BICOL UNIVERSITY

COLLEGE OF SCIENCE

Legaspi City

LAYOUT

LAB 3 ACTIVITY

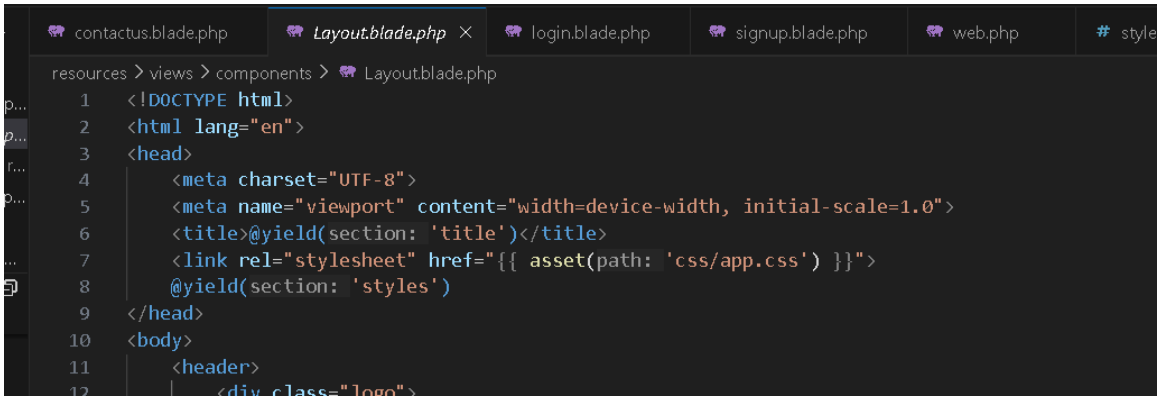
RICHARD D. BILAN JR.

BSIT 3-C

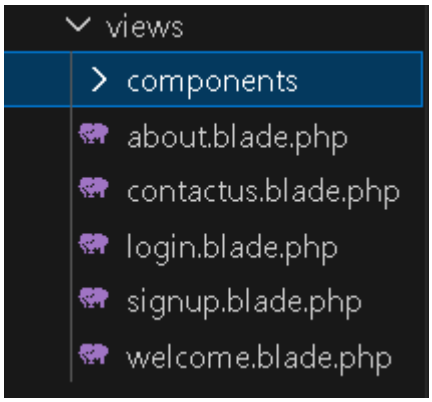
Explanations: Write a detailed explanation of your work, including:

The purpose of the layout file and how it is used.

How each view file extends the layout and inserts specific content.

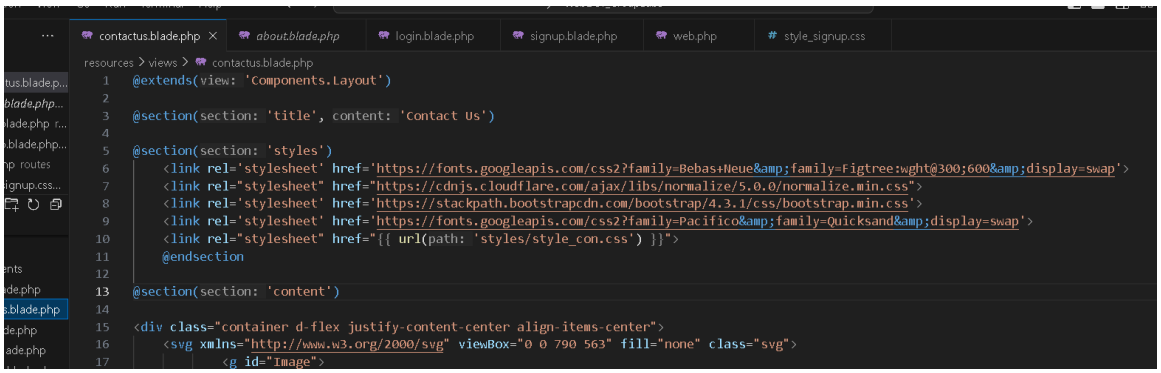


The first step we do is we create a layout page inside the new component file. This layout will be extended by individual views. The `<title>` tag uses `@yield('title')`, a Blade directive the this allows each view extending this layout to inject its own title. Also, `@yield('styles')` acts as a placeholder for additional styles that can be added by the extending views.



Next is the three views were will be extended. In this Activity, we did not create another or new file we just continue our work on previous activity which is the Activity lab2.


contactus.blade.php
about.blade.php
login.blade.php
signup.blade.php
web.php
style_signup.css
ources > views > about.blade.php
@extends(view: 'components.layout')
@section(section: 'title', content: 'About Us')
@section(section: 'styles')
<link rel='stylesheet' href='https://fonts.googleapis.com/css2?family=Bebas+Neue&family=Figtree:wght@300;600&display=swap'>
<link rel="stylesheet" href="{{ asset(path: 'styles/styles_a.css') }}">
@endsection
@section(section: 'content')

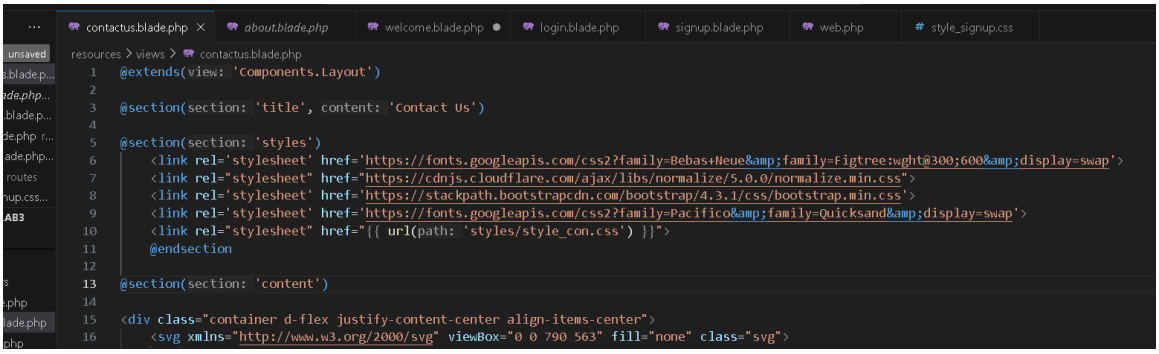


In the about page, first we extended the layout I created in the Components which is the layout folder. Using @extends('components.layout'). This allows the about view to receive the structure defined in the layout. Next, we used the @section directive to define the content for the `title`, `styles`, and `content` sections. In the `title` section, we set 'About Us' to be the title of the page. In the `styles` section, we included an external Google Fonts stylesheet and a custom CSS file using `{{ asset() }}` to dynamically link the stylesheet's path. Lastly, I defined the `content` section, where I will add the specific content for the About page. This setup enables the layout to inject page-specific content while maintaining a consistent design across the site.

the @endsection directive here, which is used to close the section block that was opened earlier in the Blade file. It's important to include @endsection after defining the content for each section to properly close it.

The **welcome.blade.php** file, we define sections for the title, additional styles, and the main content. The @section('title') directive sets the title for this specific page as 'Web Development,' and @section('styles') includes an external Google font and a specific stylesheet for the page. Finally, the @section('content') block is used to add custom content, such as images, headings, and buttons, that will appear on this page.

In this **About Us** page, we begin by extending the components. Layout file using the @extends directive, which allows us to reuse the base layout structure for this page. Then, we use the @section('title') directive to insert the page-specific title, "About Us," which will be displayed in the browser tab.



Same thing we did in the **contact us** page.

The routing setup and how it serves the views.

```
Route::get(uri: '/', action: function (): Factory|View {
    return view(view: 'welcome');
})->name(name: 'welcome');
```

Route::get('/', ...): This defines a GET route for the root URL (/).

function () { return view('welcome'); }: The route uses an anonymous function to return the welcome view when this route is accessed.

name('welcome'): This names the route welcome, allowing you to refer to this route by name elsewhere in your application.

```
Route::get(uri: '/contactus', action: function (): Factory|View {
    return view(view: 'contactus');
})->name(name: 'contact');
```

Route::get('/contactus', ...): This defines a GET route for the /contactus URL.

function () { return view('contactus'); }: The route returns the contactus view.

name('contact'): This names the route contact, which can be used in your application to generate URLs or redirects.

```
Route::get(uri: '/about', action: function (): Factory|View {
    return view(view: 'about');
})->name(name: 'about');
```

Route::get('/about', ...): This defines a GET route for the /about URL.

function () { return view('about'); }: The route returns the about view.

name('about'): This names the route about, useful for referencing the route elsewhere in your app.

When a user requests a URL, Laravel matches the request to the appropriate route based on the URL and HTTP method (GET in this case).

View Rendering: For each route, Laravel executes the associated callback function. In this setup, each callback function returns a view using the view() helper function.

Explain any challenges you faced and how you resolved them.

The challenge we faced was correctly defining and closing sections using the @section and @endsection directives. If these directives aren't used correctly, Laravel won't properly inject the content into the layout. We resolved this by reviewing the Blade syntax and ensuring every section was properly closed with @endsection.

Also, we encounter when we created the extend and implement it on each view the css become messy and it overwrite something because When extending a layout and implementing it in multiple views, a common issue is that the CSS can become messy or styles may overwrite each other. This happens because styles defined in one view can unintentionally affect elements in another, especially if the CSS is not scoped properly or if there are conflicting styles.

Explore the difference between {{$slot}} and @yield

In our Laravel Activity we did not use the {{$slot}} but I tried to explore on google what's its mean and purpose and compare it to @yield that the one we used. I figure out that The difference between {{$slot}} and @yield is subtle but important. @yield is used in the layout file to define a placeholder for content, which is then provided by the individual view files using @section. {{$slot}}, on the other hand, is used in Blade components to insert content dynamically when a component is used. In this case, we used @yield in the layout to specify where specific content (like title, styles, and main content) should be inserted by each view.