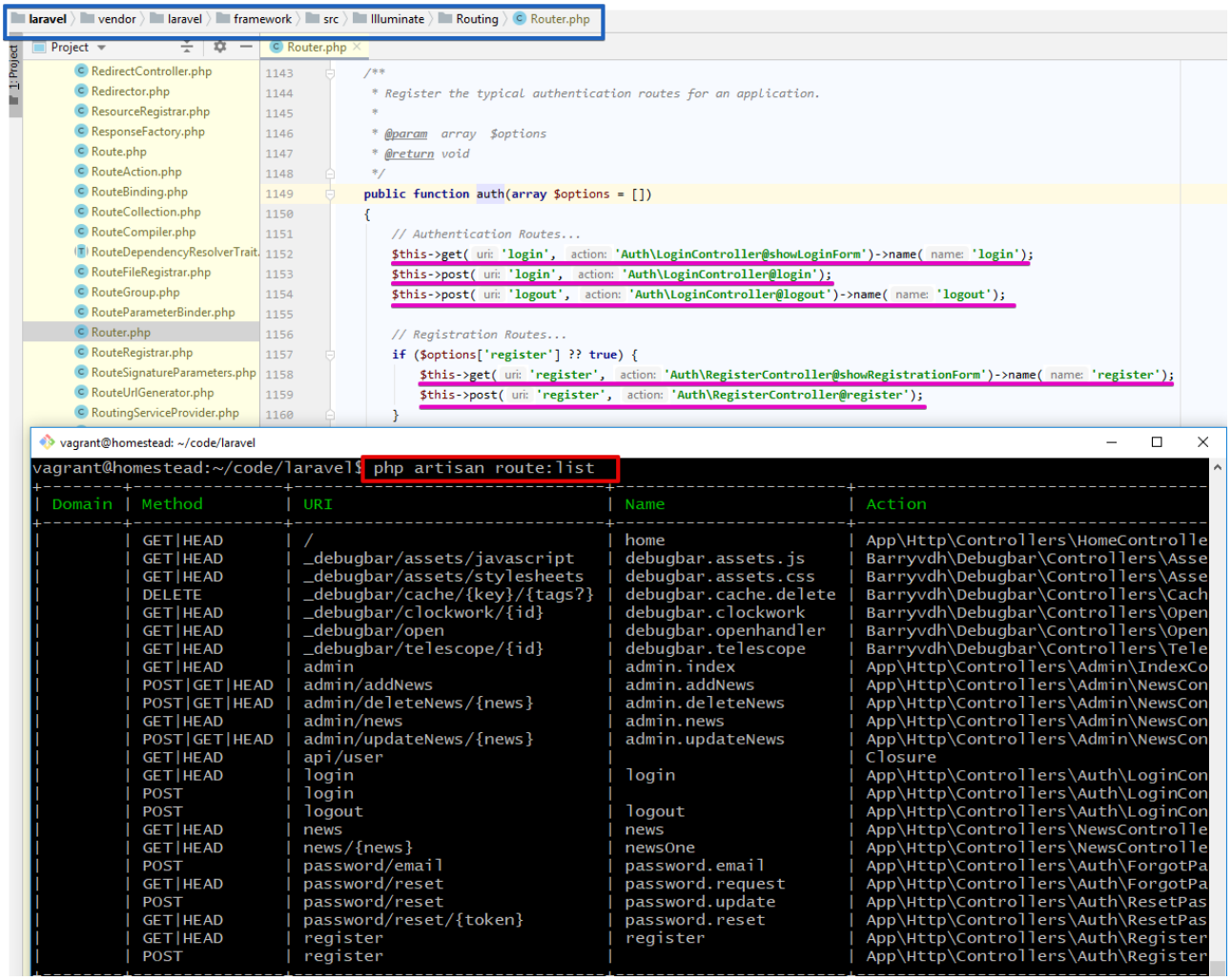
По аналогии с классом **Route** можем сделать предположение, что и класс **Auth** тоже является фасадом. Проверим это, перейдя в файл **app.php**.



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

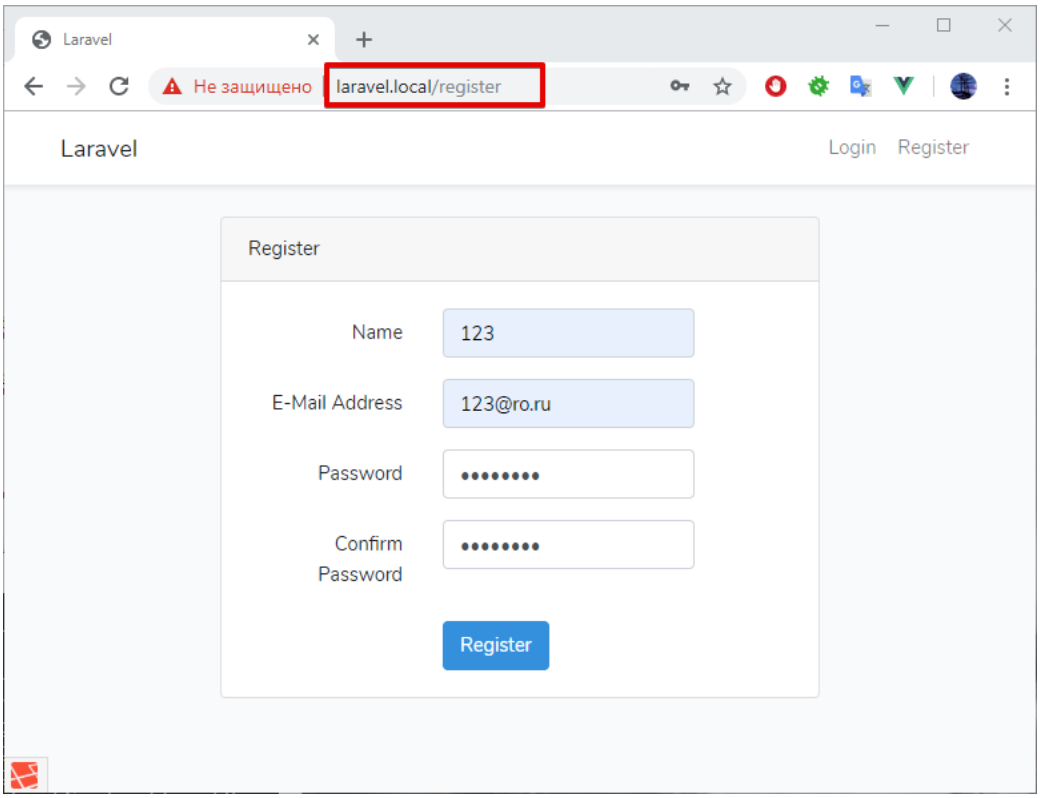


Стало ясно, что в **Auth::routes();** создает экземпляр класса **Illuminate\Routing\Router** и вызывает его метод **auth**. Теперь рассмотрим, что в нем происходит. И увидим, где спрятались новые роуты:

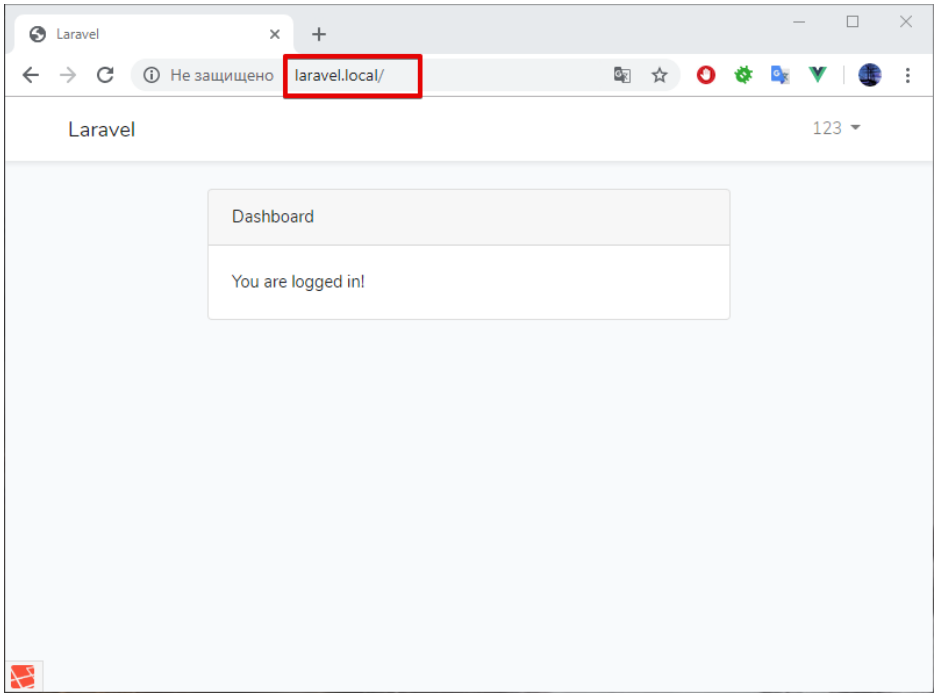


Выполнив команду **php artisan route:list**, сможем удостовериться, что такие роуты действительно используются в нашем приложении.

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

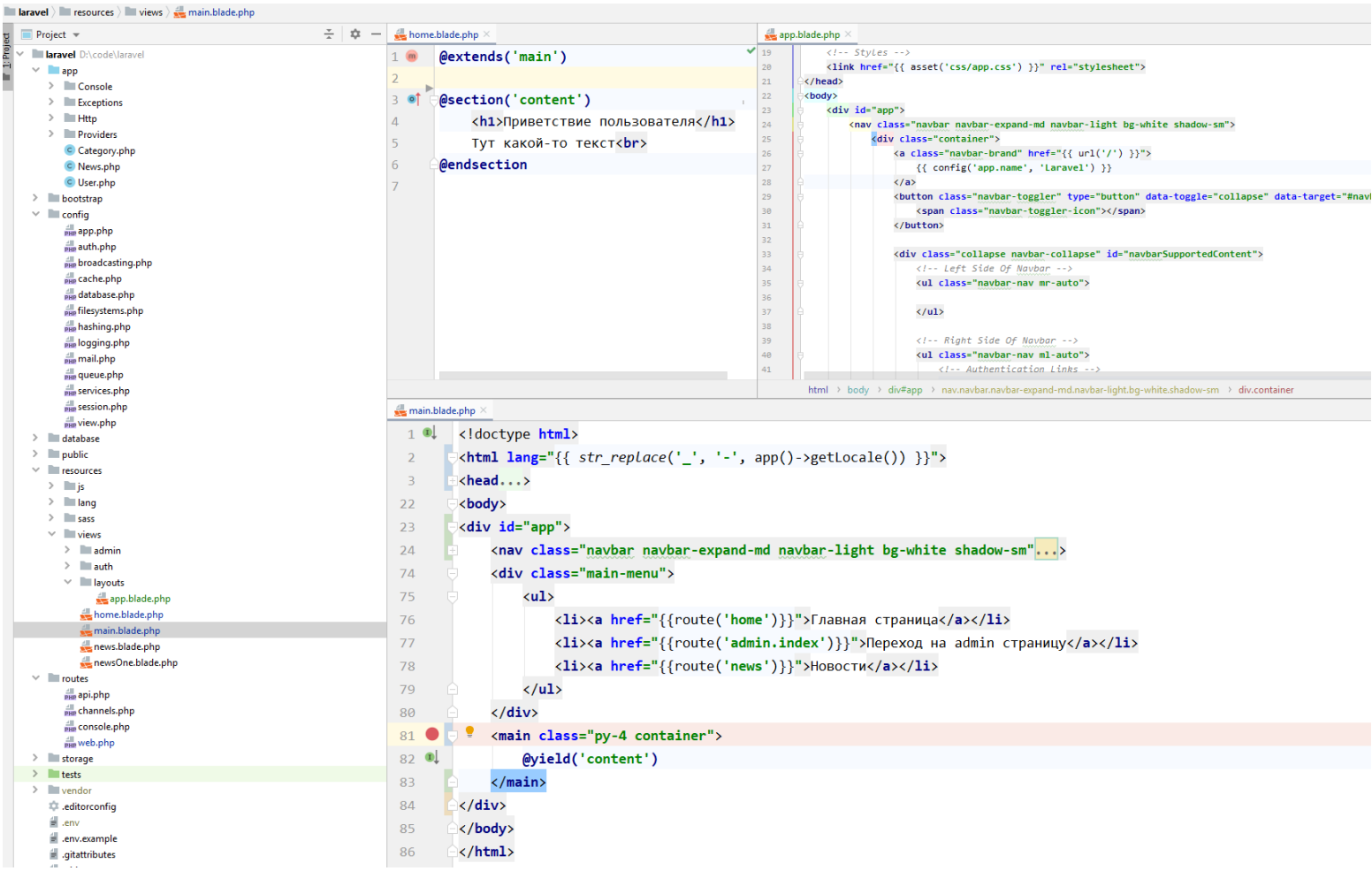


Теперь у нас есть возможность находиться на главной странице нашего приложения.

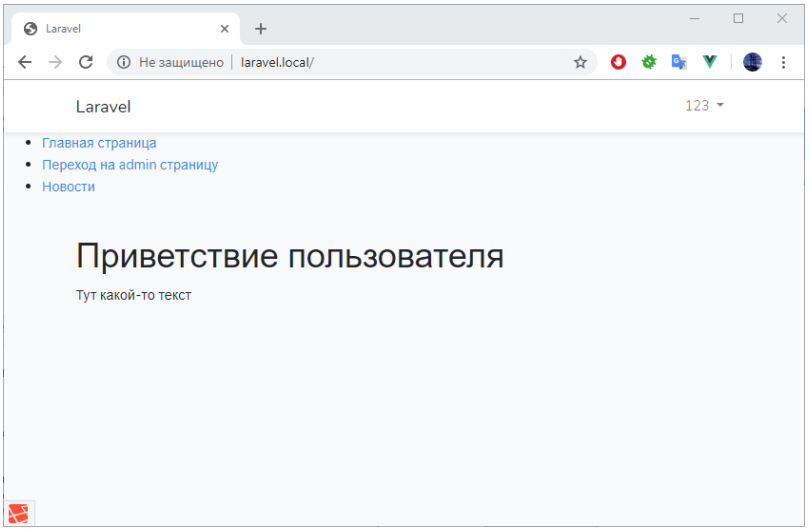


Перейдя на главную страницу, отметим, что нет ни главного меню, ни приветственного сообщения. Все это пропало, потому что при применении пользовательского интерфейса мы согласились вернуть

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

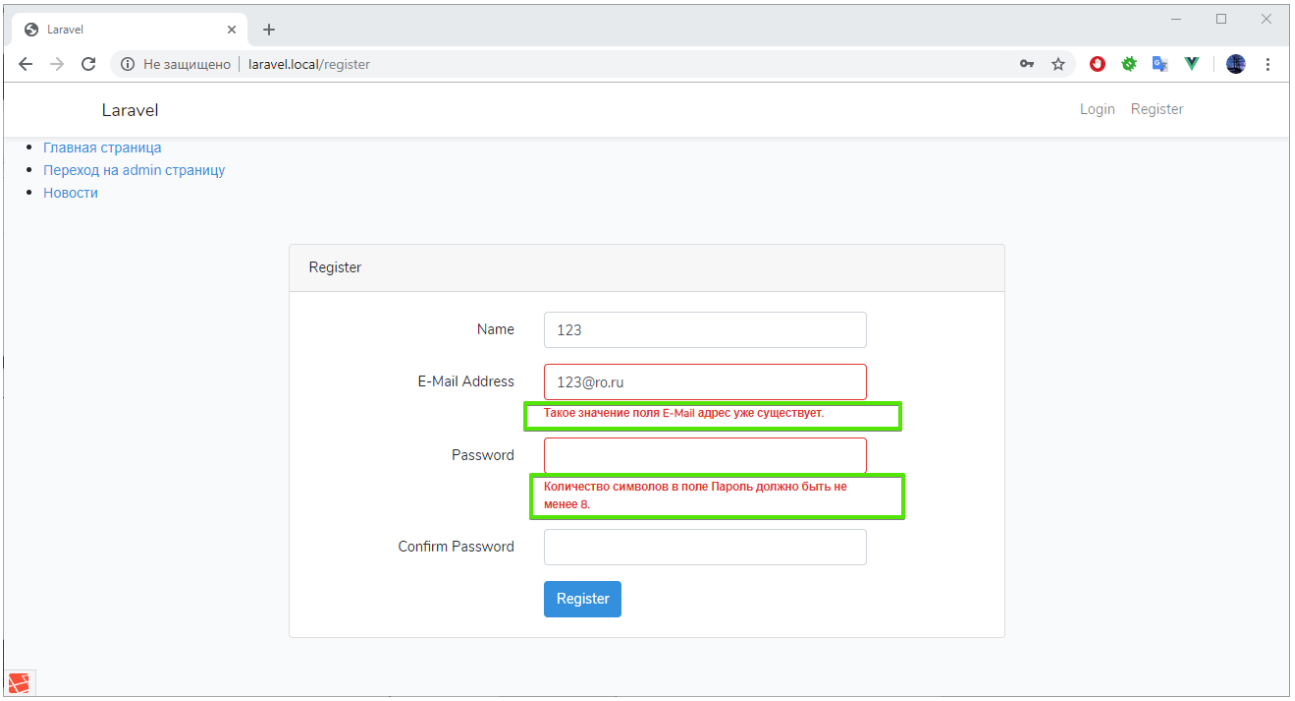


Проверим, что все хорошо, перейдя на главную страницу сайта.



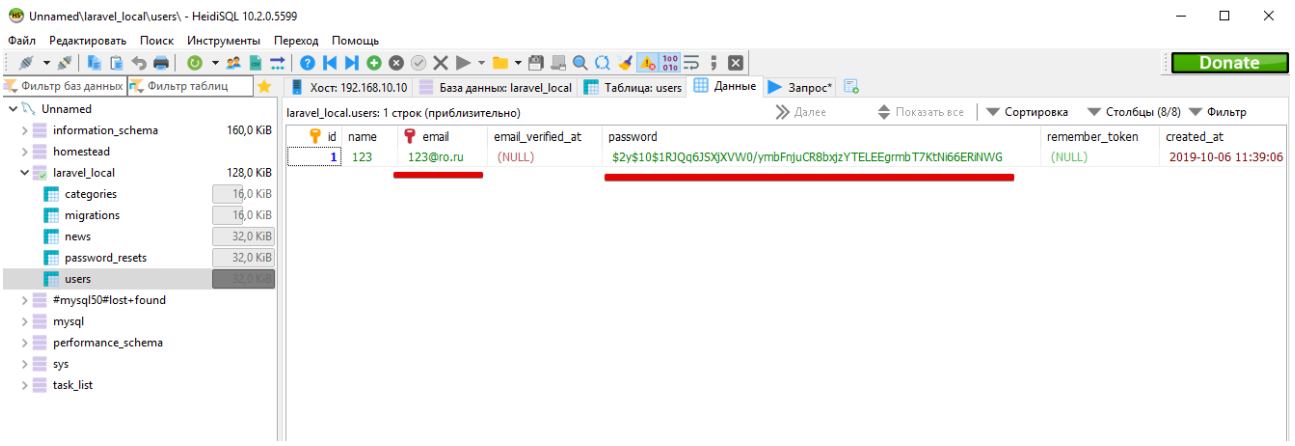
Теперь попробуем проверить встроенные плюшки в базовой аутентификации. Нажмем на имя пользователя в правом верхнем углу (в примере — 123) и в выпадающем меню выберем **Logout**.

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



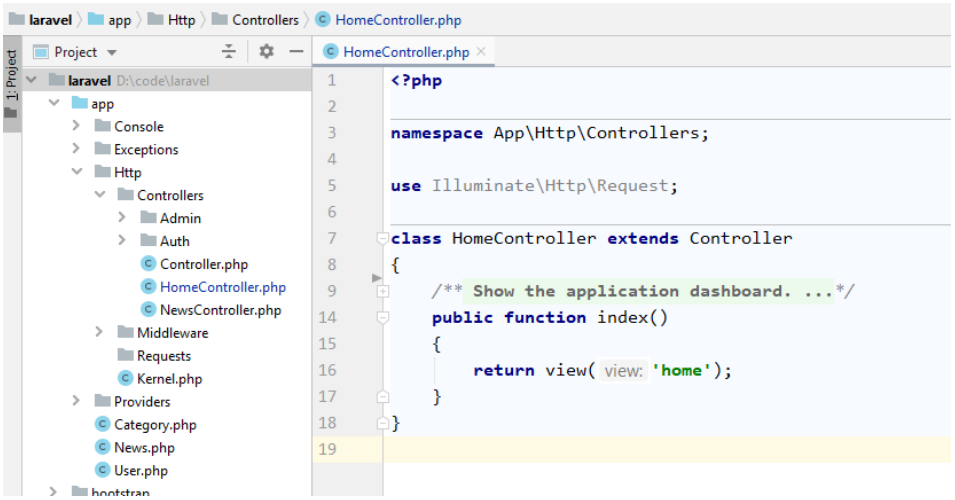
Если вы используете новый проект, то сообщения об ошибках будут выводиться на стандартном языке — английском.

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.

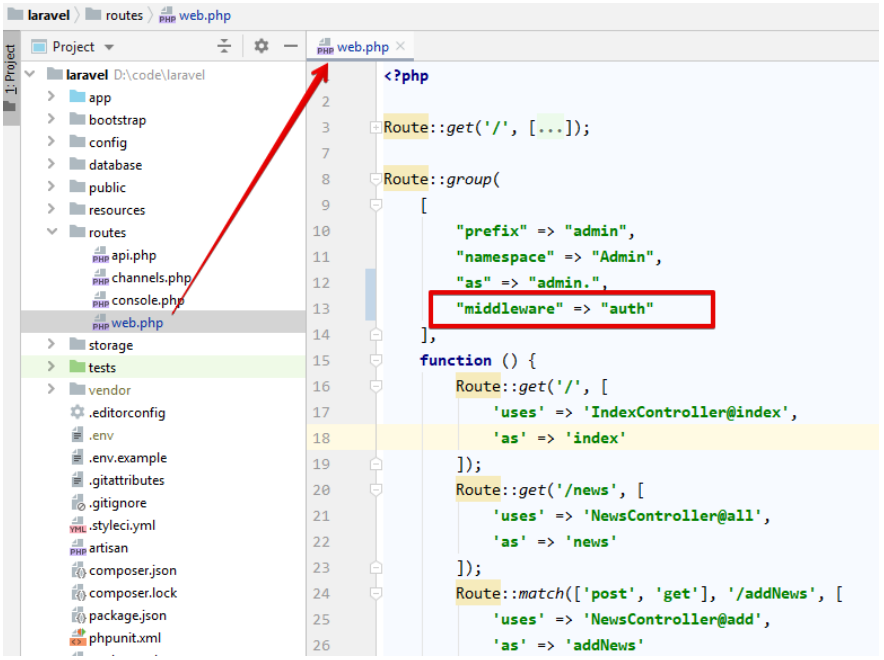


Закрываем доступ к админке сайта

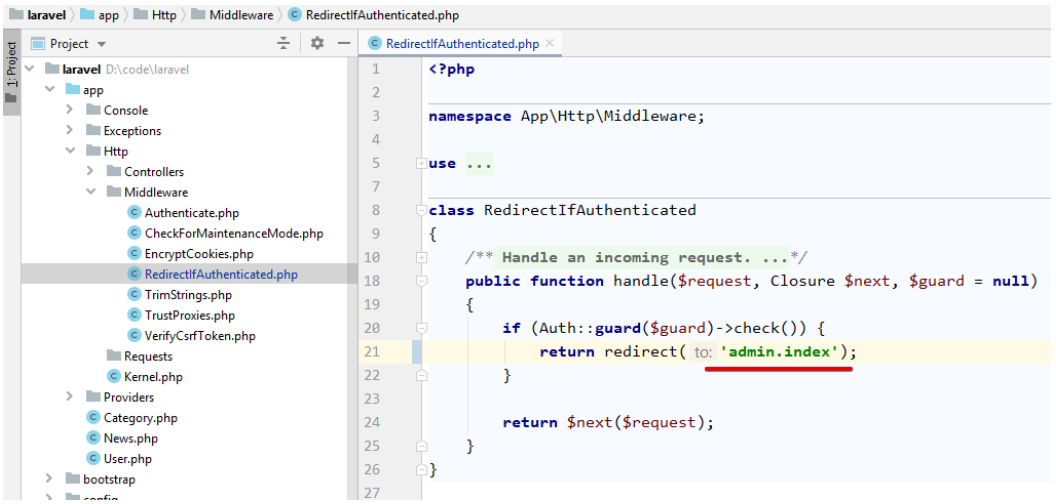
Поскольку к главной странице доступ должен быть для любого пользователя — удалим автоматически добавленный метод конструктора из **HomeController**.



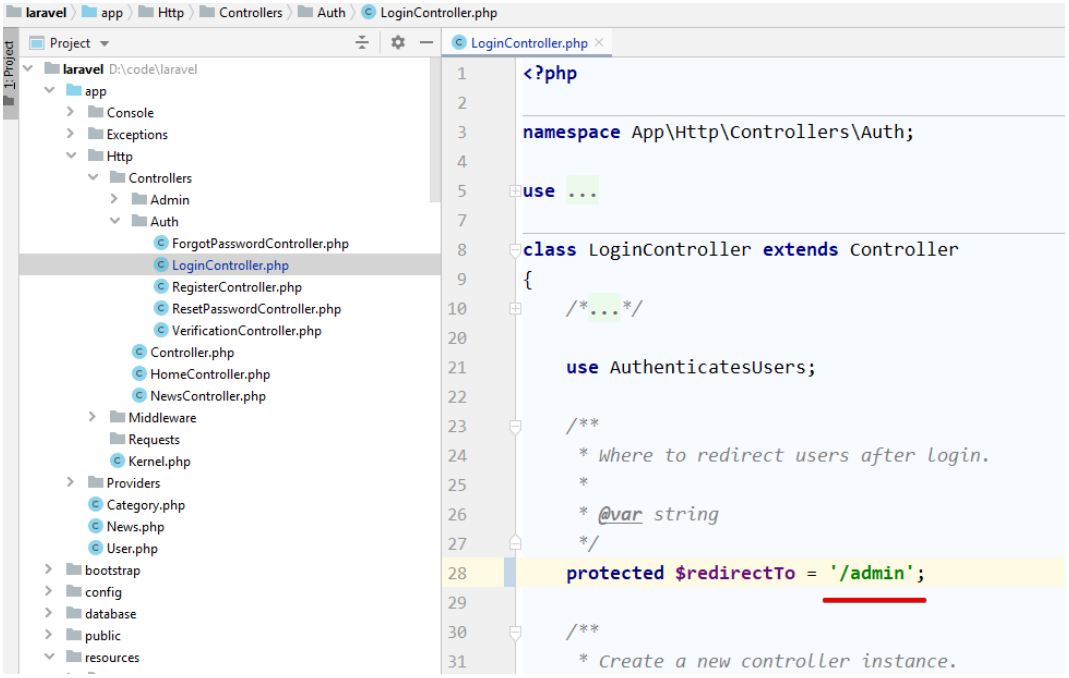
Теперь укажем, что посредник **auth** должен работать для всех маршрутов в группе, начинающихся с **/admin**. Для этого в соответствующую группу добавим указание на этот посредник.



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



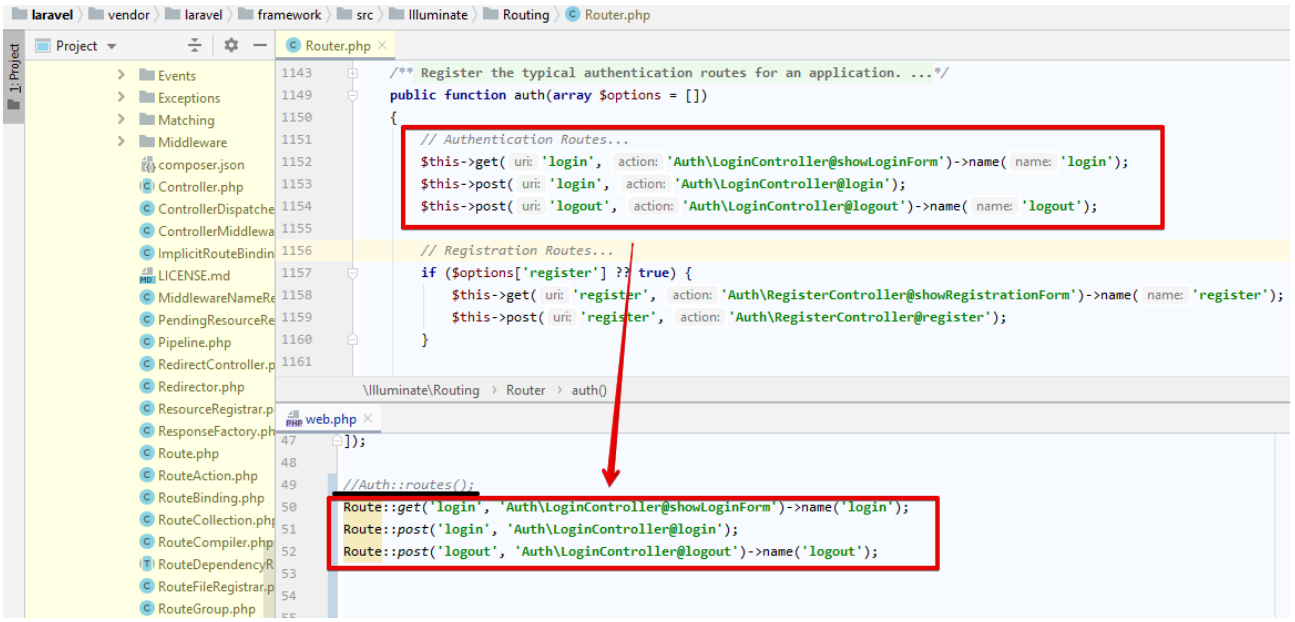
И укажем, куда переходить по умолчанию после входа в систему.



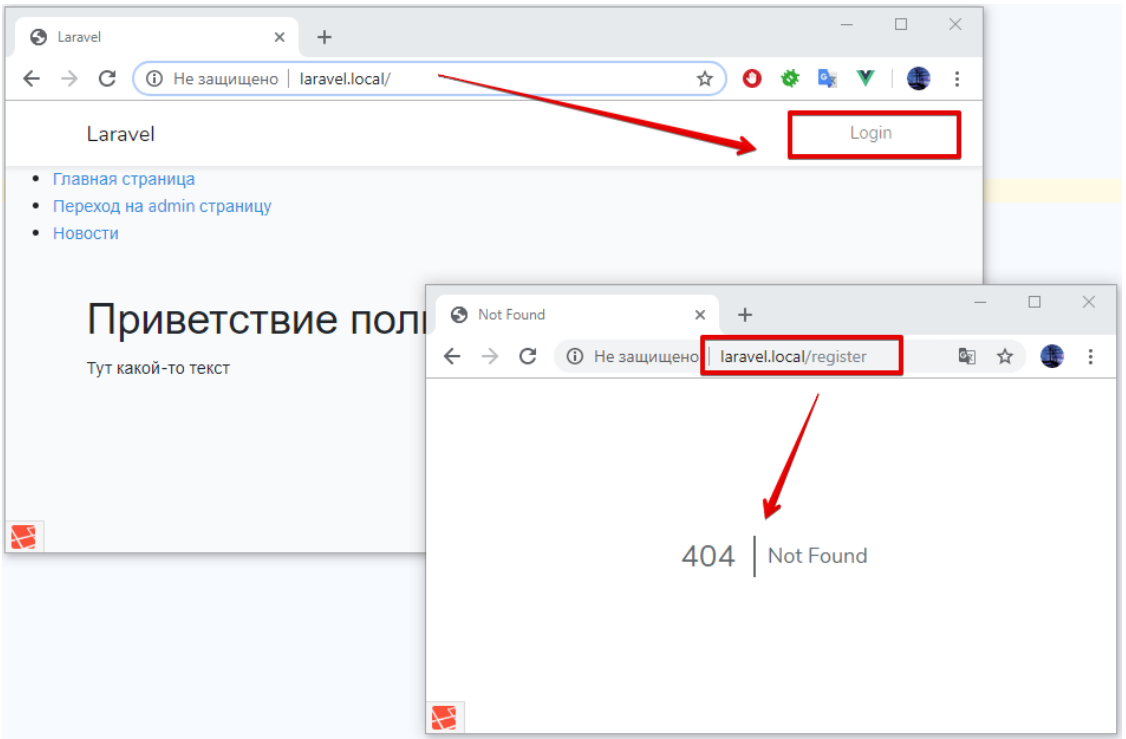
Оптимизация авторизации

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()** закомментим.



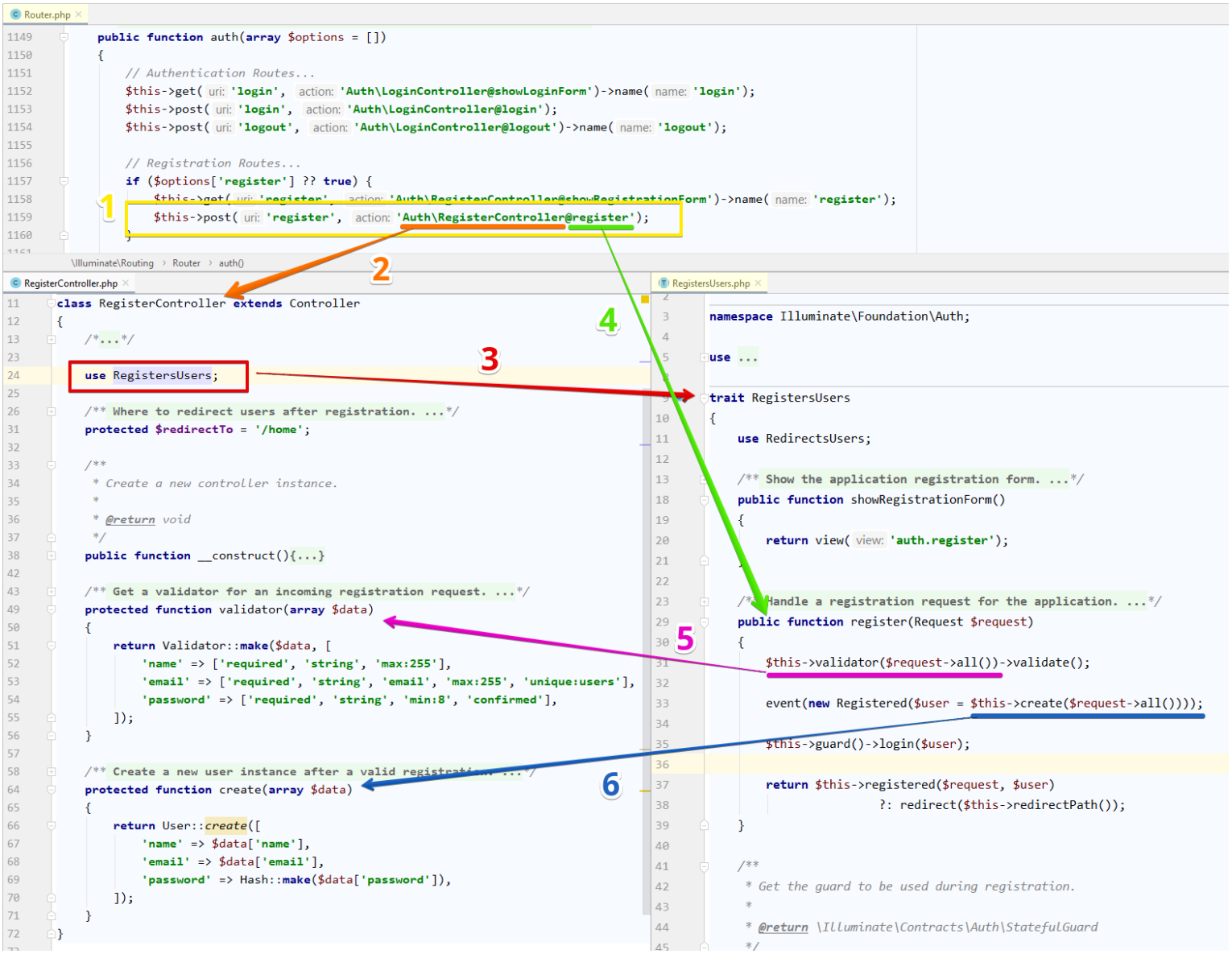
Теперь в шапке сайта у нас пропала надпись «Регистрация». Если попробовать пройти по адресу, который отвечал за регистрацию, получим сообщение о том, что страница не найдена.



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?**

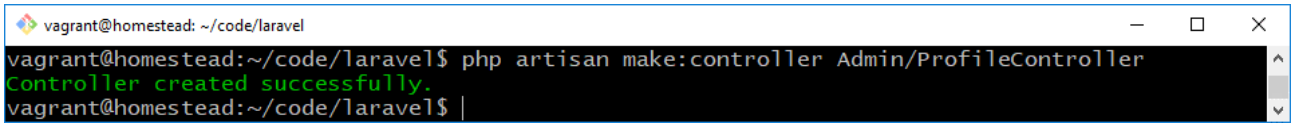


Выше представлены листинги файлов:

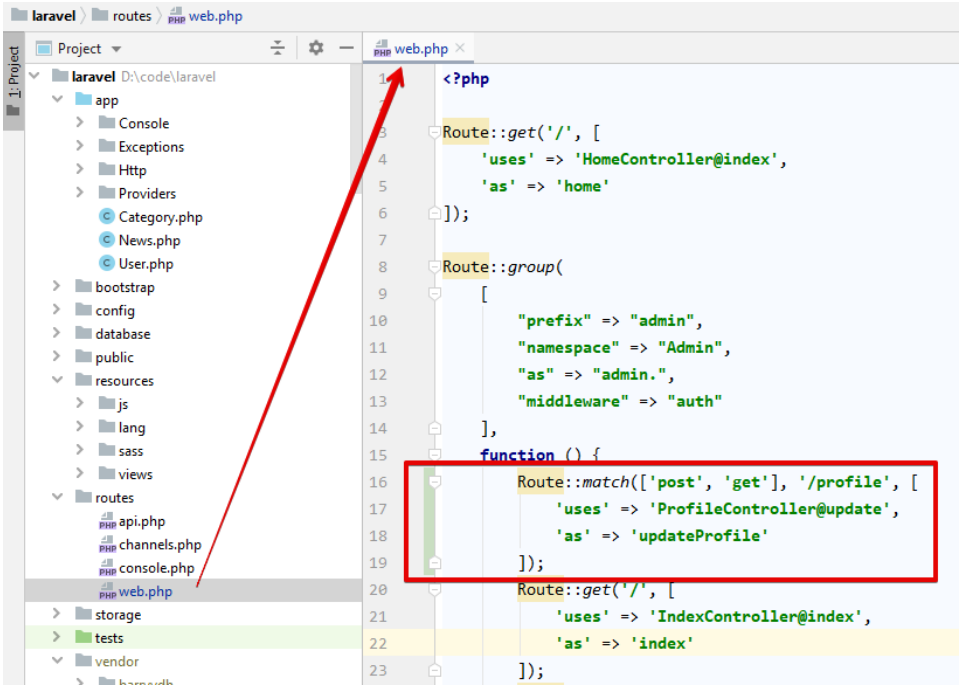
- **vendor/laravel/framework/src/Illuminate/Routing/Router.php**
- **vendor/laravel/framework/src/Illuminate/Foundation/Auth/RegistersUsers.php**
- **app/Http/Controllers/Auth/RegisterController.php**

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

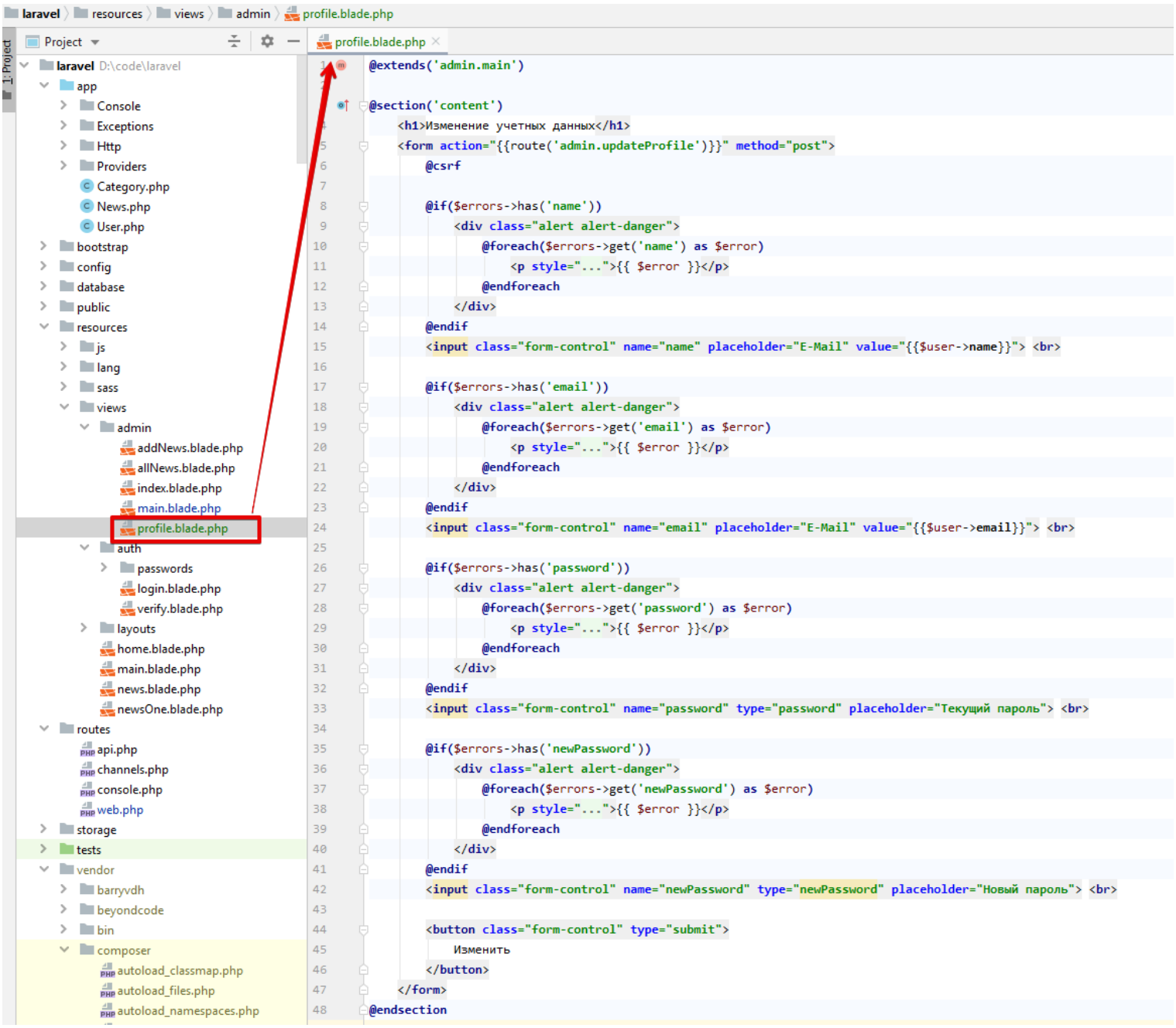
Сначала создадим новый контроллер, доступ к которому будет только у администратора системы. Поэтому создадим его в папке **Admin** следующей командой: **php artisan make:controller Admin/ProfileController**



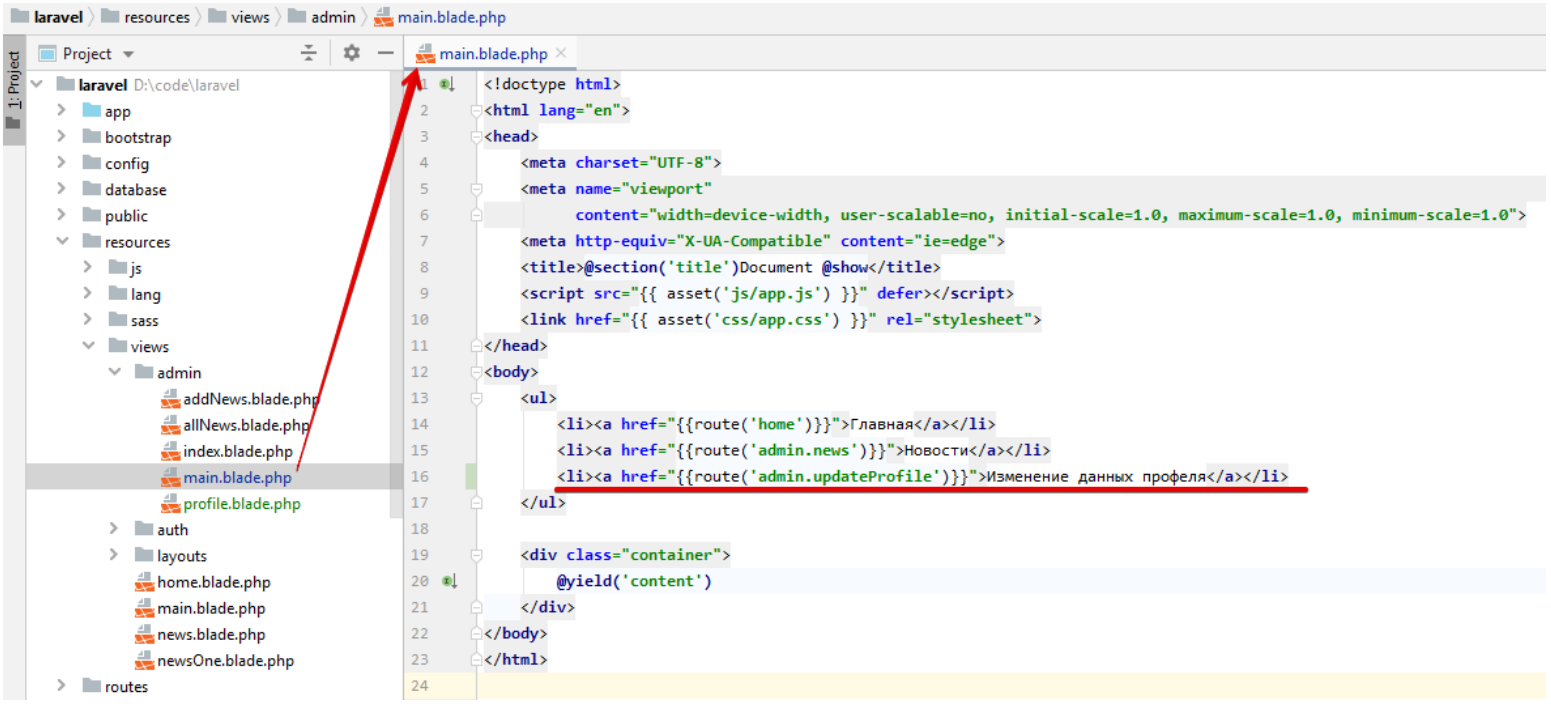
Далее добавим новый маршрут, который будет выполняться при **get**- и **post**-запросах к адресу **/admin/profile**. Он будет вызывать метод **update** в созданном контроллере:



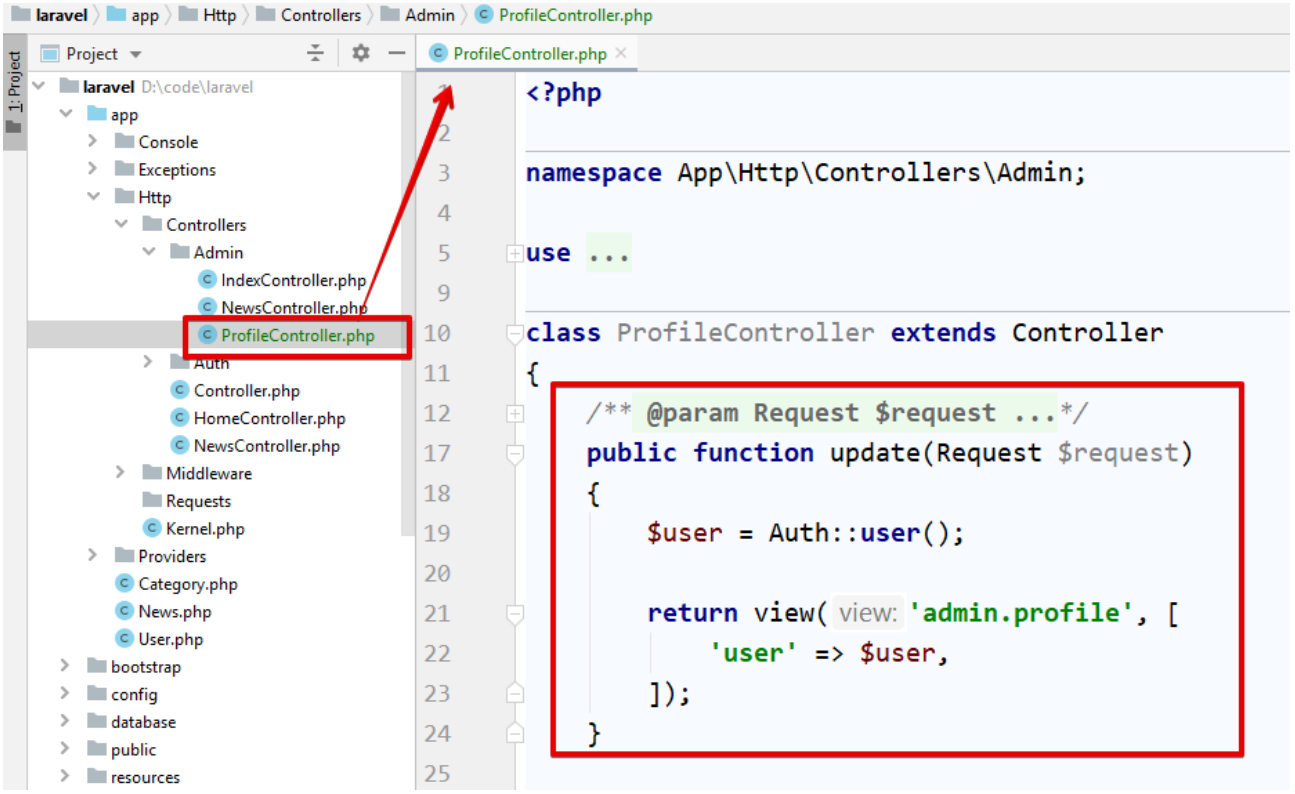
Прежде чем перейти в контроллер, создадим новый шаблон, в который поместим форму для изменения данных пользователя.



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



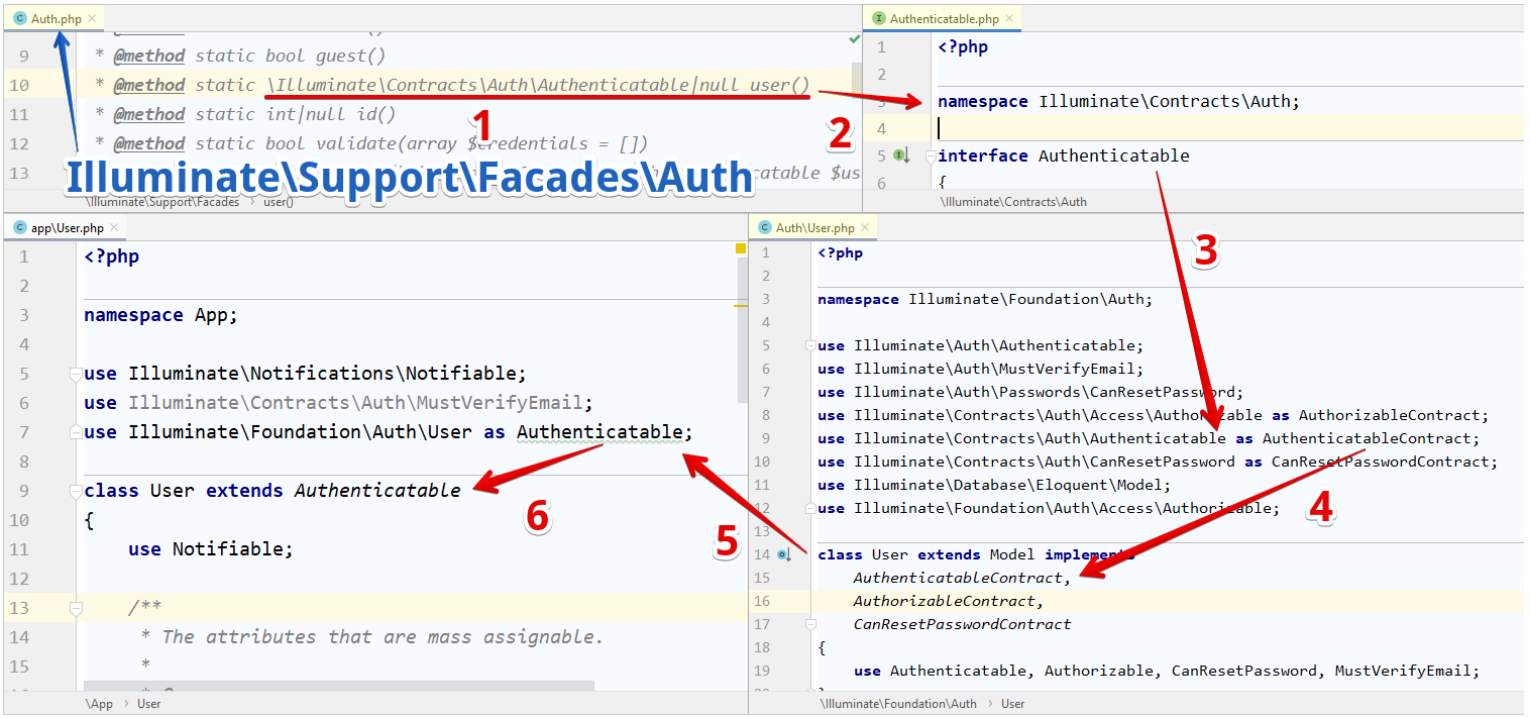
Время перейти к контроллеру. Сначала получим данные о пользователе, который у нас авторизован. В этом поможет **фасад Illuminate\Support\Facades\Auth**.



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.

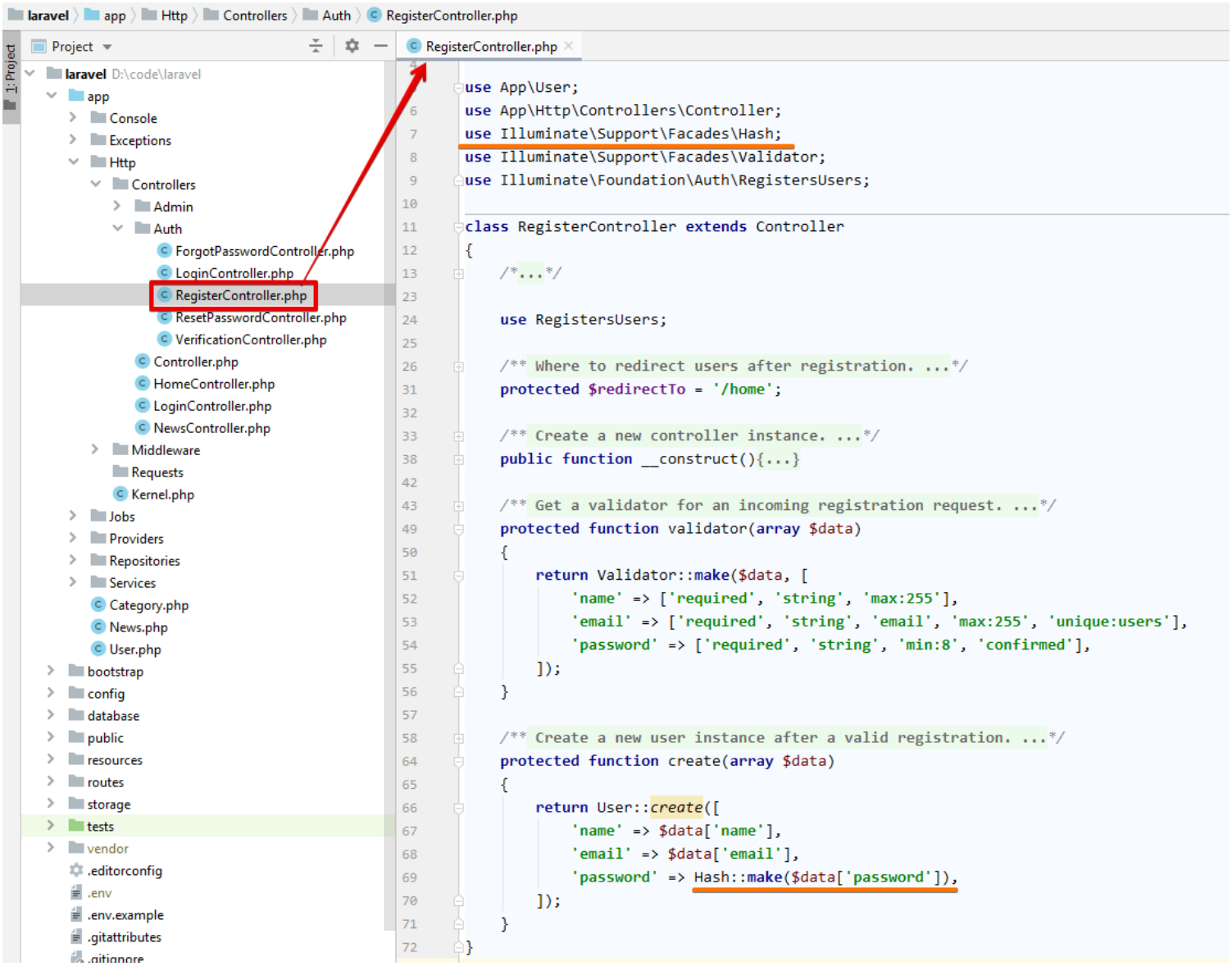
Как это работает? Схема:



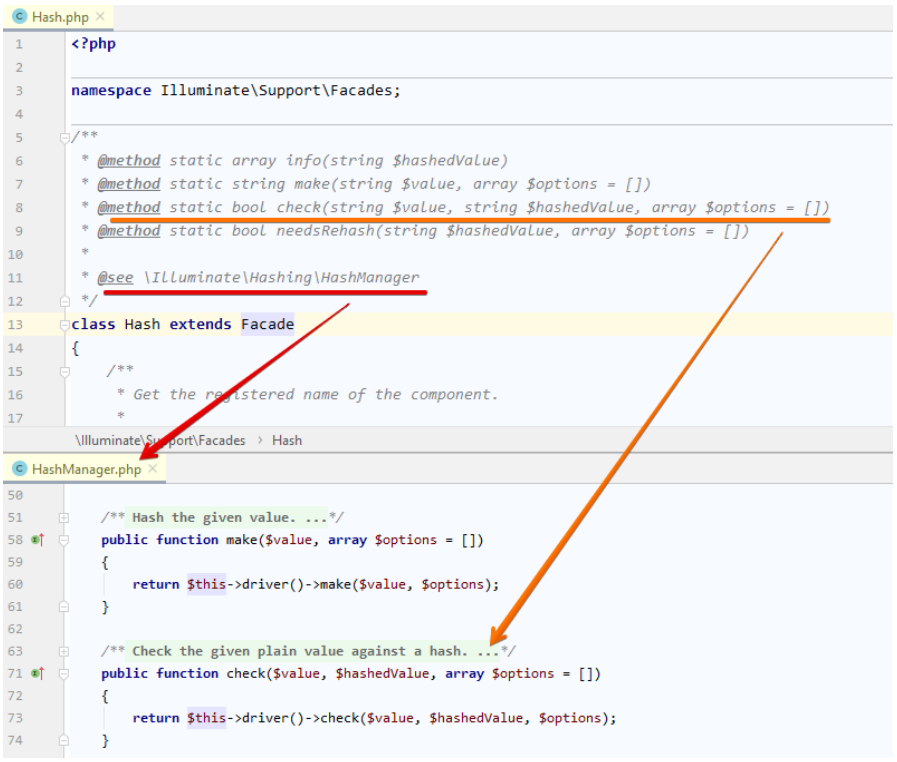
Поскольку класс **User** является классом-наследником **Illuminate\Database\Eloquent\Model**, работа по изменению информации о текущем пользователе мало чем будет отличаться от редактирования информации модели **News**. Единственное, что еще следует учесть: в базе данных пароль сохраняется в виде хеша.

Как получить этот хеш?

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

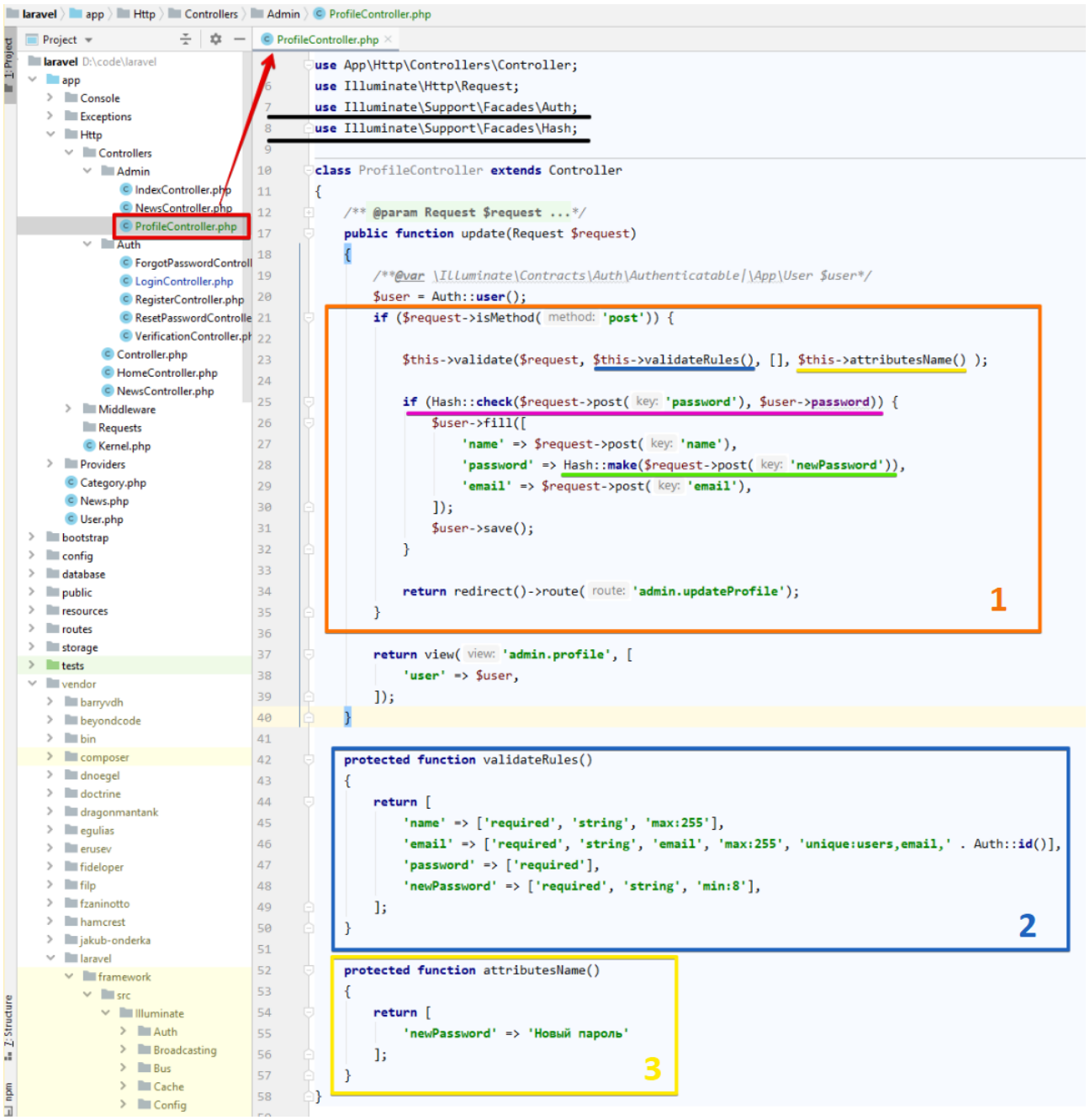


В этом классе и обнаружим, что создание хеша пароля происходило с помощью метода **make** фасада **Hash**. Исследуем данный класс — и обнаружим метод **check**, который и выполняет проверку введенного пароля пользователя с хешем из базы данных.



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.

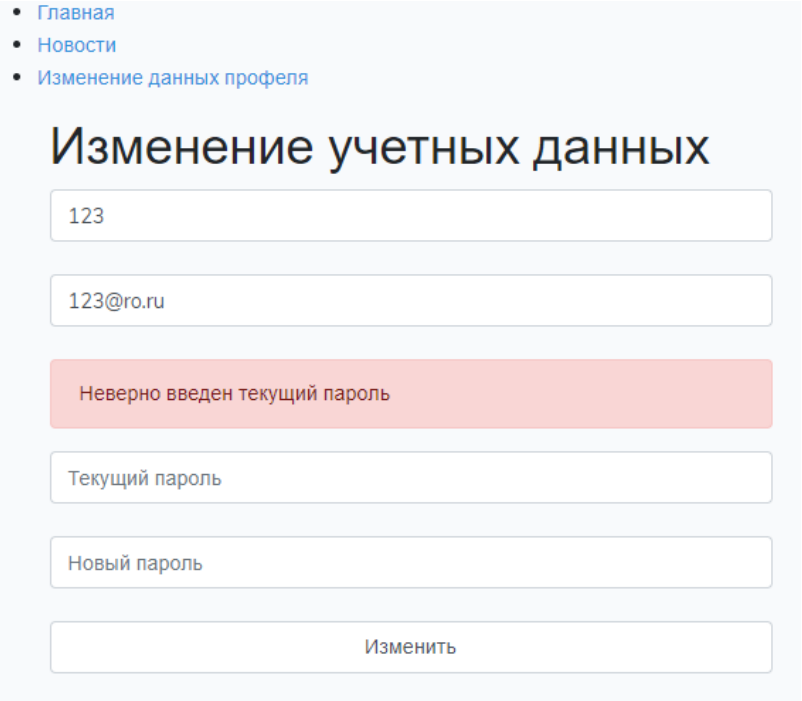


Проверяем. Кажется, что-то забыли…

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.

```php
/** @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:

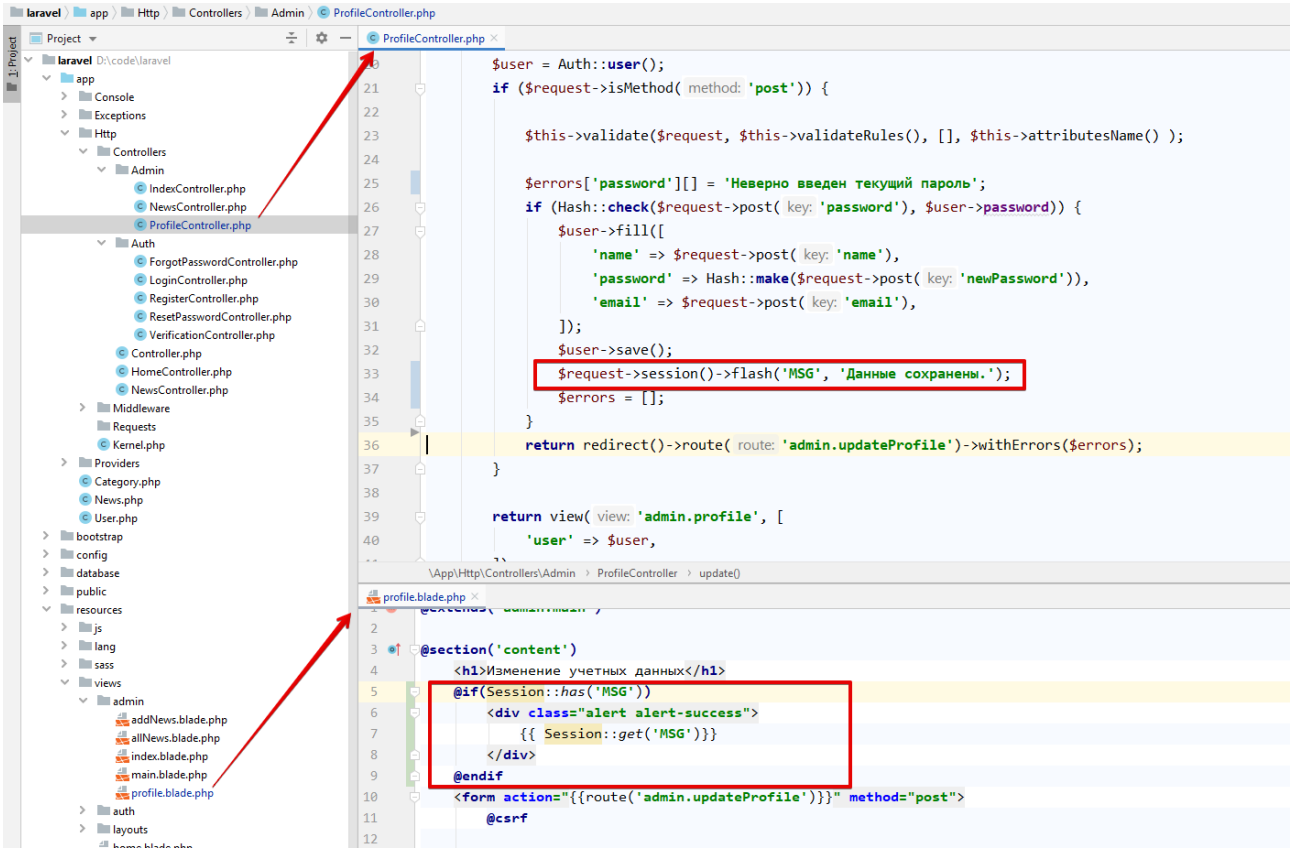


Как перенести валидацию этих данных в посредник?

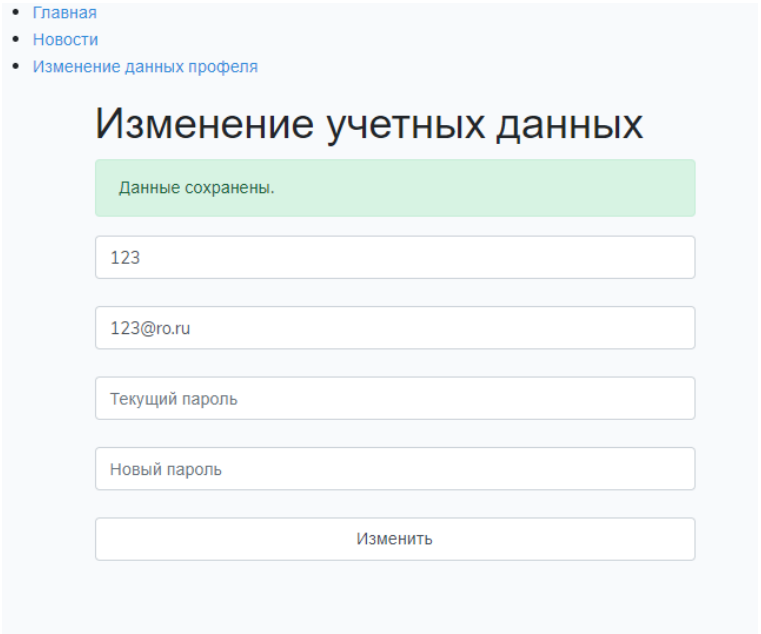
Теперь, когда ошибки выводятся, хорошо бы добавить информацию об успешном выполнении смены данных. Для этого тоже понадобится флеш-сообщение, но оно не должно попадать в ошибки.

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**.



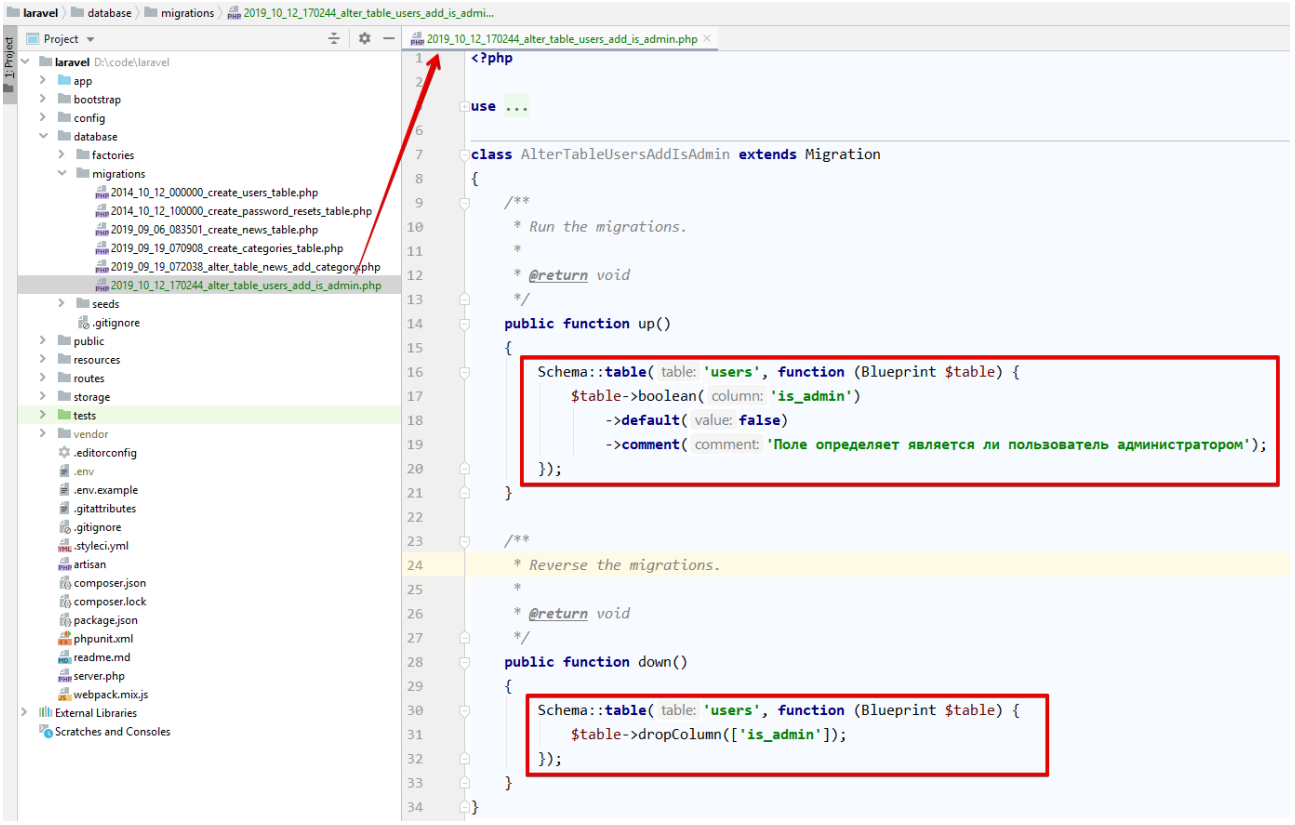
Результат выполнения:



Разграничение прав пользователей

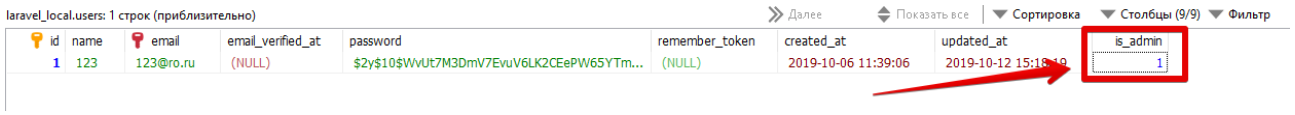
Пока в системе любой пользователь, который пройдет регистрацию и аутентификацию на сайте, становится администратором системы. Но нужно давать пользователям доступ к дополнительным новостям, блокируя возможность их править. Для этого добавим в таблице **users** дополнительное поле, которое будет отвечать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

Код для добавления нового поля:



Выполним миграцию командой **php artisan migrate**

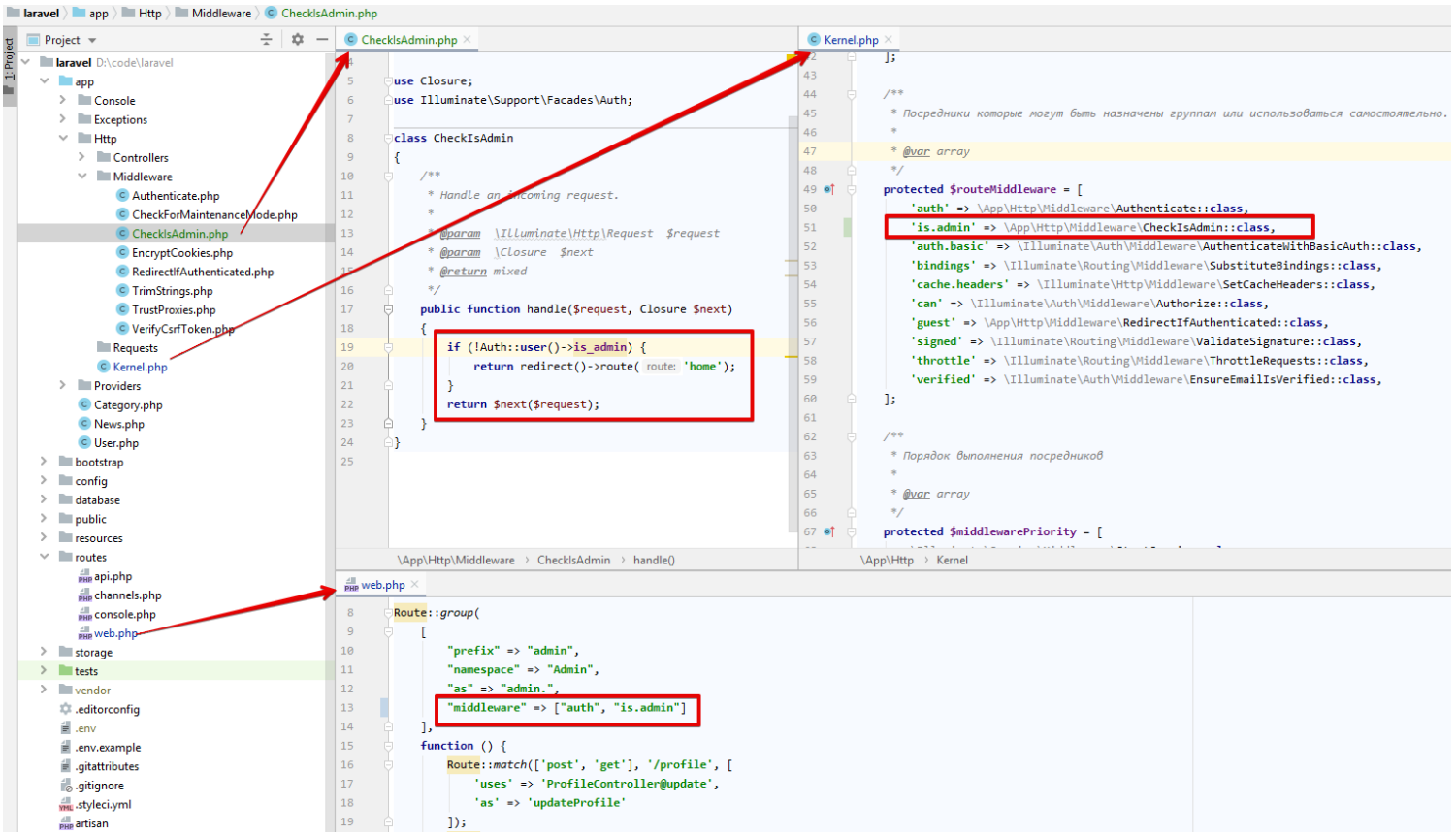
Результат выполнения миграции:



Чтобы контролировать наличие единицы в данном поле, создадим посредника:

php artisan make:middleware CheckIsAdmin

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