

Laravel

Урок 3.

Шаблонизатор

Blade

Внутренняя функциональность: шаблонизация, которая поможет красиво и удобно выводить агрегированную информацию.

Оглавление

[Теория](#)

[Представление \(view\) в MVC](#)

[Шаблонизатор Blade](#)

[Практика](#)

[Создание blade-шаблонов](#)

[Передача параметров в шаблон](#)

[Управляющие конструкции blade](#)

[Наследование шаблонов](#)

[Практическое задание](#)

[Дополнительные материалы](#)

[Используемая литература](#)

Теория

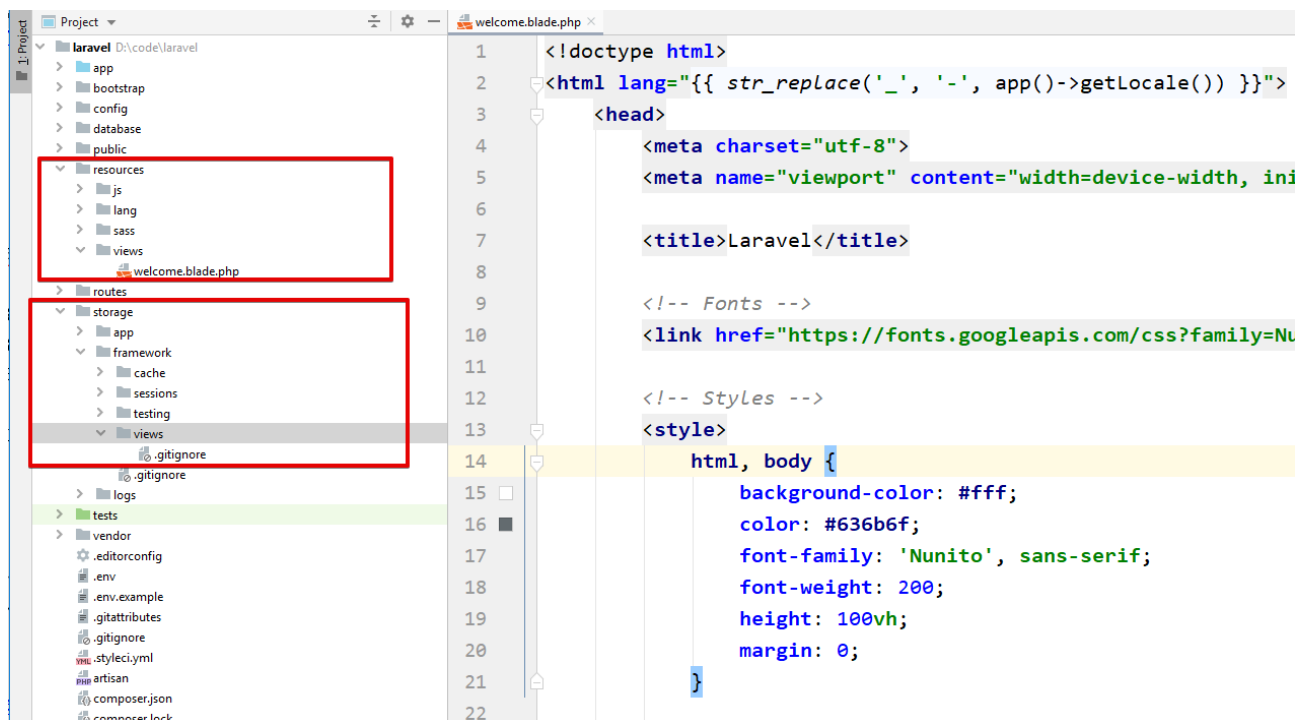
Представление (view) в MVC

Laravel реализует паттерн MVC. На этом уроке рассмотрим представления данных. Согласно правилам MVC, они должны содержать минимальное количество логики, а их работа должна быть понятной для программистов, не разбирающихся в бэкенде приложения.

В Laravel для этого используется шаблонизатор Blade.

Шаблонизатор Blade

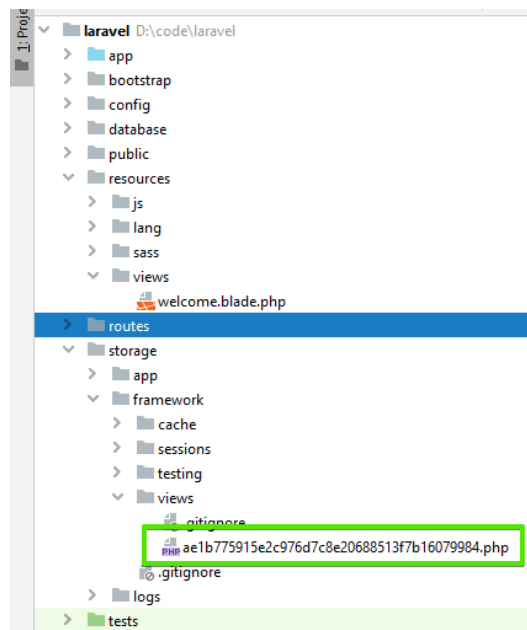
Blade предустановлен в Laravel. Он прост в использовании и постоянно развивается. Blade преобразует шаблон в код «нативного PHP» и далее использует именно его. При изменении шаблона создается новый файл для использования. Так что Blade не требует от приложения дополнительных ресурсов. В директориях папки **/resources/views** хранятся файлы самих шаблонов, а в папке **/storage/framework/views** и ее директориях — преобразованные из них файлы для выполнения.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the Laravel project structure, with the `resources/views` directory highlighted in red. The code editor shows the content of `welcome.blade.php`, which is a Blade template. The code is as follows:

```
1 <!doctype html>
2 <html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, ini
6
7     <title>Laravel</title>
8
9     <!-- Fonts -->
10    <link href="https://fonts.googleapis.com/css?family=Nu
11
12    <!-- Styles -->
13    <style>
14      html, body {
15        background-color: #fff;
16        color: #636b6f;
17        font-family: 'Nunito', sans-serif;
18        font-weight: 200;
19        height: 100vh;
20        margin: 0;
21      }
22
```

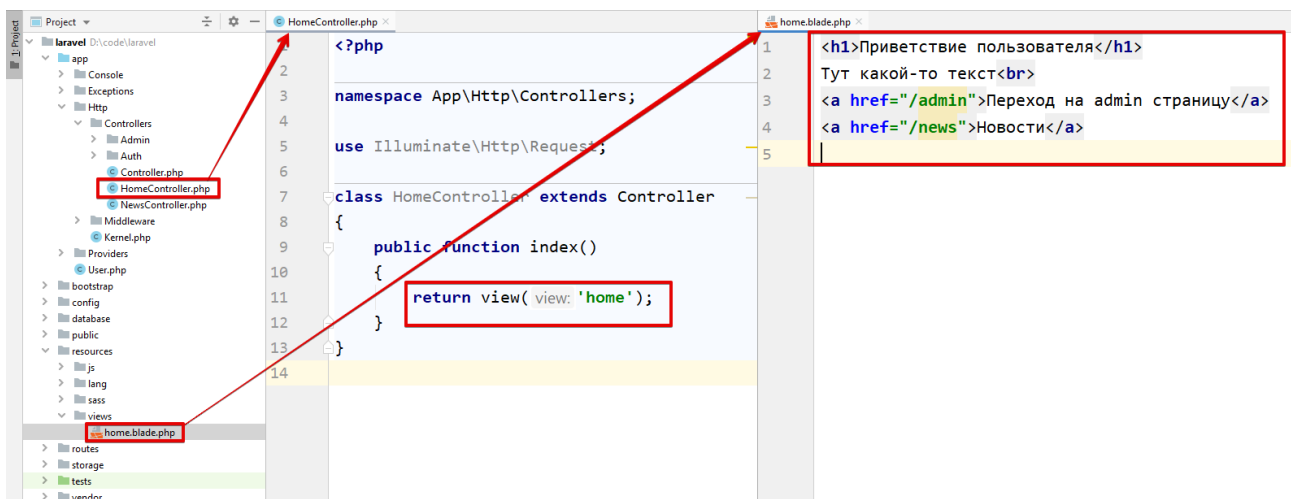
После установки Laravel можно увидеть только один blade-шаблон и отсутствие каких-либо подготовленных файлов. Но после обращения к корневому адресу появится новый подготовленный файл представления.



Практика

Создание blade-шаблонов

Чтобы создать новый blade-шаблон, потребуется файл в директории `/resources/views` или в ее подпапке с окончанием `.blade.php`. Создадим `home.blade.php` и перенесем в него все, что возвращал метод `index` в `HomeController`. А в методе `index` добавим код, возвращающий данный шаблон при помощи соответствующего хелпера `view()`. Пример реализации:



Далее создадим новый шаблон для главной страницы админ-панели. Для этого в директории шаблонов добавим папку `admin`, а в ней — `index.blade.php`. Перенесем из него выводимую информацию, немного изменив ее:

```
class IndexController extends Controller
{
    public function index()
    {
        return view( view: 'admin.index' );
    }

    public function test1()
    {
        $route = route( name: 'admin.index' );
        return <<<php
        <h1>Тест1</h1>
        <p style="...">
        <a href="{ $route }">Admin</a><br>
        </p>
        php;
    }
}
```

```
<h1>Точка входа для админа</h1>
Тут какой-то текст<br>
<p style="...">
  <a href="/admin/test1">test1</a><br>
  <a href="/admin/test2">test2</a><br>
</p>
<a href="/">Переход на главную страницу</a>
```

Чтобы получить шаблон, находящийся не в самой директории **views**, в ее подпапке **admin**, при вызове хелпера **view** указывается эта папка и название самого шаблона. Другими словами, точка в первом параметре хелпера указывает на переход к директории на уровень ниже, в подпапку.

Передача параметров в шаблон

Теперь возникает проблема — как в шаблон передать данные? Пути для ссылок, что генерировали при помощи хелпера **route()** ранее в шаблоне, заменены обычным текстом.

Решение простое: у **view()** есть возможность передать второй параметр — он способен принять массив с данными, который будет передан в шаблон.

```
class IndexController extends Controller
{
    public function index()
    {
        $params = [
            'text' => 'Тут какой-то текст',
            'ur11' => route( name: 'admin.test1' ),
            'ur12' => route( name: 'admin.test2' ),
        ];

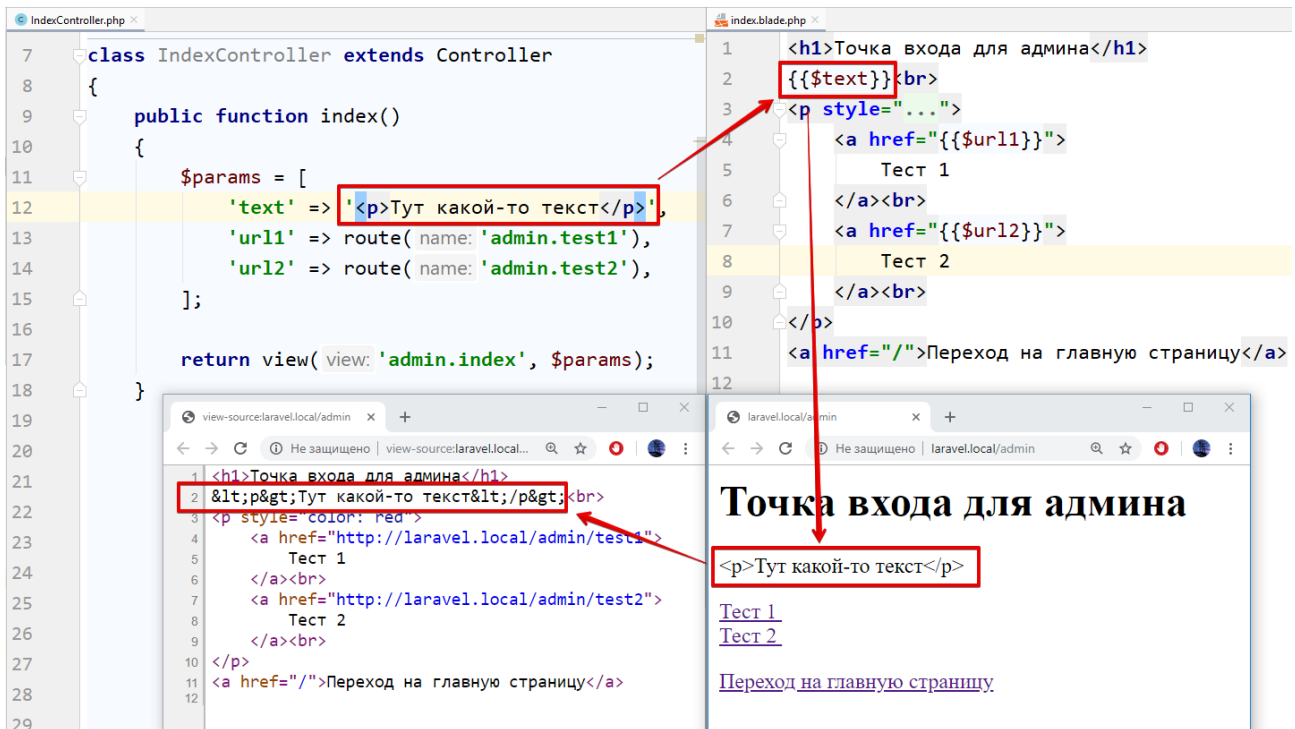
        return view( view: 'admin.index', $params );
    }
}
```

```
<h1>Точка входа для админа</h1>
<{{ $text }}><br>
<p style="...">
  <a href="{ $ur11 }">
    Тест 1
  </a><br>
  <a href="{ $ur12 }">
    Тест 2
  </a><br>
</p>
<a href="/">Переход на главную страницу</a>
```

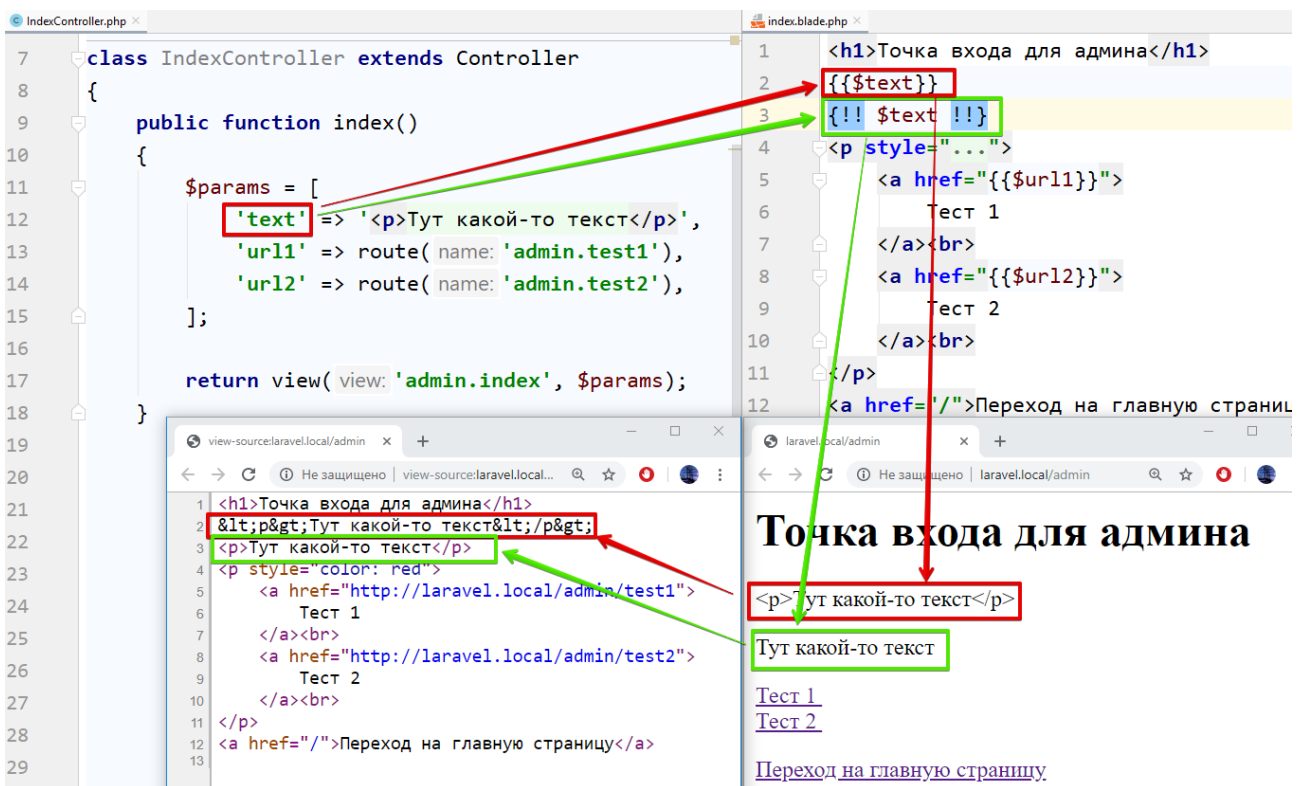
Точка входа для админа
Тут какой-то текст
[Тест 1](#)
[Тест 2](#)
[Переход на главную страницу](#)

В примере выше показано, каким образом можно передать параметры в выбранный шаблон. Обратим внимание: имя элемента массива, передаваемого в шаблон, превращается в переменную, а для ее вывода используются двойные фигурные скобки **{{}}**.

Теперь попробуем передать в шаблон html-код.



В итоге видим, что теги, переданные в переменной `$text`, были выведены в так, как указано в коде. Открыв код страницы, сможем понять, почему это произошло: дело в том, что открывающиеся и закрывающиеся угловые скобки были преобразованы в спецсимволы html. Если это не нужно, следует выводить переменную внутри `{!!}`:



Внутри шаблона доступно использование различных js-фреймворков, например Vue.js. Для вывода переменных могут тоже применять двойные скобки. Чтобы blade игнорировал их, к ним нужно в начале добавлять `@`.

```
{{ $title }} 
@{{ title }} 

```

Управляющие конструкции blade

Вернемся к новостям. На прошлом уроке новости мы получали из приватной переменной и при помощи цикла в самом контроле готовили строку — ее и возвращали пользователю в ответ на его запрос. Теперь для того, чтобы вывести данные, мы используем подготовленные шаблоны — а значит, и этот цикл должен быть перемещен в него.

В шаблонизаторе blade есть несколько конструкций, которые работают циклично:

```
@for ($i = 0; $i < 10; $i++)
    Текущее значение: {{ $i }}
@endfor

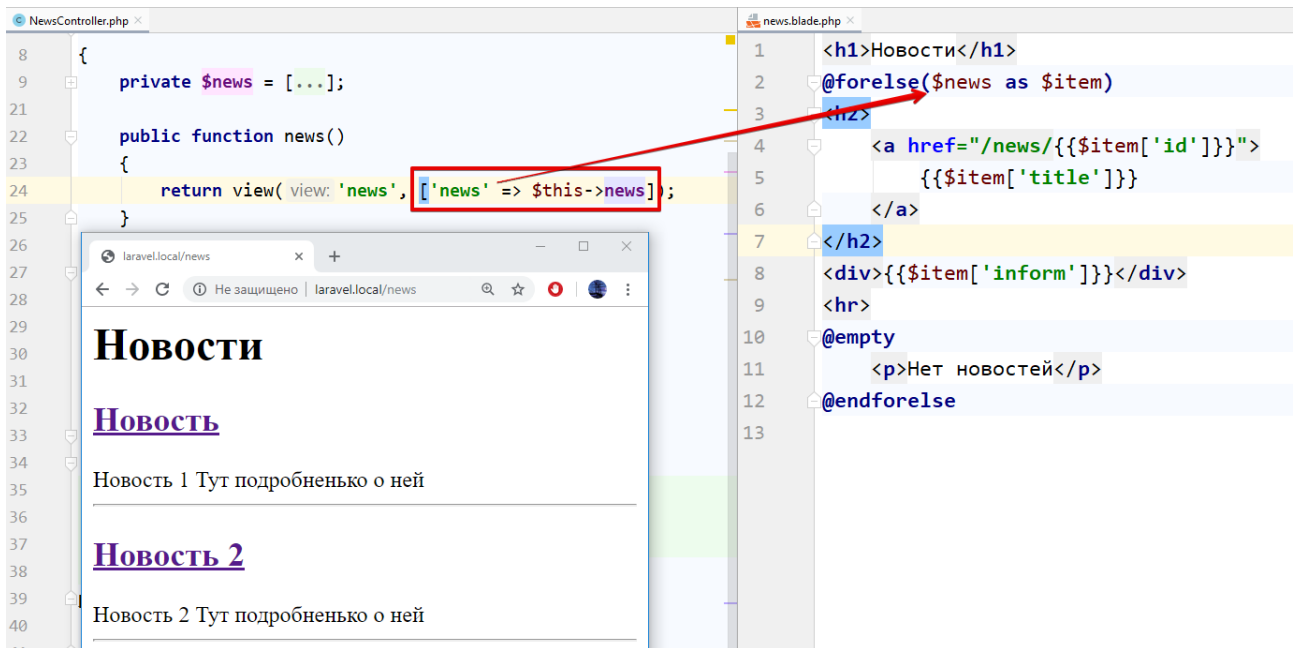
@foreach ($news as $item)
    <p>Это новость {{ $item->id }}</p>
@endforeach

@forelse ($news as $item)
    <li>{{ $item ->title }}</li>
@empty
    <p>Нет новостей</p>
@endforelse

@while (true)
    <p>Этот цикл будет длиться вечно.</p>
@endwhile
```

Как это работает? В шаблоне указываем название управляющей конструкции, а перед ней ставим **@**. Далее в круглых скобках указываются параметры данной конструкции. Это становится началом управляющей конструкции. Там, где конструкция заканчивается, опять пишется ее название, а перед ним **@end**. Все, что находится между началом и концом управляющей конструкции, будет выполняться в цикле (если он используется). При **forelse** в цикле выполняется все, что указано после начала и до **@empty**. Все, что находится от **@empty** до **@endforelse**, выполнится только один раз и только в том случае, если массив **\$news** будет пуст.

Создадим новый шаблон с именем **news.blade.php**. Перенесем все, что у нас было в **NewsController**, из метода **news()** в новый шаблон, внося правки. А в методе **news()** будем возвращать этот шаблон, передавая в него массив с новостями:



Помимо циклов в шаблоне можно использовать и конструкции типа **if**, **isset**, **empty**:

```
@if (count($records) === 1)
    Есть одна запись!
@endif
@elseif (count($records) > 1)
    Записей больше одной!
@endif
@else
    Нет записей!
@endif

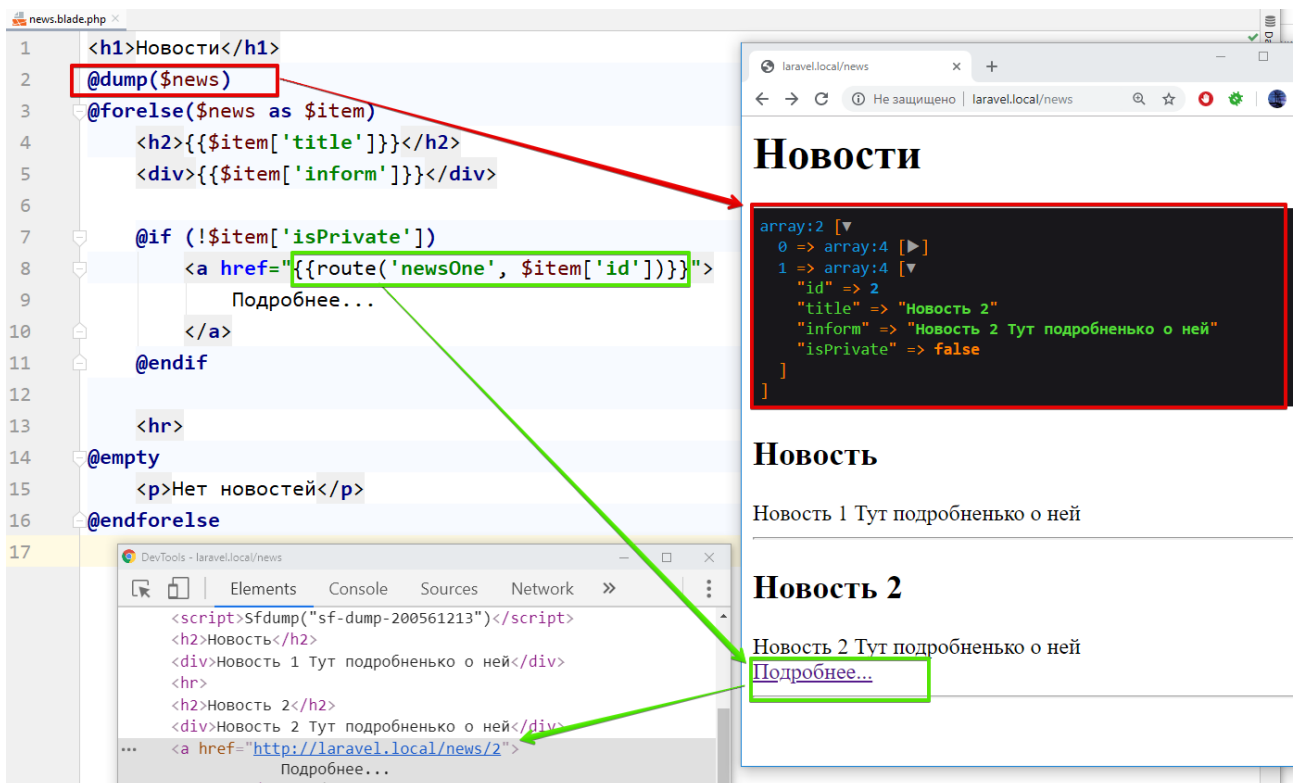
@isset($records)
    // $records переменная определена и не null...
@endisset

@empty($records)
    // $records "пуста"...
@endempty
```

Добавим в массив новостей дополнительное значение, которое будет указывать на возможность перехода на просмотр полной информации в данной новости.



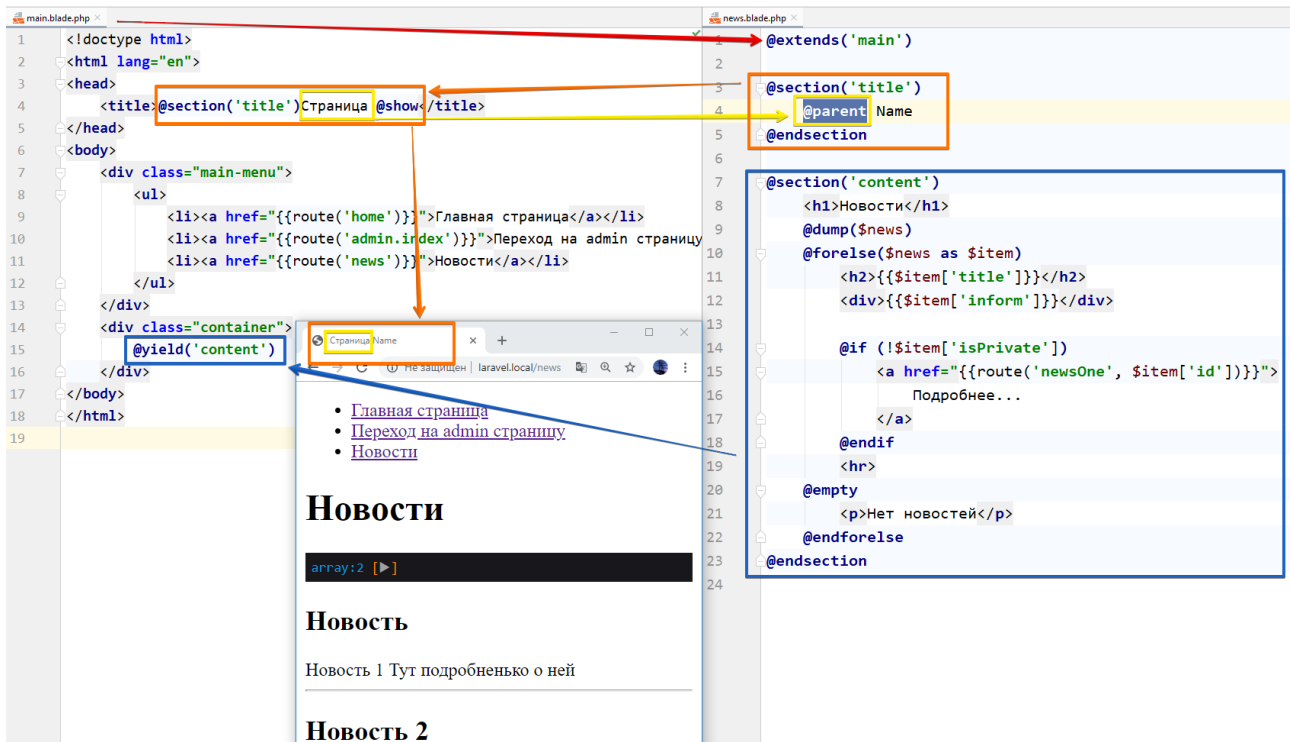
Надо правильно собрать адреса для ссылок. Прежний вариант не подходит, поскольку придется перебирать весь массив данных дополнительно. Но, оказывается, хелперы работают в шаблоне blade. Выведем структуру полученного массива новостей и добавим адреса:



Наследование шаблонов

Шаблонов сделано много — и возникает проблема дублирования частей сайта: меню, шапки, подвала.

Решается это с помощью наследования шаблонов. Принцип: создаем главный шаблон для страницы (или группы страниц) и помечаем секции, которые будут меняться в шаблонах-наследниках.



Рассмотрим подробнее алгоритм наследования шаблонов. В контроллере при помощи хелпера **view** вызывается шаблон **news**. Для него передается набор параметров. Но теперь он наследуется от нового шаблона **main** — благодаря инструкции **@extends**. В скобках инструкции указывается имя шаблона, который следует наследовать.

В шаблоне **main** используется ряд инструкций, которые позволяют определить блоки для вставки в них дополнительного контента из дочерних шаблонов:

- **@section** — определяет начало секции для вставки данных. Первым параметром принимает строку, которая будет именем секции;
- **@show** — определяет место закрытия ранее открытой секции. Контент, содержащийся между **@section** и **@show**, выводится на экран, если этот блок не переопределен в дочернем шаблоне;
- **@yield** определяет место для добавления секции из дочернего шаблона. Первым параметром принимает строку, служащую именем секции.

В шаблоне **new** используются инструкции:

- **@extends** — определяет, от какого шаблона производить наследование. В скобках указывается путь до шаблона от директории `~/resources/views`. Части имен директорий разделяются точками. К примеру, если необходимый шаблон **main** находится в папке `~/resources/views/admin`, то инструкция будет принимать строку `'admin.main'`;
- **@section** — определяет начало секции: контент, который будет добавлен в секцию наследованного шаблона с аналогичным именем;
- **@endsection** — определяет окончание секции;

- `@parent` — добавляет в секцию потомка контент из родительской секции.

Практическое задание

1. Создать blade-шаблоны страниц приложения, сделанных в предыдущей практической работе (можно использовать Twitter Bootstrap). Шаблоны и их верстка обязательно должны использовать возможности шаблонизатора, объясненные в этом уроке. Нужно добавить четыре блока: блок шапки сайта, подвала, место вывода контента и область меню. При этом шаблоны могут быть несложными.

Дополнительные материалы

1. <https://laravel.com/docs/5.8/blade>
2. <http://laravel.su/docs/5.4/views>
3. <https://laravel-guide.readthedocs.io/en/latest/blade/>

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

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