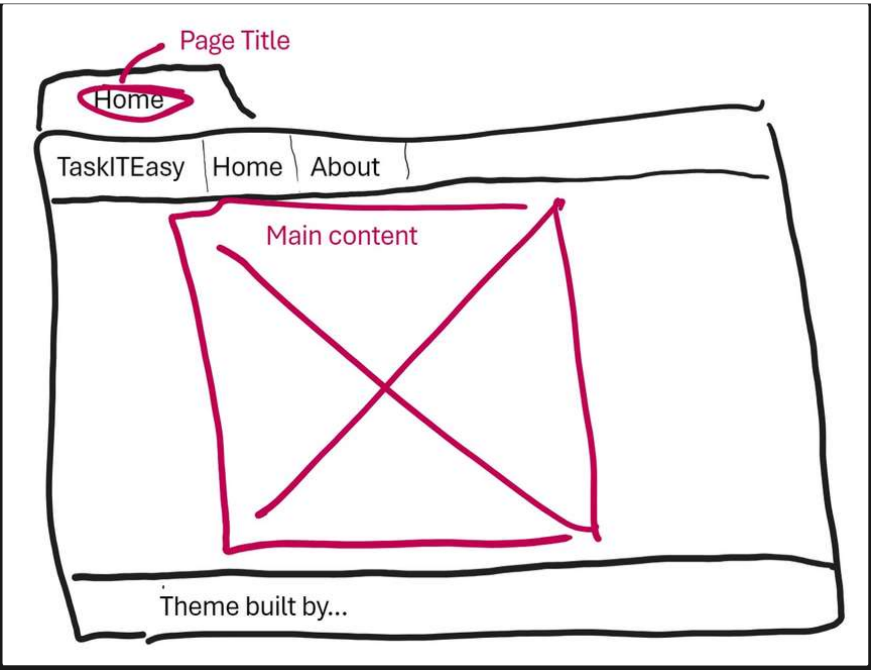
Building Layouts

The essentials

Internal consistency dictates that a web application must maintain the same general layout across various pages. It would be incredibly cumbersome and hard to maintain our application if we had to repeat the entire layout HTML in every view we create. Thankfully, it's convenient to define this layout as a single component and then use it throughout our application.

This means that you can define one single layout that has the basic HTML structure that is to be repeated across the various pages and that your actual views must then “inherit” this structure and define the actual content that must appear on that specific page. If you look at the general layout of the *TaskITEasy* app, you can see that it has two distinguished locations where specific content needs to go, the page title and main content. The rest of the layout should remain the same:



Lo-fi wireframe of the TaskITEasy layout design

In order to work with layouts, there are two things you need to do:

- Define the layout itself; which includes the common HTML structure and the declaration of the parts, often called *slots* or *sections* that should contain the specific HTML for each page
- Applying the layout in each specific view; which means that the view must declare which layout it “inherits” and provide specific content for each slot this layout requires

Laravel Blade actually has two technologies that can help you implement this: using Components and Template Inheritance. We will explain components here and leave Template Inheritance to the Digging Deeper section

Layouts Using Components

Components are a modern way of defining all kinds of reusable view code. It can serve many purposes, but one is for building layouts. You can read about the details here:

Blade components

Defining the layout component

The layout component is a blade file that should be placed in the `resources/views/components` directory. For example, imagine we are building a "todo" list application. We might define a component with a name `layout` that looks like the following:

```
<!-- resources/views/components/layout.blade.php --> <html> <head> <title>
{{ $title ?? 'Todo Manager' }}</title> </head> <body> <h1>Todos</h1> <hr/>
{{ $slot }} </body> </html>
```

This component assumes that there are two variables: `$title` and `$slot`. The `$title` is optional because the ternary operator let it render 'Todo Manager' when the variable doesn't exist.

Applying The Layout Component

Once the `layout` component has been defined, we may create a Blade view that utilizes the component. A very basic welcome page might look like this:

```
<!-- resources/views/welcome.blade.php --> <x-layout> Hello! Welcome to
your Todo Manager </x-layout>
```

The `<x-layout>` tells Blade to render the `layout` component where the HTML content inside the element will be its `$slot`. When this view is called, it will render like so:

```
<html> <head> <title>Todo Manager</title> </head> <body> <h1>Todos</h1>
<hr/> Hello! Welcome to your Todo Manager </body> </html>
```

In the next example, we will define a simple view that displays a task list:

```
<!-- resources/views/tasks.blade.php --> <x-layout> @foreach ($tasks as
$task) {{ $task }} @endforeach </x-layout>
```

As you remember, the layout also declared the `$title` slot. An implementing view is able to provide a specific title using the `<x-slot>` declaration:

```
<!-- resources/views/welcome.blade.php --> <x-layout> <x-slot:title>
Custom Title </x-slot> Hello! Welcome to your Todo Manager </x-layout>
```

The `<x-slot:title>` tells Blade that the `$title` slot is set.

Digging deeper

The SHOULDs about this topic, described as links with some extra information to help understanding the content of the link

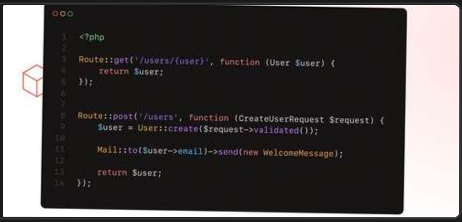
Template inheritance

As mentioned earlier, there is an alternative way of defining layouts: via Template Inheritance. Just like with components, you also need to define a layout view which can be used in implementing views. But, instead of using component slots, you use `@section` and `@yield` directives for defining sections of content and the `@extends` directive to specify which layout an implementing view is using. You can read about this here:

Laravel - The PHP Framework For Web Artisans

Laravel is a PHP web application framework with expressive, elegant syntax. We've already laid the foundation — freeing you

https://laravel.com/docs/blade#layouts-using-template-in...



```
<?php

Route::get('/users/{user}', function (User $user) {
    return $user;
});


Route::post('/users', function (CreateUserRequest $request) {
    $user = User::create($request->validated());

    Mail::to($user->email)->send(new WelcomeMessage);

    return $user;
});
```

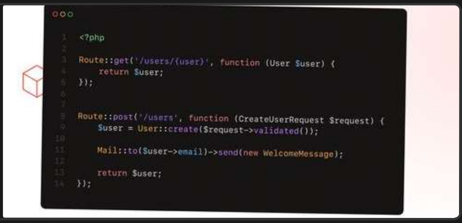
Further reading

Below you can find a link to a chapter in the the Laravel Blade documentation page that is about layouts:

Laravel - The PHP Framework For Web Artisans

Laravel is a PHP web application framework with expressive, elegant syntax. We've already laid the foundation — freeing you

https://laravel.com/docs/blade#building-layouts



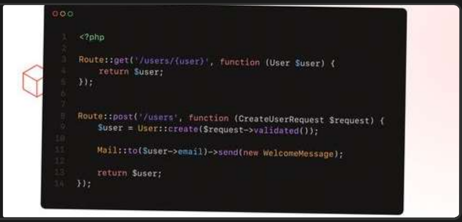
```
<?php

Route::get('/users/{user}', function (User $user) {
    return $user;
});


Route::post('/users', function (CreateUserRequest $request) {
    $user = User::create($request->validated());

    Mail::to($user->email)->send(new WelcomeMessage);

    return $user;
});
```

Beyond that, **Stacks** are also a noteworthy topic here. It might help when you want to push styles or scripts from various implementing views and components into the same location in the HTML layout structure:

Laravel - The PHP Framework For Web Artisans

Laravel is a PHP web application framework with expressive, elegant syntax. We've already laid the foundation — freeing you

https://laravel.com/docs/blade#stacks



```
<?php

Route::get('/users/{user}', function (User $user) {
    return $user;
});


Route::post('/users', function (CreateUserRequest $request) {
    $user = User::create($request->validated());

    Mail::to($user->email)->send(new WelcomeMessage);

    return $user;
});
```