Blade components

The essentials

Components are a modern way of defining all kinds of reusable view code. It can serve many purposes, but one is for building layouts. This is what you must know about components.

For example, you can create a component for the entire navigation menu - navbar - of your application. The presentation is the HTML structure of all the navigation menu items like links, dropdowns, search bar and login. It might have complex logic like dynamically show or hide menu items, pending on the user's role or current login state

In most web frameworks, components are a combination of presentation (HTML and CSS) and logic (PHP, JavaScript, etc.). Sometimes spread over different files. In Laravel it can be like views and PHP classes. For the essentials of this course, we keep it to the most simple form: *anonymous components*, which means a component with only a Blade view and no PHP class.

Creating anonymous components

Blade templates for components reside in the `resources/views/components` directory. You can just create a new `.blade.php` file there or use the Artisan command:

```
php artisan make:component navbar --view
```

It is also possible to use the Dot notation to create structure in you component directory, like:

```
php artisan make:component nav.bar --view
```

This will create a Blade file at `resources/views/components/nav/bar.blade.php`. The content of the file will be something like:

```
<div> <!-- Because you are alive, everything is possible. - Thich Nhat Hanh --> </div>
```

Here you can create the HTML structure of your component, for instance:

```
{{-- Navigation bar --}} <nav class="navbar is-primary has-text-white"> <div class="container"> <div class="navbar-brand"> {{-- the container for the brand logo --}} <a href="/" class="navbar-item"> <i class="fa-solid fa-list-check"></i> TaskITEasy </a> {{-- The navbar-burger is a hamburger menu that only appears on touch devices. --}} <a role="button" class="navbar-burger" data-target="navMenu" aria-label="menu" aria-expanded="false"> <span aria-hidden="true"></span> <span aria-hidden="true"></span> <span aria-hidden="true"></span> </a> </div> {{-- The navbar-menu is hidden on touch devices (<1024px). The modifier class `is-active` is added by means of the javascript to display it. --}} <div class="navbar-menu" id="navMenu"> <div class="navbar-start"> <a href="{{ route('home') }}" class="navbar-item">Home</a> <a href="{{ route('about') }}" class="navbar-item is-active">About</a> </div> </div> </div> </nav>
```

⚠️ Every component must be wrapped by a **single parent HTML element.** Everything within a component needs to be nested inside this one HTML element.

Rendering components

Once created, components can be used, "rendered", inside normal Blade views. You do this just by adding an HTML tag to your view. This tag starts with an 'x-' followed by the name of the component. For example, the view created above can be rendered like so:

```
<x-nav.bar />
```

Slots

In building components, you might need to send extra content to them using "slots." These slots are areas within a component where you can inject content from outside. When you use slots, you typically echo the `$slot` variable in your component's code to render this injected content.

For example, let's consider the navbar component we created above. Within the component, we only want to focus on the basic structure of the navbar, like the brand and the burger, and not the menu items themselves. We want this to be defined by the views who are using the navbar. So, we create a slot for the menu items, like so:

```
{{-- Navigation bar --}} <nav class="navbar is-primary has-text-white">
<div class="container"> <div class="navbar-brand"> {{-- the container for
the brand logo --}} <a href="/" class="navbar-item"> <i class="fa-solid
fa-list-check"></i> TaskITEasy </a> {{-- The navbar-burger is a
hamburger menu that only appears on touch devices. --}} <a role="button"
class="navbar-burger" data-target="navMenu" aria-label="menu" aria-
expanded="false"> <span aria-hidden="true"></span> <span aria-
hidden="true"></span> <span aria-hidden="true"></span> </a> </div> {{--
The navbar-menu is hidden on touch devices (<1024px). The modifier class
`is-active` is added by means of the javascript to display it. --}} <div
class="navbar-menu" id="navMenu"> <div class="navbar-start"> {{ $slot }}
</div> </div> </div> </nav>
```

We replaced the actual navigation items with the `{{ $slot }}` statement. Then, when we render the component, we can just add the menu items as its content. like so:

```
<x-nav.bar> <a href="{{ route('home') }}" class="navbar-item">Home</a> <a href="{{ route('about') }}" class="navbar-item is-active">About</a> </x-nav.bar>
```

Adding extra slots

The `{{ $slot }}` is the default slot of a component. It is possible to add extra slots like so:

```
{{-- Navigation bar --}} <nav class="navbar is-primary has-text-white"> <div class="container"> <div class="navbar-brand"> {{-- the slot for the brand logo --}} {{ $brand }} {{-- The navbar-burger is a hamburger menu that only appears on touch devices. --}} ... </div> {{-- The navbar-menu ... --}} <div class="navbar-menu" id="navMenu"> <div class="navbar-start"> {{ $slot }} </div> <div class="navbar-end"> {{ $end }} </div> </div> </div> </nav>
```

We created 2 extra slots: `brand` and `end`. If you want to add content to the slots when rendering the component, you use the `x-slot` tag:

```
<x-nav.bar> <x-slot:brand> <a href="/" class="navbar-item"> <i class="fa-solid fa-list-check"></i> TaskITEasy </a> </x-slot:brand> <a href="{{ route('home') }}" class="navbar-item">Home</a> <a href="{{ route('about') }}" class="navbar-item is-active">About</a> <x-slot:end></x-slot:end> </x-nav.bar>
```

Here we declared content for the `brand` slot but left the `end` slot empty. This means that the `end` slot will be left empty.

Digging deeper

Passing Hard-coded Data

It is possible to pass data to components when they are rendered. Data can be hard-coded, primitive values or PHP expressions or variables. Let's discuss this by the following example. We're creating a component that represents a navbar item: `nav.item`. We want its route name to be something that must be passed from the rendering view, so it can be used like:

```
<x-nav.item route="home">Home</x-nav.item> <x-nav.item
route="about">About</x-nav.item>
```

Notice the custom `route` attribute here. It can be accessed in the component with the `$route` variable:

```
<a href="{{ route($route) }}" class="navbar-item"> {{ $slot }} </a>
```

Passing Variables

It is also possible to pass variables and PHP expressions to components via attributes that use the `:` character as a prefix. In the following example a component named `alert` is rendered. It has two attributes. A hard-coded `type` and a variable `message`:

```
<x-alert type="error" :message="$message"/>
```

The content of the component may look like:

```
<div class="notification is-{{ $type }}"> {{ $message }} </div>
```

Adding extra attributes

We've talked about passing data attributes to a component. But besides data, you might also need to include other HTML attributes, like classes, that aren't directly related to the component's functionality. Usually, you'd want to send these extra attributes to the main element of the component's template. For example, let's say we have a component that represents a very simple input component:

```
<input type="$type" name="$name" value="{{ $slot }}">
```

That we render like so:

```
<x-input name="foo" type="text">/<x-input>
```

But some of the inputs should be `readonly`, by adding this as an attribute to the `<input>`. We want to specify this like:

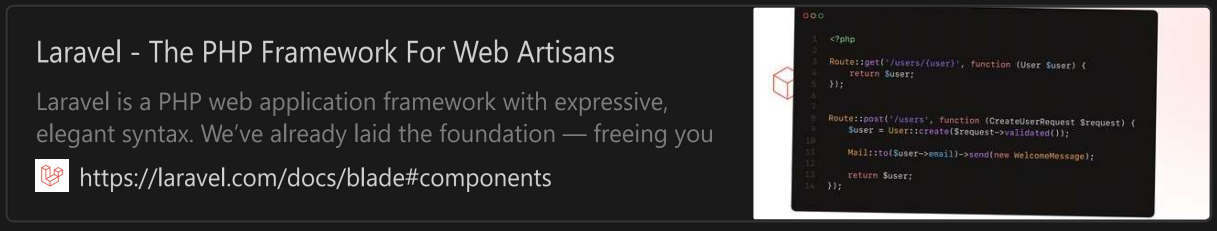
```
<x-input name="foo" type="text" readonly>/<x-input>
```

All of the attributes that are not part of the component's variables will automatically be added to the component's "attribute bag". This attribute bag is automatically made available to the component via the `$attributes` variable. All of the attributes may be rendered within the component by echoing this variable:

```
<input type="$type" name="$name" value="{{ $slot }}" {{ $attributes }}>
```

Further reading

You can read all about components here:



Specifically noteworthy topics are:

- **Default/merged attributes** - which makes it possible to pass extra classes to the component
- **Components that have classes** - which makes it possible to add more complex logic to the component