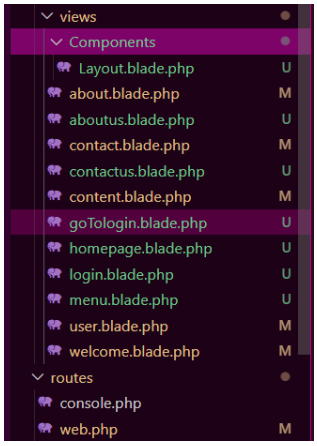
1. Using layout files in Laravel allows you to define a consistent structure for your web application. This way, you only need to write your site's main layout (like the navigation, header, and footer) once, and then inject dynamic or page-specific content into different sections using *@yield* and *@section*. This helps in minimizing code repetition and maintaining a clean, organized project structure.



2.
```
resources > views > goTologin.blade.php
1  <!-- resources/views/goTologin.blade.php -->
2  @extends('Components.Layout')
3
4  @section('title', 'Log in')
5
```

@extends('Components.Layout') in Blade template tells Laravel that this template should inherit the layout and structure defined in *Layout.blade.php*. This allows you to define common elements like headers, footers, and navigation in the *Layout.blade.php file*, while your individual views can focus on content specific to that page.

3. *@yield('content')* method in layout file is like a placeholder to display the unique content of different pages.

```
27      <main>
28          <!-- This is where the content of the child views will be displayed -->
29          @yield('content') <!--call the webpages to display its content-->
30      </main>
```

For example, the content of *goTologin.blade.php* (see below image) will be injected in the *@yield('content')* of the layout file. *@yield and @section('content')* are used together to manage dynamic content between a layout and child views (the views that extend the layout).

```
resources > views > goTologin.blade.php
1  <!-- resources/views/goTologin.blade.php -->
2  @extends('Components.Layout', ['username' => $username])
3
4  @section('title', 'About Us')
5
6  @section('content')
7      <main class="login-main">
8          <div class="background-blur"></div>
9          <div class="login-container">
10             <h1>Login</h1>
11
12             <!-- for log in form-->
13             <form action="/homepage" method="POST">
14                 @csrf
15                 <input type="text" name="username" placeholder="Enter your name" value="{{ old('username') }}" required>
16
17                 <!-- Display validation errors for the username -->
18                 @if ($errors->has('username'))
19                     <div class="error-message">
20                         {{ $errors->first('username') }}
21                     </div>
22                 @endif
23
24                 <button type="submit">Login</button>
25             </form>
26
27             <!-- Second Form for Guest Login -->
28             <form action="/homepage" method="POST">
29                 @csrf
30                 <input type="hidden" name="username" value="Guest">
31                 <button type="submit">Login as Guest</button>
32             </form>
33         </div>
34     </main>
35 @endsection
36
```

4. When it comes to routing, we didn't change much, we stick to what we use from the previous activity.

routes > web.php

```php
<?php

use Illuminate\Support\Facades\Route;

//route for the login page at the root URL "/"
Route::get('/', function () {
    return view('goTologin', ['username' => 'Guest']);
})->name('goTologin');


//route for the welcome page with dynamic username, and view the welcome message with username
Route::get('/homepage/{username}', function ($username) {
    return view('homepage', ['username' => $username]);
})->name('homepage');

//route for the about page with dynamic username
Route::get('/aboutus/{username}', function ($username) {
    return view('aboutus', ['username' => $username]);
})->name('aboutus');

//route for the content page with dynamic username
Route::get('/menu/{username}', function ($username) {
    return view('menu', ['username' => $username]);
})->name('menu');

//route for the contact page with dynamic username
Route::get('/contactus/{username}', function ($username) {
    return view('contactus', ['username' => $username]);
})->name('contactus');

//handles the log in form and username validation
Route::post('/homepage', function (\Illuminate\Http\Request $request) {
    // Validate that the username only contains alphabetic characters (a-z, A-Z)
    $request->validate([
        'username' => 'required|alpha'
    ], [
        'username.alpha' => 'The username should only contain alphabetic characters.',
```

5. *Challenges we faced and how we resolved them.*
   The layout file was straightforward and significantly enhanced our website's flexibility and efficiency. However, we did encounter some minor challenges, particularly with passing parameters across multiple pages. To address this, we made adjustments to our routing configuration, ensuring that parameters could be seamlessly passed between pages. Additionally, we implemented thorough testing to verify that the changes worked as intended, ultimately improving the overall functionality of our site.

6. Difference between @yield and $slot.
   @yield is used for defining sections in a layout that the child view can fill, while *$slot* is used within the Blade component to insert content passed from the parent view to the component.