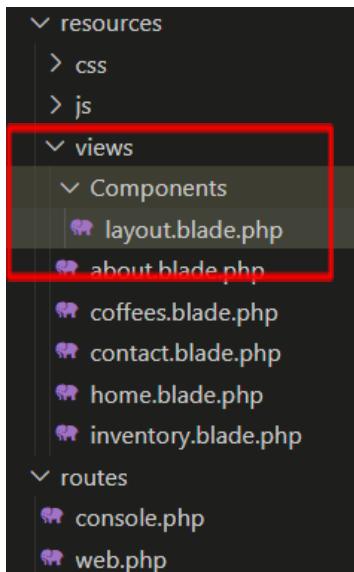


Jethro J. Manzanillo
BSIT - 3C
Web Development 2

Part 1: Creating a Layout File



In the `resources/views` directory, we created a new folder named `Components` to organize reusable Blade components.

Inside the `Components` folder, we created a file called `Layout.blade.php` to define the structure that other views will extend.

Layout.blade.php interface (Navigation bar and Footer):

Which we can see in every blade.php.

WebDev Cafe

HOME ABOUT COFFEES INVENTORY CONTACT

coffee

Address

Find us at our cozy spot, where you can enjoy great coffee and company.

+01 1234567890

itwebdev10@gmail.com

Your Email

2020 All Rights Reserved.

f t in ig

Basic Structure of Layout.blade.php:

```
layout.blade.php X
resources > views > Components > layout.blade.php
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <!-- Basic -->
6      <meta charset="utf-8">
7      <meta http-equiv="X-UA-Compatible" content="IE=edge">
8      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
9      <!-- Site Metas -->
10     <title>@yield('title', 'WebDev Cafe')</title>
11     <meta name="keywords" content="WebDev Cafe, coffee shop, cafe, coffee, website, blade">
12     <meta name="description" content="Welcome to WebDev Cafe, your cozy spot for great coffee and company.">
```

DOCTYPE Declaration and HTML Setup: Ensures that the document follows HTML5 standards. The `<html>` tag sets the document's language to English using `lang="en"`.

We use the `<title>@yield('title', 'WebDev Cafe')</title>` directive to allow individual pages to set their own titles, defaulting to "WebDev Cafe" if not specified.

```
<span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <h1><b>WebDev Cafe</b></h1>
    <ul class="navbar-nav ml-auto">
        <li class="nav-item">
            <a class="nav-link" href="{{ route('home') }}>Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{{ route('about') }}>About</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{{ route('coffees') }}>Coffees</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{{ route('inventory') }}>Inventory</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{{ route('contact') }}>Contact</a>
        </li>
    </ul>
</div>
<form class="form-inline my-2 my-lg-0">
```

The `<header>` includes the navigation links styled using `navbar-nav` `ml-auto` to align them to the right, with individual items using the `nav-item` and `nav-link` classes.

```
<div class="footer_section layout_padding">
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <h3 class="address_text">Address</h3>
                <p class="footer_text">Find us at our cozy spot, where you can enjoy great coffee and company.</p>
            </div>
            <div class="location_text">
                <ul>
                    <li>
                        <a href="#">
                            <i class="fa fa-phone" aria-hidden="true"></i>
                            <span class="padding_left_10">+01 1234567890</span>
                        </a>
                    </li>
                    <li>
                        <a href="#">
                            <i class="fa fa-envelope" aria-hidden="true"></i>
                            <span class="padding_left_10">itwebdev10@gmail.com</span>
                        </a>
                    </li>
                </ul>
            </div>
            <div class="form-group">
                <textarea class="update_mail" placeholder="Your Email" rows="5" id="comment" name="Your Email"></textarea>
            </div>
            <div class="subscribe_bt">
                <a href="#"></a>
            </div>
        </div>
```

The footer is wrapped in a `footer_section` div with classes like `layout_padding` for padding, while the `container-fluid` class ensures full-width layout. The address and contact information are styled with classes such as `address_text` for the address heading and `footer_text` for the descriptive text, while social media icons use the `footer_social_icon` class.

The address and contact information are styled with classes such as `address_text` for the address heading and `footer_text` for the descriptive text, while social media icons use the `footer_social_icon` class.

```


### Address



Find us at our cozy spot, where you can enjoy great coffee and company.



- +01 1234567890
- itwebdev10@gmail.com

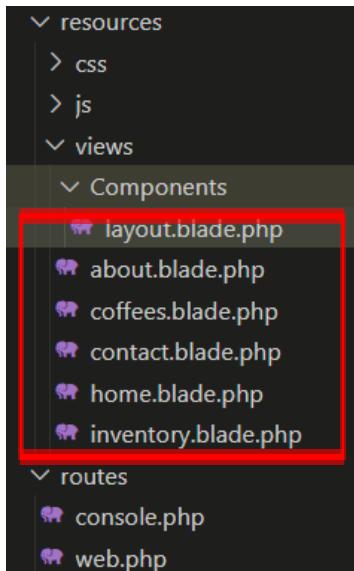


!\[Telegram Icon\]\({{ asset\('images/telegram-icon.png'\) }}\)


```

The address and contact information are styled with classes such as **address_text** for the address heading and **footer_text** for the descriptive text, while social media icons use the **footer_social_icon** class.

Part 2: Creating views

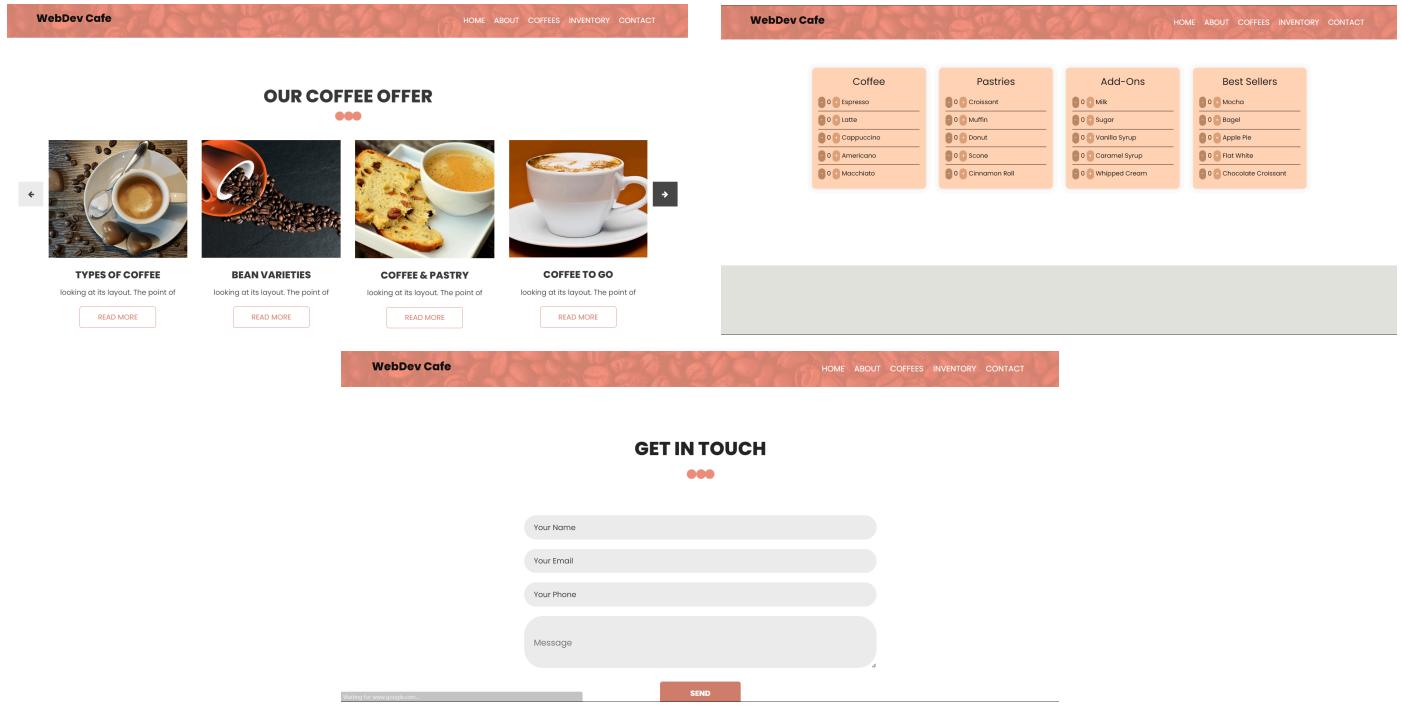


In our views, we created 5 new blade files which are **Home**, **About**, **Coffees**, **Inventory**, and **Contact**.

Layouts:

Each view has an extended layout file and includes page-specific content.

The screenshot shows the homepage of the WebDev Cafe website. At the top, there's a navigation bar with links for HOME, ABOUT, COFFEES, INVENTORY, and CONTACT. Below the navigation is a large image of a steaming coffee cup. To the right of the image, the word "coffee" is written in a large, bold, lowercase font. Underneath "coffee", the text "Tasty Of WebDev Cafe" and "more-or-less normal distribution of letters, as opposed to using" is visible. At the bottom of this section are two buttons: "About Us" and "Call Now". To the right of this content, there's a sidebar with the heading "ABOUT OUR SHOP" and three small circular icons. Further down the page, there's a photograph of the interior of a coffee shop with a counter, shelves, and hanging lights.



Part 3: Updating Routes

```
routes > 🌐 web.php
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  Route::get(uri: '/', action: function (): Factory|View {
6  |    return view(view: 'home');
7  })->name(name: 'home');
8
9  Route::get(uri: 'about', action: function (): Factory|View {
10 |    return view(view: 'about');
11 })->name(name: 'about');
12
13 Route::get(uri: 'coffees', action: function (): Factory|View {
14 |    return view(view: 'coffees');
15 })->name(name: 'coffees');
16
17 Route::get(uri: 'contact', action: function (): Factory|View {
18 |    return view(view: 'contact');
19 })->name(name: 'contact');
20
21 Route::get(uri: 'inventory', action: function (): Factory|View {
22 |    return view(view: 'inventory');
23 })->name(name: 'inventory');
```

Routes:

- We define the root URL `(/)` to return the `home` view and name this route as `home`.
- The URL `about` returns the `about` view, and we assign the route name `about`.
- For the `coffees` URL, we return the `coffees` view, and name the route `coffees`.
- The `contact` URL serves the `contact` view, with the route name `contact`.
- We define the `inventory` URL to return the `inventory` view and name this route `inventory`.

This setup ensures that each URL is linked to its respective view, and the `name()` method allows us to refer to these routes conveniently in our application.

Part 4: Explanations

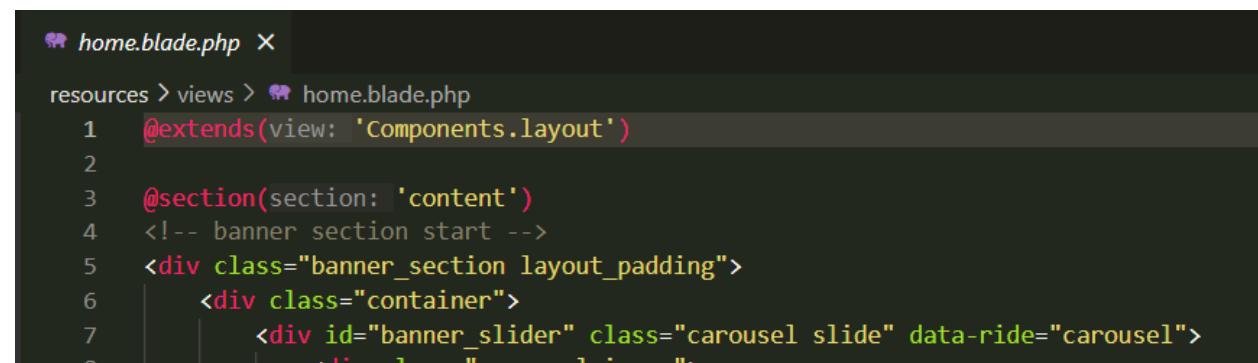
The purpose of the layout file and how it is used.

The layout file, `Layout.blade.php`, serves as a template that defines the overall structure of our web pages. It contains the common elements such as the `<head>` section with meta tags and stylesheets, the header with the navigation menu, and the footer. This layout file helps maintain consistency across all pages and reduces code duplication, as we don't have to repeat the common structure in each view.

In our project, the layout file provides placeholders for dynamic content using the `@yield` directive. This allows individual views to insert their unique content into specific sections of the layout, such as the `<title>` tag and the main content area. By doing this, we ensure that all views share a unified appearance while still being able to present unique information.

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

Each view file (like `home.blade.php`, `about.blade.php`, etc.) starts with `@extends('Components.Layout')`. This tells Laravel that the view should use the layout we created as its base. We then use `@section()` to add unique content for each view. For example, in `home.blade.php`, we use:



```
home.blade.php X
resources > views > home.blade.php
1  @extends(view: 'Components.layout')
2
3  @section(section: 'content')
4  <!-- banner section start -->
5  <div class="banner_section layout_padding">
6      <div class="container">
7          <div id="banner_slider" class="carousel slide" data-ride="carousel">
8              <div class="carousel-inner">
```

The routing setup and how it serves the views.

In the `routes/web.php` file, we define routes to connect URLs to their respective views. For example, the route:

```
routes > └─ web.php
  1  <?php
  2
  3  use Illuminate\Support\Facades\Route;
  4
  5  Route::get(uri: '/', action: function (): Factory|View {
  6  |    return view(view: 'home');
  7  })->name(name: 'home');
  8
  9  Route::get(uri: 'about', action: function (): Factory|View {
10  |    return view(view: 'about');
11  })->name(name: 'about');
12
13  Route::get(uri: 'coffees', action: function (): Factory|View {
14  |    return view(view: 'coffees');
15  })->name(name: 'coffees');
16
17  Route::get(uri: 'contact', action: function (): Factory|View {
18  |    return view(view: 'contact');
19  })->name(name: 'contact');
20
21  Route::get(uri: 'inventory', action: function (): Factory|View {
22  |    return view(view: 'inventory');
23  })->name(name: 'inventory');
```

This code makes the root URL `()` show the `home` view. We do the same for the other views like `about`, `coffees`, `contact`, and `inventory`. Each URL is linked to its own view file and given a name for easy reference.

Explain any challenges you faced and how you resolved them.

Initially, I had trouble linking styles and scripts correctly. The `{{asset('path/to/file')}}` code wasn't working on my laptop, even though my groupmates didn't have this issue. After trying different solutions and struggling for a while, I finally restarted my laptop, and that solved the problem. Using the `{{asset('path/to/file')}}` helper then worked perfectly, as it pointed to the correct path in the `public` folder.

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

`@yield('section-name')` : This is used in layout files to create placeholders for content that different views can fill. It's combined with `@section('section-name')` in views to insert specific content into those placeholders.

`{{$slot}}` : This is used in Blade components, not layouts. It defines a slot where content can be injected when the component is used. It's like a customizable placeholder for reusable components.

In short, `@yield` is for layouts and sections, while `{{$slot}}` is for components and slots.