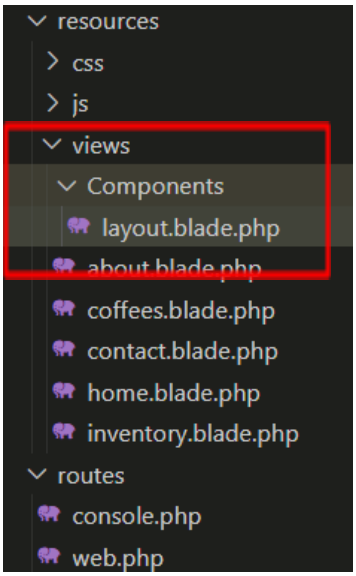
Jethro J. Manzanillo
BSIT - 3C
Web Development 2

Part 1: **Creating a Layout File**

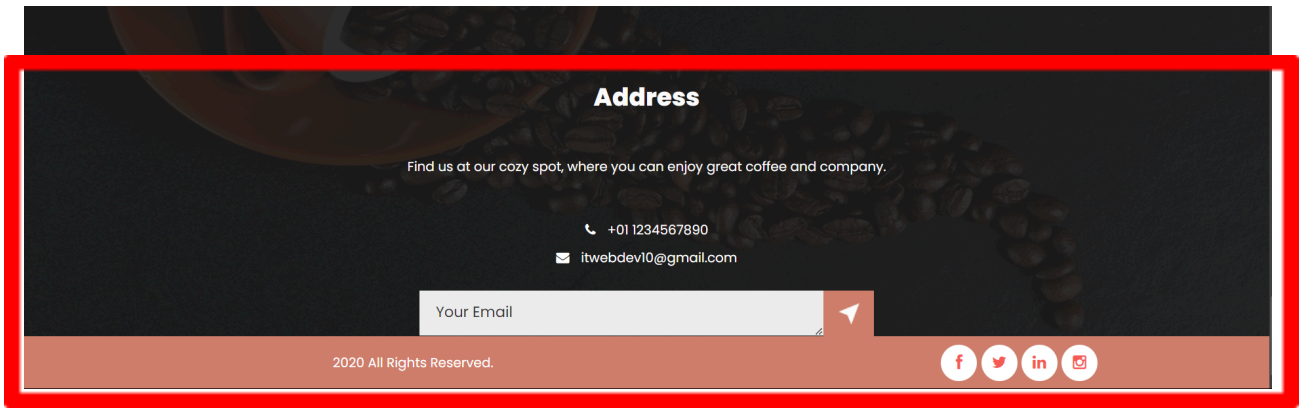


In the `resources/views` directory, we created a new folder named `Components` to organize reusable Blade components.

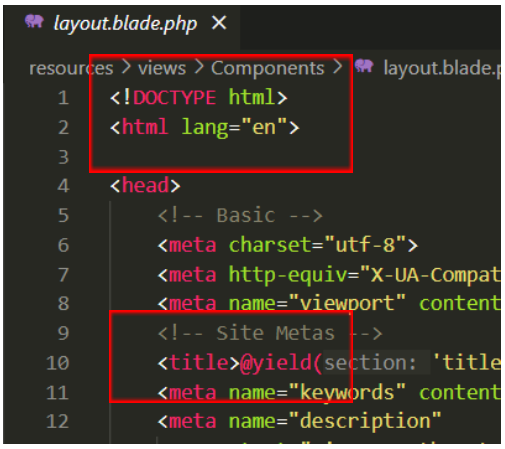
Inside the `Components` folder, we created a file called `Layout.blade.php` to define the structure that other views will extend.

Layout.blade.php interface (Navigation bar and Footer):
Which we can see in every blade.php.

coffee

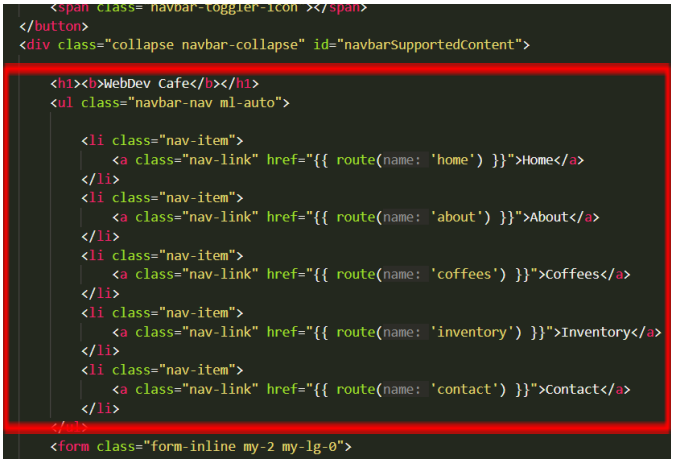


Basic Structure of Layout.blade.php:



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.



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.

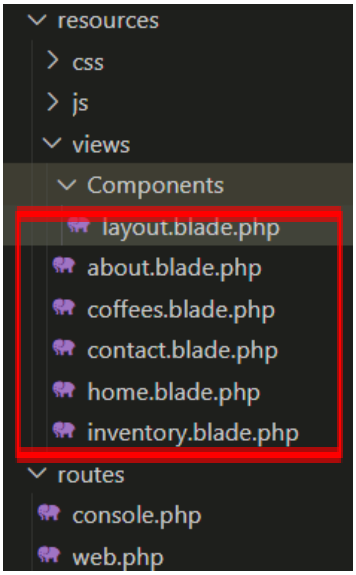
The footer is wrapped in a **`footer_section`** div with classes like **`layout_padding`** for padding, while the **`container-fluid`** class ensures full-width layout. The address and contact information are styled with classes such as **`address_text`** for the address heading and **`footer_text`**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

```blade
<div class="footer_section layout_padding">
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <h1 class="address_text">Address</h1>
                <p class="footer_text">Find us at our cozy spot, where you can enjoy great coffee and company.
                </p>
                <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 class="subscribe_bt">
                        <a href="#"><img src="{{ asset(path: 'images/teligram-icon.png') }}" alt="Telegram Icon"></a>
                    </div>
                </div>
```

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.

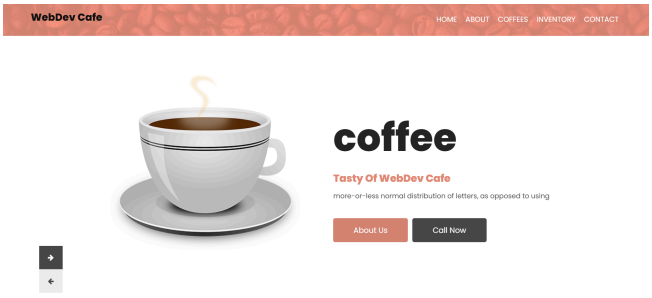
Part 2: **Creating views**

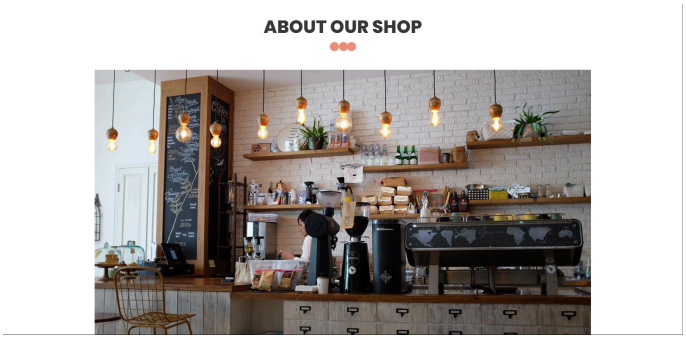


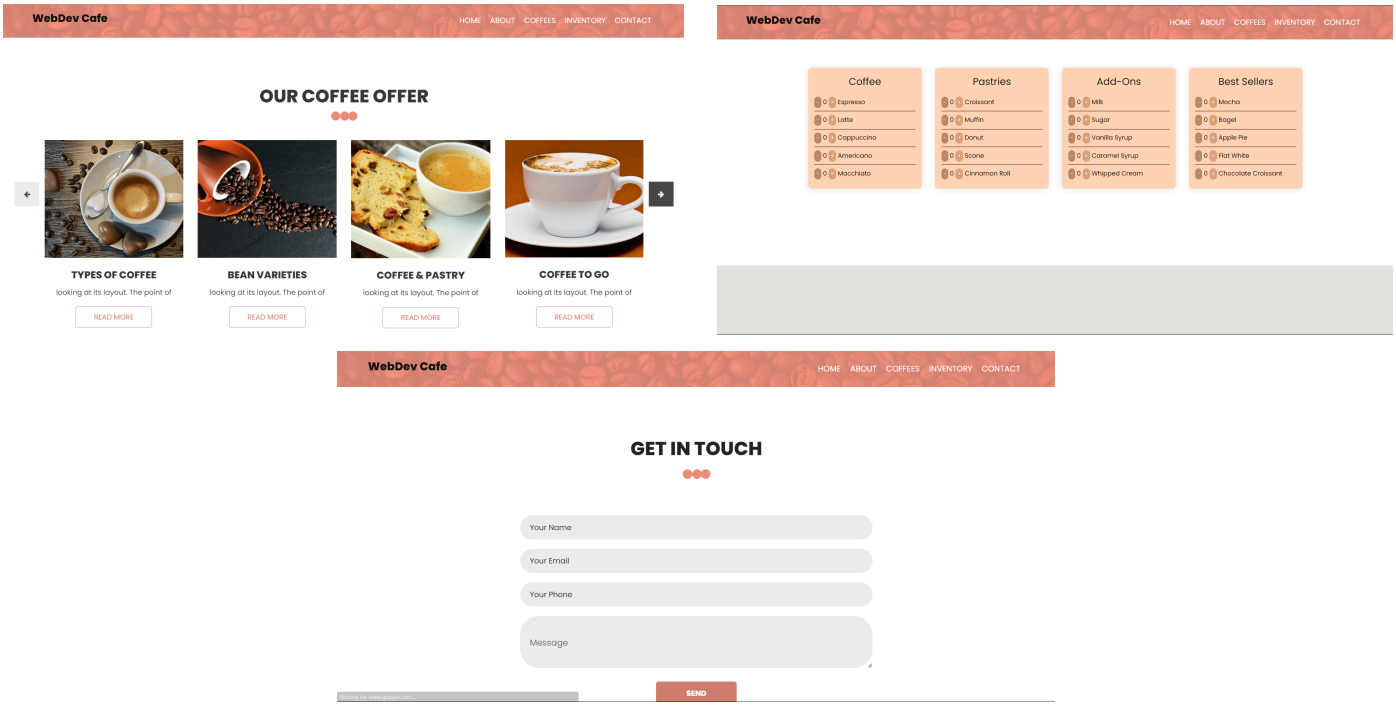
In our views, we created 5 new blade files which are **`Home, About, Coffees, Inventory, and Contact.`**

Layouts:

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







Part 3: **Updating Routes**

routes > web.php

```php
<?php

use Illuminate\Support\Facades\Route;

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

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

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

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

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

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

```php
<?php

use Illuminate\Support\Facades\Route;

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

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

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

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

Route::get(uri: 'inventory', action: function (): Factory|View {
    return view(view: 'inventory');
})->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.