GeekBrains

Laravel

Урок 2. Роутинг. Использование контроллеров

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

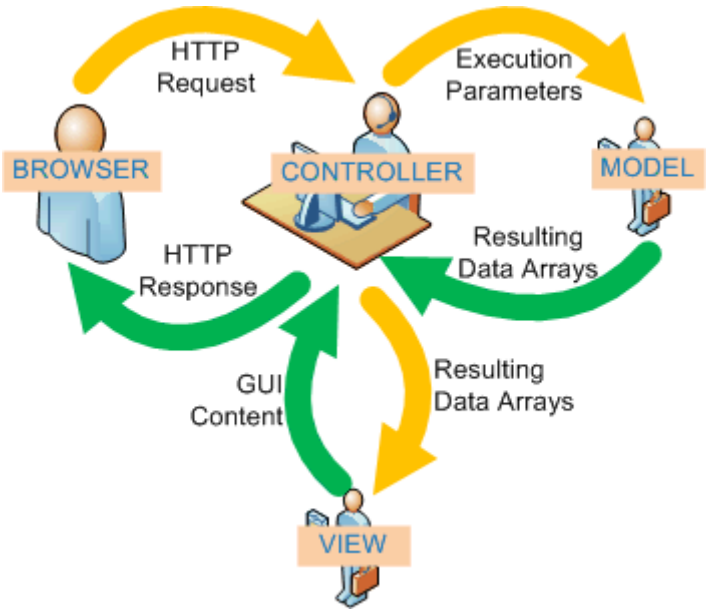
Оглавление

Теория

Принцип работы Laravel

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.

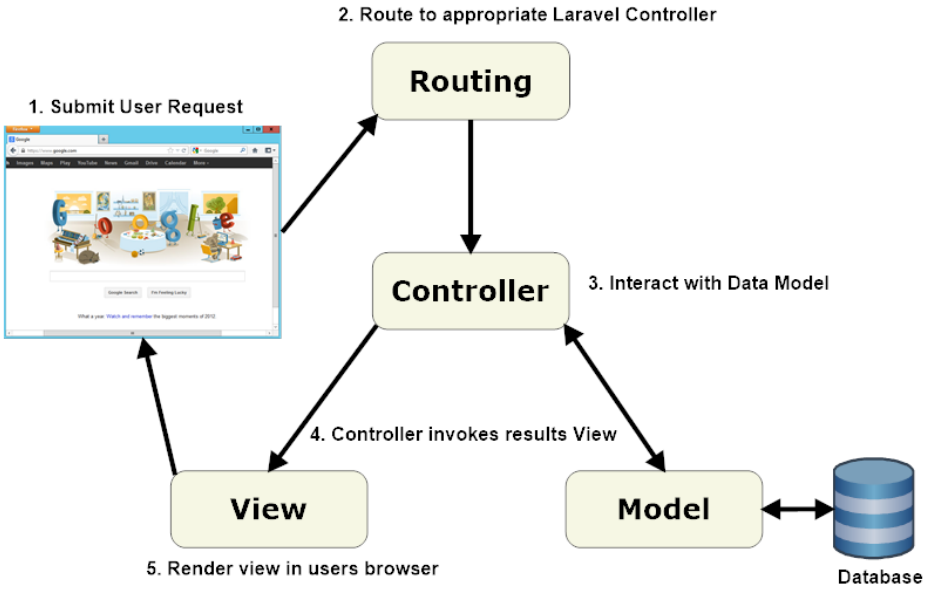
Более подробно данный подход в Laravel можно представить следующим образом (см. рисунок ниже):

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.

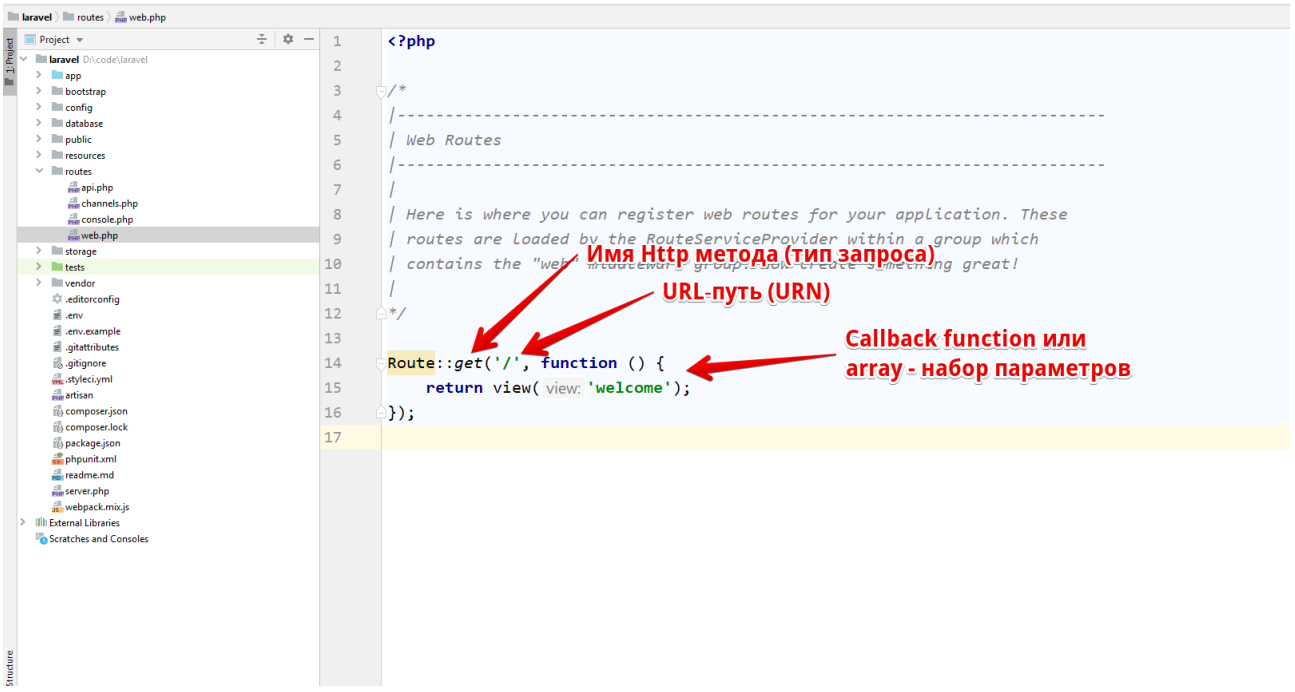
4. Данные передаются в представление (view, шаблон). Шаблон наполняется этими данными и подготавливает их для вывода в браузер пользователя.



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.

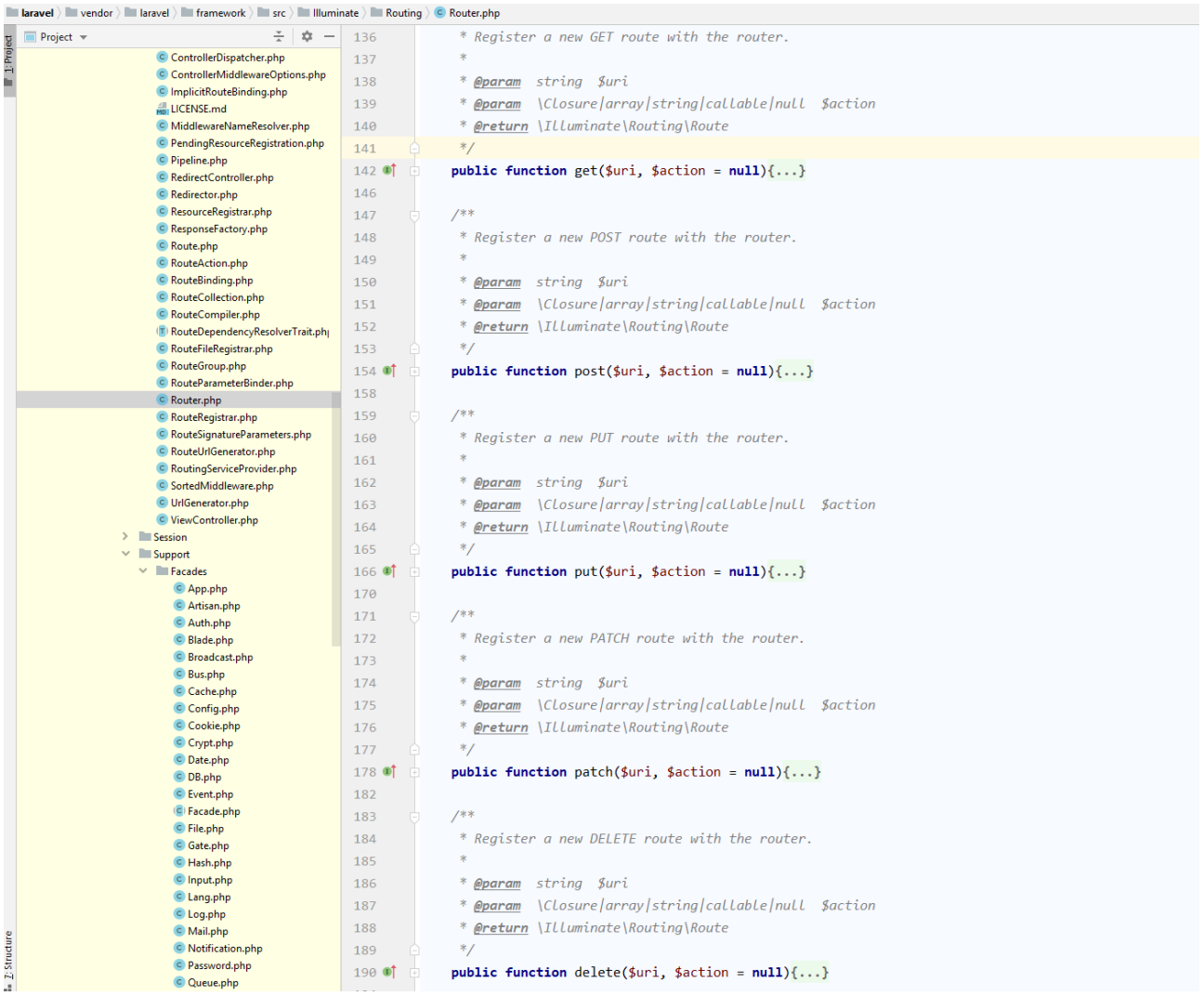
Роутинг

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.



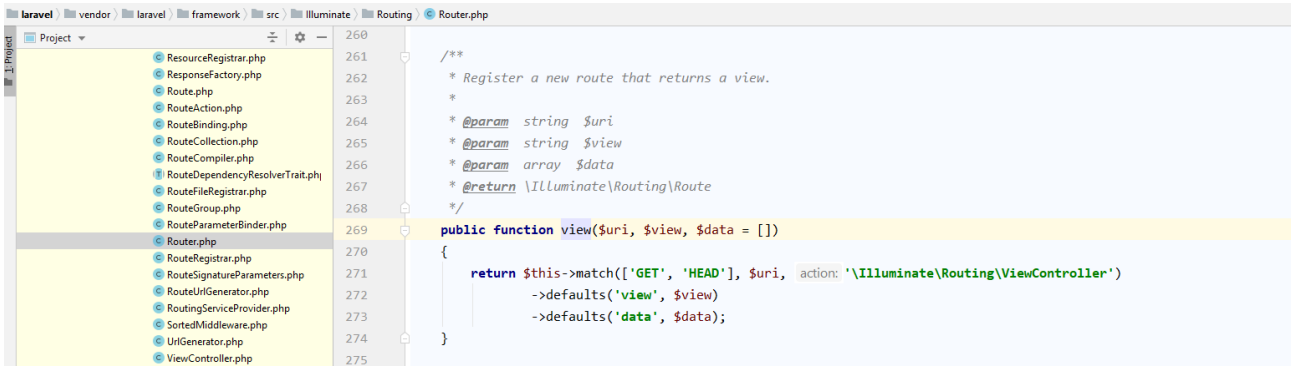
На рисунке выше мы видим вызов статического метода у класса **Route**. Он является фасадом — оберткой для получения другого класса из сервис-контейнера. Задача сервис-контейнера — обеспечивать возможность в разных местах приложения использовать одни экземпляры выбранных сервисов, не создавая их копии 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:



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.

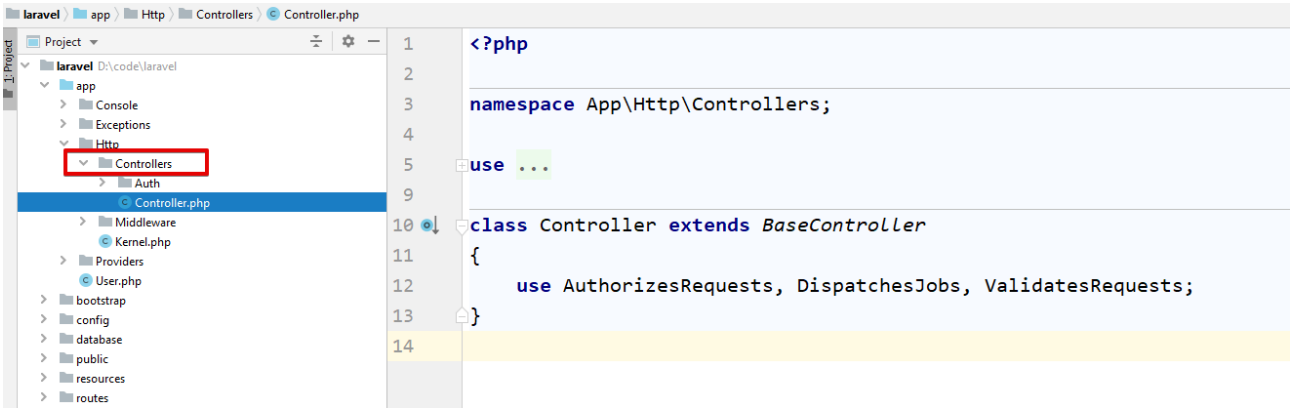
Среди методов этого класса есть тот, что будет нам возвращать указанное представление при get-запросе:



Используя данный метод, получим результат, аналогичный предыдущему.

Контроллеры

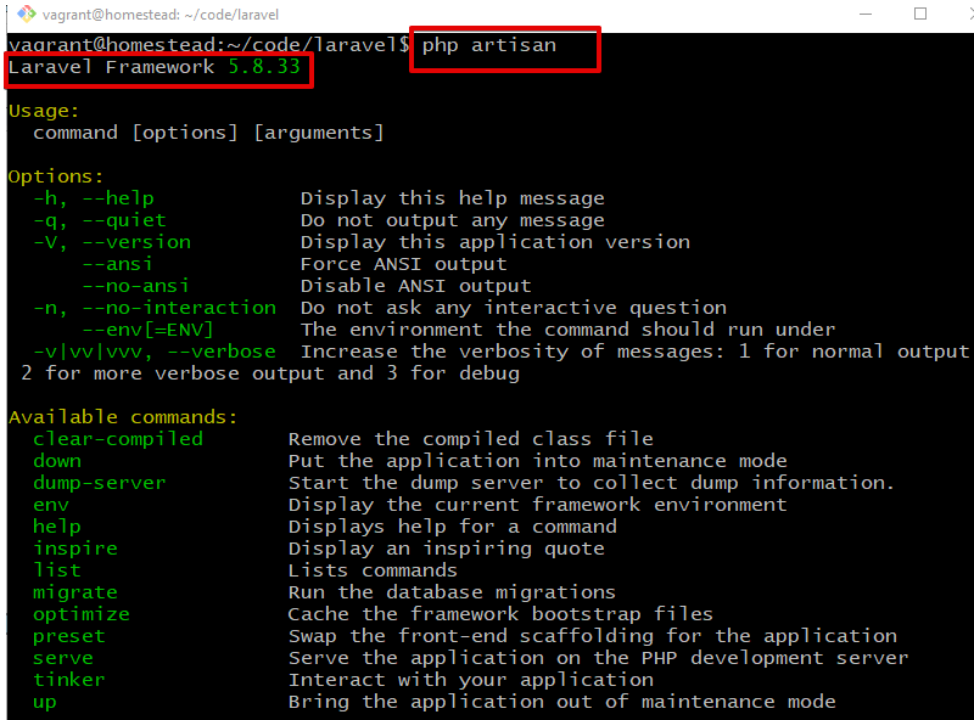
Контроллеры — классы приложения, выступающие посредниками между запросом пользователя и шаблонами представления. По умолчанию контроллеры хранятся в директории **~/code/laravel/app/Http/Controllers**. В недавно установленном Laravel мы видим только один класс-контроллер — базовый, все остальные контроллеры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



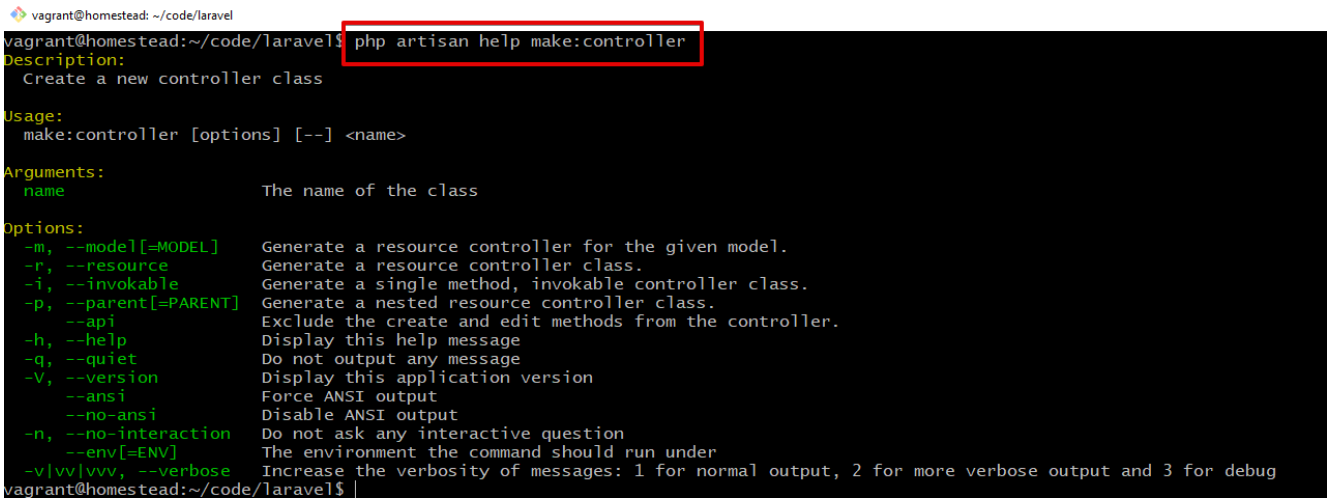
Новый класс контроллера можно создать и вручную — и при помощи специальной утилиты **artisan**.

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**.



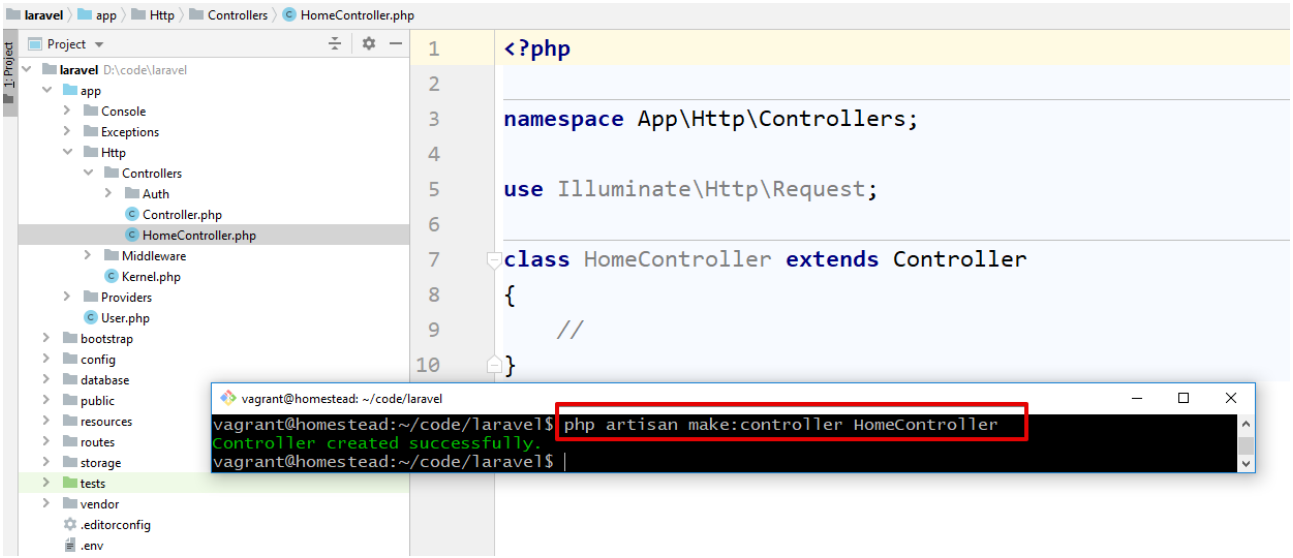
Чтобы получить подробную информацию по команде, следует указать после **artisan** слово **help** и имя нужной команды.



Практика

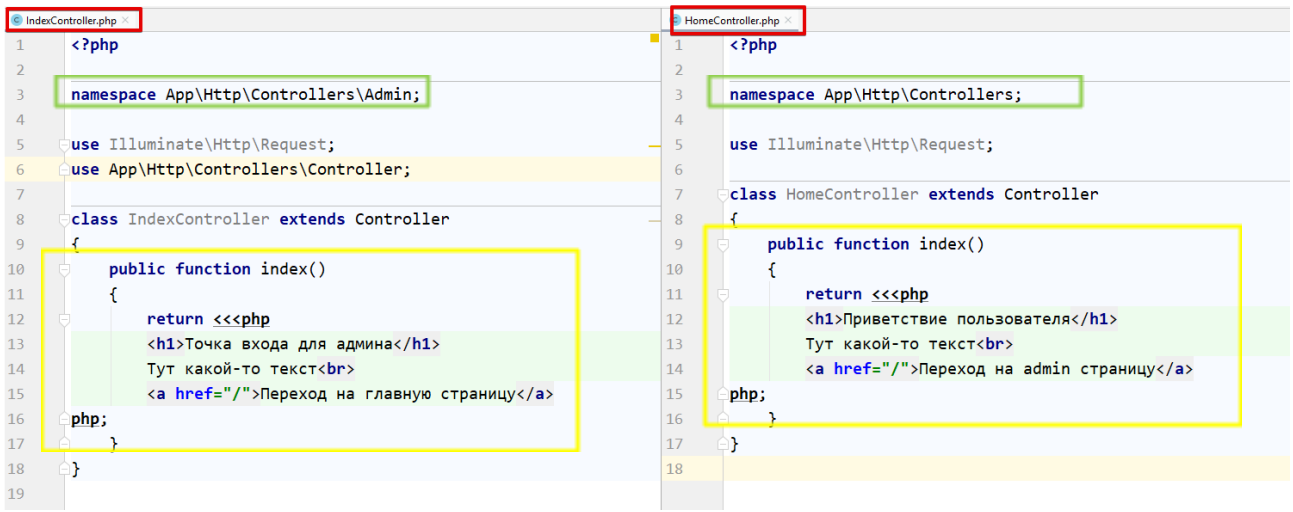
Создание контроллеров

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



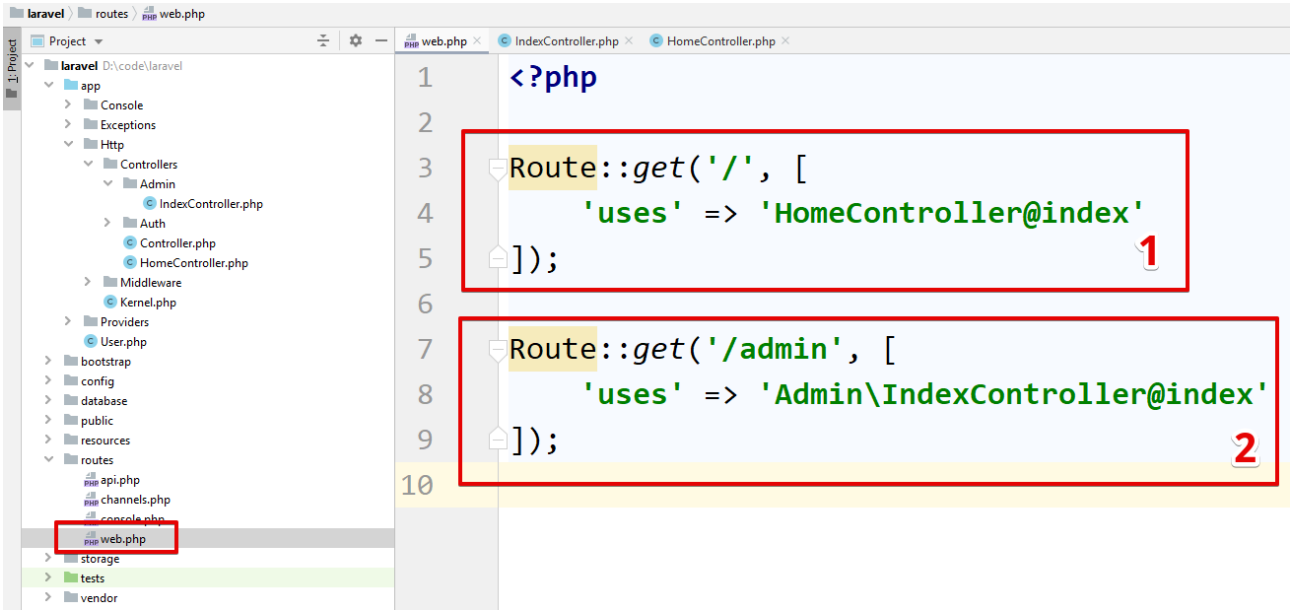
Обратим внимание: в папке **Controllers** был создан указанный нами контроллер. Добавим еще один — он будет находиться в папке **Admin**. Для этого укажем не только имя нового контроллера, но и его папку, разделив их `/`

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



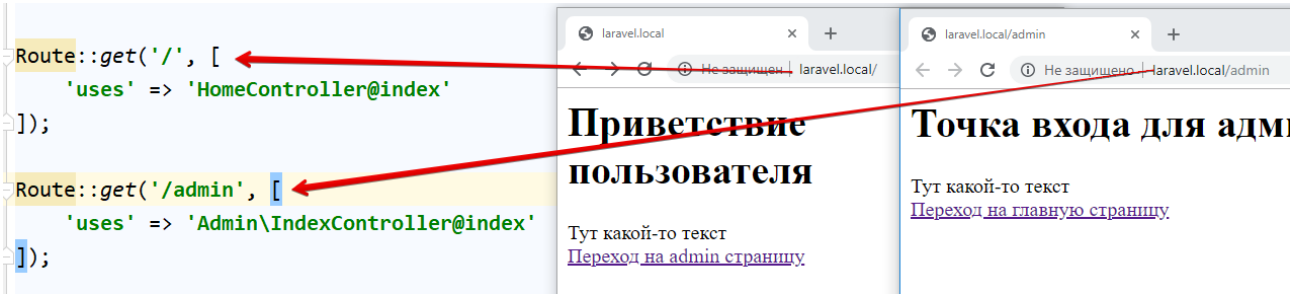
Использование роутов

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



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

```php
<?php

Route::get('/', [
    'uses' => 'HomeController@index'
]);

Route::get('/admin', [
    'uses' => 'Admin\IndexController@index'
]);

Route::get('/test', [
    'uses' => 'Admin\IndexController@test1'
]);
Route::get('/test', [
    'uses' => 'Admin\IndexController@test2'
]);
```

IndexController.php

```php
<?php

namespace App\Http\Controllers\Admin;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class IndexController extends Controller
{
    public function test1()
    {
        return <<<php
        <h1>Test1</h1>
php;
    }

    public function test2()
    {
        return <<<php
        <h1>Test2</h1>
php;
    }
```

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.

```php
// web.php
Route::get('/admin', [
    'uses' => 'Admin\IndexController@index'
]);

Route::get('/news', [
    'uses' => 'NewsController@news'
]);
```

```php
// NewsController.php
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) {
            $html .= <<<php
            <h2>
                <a href="/news/{$news['id']}">
                    {$news['title']}
                </a>
            </h2>
            <div>{$news['inform']}</div>
            <hr>
php;
        }
        return $html;
    }
}
```

/news/1

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

```php
// web.php
Route::get('/admin', [
    'uses' => 'Admin\IndexController@index'
]);

Route::get('/news', [
    'uses' => 'NewsController@news'
]);

Route::get('/news/{id}', [
    'uses' => 'NewsController@newsOne',
]);
```

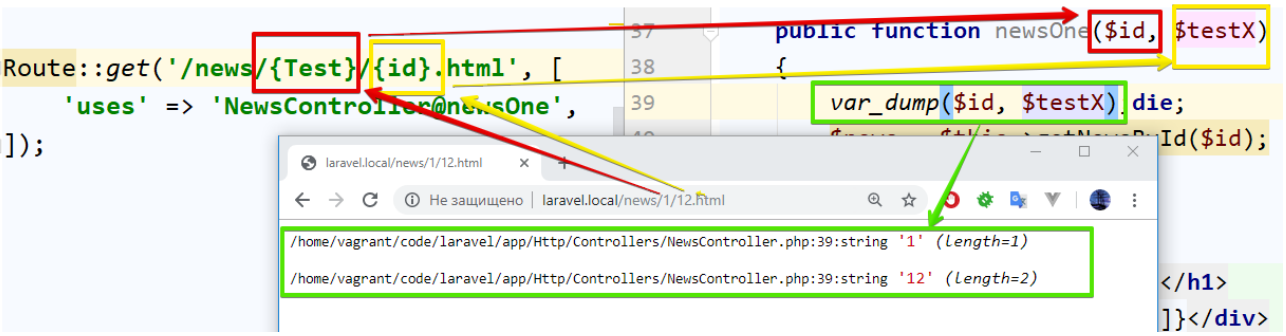
```php
// NewsController.php
    public function newsOne($id)
    {
        $news = $this->getNewsById($id);

        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.



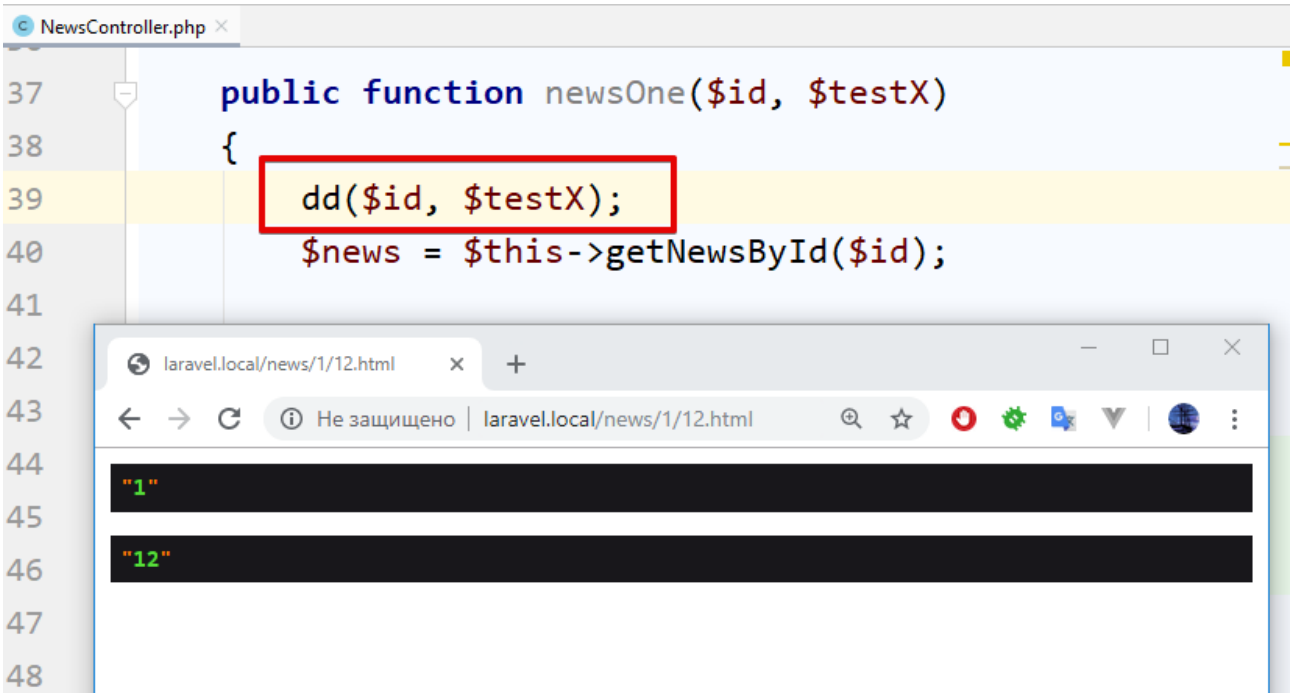
Немного о хелперах в Laravel

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()**.



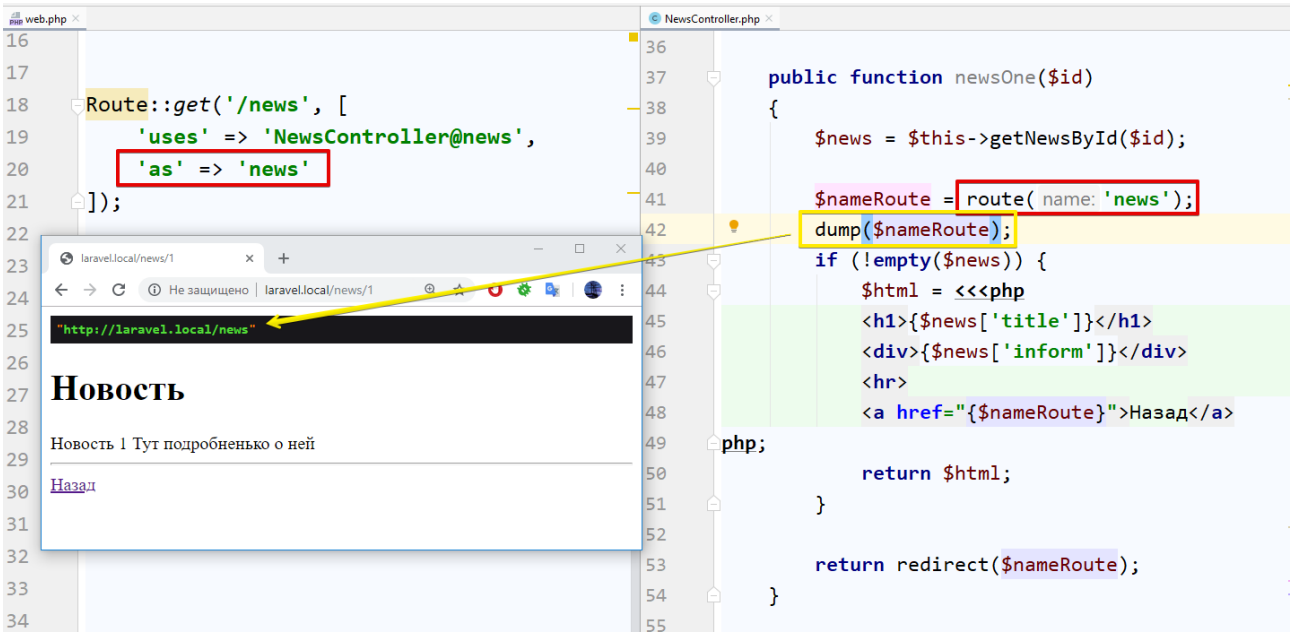
Ранее уже использовался хелпер **redirect**, перенаправляющий http-запрос на нужный адрес или роут.

В курсе мы еще столкнемся с хелперами.

Использование имен для роутов

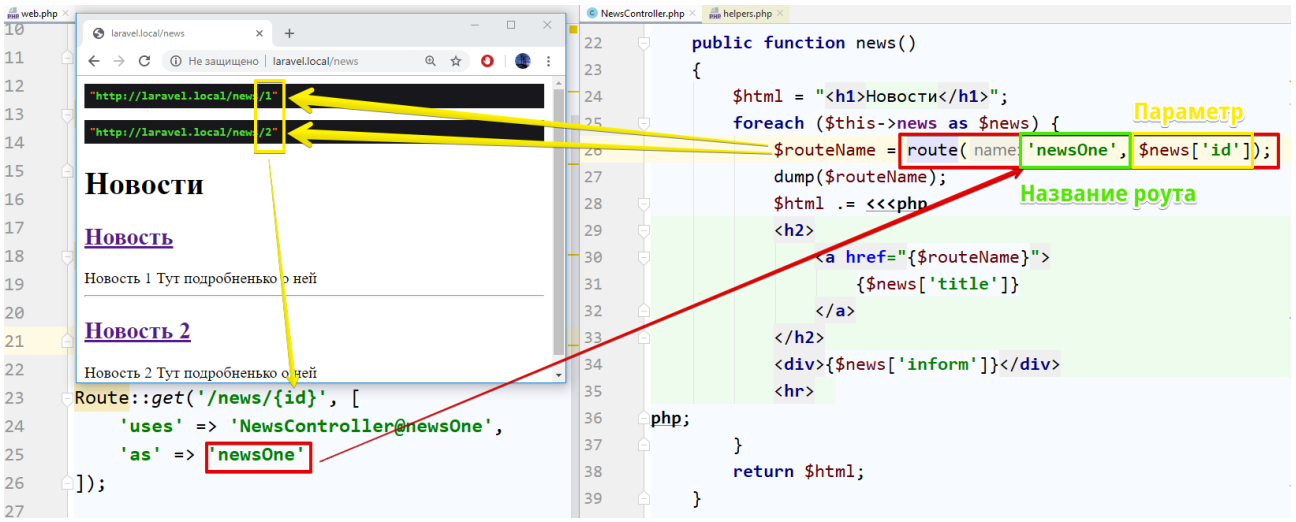
Представим ситуацию: программисты создали большое приложение с множеством страниц и задач. Смотрит на него заказчик и видит, что в адресе, по которому происходит вывод информации из личного кабинета, указано **user** — а ему хочется, чтобы там было именно **account**. И тогда происходит большой рефакторинг 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.

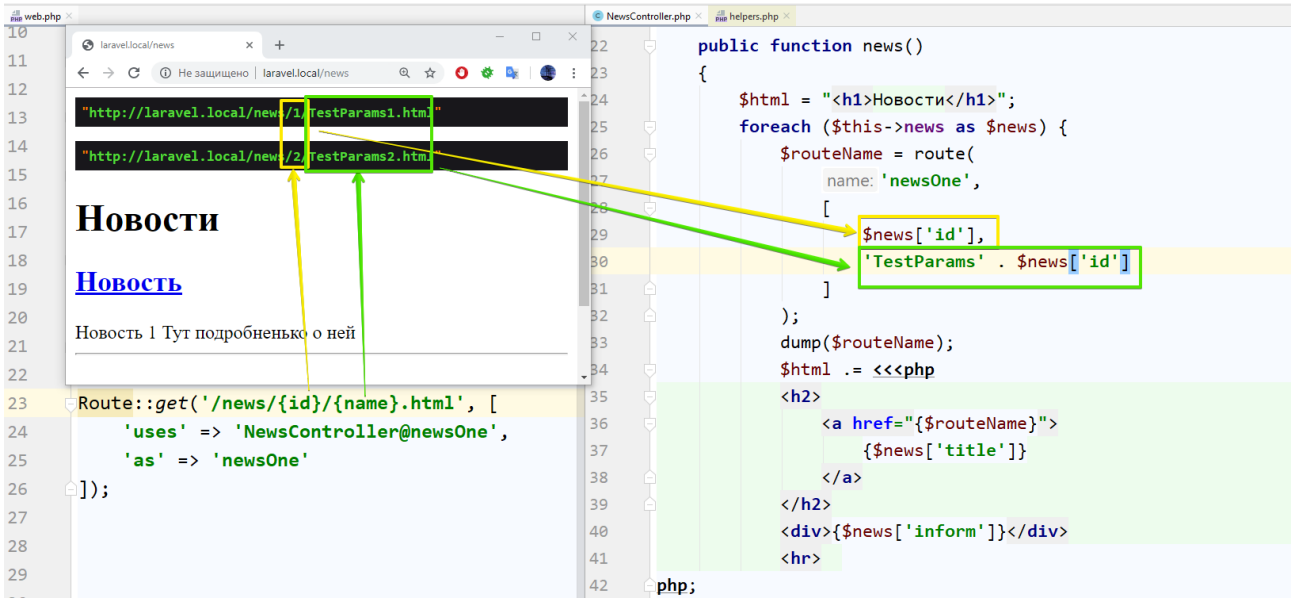


Рассмотрим более подробно пример, показанный выше. В нем для ранее созданного роута было добавлено имя — **`news`**. В **NewsController**, в методе **newsOne**, используется хелпер **route**. Он генерирует URL для указанного роута, возвращает строку. Далее в примере и будет использоваться результат выполнения 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.



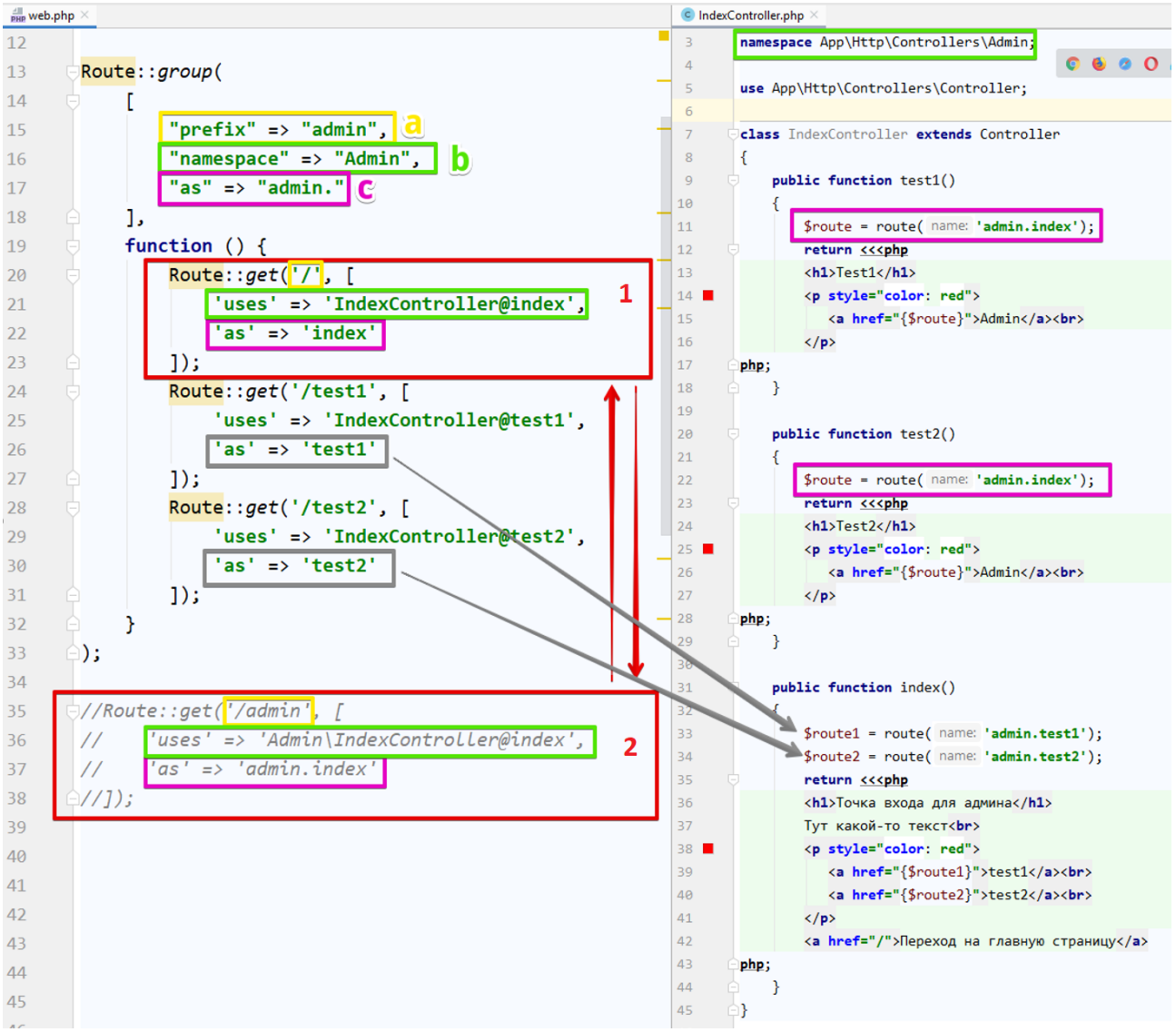
Группировки роутов

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.

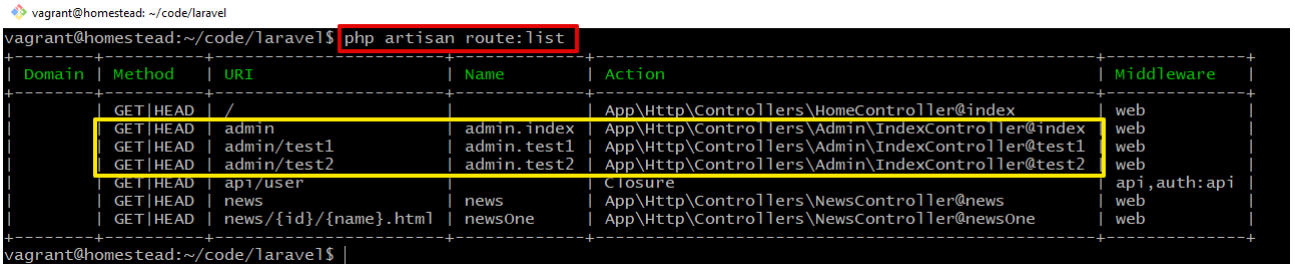
На рисунке показано, как можно использовать группировку роутов. В данном примере указано:

- 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.



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



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 следующее 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/