byteXL

UNIT - 2

HTML template to Laravel Blade template

Template Inheritance

In most of the modern webpages, a fixed theme is followed in all the webpages. Thus it is greatly effective to be able to reuse your code so that you don’t have to write again the repeating parts in your code and Blade greatly helps you in achieving this.

MASTER LAYOUT

Master page layout defines the common layout across all the web pages. All the web applications have the master page layout to define the common layout across all the web pages. The blade templating engine defines the master layout that can be extended by all the web pages. The master page layout is available in the **/resources/views/layouts/** directory.

- **Let's understand through an example.**
- First, create the folder named as '**layout**' in **resources/views/** directory.
- Now, create a new file in layout folder '**master.blade.php**'.
- We add the following code in the **master.blade.php** file.

EXTENDING MASTER LAYOUT

Now, we are going to extend the above master layout in **contact.blade.php** file as shown below:

Contact.blade.php

```
@extends('layout.master')
@section('content')
<h1>Contact Page </h1>
@stop
```

In the above code, we use the **@extends** directive. The '**@extends**' directive is used to inherit the blade layout in **contact.blade.php** file. The '@section('content')' defines the section of the content.

Now, Add the following route in **web.php** file.

```
Route::get('/contact', function () {
return view('contact');
});
```

We can also add the javascript code in **contact.blade.php** file. Suppose I added the following code in **contact.blade.php** file.

DISPLAYING VARIABLES

You may display data that is passed to your Blade views by wrapping the variable in curly braces. For example, given the following route:

```
Route::get('/', function () {
return view('welcome', ['name' => 'Samantha']);
});
```

You may display the contents of the name variable like so:

```
Hello, {{ $name }}.
```

Blade's {{ }} echo statements are automatically sent through PHP's htmlspecialchars function to prevent XSS attacks.

You are not limited to displaying the contents of the variables passed to the view. You may also echo the results of any PHP function. In fact, you can put any PHP code you wish inside of a Blade echo statement:

```
The current UNIX timestamp is {{ time() }}.
```

BLADE CONDITIONAL STATEMENT

'If' statements

```
@if ($i > 10)
<p>{{ $i }} is large.</p>
@elseif ($i == 10)
<p>{{ $i }} is ten.</p>
@else
 <p>{{ $i }} is small.</p>
 @endif
```

'Unless' statements

(Short syntax for 'if not'.)

```
 @unless ($user->hasName())
 <p>A user has no name.</p>
 @endunless
```

BLADE LOOPS

'While' loop

```
@while (true)
<p>I'm looping forever.</p>
@endwhile
```

'Foreach' loop

```
@foreach ($users as $id => $name)
<p>User {{ $name }} has ID {{ $id }}.</p>
@endforeach
```

'Forelse' Loop

(Same as 'foreach' loop, but adds a special @empty directive, which is executed when the array
expression iterated over is empty, as a way to show default content .)

```
@forelse($posts as $post)
<p>{{ $post }} is the post content.</p>
@empty
<p>There are no posts.</p>
@endforelse
```

BLADE LOOPS

'While' loop

```
@while (true)
<p>I'm looping forever.</p>
@endwhile
```

'Foreach' loop

```
@foreach ($users as $id => $name)
<p>User {{ $name }} has ID {{ $id }}.</p>
@endforeach
```

'Forelse' Loop

(Same as 'foreach' loop, but adds a special @empty directive, which is executed when the array
expression iterated over is empty, as a way to show default content .)

```
@forelse($posts as $post)
<p>{{ $post }} is the post content.</p>
@empty
<p>There are no posts.</p>
@endforelse
```

BLADE LOOPS

'While' loop

```
@while (true)
<p>I'm looping forever.</p>
@endwhile
```

'Foreach' loop

```
@foreach ($users as $id => $name)
<p>User {{ $name }} has ID {{ $id }}.</p>
@endforeach
```

'Forelse' Loop

(Same as 'foreach' loop, but adds a special @empty directive, which is executed when the array
expression iterated over is empty, as a way to show default content .)

```
@forelse($posts as $post)
<p>{{ $post }} is the post content.</p>
@empty
<p>There are no posts.</p>
@endforelse
```

EXECUTING PHP FUNCTION IN BLADE

Although it might not be proper to do such thing in a view if you intend to separate concerns

strictly, the php Blade directive allows a way to execute PHP code, for instance, to set a variable:

```
@php($varName = 'Enter content ')
(same as:)
@php
$varName = 'Enter content ';
@endphp
later:
{{ $varName }}
```

Result:

Enter content

DISPLAYING YOUR VIEW

- Of course, it's not practical to return entire HTML documents strings directly from your routes and controllers. Thankfully, views provide a convenient way to place all of our HTML in separate files.
- Views separate your controller / application logic from your presentation logic and are stored in the resources/views directory. When using Laravel, view templates are usually written using the Blade templating language. A simple view might look something like this:

```
<!-- View stored in resources/views/greeting.blade.php -->
<html>
<body>
<h1>Hello, {{ $name }}</h1>
</body>
</html>
```

CREATING AND USING BASIC VIEWS

You may create a view by placing a file with the .blade.php extension in your application's resources/views directory. The .blade.php extension informs the framework that the file contains a Blade template. Blade templates contain HTML as well as Blade directives that allow you to easily echo values, create "if" statements, iterate over data, and more.

Once you have created a view, you may return it from one of your application's routes or controllers using the global view helper:

```
Route::get('/', function () {
return view('greeting', ['name' => 'James']);
});
```

NESTED VIEWS

Let's understand the nested views through an example.

Suppose we want to know the admin details. The view of the admin details is available at the **resources/views/admin/details.blade.php** directory.

STEP 1: we create **details.blade.php** file in the admin folder, and the code of the **details.blade.php** file is given below:

```
<html>
<body>
<h1>Admin Details</h1>
</body>
</html>
```

Step 2: Now, we need to add the **display()** function in P**ostController.php** file which is returning the view of the '**admin.details**'.

```
public function display(){
return view('admin.details');
}
```

Step 3: Lastly, we will add the route in a **web.php** file

```
Route::get('/details', 'PostController@display');
```

Step 4: To see the output, enter the url '**http://localhost/laravelproject/public/details**' to the web browser.

ADDING ASSESTS

- A dynamic website almost requires the use of CSS and JavaScript. Using a Laravel asset package provides an easy way to manage these assets and include them in our views.
- Getting ready
- For this recipe, we'll need to use the code created in the *Loading a view into another view/nested views* recipe.
- How to do it...
- To complete this recipe, follow these steps:
- Open the composer.json file and add the asset package to the require section, so it looks similar to the following:
- "require": { "laravel/framework": "4.0.*", "teepluss/asset": "dev-master" }

- In the command line, run composer update to download the package as follows:
- **php composer.phar update**
- Open the app/config/app.php file and add ServiceProvider to the end of the providers array as follows:
- 'Teepluss\Asset\AssetServiceProvider'
- In the same file, in the aliases array, add the alias for the package as follows:
- 'Asset' => 'Teepluss\Asset\Facades\Asset'Copy
- In the app/filters.php file, add a custom filter for our assets.

INTEGRATING WITH BOOTSTRAP

- Creating the layout is an important part of any project. Realizing this, Laravel comes with a Blade templating engine which generates HTML based sleek designs and templates. All Laravel views built using Blade are located in **resources/views.**
- Bootstrap is well known in the development circles for impressive design options. Laravel makes it incredibly easy to use Bootstrap templates in the project's views. In this tutorial, I will demonstrate how Bootstrap templates could be used within the Laravel Blade engine.
- By default, Laravel uses NPM to install both of these frontend packages. Laravel does not dictate which JavaScript or CSS pre-processors you use, it does provide a basic starting point using Bootstrap, React, and / or Vue that will be helpful for many applications

- Integrating the Bootstrap template with Laravel is a simple process. All you need to do is cut your HTML Bootstrap into tiny Blade template contents, and then use, extend and/or include the templates in the main Blade file.

THANK YOU