

Laravel

# Урок 9. События в Laravel.

## Провайдеры.

## Работа со сторонними API

Использование провайдеров в Laravel. Добавление провайдера для работы с API VK. Авторизация пользователя через ВКонтакте.

### Оглавление

#### [Теория](#)

[Паттерн «Наблюдатель»](#)

[XML-парсеры](#)

[Сервис-провайдеры](#)

#### [Практика](#)

[Парсинг данных новостей](#)

[Создание авторизации через социальную сеть](#)

[Увеличение прав доступа для авторизованных пользователей](#)

#### [Практическое задание](#)

#### [Дополнительные материалы](#)

#### [Используемая литература](#)

# Теория

## Паттерн «Наблюдатель»

Паттерн — это общепринятый подход в решении задач программирования. Знание паттернов позволяет программисту избегать костылей и велосипедов, допускать меньше ошибок и писать поддерживаемый, красивый и чистый код.

Паттерны условно можно разделить на **порождающие**, **структурные** и **поведенческие**.

«Наблюдатель» — это поведенческий паттерн проектирования, который создает механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах. Этот паттерн позволяет, не меняя структуру одного конкретного класса, выполнять действия в других классах, которые «подписаны на события». Привязка новых классов-подписчиков и их отвязка не затрагивает класс, за которым они наблюдают.

Подробнее об этом и многих других паттернах: <https://refactoring.guru/ru/design-patterns/observer>

## XML-парсеры

XML — формат передачи и хранения данных. По сути он является аналогом HTML: его задача — выполнять разметку документа. В отличие от HTML, он является строгим языком разметки. Если, создавая html-страницу, разработчик не укажет корневой контейнер HTML или где-то не закроет открывающийся тег — документ все равно будет нормально разбираться браузером, и пользователь увидит материал. С XML такие фокусы не проходят. Документ XML с нарушением любого правила его составления будет невалидным.

Документы XML часто используются для хранения информации и ее передачи по сети. Самый часто встречающийся случай — сохранение документов MS Office. Если, к примеру, открыть документ Word, как архив, то внутри увидим набор xml-документов.

Для создания и чтения xml-документов в PHP есть встроенные расширения, такие как SimpleXML и DOM. Ссылки на данные расширения:

- <https://www.php.net/manual/ru/book.dom.php>
- <https://www.php.net/manual/ru/book.simplexml.php>

Данные расширения, как правило, работают «из коробки» PHP. Но можно столкнуться с нюансами, которые не видны на старте.

### Как быть?

Нужно взять за правило — прежде чем приступать к задаче, всегда думать, не решал ли ее кто-то прежде. В нашем примере таким решением будет библиотека **orchestral/parser**. Документация — по адресу <https://github.com/orchestral/parser>

Данная библиотека позволяет получить данные из xml-документа в виде массива.

## Сервис-провайдеры

Назначение сервис-провайдеров (провайдеров) заключается в регистрации одного или группы сервисов в приложении. Класс провайдера должен наследовать класс

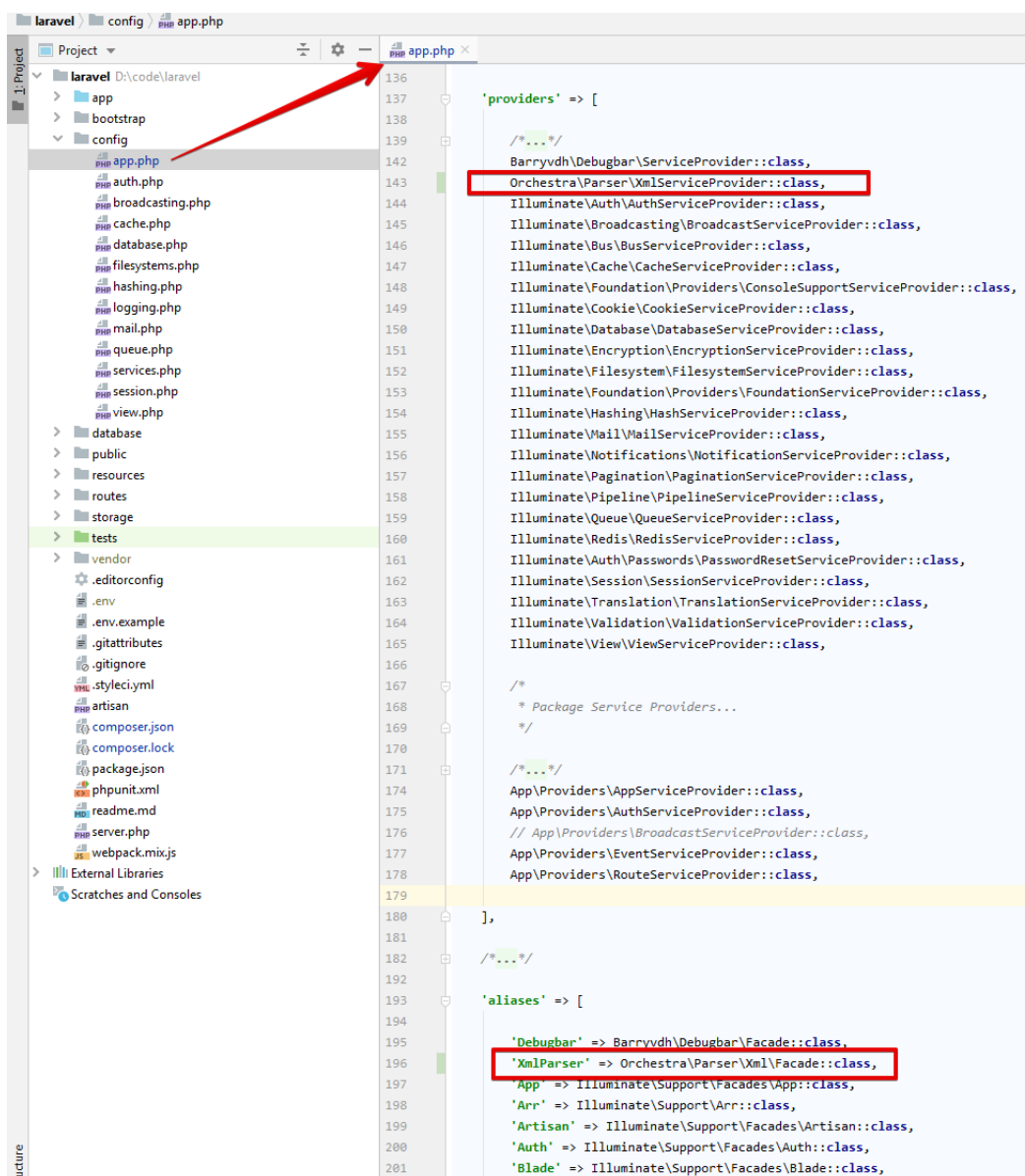
Illuminate\Support\ServiceProvider и иметь обязательный метод **register**, в котором и происходит регистрация сервисов.

# Практика

## Парсинг данных новостей

Сначала установим пакет парсера, выполнив команду `composer require "orchestra/parser" ^4.0`

После успешного скачивания пакета разместим его в списке провайдеров и добавим новый фасад в файле `app.php`.

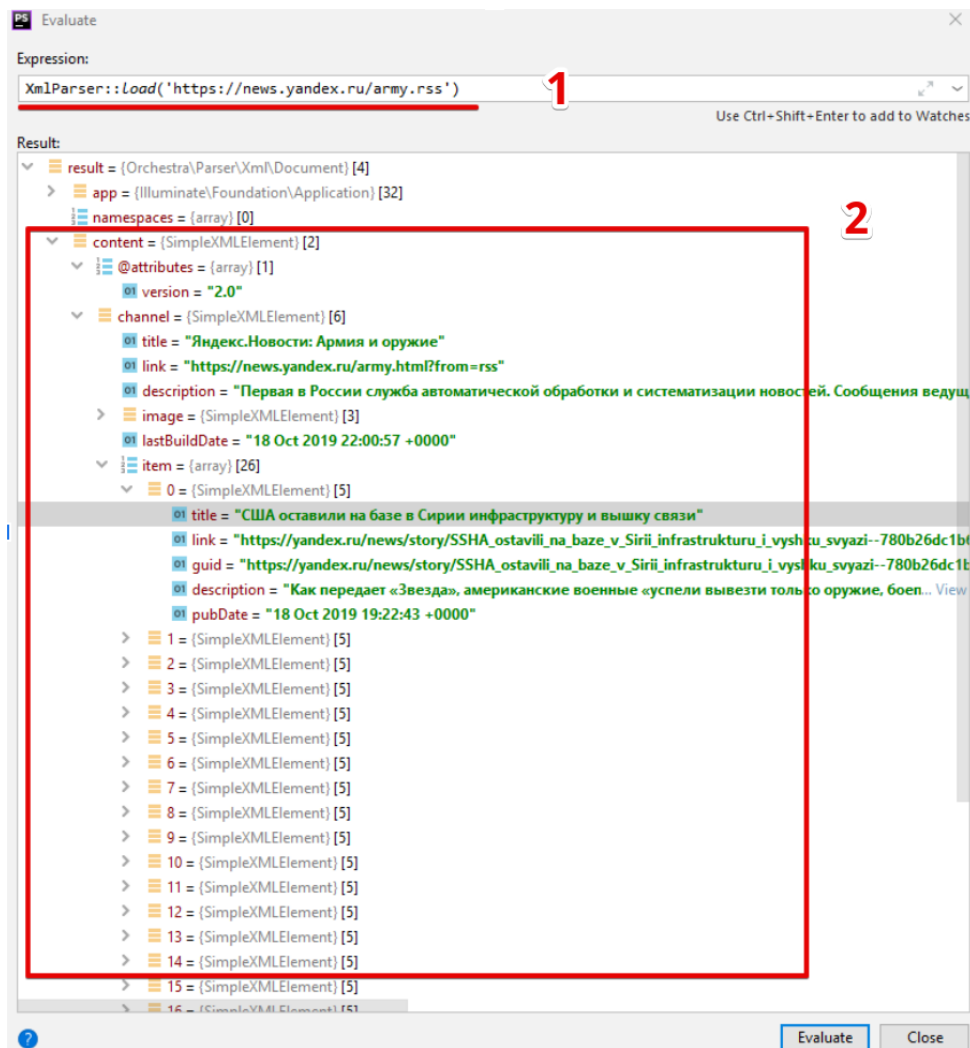


```
136
137
138
139
142 'providers' => [
143     /*...*/
144     Barryvdh\Debugbar\ServiceProvider::class,
145     Orchestra\Parser\XmlServiceProvider::class,
146     Illuminate\Auth\AuthServiceProvider::class,
147     Illuminate\Broadcasting\BroadcastServiceProvider::class,
148     Illuminate\Bus\BusServiceProvider::class,
149     Illuminate\Cache\CacheServiceProvider::class,
150     Illuminate\Foundation\Providers\ConsoleSupportServiceProvider::class,
151     Illuminate\Cookie\CookieServiceProvider::class,
152     Illuminate\Database\DatabaseServiceProvider::class,
153     Illuminate\Encryption\EncryptionServiceProvider::class,
154     Illuminate\Filesystem\FilesystemServiceProvider::class,
155     Illuminate\Foundation\Providers\FoundationServiceProvider::class,
156     Illuminate\Hashing\HashServiceProvider::class,
157     Illuminate\Mail\MailServiceProvider::class,
158     Illuminate\Notifications\NotificationServiceProvider::class,
159     Illuminate\Pagination\PaginationServiceProvider::class,
160     Illuminate\Pipeline\PipelineServiceProvider::class,
161     Illuminate\Queue\QueueServiceProvider::class,
162     Illuminate\Redis\RedisServiceProvider::class,
163     Illuminate\Auth\Passwords>PasswordResetServiceProvider::class,
164     Illuminate\Session\SessionServiceProvider::class,
165     Illuminate\Translation\TranslationServiceProvider::class,
166     Illuminate\Validation\ValidationServiceProvider::class,
167     Illuminate\View\ViewServiceProvider::class,
168     /*
169      * Package Service Providers...
170      */
171     /*...*/
172     App\Providers\AppServiceProvider::class,
173     App\Providers\AuthServiceProvider::class,
174     // App\Providers\BroadcastServiceProvider::class,
175     App\Providers\EventServiceProvider::class,
176     App\Providers\RouteServiceProvider::class,
177 ],
178
179
180
181
182 /*...*/
183
184
185 'aliases' => [
186     'Debugbar' => Barryvdh\Debugbar\Facade::class,
187     'XmlParser' => Orchestra\Parser\Xml\Facade::class,
188     'App' => Illuminate\Support\Facades\App::class,
189     'Arr' => Illuminate\Support\Arr::class,
190     'Artisan' => Illuminate\Support\Facades\Artisan::class,
191     'Auth' => Illuminate\Support\Facades\Auth::class,
192     'Blade' => Illuminate\Support\Facades\Blade::class,
```

Далее создадим новый контроллер. Для этого выполним команду: `php artisan make:controller Admin/ParserController`

В контроллере добавим метод **index**, задачей которого будет получение данных с ресурса <https://news.yandex.ru/army.rss>. Для этого воспользуемся добавленным фасадом парсера и вызовем у него метод **load**, в который и передадим ссылку на указанный ресурс. В результате выполнения

данного метода будет возвращен объект класса `Orchestra\Parser\Xml\Document`, наполненный данными из указанного ресурса.

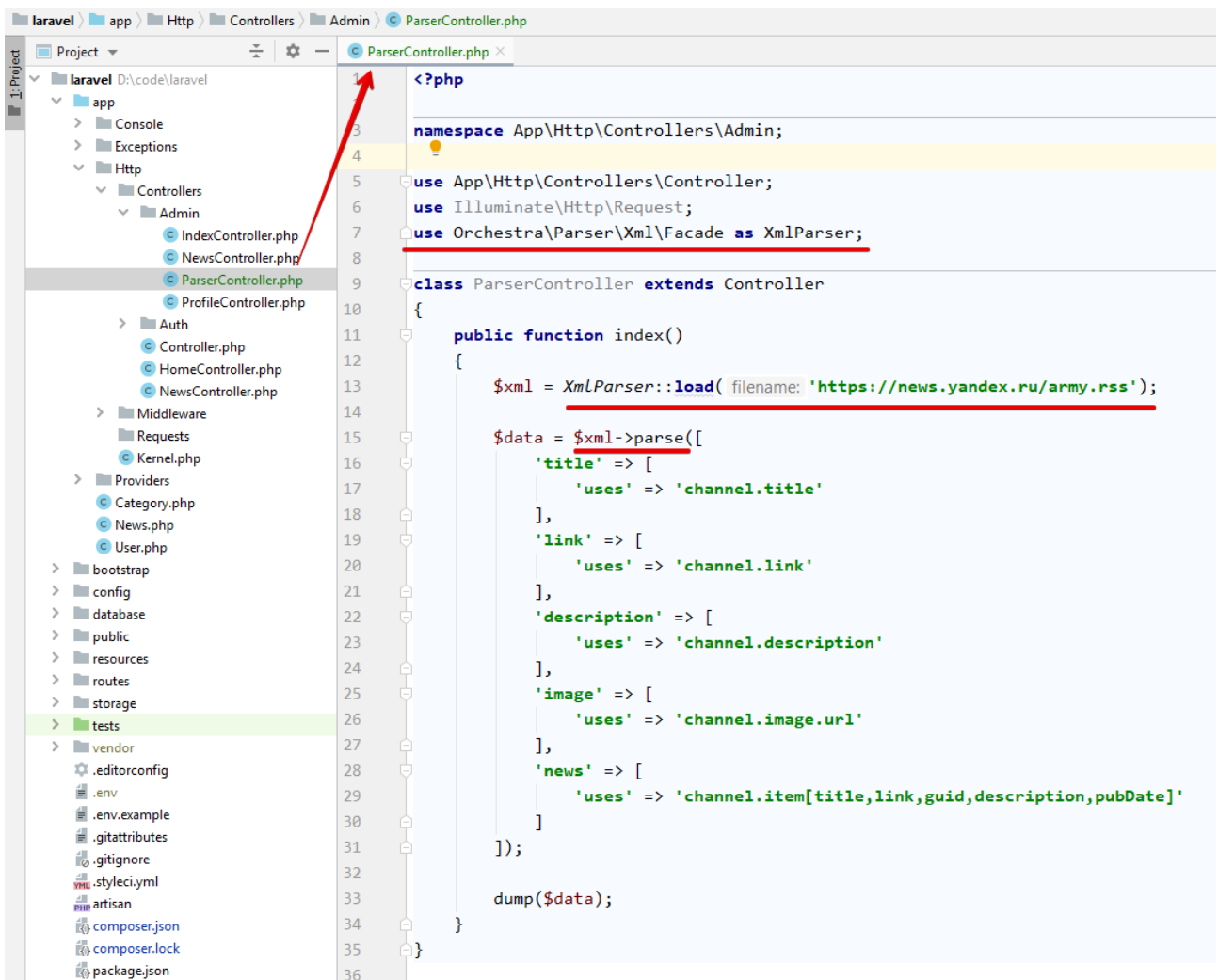


Чтобы получить необходимые данные из указанного объекта, следует передать массив с набором параметров в метод `parse` полученного объекта.

В качестве ключей массива выступают ключи, которые будут в возвращаемом массиве. В значениях указываются тоже массивы, состоящие из ключа (зарезервированного способа разбора) и значения (правила получения данных). Подробнее об этих правилах:

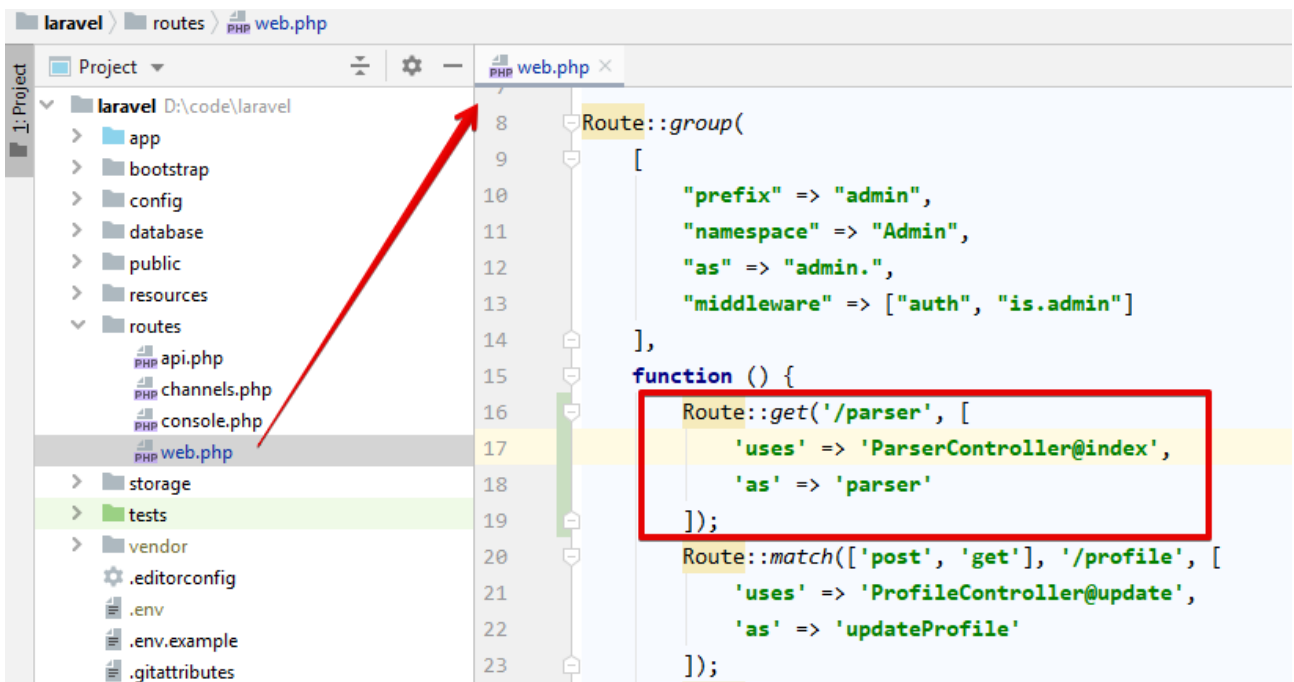
- через точку указываются узлы xml-документа, при этом игнорируется корневой;
- если данные в xml-документе повторяются, то их указывают в квадратных скобках. Эти данные будут возвращены в виде массива.

Подробности в примере:



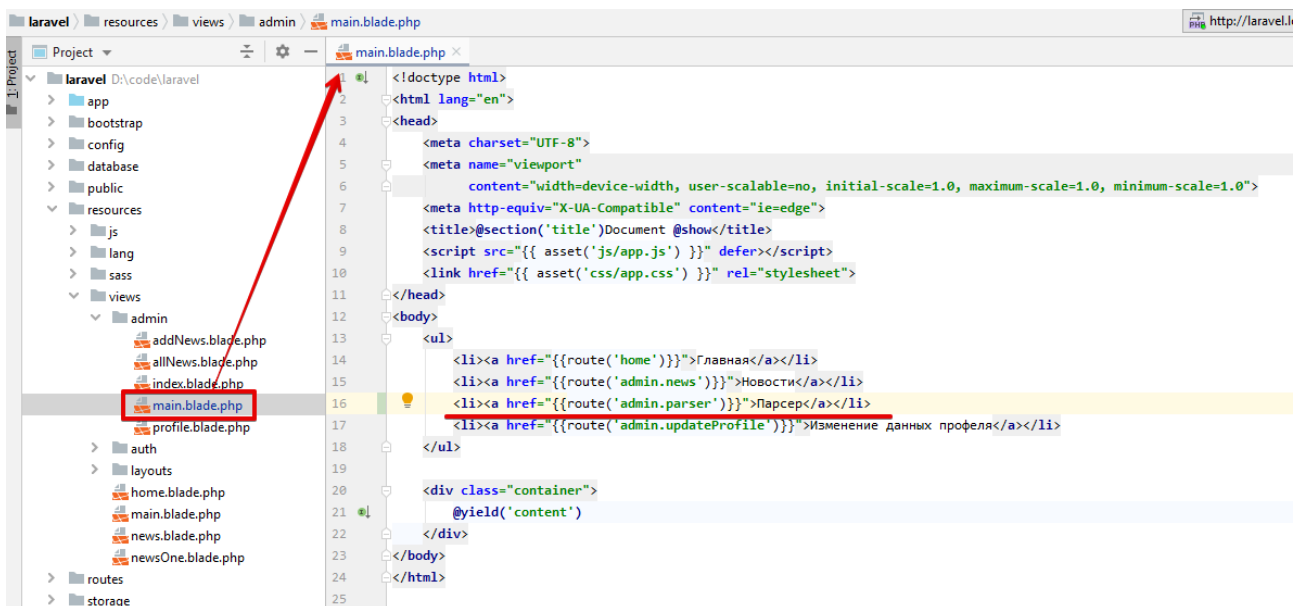
```
<?php
3
4 namespace App\Http\Controllers\Admin;
5
6 use App\Http\Controllers\Controller;
7 use Illuminate\Http\Request;
8 use Orchestra\Parser\Xml\Facade as XmlParser;
9
10 class ParserController extends Controller
11 {
12     public function index()
13     {
14         $xml = XmlParser::load( filename: 'https://news.yandex.ru/army.rss');
15
16         $data = $xml->parse([
17             'title' => [
18                 'uses' => 'channel.title'
19             ],
20             'link' => [
21                 'uses' => 'channel.link'
22             ],
23             'description' => [
24                 'uses' => 'channel.description'
25             ],
26             'image' => [
27                 'uses' => 'channel.image.url'
28             ],
29             'news' => [
30                 'uses' => 'channel.item[title,link,guid,description,pubDate]'
31             ]
32         ]);
33
34         dump($data);
35     }
36 }
```

Чтобы увидеть результат, добавим соответствующий роут для вызова метода `index` из созданного нами контроллера:

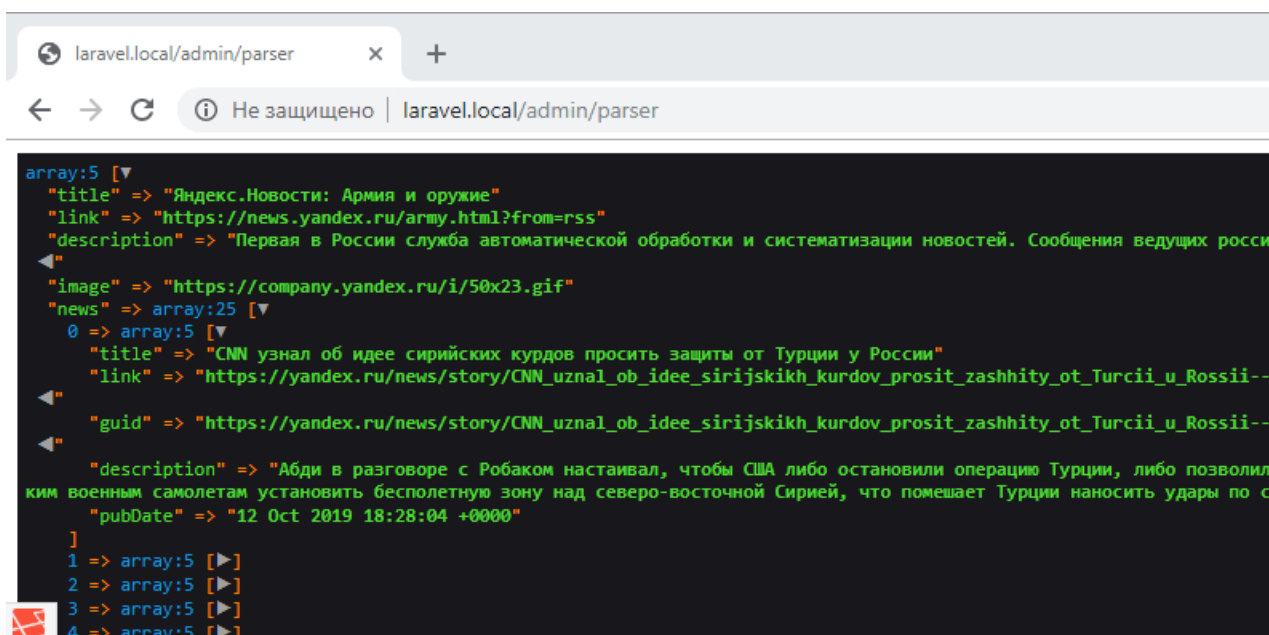


```
8 Route::group(
9     [
10         "prefix" => "admin",
11         "namespace" => "Admin",
12         "as" => "admin.",
13         "middleware" => ["auth", "is.admin"]
14     ],
15     function () {
16         Route::get('/parser', [
17             'uses' => 'ParserController@index',
18             'as' => 'parser'
19         ]);
20         Route::match(['post', 'get'], '/profile', [
21             'uses' => 'ProfileController@update',
22             'as' => 'updateProfile'
23         ]);
24     });
```

И далее добавим ссылку на этот маршрут в меню админки нашего приложения:



Запустим приложение и получим результат обработанного XML в виде массива:



## Создание авторизации через социальную сеть

Установим еще один пакет, который позволит легко авторизовываться в VK. Для установки этого пакета выполним команду `composer require socialiteproviders/vkontakte`

Добавим класс `Add \SocialiteProviders\Manager\ServiceProvider` в список провайдеров приложения:

```
137 'providers' => [  
138  
139 /*...*/  
142 Barryvdh\Debugbar\ServiceProvider::class,  
143 Orchestra\Parser\XmlServiceProvider::class,  
144 SocialiteProviders\Manager\ServiceProvider::class,  
145 Illuminate\Auth\AuthServiceProvider::class,  
146 Illuminate\Broadcasting\BroadcastServiceProvider::class,  
147 Illuminate\Bus\BusServiceProvider::class,  
148 Illuminate\Cache\CacheServiceProvider::class,  
149 Illuminate\Foundation\Providers\ConsoleSupportServiceProvider::class,  
150 Illuminate\Cookie\CookieServiceProvider::class,  
151 Illuminate\Database\DatabaseServiceProvider::class,  
152 Illuminate\Encryption\EncryptionServiceProvider::class,  
153 Illuminate\Filesystem\FilesystemServiceProvider::class,  
154 Illuminate\Foundation\Providers\FoundationServiceProvider::class,  
155 Illuminate\Hashing\HashServiceProvider::class,  
156 Illuminate\Mail\MailServiceProvider::class,  
157 Illuminate\Notifications\NotificationServiceProvider::class,  
158 Illuminate\Pagination\PaginationServiceProvider::class,  
159 Illuminate\Pipeline\PipelineServiceProvider::class,  
160 Illuminate\Queue\QueueServiceProvider::class,  
161 Illuminate\Redis\RedisServiceProvider::class,  
162 Illuminate\Auth\Passwords>PasswordResetServiceProvider::class,  
163 Illuminate\Session\SessionServiceProvider::class,  
164 Illuminate\Translation\TranslationServiceProvider::class,  
165 Illuminate\Validation\ValidationServiceProvider::class,  
166 Illuminate\View\ViewServiceProvider::class,  
167
```

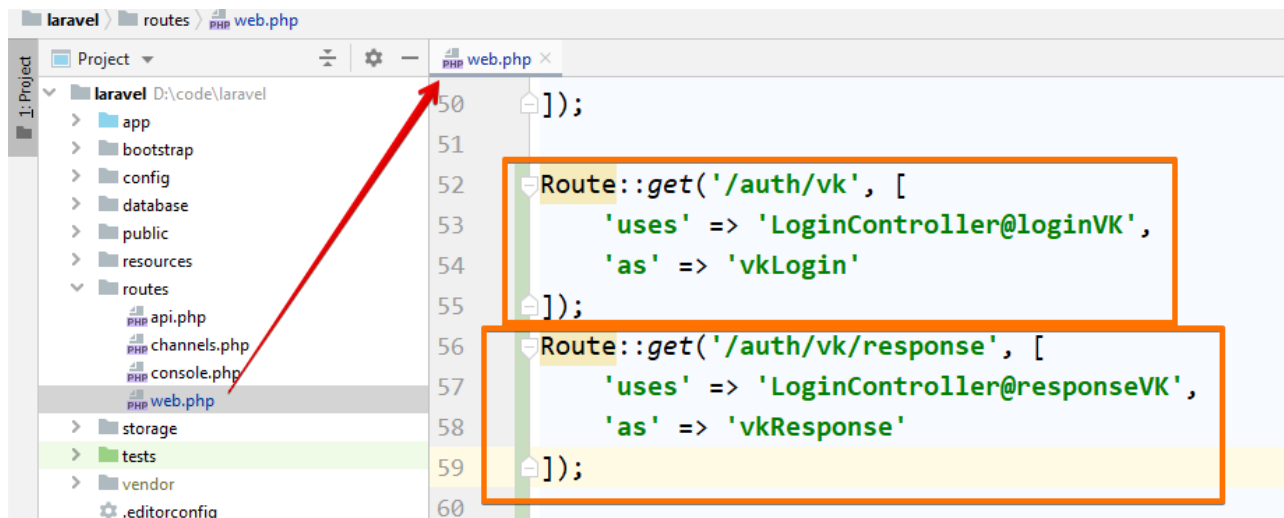
Данный пакет использует паттерн «Наблюдатель» — для реализации этого паттерна в облегченном виде в Laravel есть класс **App\Providers\EventServiceProvider**. В его свойстве **listen** указываются классы, которые подписываются на события в других классах. Класс, который выступает источником события, указывается в качестве ключа массива, а классы-подписчики — в виде его значений. Пакет **socialiteproviders/vkontakte** — это надстройка над пакетом **socialite**. В его реализации «Наблюдатель» используется для получения событий авторизации в конкретной социальной сети с дальнейшим редиректом к ее сервисам с указанными параметрами. А также — для отслеживания редиректа со стороны авторизирующего сервиса и предобработки полученной информации от него.

Создадим новый контроллер, который будет выполнять авторизацию. Для этого выполним команду **php artisan make:controller LoginController**

В созданный нами контроллер добавим два метода. Задача метода **loginVK** — совершить редирект браузера на страницу авторизации VK, передав необходимые параметры приложения в строке запроса. Метод **responseVK** будет принимать ответ со стороны социальной сети. Создание запроса и его наполнение параметрами, получение ответа — все это будет выполнять используемая нами библиотека, поэтому код в контроллере будет минимальным.

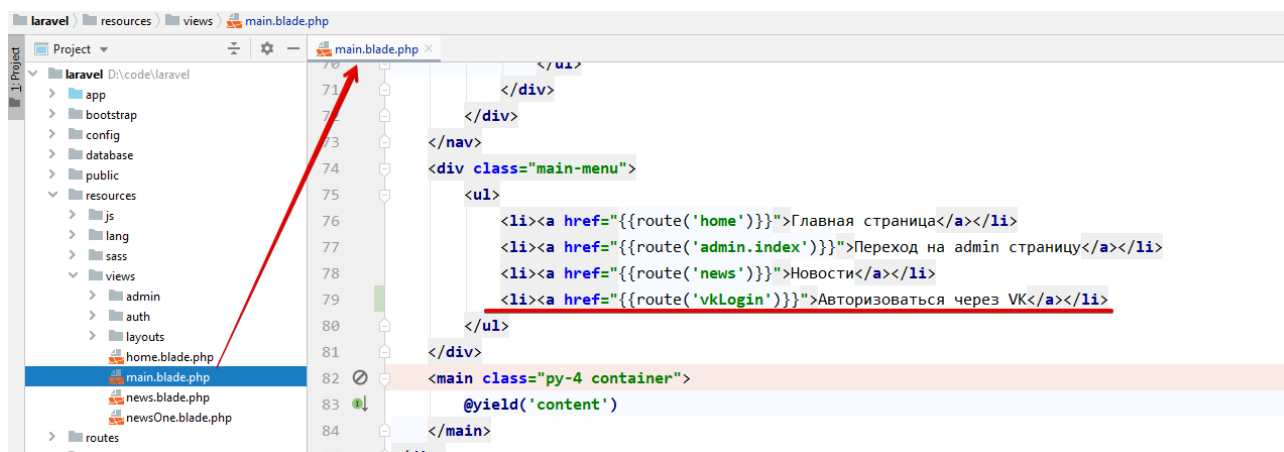


Теперь, когда наши методы готовы, создадим для них соответствующие маршруты. Данные маршруты должны быть доступны для любого пользователя, и поэтому мы не помещаем их в группу маршрутов для администратора.



```
50 });
51
52 Route::get('/auth/vk', [
53     'uses' => 'LoginController@loginVK',
54     'as' => 'vkLogin'
55 ]);
56
57 Route::get('/auth/vk/response', [
58     'uses' => 'LoginController@responseVK',
59     'as' => 'vkResponse'
60 ]);
```

Чтобы авторизация началась, следует добавить ссылку на маршрут, обращенный к методу **loginVK**.



```
70 </ul>
71 </div>
72 </div>
73 </nav>
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 <li><a href="{{route('vkLogin')}}">Авторизоваться через VK</a></li>
80 </ul>
81 </div>
82 <main class="py-4 container">
83 @yield('content')
84 </main>
```

Чтобы пакет **socialiteproviders** знал, какие данные следует передать, укажем их в настройках. Сами ключи сохраним в файле **env**, а получать их из этого файла будем посредством функции **env**.

В файле **services.php** добавим новый элемент массива: ключом будет название социальной сети, а в качестве значения укажем еще один массив, который будет получать данные из файла **env**.

```
<?php
return [
    /*...*/

    'vkontakte' => [
        'client_id' => env( key: 'VKONTAKTE_KEY'),
        'client_secret' => env( key: 'VKONTAKTE_SECRET'),
        'redirect' => env( key: 'VKONTAKTE_REDIRECT_URI')
    ],

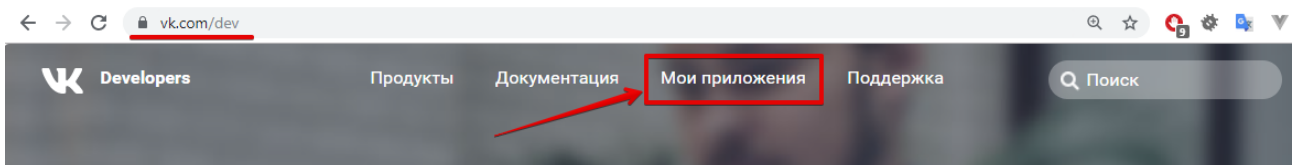
    'mailgun' => [
        'domain' => env( key: 'MAILGUN_DOMAIN'),
        'secret' => env( key: 'MAILGUN_SECRET'),
        'endpoint' => env( key: 'MAILGUN_ENDPOINT', default: 'api.mailgun.net'),
    ],

    'postmark' => [
        'token' => env( key: 'POSTMARK_TOKEN'),
    ],
];
```

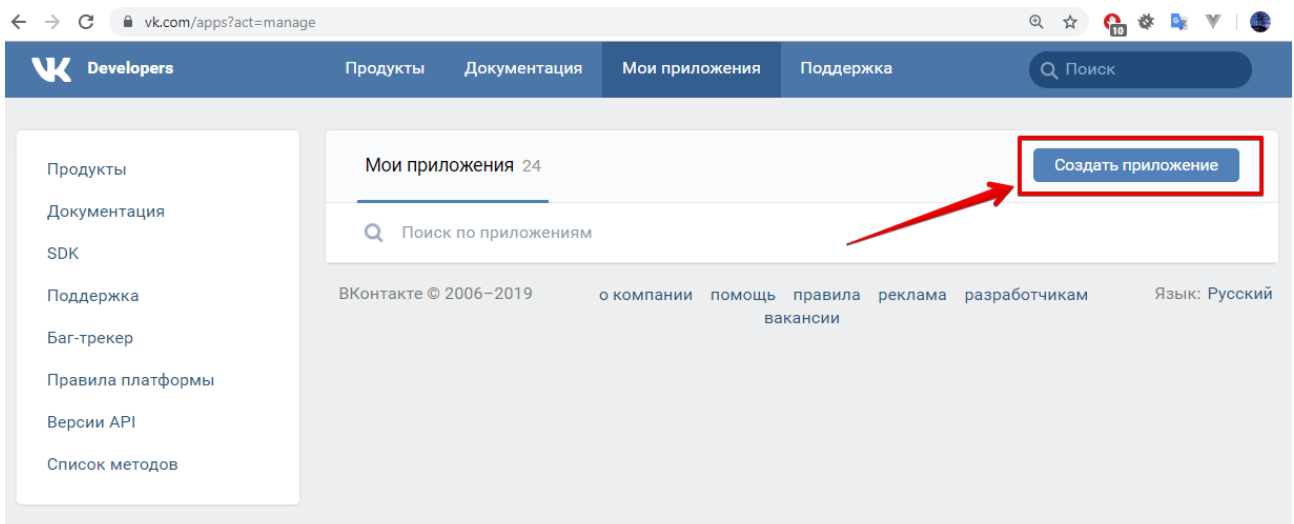
### Где взять данные для авторизации приложения в API VK?

Ответ очевиден — ВКонтакте.

Перейдем по ссылке <https://vk.com/dev> и щелкнем по «Мои приложения»:



Далее следует зарегистрировать (создать) новое приложение в VK. Для этого нажмем на кнопку «Создать приложение».



В новом окне заполним базовую форму. Пример:

### Создание приложения

Название:

Платформа:  Встраиваемое приложение  
 Standalone-приложение  
 Сайт

Адрес сайта:

Базовый домен:

Ваша система не обязана иметь реальный адрес в сети, подойдет и локальный.

Жмем «Подключить сайт». Появится новое окно, в котором следует перейти в «Настройки».

### Информация

Название:

Описание:

Тип турнирной таблицы:

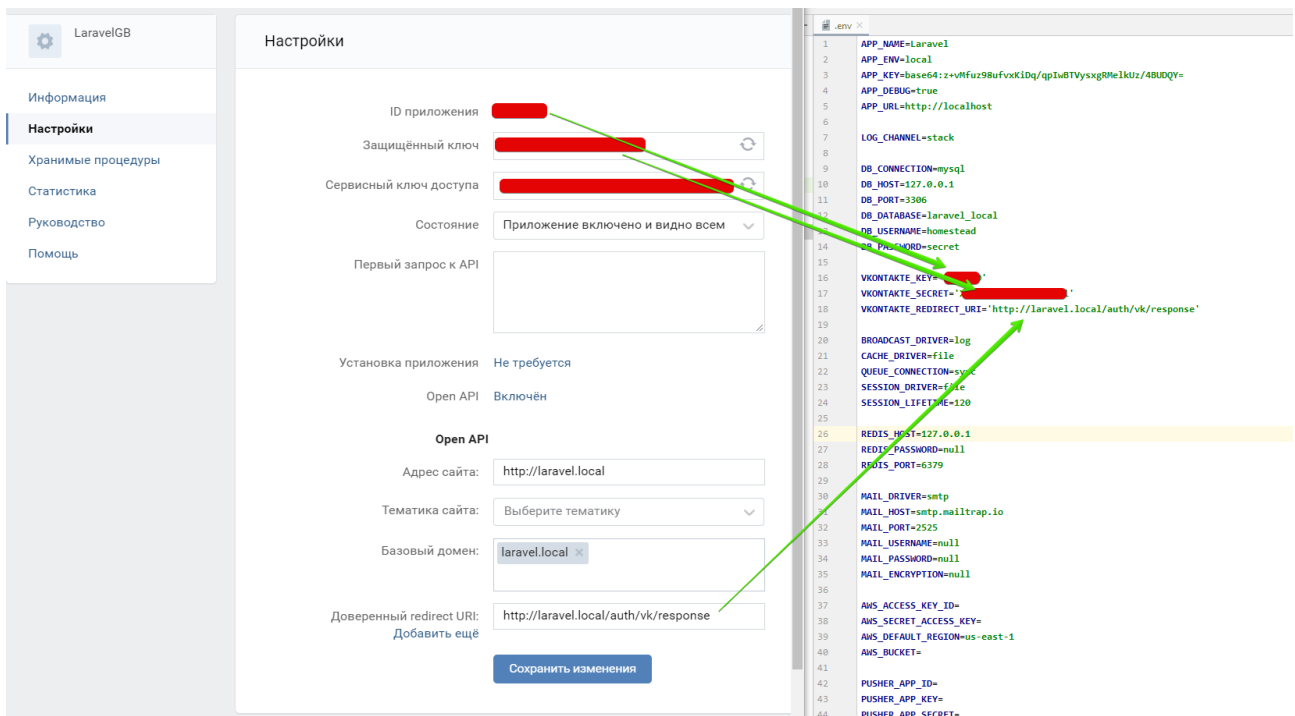
Сообщество:

Иконка 32x32:

Настройка:

В настройках нас и ждут данные, необходимые для авторизации. Перенесем их в файл **env**.

Еще нужно указать, какой адрес может использовать приложение для обратного редиректа (доверенный redirect URI). В нашем случае это роут, который ведет к методу **responseVK**.



Сохраняемся — жмем кнопку «Сохранить изменения».

Подробнее о механизме авторизации можно прочитать тут: [https://vk.com/dev/access\\_token](https://vk.com/dev/access_token).

Перейдем обратно в нашу систему и нажмем на «Авторизоваться через VK». Если вы переходите по этой ссылке впервые, то сначала увидите запрос подтверждения: согласны ли вы предоставить указанному сайту данные. Подтвердив согласие, браузер вернет вас в систему. Если увидите данные вашего пользователя — вы все сделали верно.

Теперь самое время сохранить эти данные. Но прежде чем сделать это, уточним некоторые аспекты авторизации пользователя через социальную сеть.

1. Пользователя в базу добавляем только при первой авторизации.
2. При добавлении пользователя учитываем, что он не админ.
3. Уточняем, каким образом авторизуется пользователь: через стандартный логин и пароль (возможно, в дальнейшем это потребуется) или через конкретную социальную сеть.

Для выполнения этих критериев следует определиться, где хранить данную информацию. Самый логичный вариант — сохранять в таблицу **users**.

Добавим миграцию, которая добавит в имеющуюся таблицу дополнительные поля. Команда для создания миграции: **php artisan make:migration AlterTableAddSocAuth**

После того как новый класс будет создан, добавим в него код из скриншота ниже.

```

1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 class AlterTableAddSocAuth extends Migration
8 {
9     public function up()
10    {
11        Schema::table('users', function (Blueprint $table) {
12            $table->string('column: 'id_in_soc', length: 20)
13                ->default('value: ''')
14                ->comment('comment: 'id в социальной сети');
15            $table->enum('column: 'type_auth', ['site', 'vk', 'fb'])
16                ->default('value: 'site')
17                ->comment('comment: 'Указывает на то, какой тип авторизации использует пользователь');
18            $table->string('column: 'avatar', length: 150)->default('value: ''')->comment('comment: 'Ссылка на аватар');
19            $table->index('columns: 'id_in_soc');
20        });
21    }
22
23    public function down()
24    {
25        Schema::table('users', function (Blueprint $table) {
26            $table->dropColumn(['id_in_soc', 'type_auth', 'avatar']);
27        });
28    }
29 }

```

Выполним миграции командой `php artisan migrate`

И еще добавим возможность автоматического заполнения данными нового свойства в модель User.

```

1 <?php
2
3 namespace App;
4
5 use ...
6
7
8
9 class User extends Authenticatable
10 {
11     use Notifiable;
12
13     /** The attributes that are mass assignable. ...*/
14     protected $fillable = [
15         'name', 'email', 'password', 'id_in_soc', 'type_auth', 'avatar'
16     ];
17
18     /**
19      * The attributes that should be hidden for arrays.
20      *
21      * @var array
22      */
23 }

```

Теперь следует подготовить алгоритм, который будет добавлять новых пользователей, искать уже зарегистрированных и выполнять авторизацию на основе данных, полученных после ответа `api`.

Местом сохранения алгоритма будет репозиторий. В Laravel из коробки не предусмотрено использование репозитория (в фреймворке используется паттерн `activeRecord`). Но не предусмотрено — не значит запрещено. Создадим соответствующую папку и класс:

```
1 <?php
2
3 namespace App\Repositories;
4
5 use App\User;
6 use SocialiteProviders\Manager\OAuth2\User as UserOAuth;
7
8 class UserRepository
9 {
10     /** Поиск или создание нового пользователя по данным из социальной сети ...*/
11
12     public function getUserBySocId(UserOAuth $user, string $socName)
13     {
14         $userInSystem = User::query()
15             ->where( column: 'id_in_soc', $user->id)
16             ->where( column: 'type_auth', $socName)
17             ->first();
18
19         if (empty($userInSystem)) {
20             $userInSystem = new User();
21             $userInSystem->fill([
22                 'name' => !empty($user->getName()) ? $user->getName(): '',
23                 'email' => !empty($user->getEmail()) ? $user->getEmail(): '',
24                 'password' => '',
25                 'id_in_soc' => !empty($user->getId()) ? $user->getId(): '',
26                 'type_auth' => $socName,
27                 'avatar' => !empty($user->getAvatar()) ? $user->getAvatar(): '',
28             ]);
29             $userInSystem->save();
30         }
31
32         return $userInSystem;
33     }
34 }
35
36
37
38 }
```

### Что выполняет данный класс?

В класс **App\Repositories\UserRepository** добавлен один метод, который принимает два параметра:

- экземпляр класса **SocialiteProviders\Manager\OAuth2\User**, которому задан алиас (псевдоним) **UserOAuth**;
- строковое значение, определяющее, какая социальная сеть используется для авторизации. В нашем случае второй параметр — **vk**.

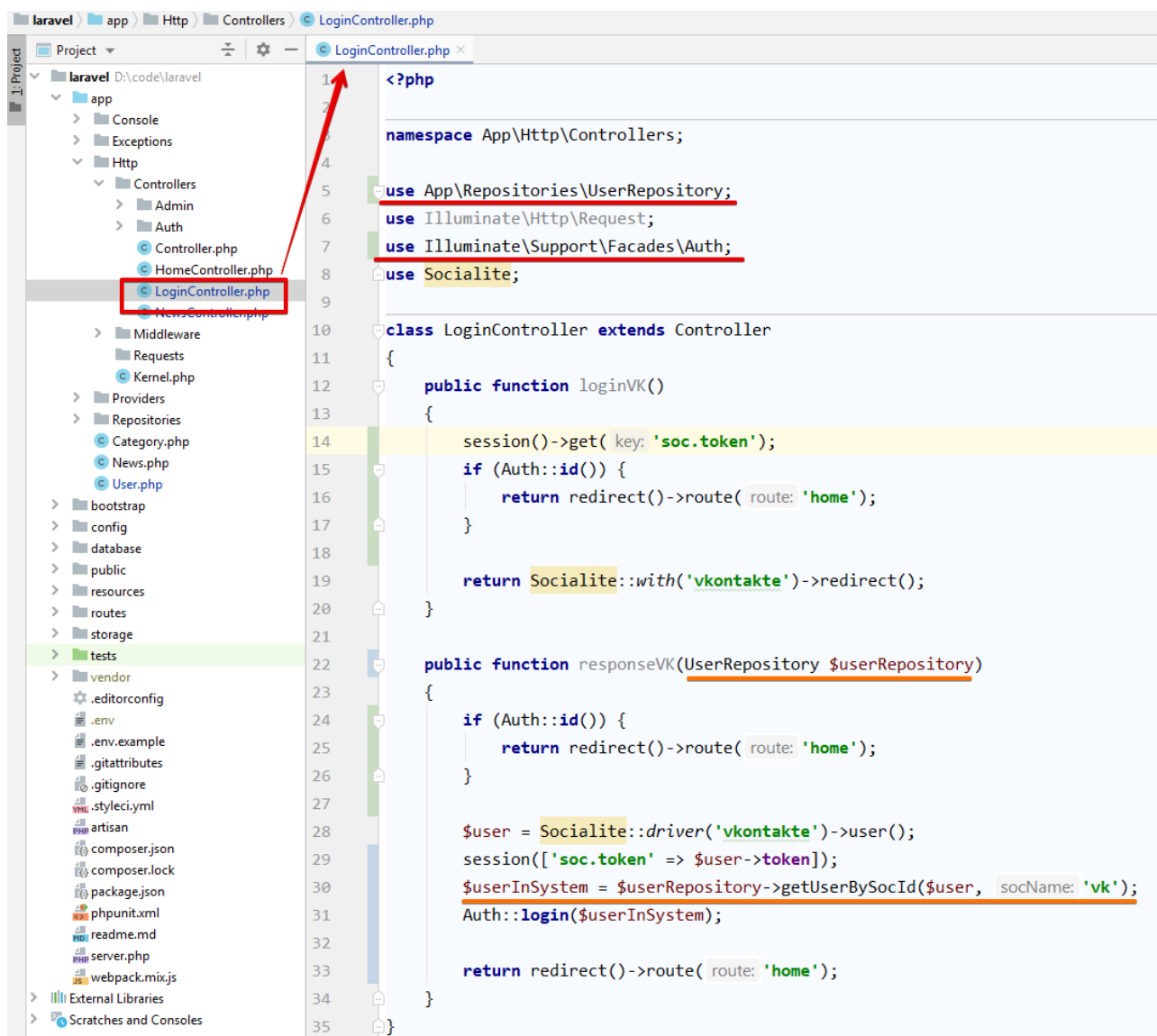
Экземпляр класса **SocialiteProviders\Manager\OAuth2\User** наполняется из ответа от сервиса VK в методе **responseVK** и сохраняется в переменную **user**. Эту переменную и будем передавать в качестве первого параметра в метод **UserRepository**.

Задача метода **getUserBySocId** — определить, существует ли пользователь в базе с **id\_in\_soc**, эквивалентным полученному от сервиса VK, и с **type\_auth**, равным **vk**, и вернуть его. Если такой пользователь отсутствует в базе — создать его и вернуть.

Вернемся в контроллер. Чтобы не создавать новый класс репозитория через **new**, укажем его в качестве параметра — Laravel сам создаст его и передаст в метод объект данного класса.

Последнее, что остается сделать:

- вызвать у переменной **\$userRepository**, которая содержит объект класса репозитория, метод **getUserBySocId**;
- получить пользователя;
- авторизовать его в системе, используя метод **login фасада Auth**;
- перенаправить пользователя на страницу — например, на домашнюю.



```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Repositories\UserRepository;
6 use Illuminate\Http\Request;
7 use Illuminate\Support\Facades\Auth;
8 use Socialite;
9
10 class LoginController extends Controller
11 {
12     public function loginVK()
13     {
14         session()->get( key: 'soc.token');
15         if (Auth::id()) {
16             return redirect()->route( route: 'home');
17         }
18         return Socialite::with('vkontakte')->redirect();
19     }
20
21     public function responseVK(UserRepository $userRepository)
22     {
23         if (Auth::id()) {
24             return redirect()->route( route: 'home');
25         }
26
27         $user = Socialite::driver('vkontakte')->user();
28         session(['soc.token' => $user->token]);
29         $userInSystem = $userRepository->getUserBySocId($user, socName: 'vk');
30         Auth::login($userInSystem);
31
32         return redirect()->route( route: 'home');
33     }
34 }
35
```

## Увеличение прав доступа для авторизованных пользователей

Теперь добавим возможность для зарегистрированных в системе пользователей просматривать приватные новости. Для этого внесем небольшие правки в **NewsController**: проверим, что фасад **Auth** возвращает **id** авторизованного пользователя. Если это происходит — пропускаем условие на

выборку новостей, у которых `is_private = 0`. Если `id` не существует — значит, пользователь не авторизован в системе, и данное условие будет использоваться в запросе.



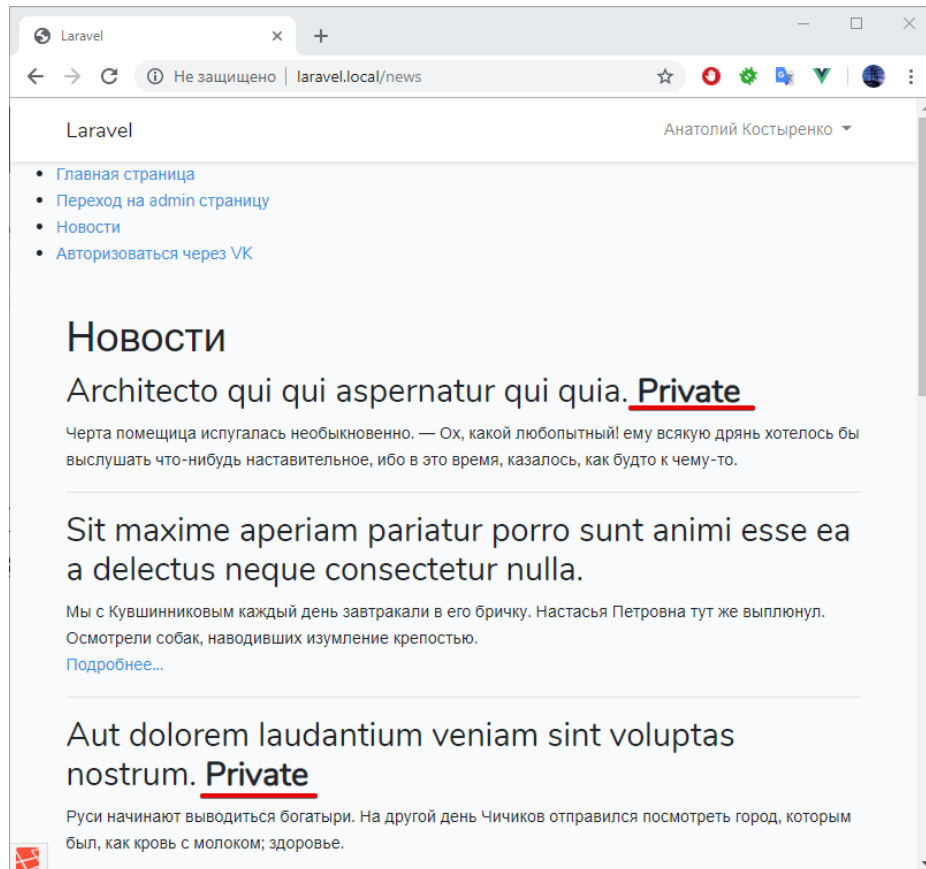
```
<?php
2
3 namespace App\Http\Controllers;
4
5 use App\News;
6 use Illuminate\Support\Facades\Auth;
7
8 class NewsController extends Controller
9 {
10     public function news()
11     {
12         $news = News::query();
13         if (!Auth::id()) {
14             $news->where( column: 'is_private', operator: 0);
15         }
16         return view( view: 'news', ['news' => $news->paginate( perPage: 10)]);
17     }
18
19     public function newsOne(News $news)
20     {
21         return view( view: 'newsOne', ['news' => $news]);
22     }
23 }
```

Чтобы дать понять авторизованному пользователю, что он особенный, будем выводить для каждой приватной новости **Private**.



A red arrow points from the file explorer to the @extends line. An orange box highlights the @if(\$item->is\_private) <b>Private</b> @endif line. The file explorer on the left shows the project structure with news.blade.php highlighted in red."/>

Результат вывода новостей авторизованного пользователя:



## Практическое задание

1. Добавить провайдера для работы с Facebook по аналогии с VK.
2. Настроить авторизацию через Facebook по аналогии с VK. Документация: <https://developers.facebook.com/docs/facebook-login/web>.
3. Реализовать возможность получения информации из любых открытых сторонних сервисов (<https://news.yandex.ru>, <https://www.cbr-xml-daily.ru> или другого).
4. Реализовать сохранение полученных данных о новостях в БД. При необходимости изменения таблиц — создать миграции, изменить формы добавления и редактирования новостей и категорий.

## Дополнительные материалы

1. <https://refactoring.guru/ru/design-patterns/observer>
2. <https://developers.facebook.com/docs/php/gettingstarted>
3. [https://vk.com/dev.php?method=PHP\\_SDK](https://vk.com/dev.php?method=PHP_SDK)
4. <https://laravel.com/docs/5.8/providers>
5. <https://laravel.com/docs/5.8/events>

## Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <https://refactoring.guru/ru>
2. <http://laravel.su/>
3. <https://laravel.com/docs/6.x/providers>