**View – 'представление'**

**Шаблон(view)** - образец для генерирования веб-страницы. Хранятся в resources\views.

**context** - ассоциативный массив содержащий данные для формирования страницы.

**Blade** - шаблонизатор встроенный в Laravel, поэтому шаблоны имеют расширение .blade.php

Шаблонизатор вызывается методом view() и для рендеренга страницы принимает шаблон(view, без расширение .blade.php) и context:

view('home', $context);

Путь к шаблону указывается относительно домашней папки для шаблонов resources\views. Путь к вложенным(находящимся в папке) шаблонам может быть указан как через слеш main\home, так и через точечную нотацию main.home

**директивы** - встроенные в шаблонизатор функции(зачастую, дублирующие функции PHP).

При первом рендеринге шаблона он компилируется в РНР-код и сохраняется в storage\framework\views. При внесении изменений в шаблон, компиляция происходит повторно.

**Настройки шаблонизатора** хранятся в модуле config\view.php.

paths - массив путей, по которым шаблонизатор будет искать файлы шаблонов.

compiled- путь к папке, в которой будут храниться откомпилированные шаблоны.

**- Директивы вывода данных(11.2.1):**

**{{ значение }}** - выводит заданное значение, которым может быть явно заданная величина, значение переменной, свойства, константы, результат выполнения функции, метода и произвольное выражение РНР:

<h4>{{ $bb->title }}</h4>

<а href="{{ route('about') }}">О сайте</а>

Перед выводом все присутствующие в значении недопустимые символы преобразуются в соответствующие НТМL-литералы:

{{ '<h2>Дома</h2>' }} // Результат: &lt;h2&gt;Дoмa&lt;/h2&gt;

**{!! значение !!}**- выводит заданное значение без преобразования недопустимых символов в литералы:

{!! '<h2>Дома</h2>' !!} // Результат: <h2>Дома</h2>

**@json( значение, настройка, глубина)** - выводит значение, преобразованное в формат JSON. Для преобразования использует функцию РНР json\_encode() , которой передаются заданные в директиве параметры настройки и глубина:

<script>

let rubric = @json($rubric);

let bbs = @json($bbs, JSON PRETTY PRINT);

</script>

**- Условные директивы и директивы выбора(11.2.2.1):**

**@if** ... @elseif ... @else ... @endif - аналог условного выражения РНР:

@if(count($bbs) == 0)

<р>Объявлений нет</р>

@elseif(count($bbs) == 1)

<р>Всего одно объявление</р>

@else

<р>Много объявлений</р>

@endif

**@unless** ... @endunless - выводит содержимое unless, если условие ложно, и содержимое else - в противном случае:

@unless(!$bb->publish)

<р>checked</р>

@else

<р>have no</р>

@endunless

**@isset** ... @else ... @endisset - выводит содержимое isset, если все переменные хранят значения, отличные от null, и содержимое else - в противном случае:

@issеt(<переменная 1>, <переменная 2> . . . <переменная п>)

<содержимое isset>

@else

<содержимое else>

@endisset

**@empty** ... @else ... @endempty - выводит содержимое еmрtу, если переменная хранит «пустое» значение, и содержимое else - в противном случае. «пустым>) считаются «пустая»

строка, строка 'о', числа о, о. о, величины false, null и «пустой» массив.

@ernpty($bbs)

<р>Объявлений нет</р>

@else

<р>Объявления есть</р>

@endernpty

**@env** ... @else ... @endenv - выводит содержимое env, если сайт работает в заданном режиме или в одном из режимов, указанных в массиве, и содержимое else - в противном случае:

@еnv(<режим работы сайта>l<массив режимов>)

<содержимое env>

@else

<содержимое else>

@endenv

@env( 'local')

<р>Сайт работает в режиме разработки.</р>

@endenv

@env( ['local', 'staging' ] )

<р>Сайт работает в режиме разработки или отладки.</р>

@else

<р>Сайт работает в режиме эксплуатации.</р>

@endenv

**@production** ... @else ... @endproduction- выводит содержимое production, если сайт работает в эксплуатационном режиме, и содержимое else- в противном случае:

@production

<содержимое production>

@else

<содержимое else>

@endproduction

**@switch** ... @case ... @default ... @endswitch - аналоги выражения выбораРНР:

@switсh(<значение>)

@саsе(<величина 1>)

<содержимое 1 - выводится, если значение == величине 1>

@break

@саsе(<величина 2>)

<содержимое 2 - выводится, если значение == величине 2>

@break

@саsе(<величина n>)

<содержимое п - выводится, если значение == величине n>

@break

@default

<содержимое default - выводится, если значение != ни одной величине>

@endswitch

@switch(Auth::user()->role)

@case('admin')

<р>Администратор</р>

@case('editor')

<р>Редактор</р>

@case('author')

<р>Автор объявлений</р>

@default

<р>Неизвестная роль</р>

@endswitch

**- Директивы циклов(11.2.2.2):**

**@foreach** ... @endforeach- аналог цикла по массивуРНР:

@foreach($rubrics as $rubric)

<h2>{{ $rubric->name }}</h2>

@endforeach

**@forelse** ... @empty ... @endforelse - аналог цикла по массиву РНР, выводящий содержимое empty, если массив «пуст»:

@forelse($rubrics as $rubric)

<h2>{{ $rubric->name }}</h2>

@empty

<р>Рубрик нет</р>

@endforelse

**@for** ... @endfor - аналог цикла со счетчиком РНР:

@for($i = 1; $i < 11; $i++)

{{ $i }} &nbsp;

@endfor

**@while** ... @endwhile - аналог цикла с предусловием РНР:

@while(<ycлoвиe>)

<тело цикла>

@endwhile

**@break** ( условие ) - аналог оператора прерывания цикла РНР break. Если задано условие, цикл прервется, только если оно истинно:

@foreach($rubrics as $rubric)

@break($rubric->id > 7)

<h2>{{ $rubric->name }}</h2>

@endforeach

**@continue** (<условие>) - аналог оператора прерывания текущей итерации цикла РНР

continue. Если задано условие, текущая итерация цикла прервется, только если оно истинно.

**loop** - переменная доступная в теле любого цикла, хранящая объект с полезными сведениями о текущем цикле. Вот свойства этого объекта:

**index** - номер текущей итерации цикла, начиная с 0;

**iteration** - тo же самое, что и index, но начиная с 1;

**remaining** - количество оставшихся итераций (если это не цикл с предуслови-

ем);

**count** - количество элементов в перебираемом в цикле массиве;

**first** - true, если это первая итерация цикла, false-в противном случае;

**last** - true, если это последняя итерация цикла, false-в противном случае;

**even** - true, если это четная итерация цикла, false-в противном случае;

**odd** - true, если это нечетная итерация цикла, false-в противном случае;

**depth** - уровень вложенности цикла ( 1 - для внешнего, 2 - для вложенного

в него и т. д.);

**parent** - хранит объект со сведениями о цикле предыдущего уровня вложенности.

Пример:

@foreach($rubrics as $superrubric)

@if ($loop->first) <hl>Рубрики</hl> @endif

<h2>{{ $loop->iteration }}. {{ $superrubric->name }}</h2>

@foreach($superrubric->rubrics()->get() as $subrubric)

<hЗ>{{ $loop->parent->iteration }}.{{ $loop->iteration }}.

{{ $subrubric->name }}</hЗ>

@endforeach

@endforeach

**- Прочие директивы(11.2.3):**

**@php** ... @endphp - помещает в шаблон фрагмент РНР-кода:

@php

$rubric\_count = count($rubrics);

@endphp

**{{--** Комментированный текст **--}}**

**- Запрет на обработку директив(11.2.4):**

Многие JаvаSсriрt-фреймворки, как и Laravel, используют для записи своих директив двойные фигурные скобки. Чтобы указать шаблонизатору Laravel не обрабатывать такие директивы, а оставить их как есть, достаточно поставить перед ними символы@:

{{ $rubric->name }} - Обрабатывается La ravel

@ {{ rubric. name }} - Не обрабатывается Laravel

Чтобы Laravel не обрабатывал управляющие директивы, принадлежащие другим фреймворкам, перед ними также нужно поставить символы@:

@@if (bb.publish)

...

@@endif

**@verbatim** ... @endverbatim указывает Laravel не обрабатывать все ее содержимое:

@verbatim

<h2>{{ bb.title }}</h2>

<р>{{ bb.content }}</р>

<р>{{ bb.price }}</р>

@endverbatim

**- Веб-форма(11.3.1):**

**@csrf** - электронный жетон безопасности выводимой веб-формы:

<form ... >

@csrf

...

</form >

Эта директива создает скрытое поле, хранящее электронный жетон безопасности. Получив данные, введенные в веб-форму, фреймворк сравнивает жетон, извлеченный из скрытого поля, с ранее сохраненным в серверной сессии. Если оба жетона идентичны, данные из веб-формы считаются -заслуживающими доверия и принимаются к обработке. В противном случае Laravel полагает, что имел место случай межсайтовой атаки, и выводит сообщение с кодом статуса 419 (страница устарела).

Формирование и проверку электронного жетона безопасности осуществляет посредник App\Http\Middleware\VerifyCsrfToken. Изначально он входит в группу web и, таким образом,

связывается со всеми веб-маршрутами;

**- @method, дополнительные НТТР-метод**. В Laravel существует больше методов клиентского запроса (get, post, put, patch, delete, options, match, any, view, redirect, permanentRedirect) чем в протоколе HTTP (GET и POST). Поэтому, если запрос использует метод не предусмотренный протоколом, необходимо форму передать HTTP методом POST, а в теле формы директивой @method передать необходимый метод Laravel запроса:

<form action=" ... " method="POST">

@csrf

@method( 'DELETE')

<input type="submit" vаluе="Удалить">

</form>

Вместо директивы @method можно использовать функцию:

{{ **method\_field**('DELETE') }}

**- Вывод элементов управления(11.3.2):**

Смотри параграф и конспект контроллер Извлечение ранее введенных данных на странице(10.2.5.).

Элементы бывают: поле ввода, флажок, переключатель, список с возможностью выбора только одного пункта, список с возможностью выбора нескольких пунктов. Введенное ранее значение (не прошедшее валидацию и возвращенное контроллером) "заносится в поле" при помощи метода **old(content, default)**.

В значение по умолчанию (default) могут подставляться значения из DB при редактировании данных.

Так-же при создании новой записи в значение по умолчанию (default) автор предлагает

вносить данные "для примера заполнения" аж из модели свойства attributes (смотри 10.4)

**- Вывод сообщений об ошибках ввода(11.3.3):**

В любом шаблоне присутствует переменная errors, хранящая список допущенных при вводе ошибок. Этот список можно извлечь и вывести на странице:

<input narne= "title" ... >

@if($errors->has('title'))

<ul>

@foreach ($errors->get('title') as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

@endif

Также можно использовать директиву шаблонизатора @error ... @enderror

@error('inputName')

{{ $message }}

@enderror

Если имя хранилища изменено

@еrrоr(<наименование элемента управления>[, <имя хранилища ошибок>]

<содержимое>

@enderror

**- Наследование шаблонов(11.4):**

Шаблоны находятся в папке resources\views, базовые шаблоны в папке resources\views\layouts. Основной базовый шаблон, от которого наследуются остальные шаблоны, должен иметь имя app.blade.php.

Базовый шаблон содержит элементы, присутствующие на всех страницах сайта: шапку, поддон, главную панель навигации, элементы разметки и пр. Производный шаблон должен

иметь уникальное содержимое.

Реализовать наследование шаблонов можно двумя способами:

**первый**(вставка содержимого производного шаблона в базовый)**:**

В коде **базового шаблона** отмечаются места, куда будет вставлено содержимое секций производных шаблонов директивой шаблонизатора **@yield**:

@уiеld(<имя секции>[, <не обязательное содержимое, выводимое, если секция не была создана в производном шаблоне>])

в коде **производного шаблона**:

В первой строке кода шаблона директивой **@extends** указывается базовый шаблон:

@extends (<имя базового шаблона>)

Директивой **@section**, имеющей два формата записи, создаются нужные секции:

@section(<имя создаваемой секции>, <содержимое секции>)

@section(<имя открываемой секции>)

<содержимое секции>

@endsection (<имя закрываемой секции>) или @endsection

**виорой**(замена или дополнение содержимым производного шаблона в базовом)**:**

В коде **базового шаблона** создаются секция директивой **@section** после чего она выводятся на экран директивой **@yield**:

@section('main')

<hl>Список объявлений</hl>

@endsection

@yield('main')

Вместо комбинации директив @endsection и @yield можно использовать директиву **@show**:

@section('main')

<hl>Список объявлений</hl>

@show

в коде **производного шаблона**:

В первой строке кода шаблона директивой **@extends** указывается базовый шаблон:

@extends (<имя базового шаблона>)

Для замены содержимого базового шаблона необходимо воспользоваться директивой **@section**:

@section('main')

...

@endsection

Для добавления к содержимому базового шаблона необходимо воспользоваться директивой **@parent** вставив её в директиву @section:

@section('main')

@parent

...

@endsection

В параграфе рассмотрена директива @hassection реализующая логику if else при выводе содержимого производного шаблона.

А так-же очень полезный пример, позволяющая менять title базового шаблона из

производного шаблона средствами директив @уiеld @section.

**- Стеки(11.5):**

Стек аналогичен секции, за тем исключением, что содержимое, помещаемое в стек, не заменяет уже присутствующее в нем, а дополняет его. Стеки удобно применять для задания внешних таблиц стилей и веб-сценариев, привязываемых к странице, в производных шаблонах.

Директива **@stack**(<имя стека>) указания места шаблона, куда будет помещено содержимое стека.

Директива **@push** добавляет содержимое в конец стека.

Директива **@prepend** добавляет содержимое в начало стека.

Пример:

{{-- Базовый шаблон --}}

<body>

@push('inBody')

<div>from parent blade (must be in the center)</div>

@endpush

@stack('inBody')

</body>

{{-- Производный шаблон --}}

@extends('stack.parent')

@push('inBody')

<div>from child blade, push (should be down)</div>

@endpush

@prepend('inBody')

<div>from child blade, prepend (must be at the top)</div>

@endprepend

**имхо** в Laravel 9 сначала обрабатываются @push и @prepend из производного шаблона, а затем из базового, поэтому порядок получился другим чем в книге, @push из базового

шаблона получиля в конце (возможно из-за того, что пробовал в body а не в title).

**- ИМХО. Включаемые шаблоны и Компоненты во view Laravel - аналог трейтов в php и оба реализуют принцип DRY. Включаемые шаблоны используются при одинаковой структуре "таблицы" но разном её наполнении во включающих шаблонах, а компонент при полностью идентичных "таблицах" на включающих шаблонах. Поэтому имеют различая:**

**Включаемый шаблон берет данные из включающего шаблона, поэтому контроллер включающего шаблона должен обеспечить данными включаемый шаблон;**

**Компонент сам "обеспечивает" себя данными при помощи создаваемого класса функционально похожего на контроллер, со своей логикой и возможностью работать**

**с моделями.**

**- Включаемые шаблоны(11.6):**

Включаемый шаблон хранит в себе фрагмент кода который используется в нескольких включающих шаблонах. Включаемый шаблон имеет доступ ко всем данным включающего шаблона. Включаемый шаблон должен находится в папке views\includes или views\shared.

Пример всего кода включаемого шаблона формирующего таблицу table.blade.php:

@if(isset($context))

<table border=1>

<tr><th>array values</th></tr>

@foreach($context as $value)

<tr><td>{{ $value }}</td></tr>

@endforeach

</table>

@else

have no context for table

@endif

Для вставки включаемого шаблона, во включающем шаблоне используются директивы:

**@include** - вставляет включаемый шаблон с заданным путем

@include('includes.table')

**@each** - работает аналогично циклу foreach, принимает "массив массивов", для каждого массива выводит на странице включаемый шаблон и передает в него переименованный массив.

@each (<путь к включаемому шаблону>,

<массив массивов или коллекция>,

<имя для переименования массива передаваемого включаемому шаблону>,

<путь к включаемому запасному шаблону, применяющемуся, если массив пуст>).

@each('includes.table', $array, 'context', 'includes.empty')

Подробный разбор директивы смотри в практике.

**@includeWhen**- вставляет включаемый шаблон с заданным путем, если результат

вычисления условия равен true, в противном случае ничего не делает:

@includeWhen(count($bbs) > 0, 'shared.bbs')

**@includeUnless** - вставляет включаемый шаблон с заданным путем, если результат

вычисления условия равен false, в противном случае ничего не делает

**@includelf**- вставляет включаемый шаблон с заданным путем, только если тот существует.

**@includeFirst**- вставляет первый существующий включаемый шаблон из содержащихся в массиве:

@includeFirst(<мaccив с путями к включаемым шаблонам>,

<ассоциативный массив с дополнительными данными>)

**- Псевдонимы включаемых шаблонов(11.6.1):**

Часто используемым включаемым шаблонам можно дать короткие псевдонимы. Это выполняется в теле метода boot()провайдера App\Providers\

AppServiceProvider вызовом у фасада Illurninate\Support\Facades\Blade метода

include():

Blade::include(<путь к включаемому шаблону> , <псевдоним>=null)

Если псевдоним не указан, в качестве такового будет использован последний элемент заданного пути (например shared.errors получит псевдоним errors).

Для вставки включаемого шаблона по заданному псевдониму применяется директива формата: @<псевдоним> [ (<ассоциативный массив с дополнительными данными>)].

Пример:

use Illurninate\Support\Facades\Blade;

class AppServiceProvider extends ServiceProvider{

public function boot() {

Blade::include('shared.method', 'httpMethod');

}

}

...

@httpMethod('РАТСН' )

**- Компоненты(11.7):**

Компонент это комбинация включающегося шаблона, хранящего в себе фрагмент кода использующегося в нескольких включающих шаблонах, и связанной с включающимся шаблоном программной логики, формирующей его контекст.

Создание полнофункционального компонента выполняется командой:

php artisan make:component <имя компонента>

Компонент может быть упрощенным не иметь шаблона либо контроллера. Принимать данные с включающего шаблона, принимать HTML код, принимать данные при помощи слота. Компонент может динамическим изменятся в зависимости от ситуации.

ИМХО. Компонент узкоспециализированный и многофункциональный инструмент и должен

применяться в зависимости от задачи, поэтому смотри учебник)))

**- Передача данных в несколько шаблонов(11.8):**

При необходимости иметь доступ к переменной в нескольких шаблонах приложения, в книге, предлагается три варианта. Суть всех способов сводится к регистрации в App\Providers\

AppServiceProvider в методе boot кастомных решений возвращающих необходимую переменную. Подробную реализацию методов смотри в книге, дальше будет краткое

описание и различия этих методов:

**- Разделяемые значения(11.8.1):**

Применяется при необходимости иметь доступ на странице к, неизменяемым в процессе работы приложения, переменным, константам. Пример:

use **Illuminate\Support\Facades\View;**

class AppServiceProvider extends ServiceProvider {

...

public function boot() {

**View::share**('copyright', '© команда разработчиков');

}

}

**- Составители значений(11.8.2):**

Может передать значения не всем, а заданным шаблонам. Оформляется в виде класса, рекомендуется объявить в пространстве имен App\Http\View\Composers, создав необходимые папки самостоятельно. Пример:

namespace App\Http\View\Composers;

use App\Models\Rubric;

class RubricsComposer {

public function \_construct() { }

public function compose($view) {

$view->with('rubrics', Rubric::orderBy('name')->get());

}

}

Регистрация выполняется в провайдере App\Providers\AppServiceProvider в методе boot() вызовом у фасада View метода composer(<обозначение шаблона>, <путь к классу составителя значений>). В качестве обозначения шаблона можно указать:путь к шаблону, массив с путями к шаблонам, \* если задаваемые значения должны добавляться в контексты всех шаблонов.

**Внимание!** если в контексте шаблона уже присутствует переменная с таким же именем, она

будет переопределена.

**- Создатели значений(11.8.3):**

Создатели значений аналогичны составителям с отличиями:

- для занесения значений в контекст шаблона применяется метод create(), имеющий тот же формат вызова, что и compose() :

class RubricsCreator {

...

public function create($view){

$view->with('rubrics', Rubric::orderBy('name')->get());

}

}

- регистрация создателя значений выполняется методом creator() фасада View, имеющего тот же формат вызова, что и метод cornposer() :

use App\Http\View\Creators\RubricsCreator;

class AppServiceProvider extends ServiceProvider{

...

public function boot() {

View::creator(['index', 'rubric'], RubricsCreator::class);

}

}

**Внимание!** Поскольку создатель значений выполняется непосредственно при инициализации шаблонов, есть возможность заменить предоставляемое им значение каким-либо другим, вызвав у объекта шаблона метод with() (см.разд. 9.5.1.1). Пример:

return view('index')->with(['rubrics' => $bbs]);

**- Обработка статических файлов(11.9):**

Статические файлы пересылаются клиенту без обработки. К ним относятся внешние таблицы стилей, веб-сценарии,изображения, документы, архивы и пр.

Обработка статических файлов различается в зависимости от их местоположения:

- файлы хранящиеся непосредственно в папке сайта public:

<link href="/styles/main.css" rel="stylesheet" type="text/css">

- файлы хранятся во вложенной в папку public:

в локальной настройке ASSET\_URL или рабочей настройке asset\_url указать

путь к этой папке относительно папки public обязательно с начальным слешем:

ASSET URL=/assets

использовать для генерирования интернет-адресов файлов функцию asset():

<link href="{{ asset('/styles/main.css') }}" rel="stylesheet" type="text/css">

- файлы будут обслуживаться другим веб-сервером:

в настройках ASSET\_URL или asset\_url указать полный интернет-адрес этой папки:

ASSET\_URL=cdn.somesite/public

использовать для генерирования интернет-адресов файлов функцию asset():

<link href="{{ asset('/styles/main.css', false) }}" rel="stylesheet" type="text/css">

**- Пагинация(12)**(https://laravel.com/docs/9.x/pagination)**:**

смотри Controller

**- Разграничение доступа в шаблонах(13.8.1.2):**

**@auth** ... @else ... @endauth - выводит содержимое if, если был выполнен вход, и содержимое else - в противном случае:

@auth

<содержимое if>

@else

<содержимое else>

@endauth

@auth

<form action="{{ route('logout') }}" method="POST">

@csrf

<input type="submit" value="Bыxoд">

</form>

@else

<а href="{{ route('login') }}">Вход</а>

@endauth

**@guest** ... @else ... @endguest - выводит содержимое if, если вход не был выполнен, и содержимое else - в противном случае:

@guest

<содержимое if>

@else

<содержимое else>

@endguest

**- Разграничение доступа посредством политик(13.8.3.2):**

Политики можно применять в шаблонах для отображения определенной информации только **авторизированным** пользователям (стр.311).

Большая тема, описана в Auth, не хочется повторятся.