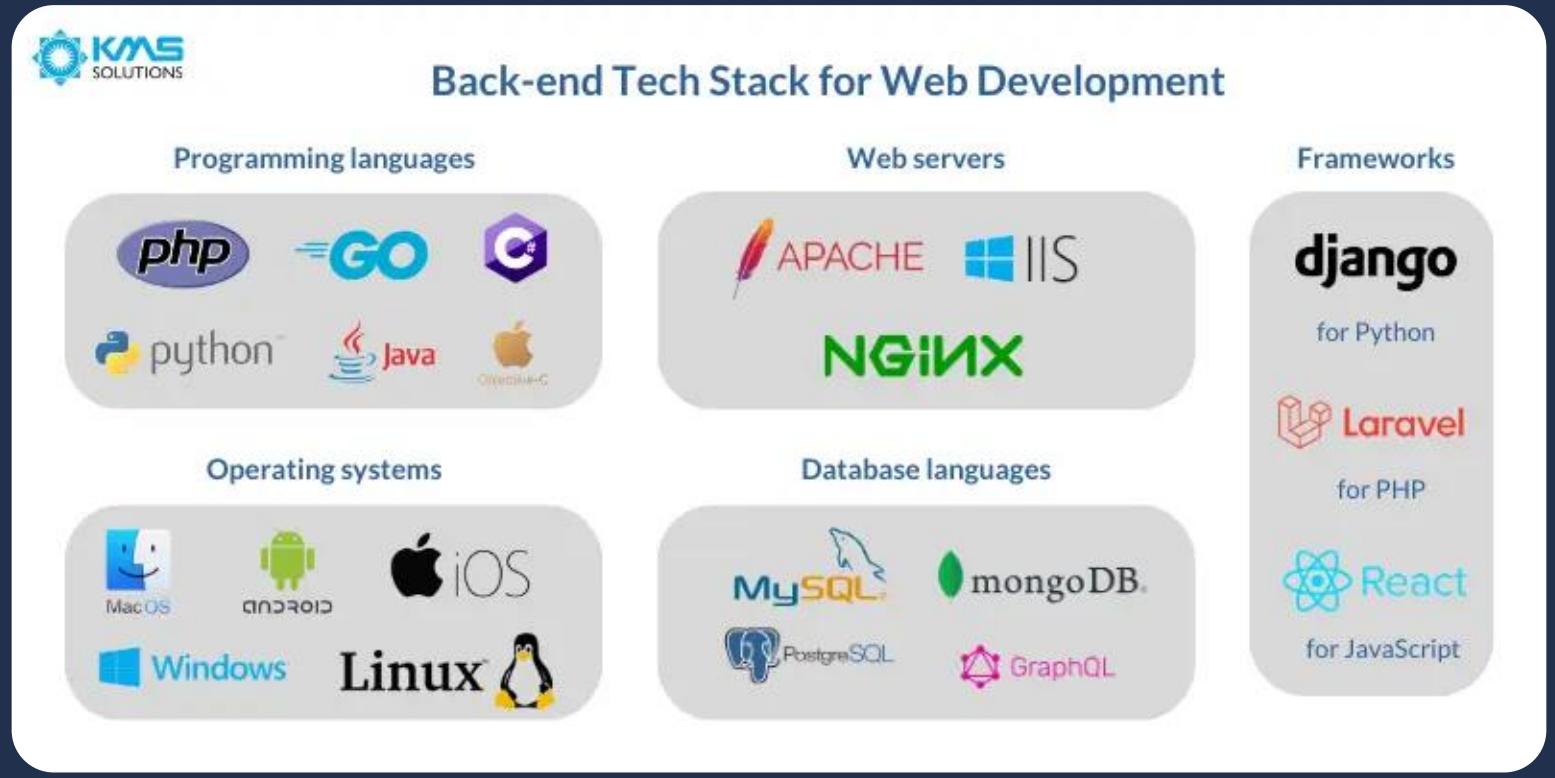
CSE485 – Công nghệ Web

dungkt@tlu.edu.vn



Bài 7. Laravel Framework (Phần 2)

NỘI DUNG

1. Giới thiệu về khuôn mẫu Blade
2. Cú pháp và chỉ thị của Blade
3. Tạo bố cục và kế thừa trong Blade
4. Tạo và sử dụng component
5. Truyền dữ liệu view trong Blade

1. Giới thiệu khuôn mẫu Blade

- **Blade** là hệ thống template của Laravel, được thiết kế để viết mã PHP một cách sạch sẽ và dễ đọc. Blade cung cấp một cách tiếp cận hiện đại để tạo ra các chế độ xem (**views**) dựa trên khuôn mẫu, giúp bạn tách biệt logic ứng dụng và giao diện người dùng.
- Đặc điểm của Blade:
  - Cú pháp rõ ràng và dễ hiểu.
  - Kế thừa layout và tạo components để tái sử dụng giao diện.
  - Hiệu suất cao: Blade được biên dịch thành mã PHP thuần túy và được lưu trữ dưới dạng cache để tăng tốc độ tải.
- Tài liệu gốc chi tiết: Blade Templates - Laravel 10.x - The PHP Framework For Web Artisans

2. Cú pháp và chỉ thị của Blade

- **Blade sử dụng một số cú pháp đặc biệt để thực hiện các hành động khác nhau:**
  - Hiển thị dữ liệu:
    - Cú pháp: {{ $variable }}
    - Mục đích: Hiển thị nội dung của biến. Blade tự động "thoát" (escape) dữ liệu để ngăn chặn các vấn đề bảo mật như XSS (Cross-Site Scripting).
    - Ví dụ:

```
Hello, {{ $name }}!
```

  - Chạy mã PHP:
    - Cú pháp: @php ... @endphp
    - Mục đích: Cho phép bạn thêm mã PHP ngay trong template Blade.
    - Ví dụ:

```
@php
    $counter = 0;
@endphp
```

2. Cú pháp và chỉ thị của Blade

- **Blade sử dụng một số cú pháp đặc biệt để thực hiện các hành động khác nhau:**
  - Cấu trúc điều khiển:
    - Blade cung cấp các chỉ thị điều khiển luồng giống như các cấu trúc điều khiển trong PHP:
    - @if, @elseif, @else

```
@if ($condition)
    // Mã thực thi khi $condition là true
@elseif ($otherCondition)
    // Mã thực thi khi $otherCondition là true
@else
    // Mã thực thi trong các trường hợp còn lại
@endif
```

    - @foreach

```
@foreach ($array as $element)
    // Mã thực thi với mỗi phần tử của $array
@endforeach
```

2. Cú pháp và chỉ thị của Blade

- **Blade sử dụng một số cú pháp đặc biệt để thực hiện các hành động khác nhau:**
    - Cấu trúc điều khiển:
        - Blade cung cấp các chỉ thị điều khiển luồng giống như các cấu trúc điều khiển trong PHP:
        - @switch .. case

```
@switch($variable)
    @case(1)
        // Mã cho trường hợp $variable == 1
        @break

    @case(2)
        // Mã cho trường hợp $variable == 2
        @break

    @default
        // Mã cho các trường hợp còn lại
@endswitch
```

3. Tạo bố cục và kế thừa trong Blade

- **Tạo layout cơ bản**: layouts/app.blade.php được kế thừa bởi pages/home.blade.php

```
<html>
    <head>
        <title>App Name - @yield('title')</title>
    </head>
    <body>
        @section('sidebar')
            This is the master sidebar.
        @show

        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

```
@extends('layouts.app')

@section('title', 'Home Page')

@section('sidebar')
    @parent

    <p>This is appended to the master sidebar.</p>
@endsection

@section('content')
    <p>This is my body content.</p>
@endsection
```

4. Tạo và sử dụng component

- **Tạo component:** php artisan make:component Alert
- Sử dụng trong view

```
<x-alert type="error" message="This is an error alert!"/>
```

5. Truyền dữ liệu cho view

- **Truyền dữ liệu từ Controller sang View:**
  - Sử dụng phương thức với with

```
return view('view-name')->with('variableName', $value);
return view('view-name')->with(['variable1' => $value1, 'variable2'
=> $value2]);
```

  - Sử dụng một phần của phương thức view

```
return view('view-name', ['variableName' => $value]);
```

  - Sử dụng phương thức compact

```
$variable1 = 'value1';
$variable2 = 'value2';
return view('view-name', compact('variable1', 'variable2'));
```

  - Sử dụng phương thức ma thuật

```
return view('view-name')->withVariableName($value);
```

5. Truyền dữ liệu cho view

- **Truyền dữ liệu từ Controller sang View:**
  - Truyền Dữ liệu thông qua View Composers
    - View composers là các callback hoặc class methods được gọi khi một view cụ thể được render. Đây là cách linh hoạt để truyền dữ liệu đến view, đặc biệt khi dữ liệu cần được sử dụng trong nhiều view.

```
View::composer('view-name', function ($view) {
    $view->with('variableName', $value);
});
```

  - Sử dụng Shared Data
    - Bạn có thể chia sẻ dữ liệu với tất cả views thông qua phương thức share của facade View.

```
View::share('variableName', $value);
```

Lựa chọn phương pháp truyền dữ liệu tùy thuộc vào trường hợp sử dụng cụ thể của bạn. Nếu bạn chỉ cần truyền một vài biến đơn giản, with, view hoặc compact là lựa chọn tốt. Đối với dữ liệu cần có sẵn trong nhiều view, việc sử dụng View Composers hoặc Shared Data có thể phù hợp hơn.

“Câu hỏi & Thảo luận”

THE END!