

Laravel

Урок 2. Роутинг. Использование контроллеров

Принципы работы Laravel. Роутинг, контроллеры. Основы маршрутизации. Какие роуты (маршруты) необходимы для работы агрегатора данных.

Оглавление

[Теория](#)

[Принцип работы Laravel](#)

[Роутинг](#)

[Контроллеры](#)

[Artisan](#)

[Практика](#)

[Создание контроллеров](#)

[Использование роутов](#)

[Передача параметров в route через адресную строку](#)

[Немного о хелперах в Laravel](#)

[Использование имен для роутов](#)

[Группировки роутов](#)

[Практическое задание](#)

[Дополнительные материалы](#)

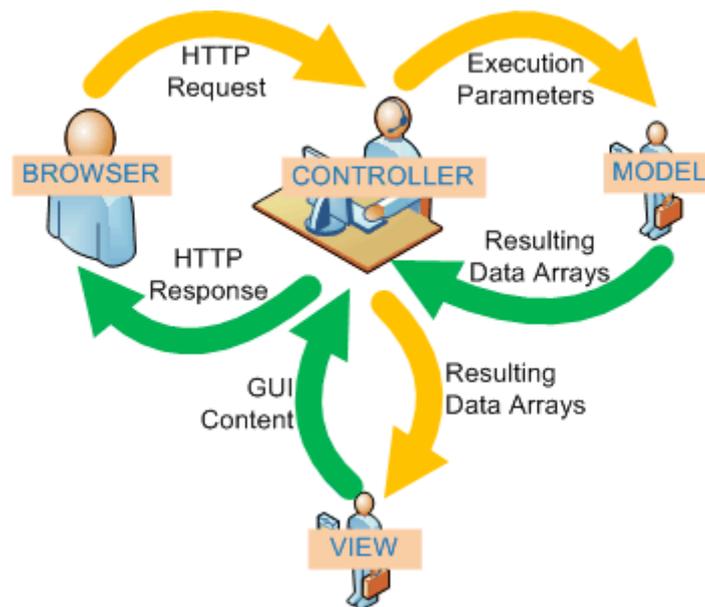
[Используемая литература](#)

Теория

Принцип работы Laravel

Фреймворк Laravel написан с использованием шаблона проектирования MVC. Он позволяет отделить логику самого приложения от его представления (шаблонов). Такое разделение дает возможность работать в одном проекте параллельно фронтендерам и бэкенд-разработчикам.

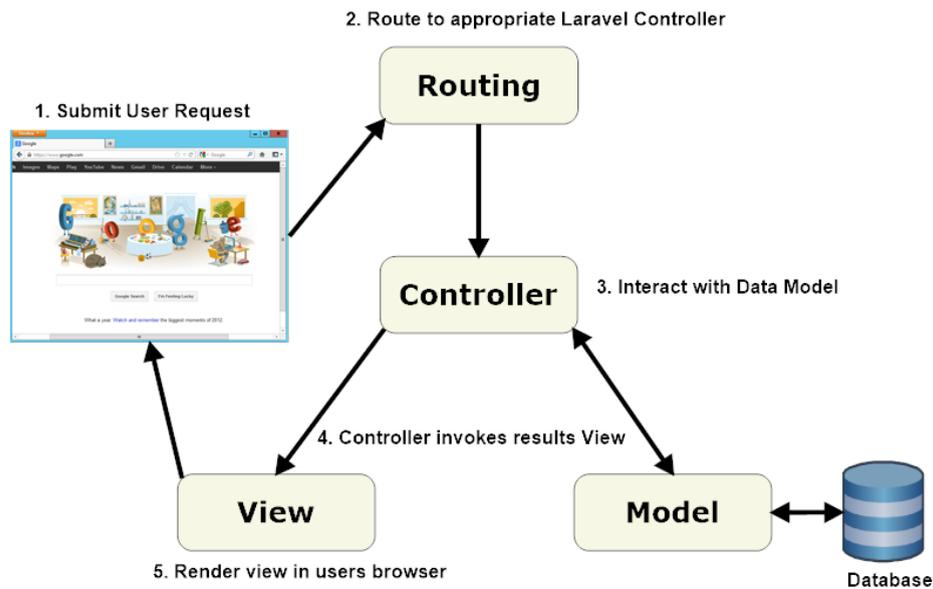
Схема работы паттерна:



Контроллер в этом шаблоне отвечает за прием и нормализацию данных из запроса пользователя и передачу их в модель. Модель — за расчеты, выборку и изменения данных из БД. Также вся логика обработки данных в идеале должна содержаться в ней. Вид отвечает за вывод информации на экран с минимальным набором логических изменений.

Более подробно данный подход в Laravel можно представить следующим образом (см. рисунок ниже):

1. Пользователь отправляет запрос в приложение.
2. По URL-пути определяется необходимый роут (набор правил для выполнения действий при конкретном запросе пользователя), который в свою очередь передает запрос пользователя в контроллер.
3. Контроллер может вызывать нужную модель (когда это необходимо), которая обрабатывает переданную для нее информацию из контроллера. При необходимости модель вносит изменения в базу данных или получает дополнительную информацию из нее. Обработанные данные возвращаются в контроллер.
4. Данные передаются в представление (view, шаблон). Шаблон наполняется этими данными и подготавливает их для вывода в браузер пользователя.



Но, как мы видели на предыдущем уроке, данные из роута могут сразу выводить информацию на экран. Такое использование, как правило, не самый удачный и эффективный вариант.

Роутинг

Роутинг — создание маршрутизации в приложении. Роуты — набор правил, определяющих, что делать с запросом пользователя по методу запроса, и его URL-путь. Таких наборов правил в Laravel несколько. В этом курсе рассмотрим роуты для веба. Эти правила хранятся в файле `~/code/laravel/routes/web.php`. Это тот файл, который мы изменили на предыдущем занятии.

```

1 <?php
2
3 /*
4  |-----|
5  | Web Routes |
6  |-----|
7  |
8  | Here is where you can register web routes for your application. These
9  | routes are loaded by the RouteServiceProvider within a group which
10 | contains the "web" middleware group. Now create something great!
11 |
12 /*
13
14 Route::get('/', function () {
15     return view('welcome');
16 });
17
  
```

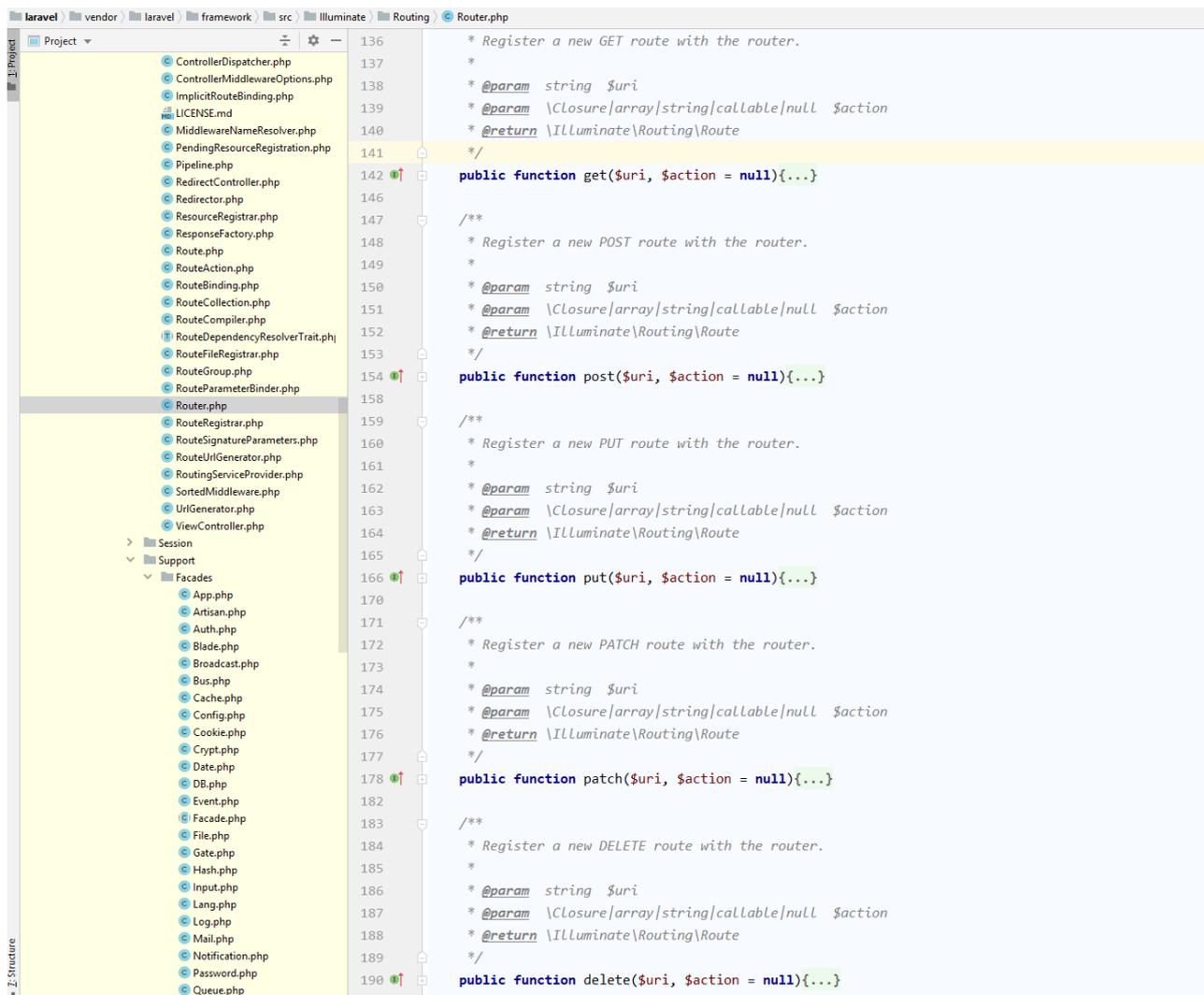
Имя Http метода (тип запроса) — `get`

URL-путь (URN) — `'/'`

Callback function или array - набор параметров — `function () { return view('welcome'); }`

На рисунке выше мы видим вызов статического метода у класса **Route**. Он является фасадом — оберткой для получения другого класса из сервис-контейнера. Задача сервис-контейнера — обеспечивать возможность в разных местах приложения использовать одни экземпляры выбранных сервисов, не создавая их копии после первой инициализации. В нашем случае получаем класс `Illuminate\Routing\Router`, в котором и находится динамический метод `get`. При первом обращении к

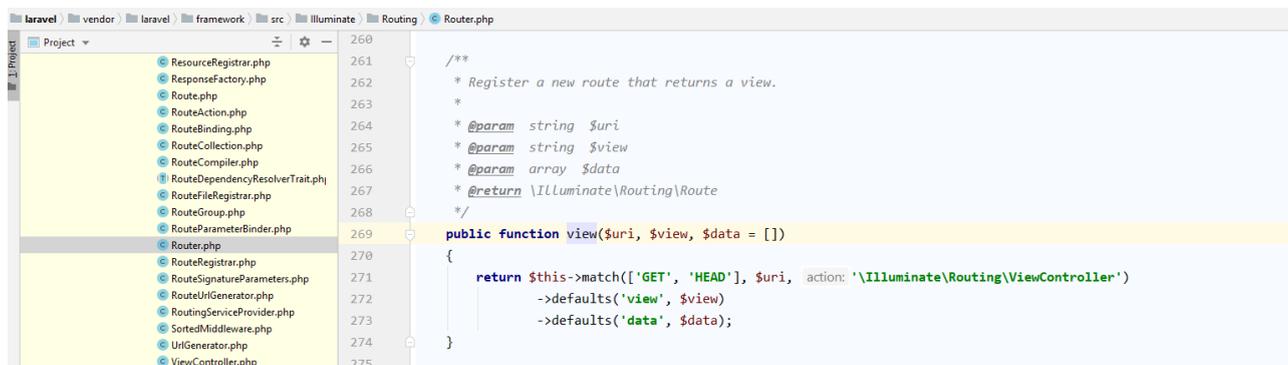
фасаду **Route** будет создан экземпляр класса **Illuminate\Routing\Router**, в котором и будут выполняться этот и все последующие методы **get** или другие методы данного класса. Листинг данного класса:



```
136 * Register a new GET route with the router.
137 *
138 * @param string $uri
139 * @param \Closure|array|string|callable|null $action
140 * @return \Illuminate\Routing\Route
141 */
142 public function get($uri, $action = null){...}
143
144 /**
145  * Register a new POST route with the router.
146  *
147  * @param string $uri
148  * @param \Closure|array|string|callable|null $action
149  * @return \Illuminate\Routing\Route
150  */
151 public function post($uri, $action = null){...}
152
153 /**
154  * Register a new PUT route with the router.
155  *
156  * @param string $uri
157  * @param \Closure|array|string|callable|null $action
158  * @return \Illuminate\Routing\Route
159  */
160 public function put($uri, $action = null){...}
161
162 /**
163  * Register a new PATCH route with the router.
164  *
165  * @param string $uri
166  * @param \Closure|array|string|callable|null $action
167  * @return \Illuminate\Routing\Route
168  */
169 public function patch($uri, $action = null){...}
170
171 /**
172  * Register a new DELETE route with the router.
173  *
174  * @param string $uri
175  * @param \Closure|array|string|callable|null $action
176  * @return \Illuminate\Routing\Route
177  */
178 public function delete($uri, $action = null){...}
179
180
```

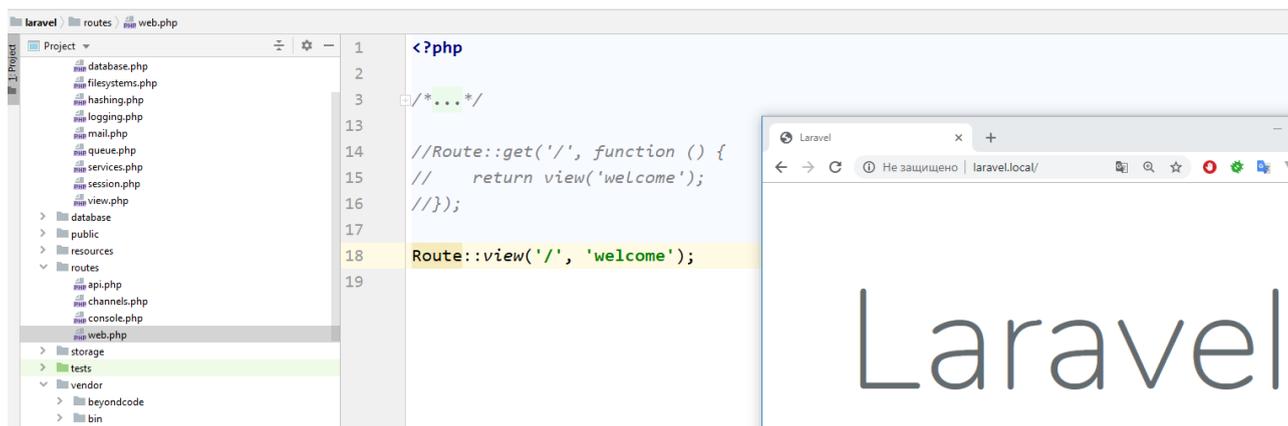
В этом классе мы видим и другие методы, которые можно использовать для определения роута по остальным методам http-запросов. В аннотации к этим методам можно увидеть, какие параметры должны быть переданы для их работы.

Среди методов этого класса есть тот, что будет нам возвращать указанное представление при get-запросе:



```
260 /**
261  * Register a new route that returns a view.
262  *
263  * @param string $uri
264  * @param string $view
265  * @param array $data
266  * @return \Illuminate\Routing\Route
267  */
268 public function view($uri, $view, $data = [])
269 {
270     return $this->match(['GET', 'HEAD'], $uri, $action: '\\Illuminate\\Routing\\ViewController')
271         ->defaults('view', $view)
272         ->defaults('data', $data);
273 }
274
275
```

Используя данный метод, получим результат, аналогичный предыдущему.



Контроллеры

Контроллеры — классы приложения, выступающие посредниками между запросом пользователя и шаблонами представления. По умолчанию контроллеры хранятся в директории `~/code/laravel/app/Http/Controllers`. В недавно установленном Laravel мы видим только один класс-контроллер — базовый, все остальные контроллеры должны его наследовать.



Новый класс контроллера можно создать и вручную — и при помощи специальной утилиты **artisan**.

Artisan

Artisan — интерфейс командной строки, который используется в Laravel для выполнения команд, ускоряющих разработку приложения. Чтобы использовать его, следует перейти в каталог приложения Laravel и ввести команду `php artisan`. Она выведет версию установленного фреймворка и список команд, которые доступны для выполнения через **artisan**.

```
vagrant@homestead: ~/code/laravel
vagrant@homestead:~/code/laravel$ php artisan
Laravel Framework 5.8.33

Usage:
  command [options] [arguments]

Options:
  -h, --help            Display this help message
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
                      --ansi             Force ANSI output
                      --no-ansi        Disable ANSI output
  -n, --no-interaction Do not ask any interactive question
  --env[=ENV]          The environment the command should run under
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output
                      2 for more verbose output and 3 for debug

Available commands:
  clear-compiled  Remove the compiled class file
  down           Put the application into maintenance mode
  dump-server    Start the dump server to collect dump information.
  env           Display the current framework environment
  help         Displays help for a command
  inspire     Display an inspiring quote
  list       Lists commands
  migrate    Run the database migrations
  optimize   Cache the framework bootstrap files
  preset     Swap the front-end scaffolding for the application
  serve     Serve the application on the PHP development server
  tinker   Interact with your application
  up       Bring the application out of maintenance mode
```

Чтобы получить подробную информацию по команде, следует указать после **artisan** слово **help** и имя нужной команды.

```
vagrant@homestead: ~/code/laravel
vagrant@homestead:~/code/laravel$ php artisan help make:controller
Description:
  Create a new controller class

Usage:
  make:controller [options] [--] <name>

Arguments:
  name                The name of the class

Options:
  -m, --model[=MODEL]  Generate a resource controller for the given model.
  -r, --resource        Generate a resource controller class.
  -i, --invokable      Generate a single method, invokable controller class.
  -p, --parent[=PARENT] Generate a nested resource controller class.
  --api               Exclude the create and edit methods from the controller.
  -h, --help          Display this help message
  -q, --quiet         Do not output any message
  -V, --version       Display this application version
  --ansi             Force ANSI output
  --no-ansi          Disable ANSI output
  -n, --no-interaction Do not ask any interactive question
  --env[=ENV]        The environment the command should run under
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
vagrant@homestead:~/code/laravel$
```

Практика

Создание контроллеров

Чтобы создать контроллер, воспользуемся консолью. Введем команду: ``php artisan make:controller HomeController``

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller
{
    //
}
```

```
vagrant@homestead: ~/code/laravel
vagrant@homestead:~/code/laravel$ php artisan make:controller HomeController
Controller created successfully.
vagrant@homestead:~/code/laravel$
```

Обратим внимание: в папке **Controllers** был создан указанный нами контроллер. Добавим еще один — он будет находиться в папке **Admin**. Для этого укажем не только имя нового контроллера, но и его папку, разделив их ``/``

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller
{
    //
}
```

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

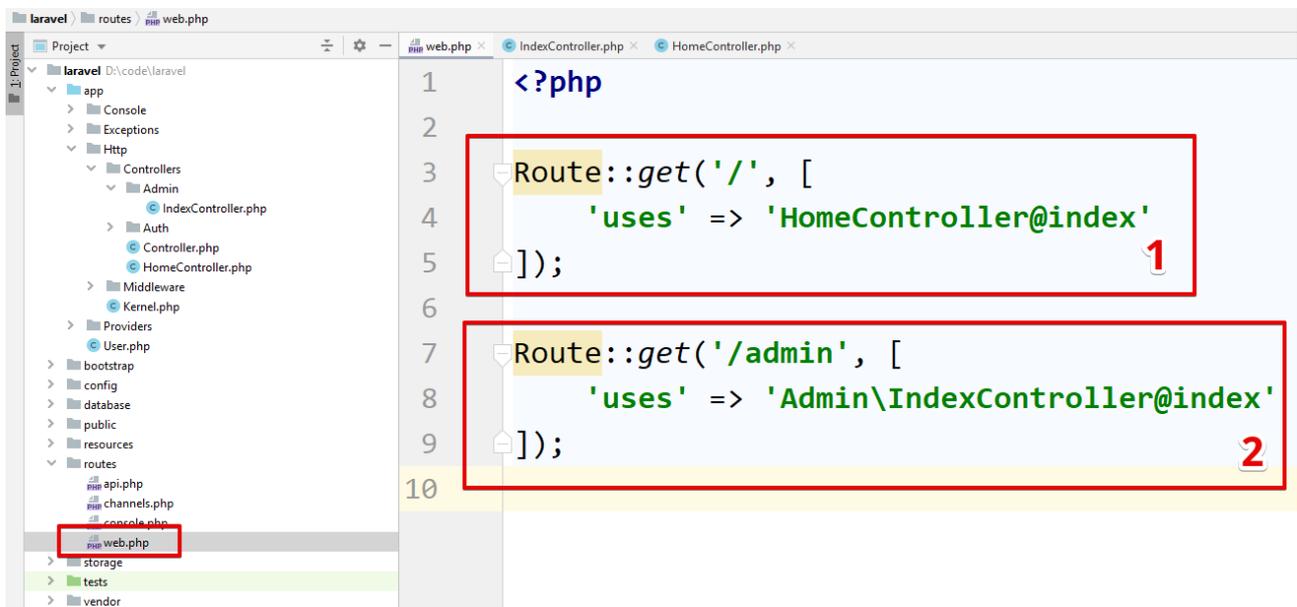
Теперь в каждом контроллере создадим по одному методу — они будут возвращать небольшую html-разметку:

```
IndexController.php
<?php
namespace App\Http\Controllers\Admin;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
class IndexController extends Controller
{
    public function index()
    {
        return <<<php
        <h1>Точка входа для админа</h1>
        Тут какой-то текст<br>
        <a href="/">Переход на главную страницу</a>
    }
}

HomeController.php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class HomeController extends Controller
{
    public function index()
    {
        return <<<php
        <h1>Приветствие пользователя</h1>
        Тут какой-то текст<br>
        <a href="/">Переход на админ страницу</a>
    }
}
```

Использование роутов

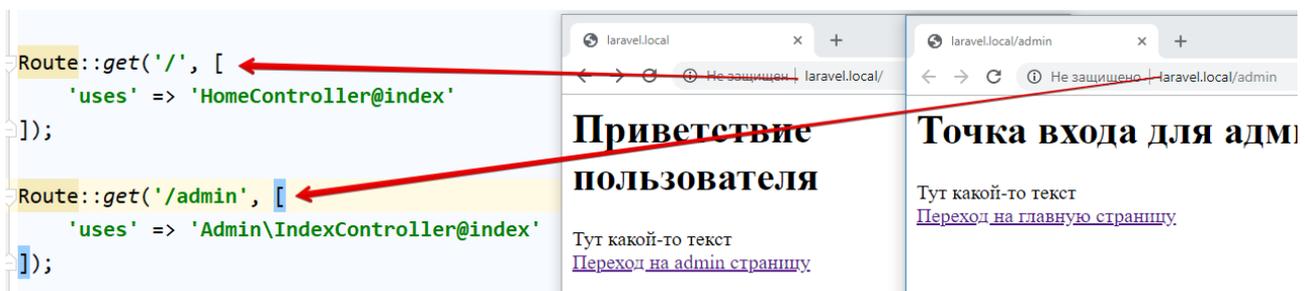
Теперь, когда созданы несколько контроллеров с необходимыми методами, добавим несколько роутов для перенаправления на эти методы.



```
1 <?php
2
3 Route::get('/', [
4     'uses' => 'HomeController@index'
5 ]);
6
7 Route::get('/admin', [
8     'uses' => 'Admin\IndexController@index'
9 ]);
10
```

На рисунке выше показаны два роута. Для первого добавленного нами роута используется метод **get**, и в него передаются два параметра: строка и массив. Строка определяет URL-путь, которому соответствует данный роут, а в массиве передаются его параметры. Ключ элемента массива отвечает за имя параметра, а значение элемента массива — то, которое мы хотим для него передать. В нашем случае передается только один параметр `uses` со значением `HomeController@index`. Данный параметр указывает на то, какой контроллер нужно вызвать и какой метод в нем следует использовать. Имя метода от названия контроллера отделено при помощи `@`.

Второй роут тоже будет работать при обращении к приложению методом **get**, но при другом URL-пути. В качестве параметра роута передан также массив с одним элементом. Ключ этого элемента такой же, как и в первом случае — значит, он отвечает за то, какой метод и в каком контроллере будет вызван. Поскольку **IndexController** находится в пространстве имен `App\Http\Controllers\Admin`, а ожидается, что он будет в `App\Http\Controllers`, то в значение параметра указана недостающая часть полного имени класса `Admin`. После `@` указан метод, который следует вызвать. Пример выполнения:



Теперь добавим еще несколько методов в один из контроллеров и роуты с одинаковыми URL-путями и методами запросов, но с вызовом разных методов.

```
web.php × HomeController.php × IndexController.php ×
1 <?php
2
3 Route::get('/', [
4     'uses' => 'HomeController@index'
5 ]);
6
7 Route::get('/admin', [
8     'uses' => 'Admin\IndexController@index'
9 ]);
10
11 Route::get('/test', [
12     'uses' => 'Admin\IndexController@test1'
13 ]);
14 Route::get('/test', [
15     'uses' => 'Admin\IndexController@test2'
16 ]);
17
18
19
20
21
22
23
1 <?php
2
3 namespace App\Http\Controllers\Admin;
4
5 use Illuminate\Http\Request;
6 use App\Http\Controllers\Controller;
7
8 class IndexController extends Controller
9 {
10     public function test1()
11     {
12         return <<<php
13             <h1>Test1</h1>
14         php;
15     }
16
17     public function test2()
18     {
19         return <<<php
20             <h1>Test2</h1>
21         php;
22     }
23 }
```

Как вы думаете, какой в итоге метод будет вызван и что будет выведено в браузере?

Предлагаем вам самостоятельно ответить на этот вопрос и объяснить, почему так происходит.

Передача параметров в route через адресную строку

Создадим новый контроллер, ответственный за вывод новостей. Добавим в него приватную переменную, которая будет содержать массив данных, и метод, который красиво выведет все эти данные пользователю.

На рисунке ниже представлен листинг данного задания. На 26-й строке этого скриншота создается новый путь, который мы пока не знаем как указать в **web.php**. Его особенность в том, что последняя часть этого адреса будет динамичной — в конце мы будем передавать id новости, которую хотим получить.

```
web.php × IndexController.php × NewsController.php × HomeController.php × NewsController.php ×
12
13 Route::get('/admin', [
14     'uses' => 'Admin\IndexController@index'
15 ]);
16
17
18 Route::get('/news', [
19     'uses' => 'NewsController@news'
20 ]);
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

namespace App\Http\Controllers;
use Illuminate\Http\Request;
class NewsController extends Controller
{
    private $news = [
        [
            'id' => 1,
            'title' => 'Новость',
            'inform' => 'Новость 1 Тут подробненько о ней',
            [...]
        ]
    ];

    public function news()
    {
        $html = "<h1>Новости</h1>";
        foreach ($this->news as $news) {
            $html .= <<<php
            <h2>
26         <a href="/news/{ $news['id'] }">
27             { $news['title'] }
28         </a>
29         </h2>
30         <div>{ $news['inform'] }</div>
31         <hr>
32     </php>
33     }
34     return $html;
35 }
}
```

Чтобы в роуте был указан такой динамичный параметр, его достаточно поместить в фигурные скобки, а в самом методе указать его имя в качестве параметра:

```
web.php × NewsController.php ×
12
13 Route::get('/admin', [
14     'uses' => 'Admin\IndexController@index'
15 ]);
16
17
18 Route::get('/news', [
19     'uses' => 'NewsController@news'
20 ]);
21
22 Route::get('/news/{id}', [
23     'uses' => 'NewsController@newsOne',
24 ]);
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

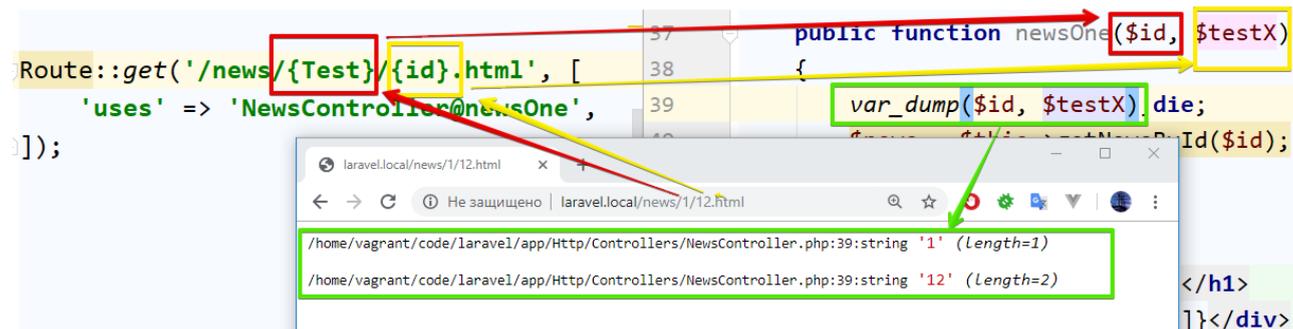
public function newsOne($id)
{
    $news = $this->getNewsById($id);

    if (!empty($news)) {
        $html = <<<php
        <h1>{ $news['title'] }</h1>
        <div>{ $news['inform'] }</div>
        <hr>
        <a href="/news">Назад</a>
        </php>
        return $html;
    }

    return redirect(to: '/news');
}

private function getNewsById($id)
{
    foreach ($this->news as $news) {
        if ($news['id'] == $id) {
            return $news;
        }
    }
    return [];
}
```

Таких параметров при необходимости может быть передано и несколько. Предлагаем взглянуть на рисунок ниже и сделать дополнительные выводы по использованию динамических значений в роутах.



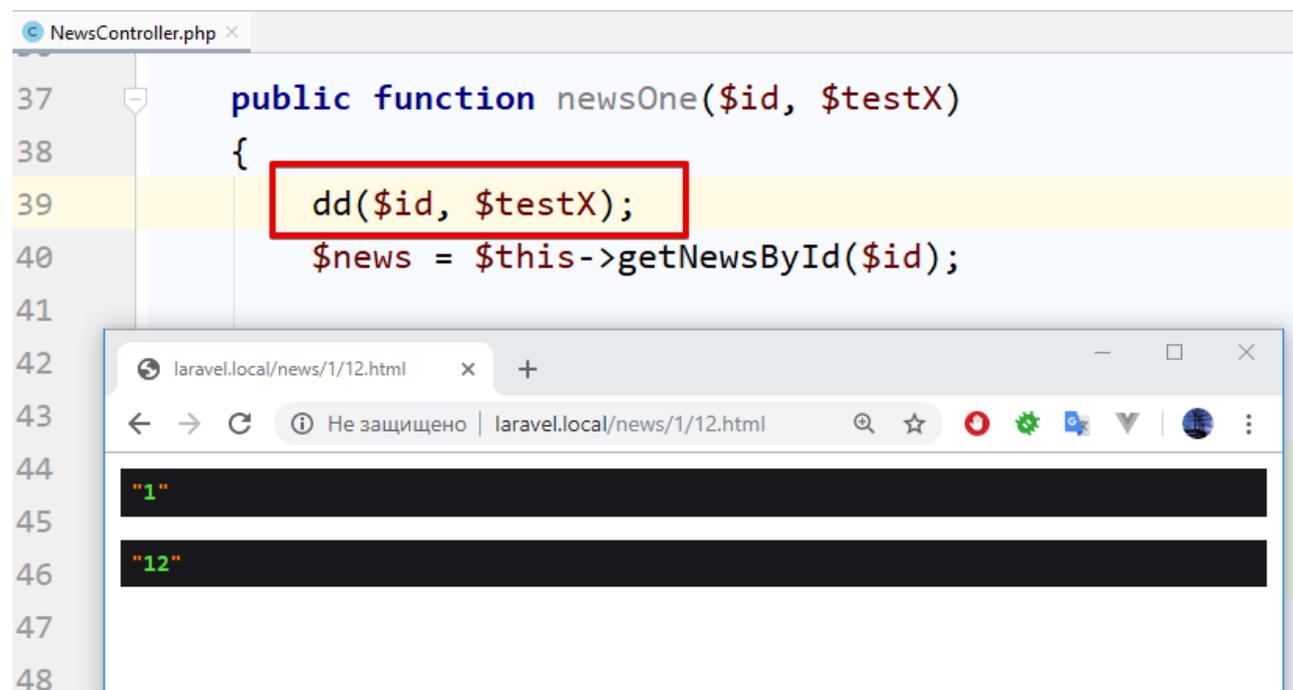
Немного о хелперах в Laravel

В Laravel используется множество вспомогательных функций (хелперов), которые заметно облегчают разработку. Более подробно о них: <https://laravel.com/docs/5.8/helpers>

Посмотрим на работу некоторых из них:

- **dump()** — выводит значение переменной. Работает аналогично **var_dump()**, но его вывод не зависит от настроек вашего сервера;
- **dd()** — действует, как **dump()**, но завершает работу приложения.

Таким образом, вывод значения переменных и остановку выполнения кода мы можем заменить хелпером **dd()**.



Ранее уже использовался хелпер **redirect**, перенаправляющий http-запрос на нужный адрес или роут.

В курсе мы еще столкнемся с хелперами.

Использование имен для роутов

Представим ситуацию: программисты создали большое приложение с множеством страниц и задач. Смотрит на него заказчик и видит, что в адресе, по которому происходит вывод информации из личного кабинета, указано **user** — а ему хочется, чтобы там было именно **account**. И тогда происходит большой рефакторинг всего, что должно переходить в этот раздел. А один разработчик пропускает это совещание — и после больничного выкатывает свою новую фичу, которую проверил раньше, но не протестировал теперь, и получает 404-е ошибки.

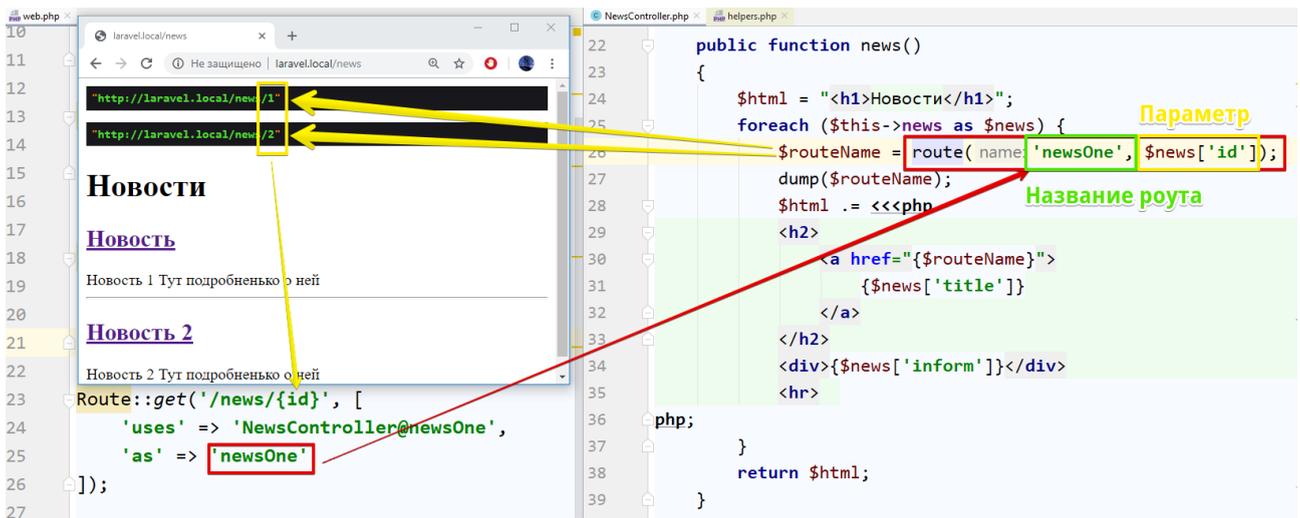
Чтобы избежать такой ситуации, в Laravel есть функциональность, которая позволяет давать каждому роуту имя и использовать в самом приложении именно его. Если в роуте понадобится изменить путь, то эти правки нужно будет сделать только в нем самом.

```
16
17
18 Route::get('/news', [
19     'uses' => 'NewsController@news',
20     'as' => 'news'
21 ]);
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37 public function newsOne($id)
38 {
39     $news = $this->getNewsById($id);
40
41     $nameRoute = route( name: 'news' );
42     dump($nameRoute);
43     if (!empty($news)) {
44         $html = <<<php
45             <h1>{$news['title']}</h1>
46             <div>{$news['inform']}</div>
47             <hr>
48             <a href="{ $nameRoute }">Назад</a>
49         php;
50     }
51     return $html;
52
53     return redirect($nameRoute);
54 }
55
```

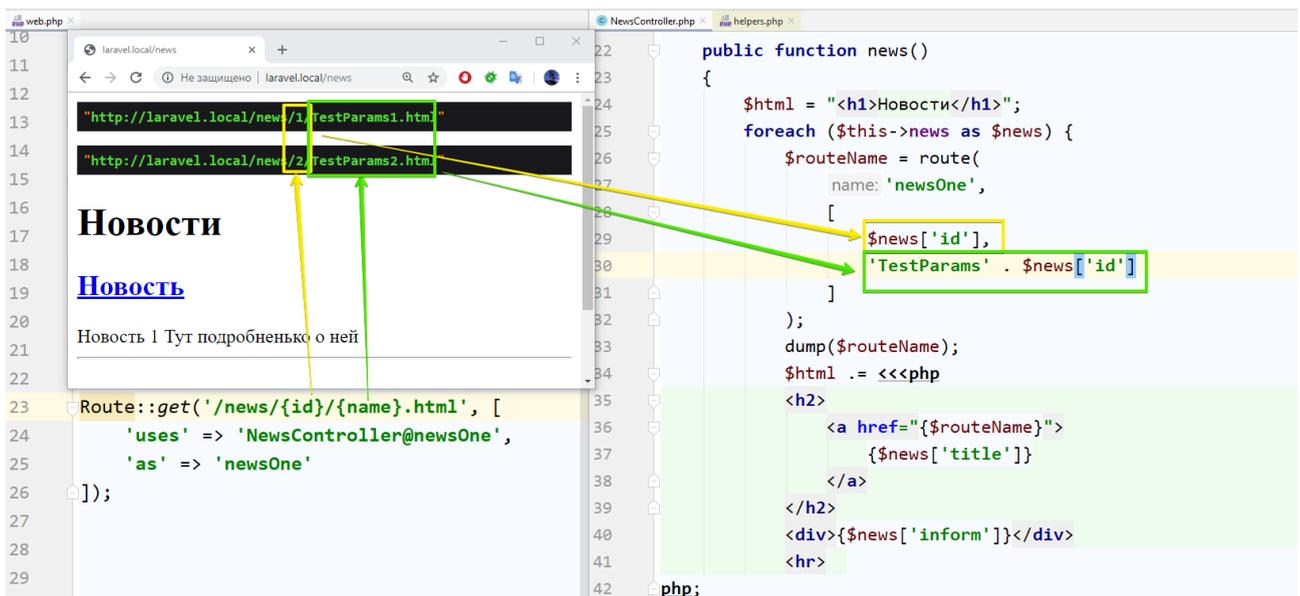
Browser window: `http://laravel.local/news`
Новость
Новость 1 Тут подробенько о ней
[Назад](#)

Рассмотрим более подробно пример, показанный выше. В нем для ранее созданного роута было добавлено имя — `'news'`. В **NewsController**, в методе **newsOne**, используется хелпер **route**. Он генерирует URL для указанного роута, возвращает строку. Далее в примере и будет использоваться результат выполнения данного хелпера. И если в **web.php** будет изменен адрес, то вносить правки в работу контроллера не потребуется.

Чтобы передать в хелпер **route** динамические параметры (например, `id` необходимой новости), следует поступить так:



Если в роуте требуется несколько параметров, то вместо второго, переданного в хелпере, передается массив.



Группировки роутов

Группировку роутов можно использовать, если надо применить к ним общие правила: общие префиксы в доменном имени или название роута, правила безопасности и другие. Группа роутов может включать в себя и другие группы.

На рисунке показано, как можно использовать группировку роутов. В данном примере указано:

- начало URL-пути — **'admin'** (a);
- добавленное в класс **IndexController** пространство имени **'Admin'** (b);
- Добавлен префикс в название роута — **'admin.'** (c).

В итоге преобразований код 1 и код 2 эквивалентны.

```

web.php
12
13 Route::group(
14     [
15         "prefix" => "admin", a
16         "namespace" => "Admin", b
17         "as" => "admin." c
18     ],
19     function () {
20         Route::get('/', [
21             'uses' => 'IndexController@index', 1
22             'as' => 'index'
23         ]);
24         Route::get('/test1', [
25             'uses' => 'IndexController@test1',
26             'as' => 'test1'
27         ]);
28         Route::get('/test2', [
29             'uses' => 'IndexController@test2',
30             'as' => 'test2'
31         ]);
32     });
33
34
35 //Route::get('/admin', [
36 //    'uses' => 'Admin\IndexController@index', 2
37 //    'as' => 'admin.index'
38 //]);
39
40
41
42
43
44
45
IndexController.php
3 namespace App\Http\Controllers\Admin;
4
5 use App\Http\Controllers\Controller;
6
7 class IndexController extends Controller
8 {
9     public function test1()
10    {
11        $route = route( name: 'admin.index');
12        return <<<php
13            <h1>Test1</h1>
14            <p style="color: red">
15                <a href="{ $route }">Admin</a><br>
16            </p>
17        </php>;
18    }
19
20    public function test2()
21    {
22        $route = route( name: 'admin.index');
23        return <<<php
24            <h1>Test2</h1>
25            <p style="color: red">
26                <a href="{ $route }">Admin</a><br>
27            </p>
28        </php>;
29    }
30
31    public function index()
32    {
33        $route1 = route( name: 'admin.test1');
34        $route2 = route( name: 'admin.test2');
35        return <<<php
36            <h1>Точка входа для админа</h1>
37            Тут какой-то текст<br>
38            <p style="color: red">
39                <a href="{ $route1 }">test1</a><br>
40                <a href="{ $route2 }">test2</a><br>
41            </p>
42            <a href="/">Переход на главную страницу</a>
43        </php>;
44    }
45 }

```

Чтобы посмотреть результаты используемых роутов в приложении, следует в терминале обратиться к интерфейсу **artisan**, введя команду `php artisan route:list`

```

vagrant@homestead: ~/code/laravel
vagrant@homestead:~/code/laravel$ php artisan route:list

```

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		App\Http\Controllers\HomeController@index	web
	GET HEAD	admin	admin.index	App\Http\Controllers\Admin\IndexController@index	web
	GET HEAD	admin/test1	admin.test1	App\Http\Controllers\Admin\IndexController@test1	web
	GET HEAD	admin/test2	admin.test2	App\Http\Controllers\Admin\IndexController@test2	web
	GET HEAD	api/user		Closure	api, auth:api
	GET HEAD	news	news	App\Http\Controllers\NewsController@news	web
	GET HEAD	news/{id}/{name}.html	newsOne	App\Http\Controllers\NewsController@newsOne	web

```

vagrant@homestead:~/code/laravel$

```

Практическое задание

1. Добавить в родительский контроллер методы для хранения данных, которые будет возвращать массивы. Массивы должны содержать информацию о категориях новостей (минимум 5), и новостях (минимум 4 для каждой категории).

2. Добавить в проект несколько контроллеров для вывода следующих страниц:
 - 1.1. Страница приветствия. Небольшая статическая страница с некоторой информацией о будущем агрегаторе новостей.
 - 1.2. Страница категорий новостей. Данная страница должна выводить все категории из данных созданных в первом задании.
 - 1.3. Страница вывода новостей по конкретной категории. Переход на эту страницу должен осуществляться по ссылке на странице категорий. Ссылка должна содержать параметр, который будет определять какие именно новости будут выведены. Новости получать из метода созданного в первом задании.
 - 1.4. Страница вывода конкретной новости. Переход на эту страницу должен осуществляться со страниц вывода новостей по конкретной категории, по ссылке, содержащей параметр определяющий какую конкретно новость нужно вывести. Данные получаем из методов созданных на первом задании.
 - 1.5. Страница авторизации. Страница должна содержать форму, в которой используются следующие элементы:
 - 1.5.1. Поле ввода логина
 - 1.5.2. Поле для ввода пароля
 - 1.5.3. Чекбокс для указания, что следует “Запомнить меня”
 - 1.5.4. Кнопка
 - 1.6. Страница добавления новости. Страница должна содержать следующие элементы формы:
 - 1.6.1. Поле для указания название новости
 - 1.6.2. Поле для подробного описания новости
 - 1.6.3. Поле для краткого описания новости

Дополнительные материалы

1. <https://laravel.ru/docs/v5/facades>
2. <http://laravel.su/docs/5.0/helpers>
3. <https://laravel.com/docs/5.8/facades>
4. <http://laravel.su/docs/5.4/lifecycle>
5. <https://laravel.com/docs/5.8/lifecycle>
6. <https://laravel.com/docs/5.8/routing>
7. <https://laravel.com/docs/5.8/controllers>

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <https://laravel.com/docs/5.8>
2. <http://laravel.su/>