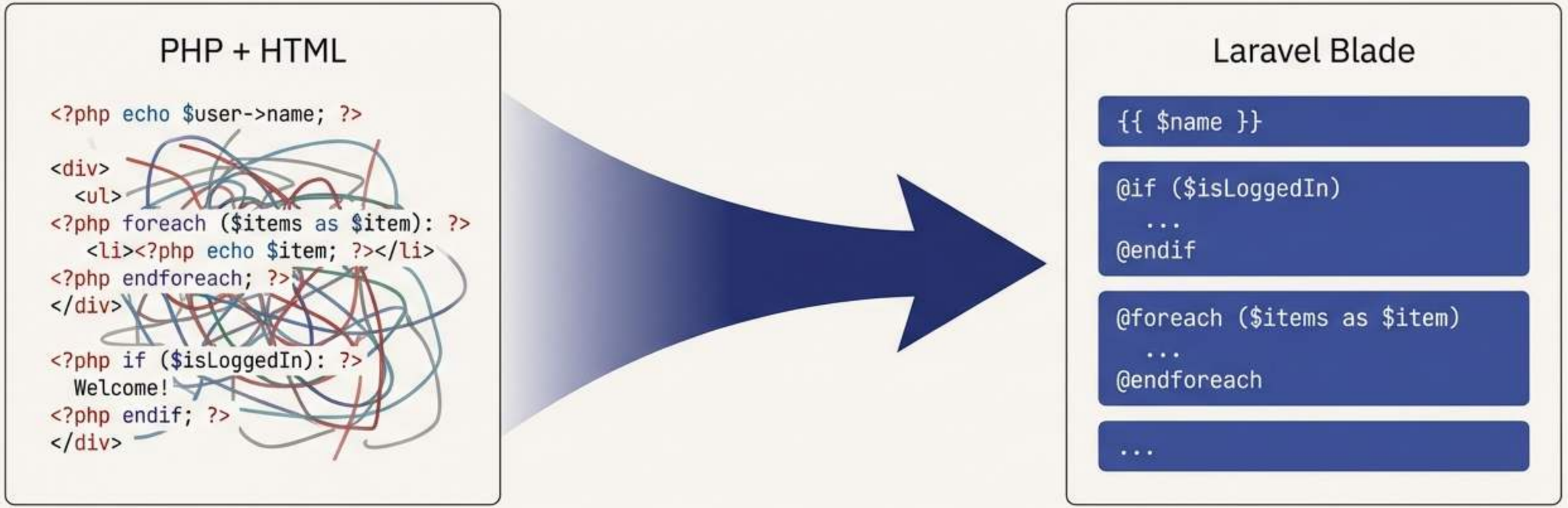
Laravel Blade: الطريق الأسهل والأقوى لبناء واجهات المستخدم

شرح كامل ومبسط من الأساسيات للاحتراف



Blade هو محرك القوالب البسيط والقوي المدمج في Laravel. بدلاً من خلط كود PHP مع HTML بشكل فوضوي، يقدم Blade طريقة نظيفة ومنظمة لعرض البيانات، التحكم في التدفق، وبناء مكونات قابلة لإعادة الاستخدام. هذا الدليل سيأخذك في رحلة لإتقان Blade، خطوة بخطوة.

NotebookLM

عرض البيانات: أساس كل صفحة ويب

المفهوم: الطباعة الآمنة للبيانات

لعرض البيانات التي تم تمريرها إلى الـ View، استخدم الأقواس الأقواس المعقوصة المزدوجة {{ }}.
يقوم Blade تلقائياً بحماية تطبيقك من هجمات XSS عن طريق تمرير كل المخرجات عبر دالة htmlspecialchars الخاصة بـ PHP.

Controller/Route

```php
Route::get('/', function () {
    return view('greeting', ['name' => 'Finn']);
});
```

Blade View

```
مرحبًا، {{ $name }}.
```

المفهوم: عرض البيانات بدون حماية (Unescaped)

في بعض الأحيان، قد تحتاج لعرض HTML كما هو.
استخدم الصيغة {!! !!} بحذر شديد، وفقط مع البيانات التي تثق بها تماماً لتجنب الثغرات الأمنية.

Controller/Route

```php
$nameWithHtml = '<em>Samantha</em>';
```

Blade View

```
مرحبًا، {!! $nameWithHtml !!}.
```

NotebookLM

التوافق مع JavaScript: تجنب تضارب الصيغ

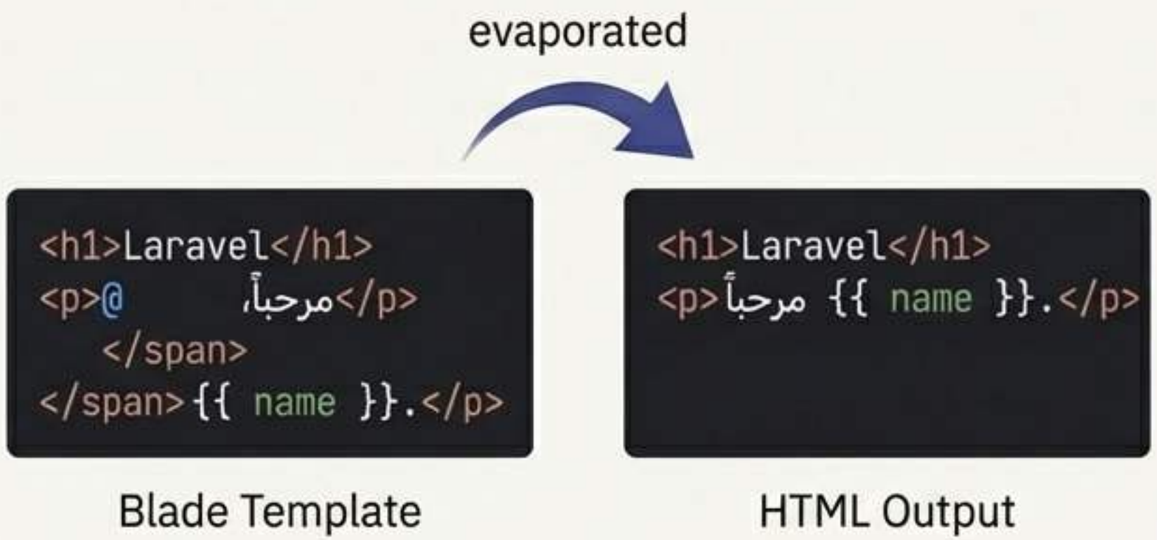
المفهوم: تجاهل كتلة كاملة من الكود

إذا كان لديك جزء كبير من الكود يحتوي على متغيرات JavaScript، يمكنك استخدام الأمر `@verbatim` لتجنب إضافة `@` في كل مرة.

```
@verbatim
    <div class="container">
        Hello, {{ name }}.
    </div>
@endverbatim
```

المفهوم: تجاهل تعبير واحد

العديد من أطر عمل JavaScript تستخدم نفس الأقواس المعقوصة. لمنع Blade من معالجة هذه التعبيرات، استخدم الرمز `@` قبلها.



NotebookLM

التحكم في التدفق: الشروط والقرارات

المفهوم: الشروط الأساسية

يوفر Blade اختصارات نظيفة لكل هياكل التحكم في PHP. استخدم @if , @elseif , @else و @endif للتحكم في عرض المحتوى بناءً على شروط معينة.

```
@if (count($records) === 1)
    لـدي سجل واحد فقط!
@elseif (count($records) > 1)
    لـدي عدة سجلات!
@else
    لـيس لـدي أي سجلات.
@endif
```

المفهوم: اختصارات مفيدة

استخدم @unless (وهي عكس @if)، و @isset و @empty للتحقق من وجود المتغيرات أو ما إذا كانت فارغة'.

```
@unless (Auth::check())
    أنـت غير مسجل الـدخول.
@endunless
```

```
@isset($records)
    // الـمتغير $records مُعرّف ولـيس
@endisset
```

NotebookLM

التعامل مع المجموعات: الحلقات التكرارية

المفهوم: المتغير المساعد $loop

داخل حلقة @foreach، يمكنك استخدام المتغير $loop للحصول على معلومات قيمة حول التكرار الحالي.

الوصف	النوع	الخاصية
هل هذا هو التكرار الأول؟	boolean	$loop->first
هل هذا هو التكرار الأخير؟	boolean	$loop->last
رقم التكرار الحالي (يبدأ من 1).	int	$loop->iteration
العدد الإجمالي للعناصر في المجموعة.	int	$loop->count

المفهوم: التكرار مع التحقق من الفراغ

يوفر Blade أوامر بسيطة لكل هياكل التكرار (@for , @foreach). الأمر @forelse مفيد بشكل خاص لأنه يجمع بين حلقة التكرار والتحقق والتحقق مما إذا كانت المجموعة فارغة في أمر واحد.

```
@forelse ($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>لا يوجد مستخدمين لعرضهم</p>
@endforelse
```

NotebookLM

تنظيم الكود: استدعاء الأجزاء الفرعية (Subviews)

المفهوم: تضمين views

يتيح لك الأمر @include تضمين view من Blade داخل view آخر. كل المتغيرات المتاحة في الـ view الأصلي تكون متاحة في الـ view المُضمَّن.

```
<div>
    @include('shared.errors')

    <form>
        <!-- Form Contents -->
    </form>
</div>
```

تمرير بيانات إضافية:

يمكنك تمرير مصفوفة من البيانات الإضافية إلى الـ view المُضمَّن.

```
@include('view.name', ['status' => 'complete'])
```

ملاحظة هامة

على الرغم من فائدة **@include**، إلا أن '**المكونات**' (Components) توفر وظائف مماثلة مع مزايا أكبر مثل ربط البيانات والخصائص، ومترائص، وهي الطريقة الحديثة الموصى بها والتي سنتناولها الآن.

NotebookLM

نقلة نوعية: عالم مكونات Blade

المكونات (Components) هي الطريقة الحديثة والأكثر قوة لبناء واجهات مستخدم معزولة وقابلة لإعادة الاستخدام في Blade. في ش سر فكر فيها كعناصر HTML مخصصة يمكنك إنشاؤها بنفسك.

لماذا المكونات أفضل؟

Components (الطريقة الحديثة)	(الطريقة القديمة) Includes/Sections
• تعتمد على التكوين (Composition) • تدفق بيانات واضح وصريح (Props) • قابلة لإعادة الاستخدام بشكل كامل • إدارة الخصائص (attributes) أسهل بكثير	تعتمد على الوراثة (Inheritance) تدفق البيانات ضمني وغير واضح أقل مرونة

أنواع المكونات

Class-based: مكون له ملف PHP Class ط ملف وملف view. يوفر منطقاً قوياً ومعالجة للبيانات اللبيانات.

Anonymous: مكون عبارة عن ملف view فقط . مثالي للمكونات البسيطة التي تركز على العرض.

NotebookLM

بناء أول مكون: لنصنع عنصر تنبيه (Alert)

الإنشاء (Create)

استخدم أمر Artisan لإنشاء ملفات المكون.

```
php artisan make:component Alert
```

تحديد البيانات (Define Data)

في `app/View/Components/Alert.php`، عرّف الخصائص التي سيستقبلها المكون في الـ constructor.

```
class Alert extends Component {
    public function __construct(
        public string $type,
        public string $message,
    ) {}
    // ...
}
```

التصميم (Design the View)

في `resources/views/components/alert.blade.php`، استخدم المتغيرات التي قمت بتعريفها.

```
<div class="alert alert-{{ $type }}">
    {{ $message }}
</div>
```

الاستخدام (Use)

استدعِ المكون في أي view. استخدم `:` لتمرير متغيرات PHP.

```
<x-alert type="error" :message="$errorMessage" />
```

NotebookLM

الخصائص الإضافية (Attributes): مرونة لا حدود لها

أي خصائص (attributes) تمررها إلى المكون ولم يتم تعريفها في الـ constructor، يتم تجميعها تلقائياً في متغير خاص يسمى `$attributes`. يمكنك طباعة هذا المتغير لإضافة هذه الخصائص إلى عنصر في الـ view الخاص بالمكون.

المثال الأساسي

الاستخدام (Usage)

```
<x-alert type="error"
    :message="$msg"
    class="mt-4"
    id="my-alert"
/>
```

عرض المكون (Component View)

```
<div {{ $attributes }}>...</div>
```

HTML الناتج (Resulting HTML)

```
<div class="alert alert-error mt-4"
    id="my-alert">...</div>
```

دمج الخصائص (Merging Attributes)

يمكنك دمج الخصائص الافتراضية (مثل الـ classes الأساسية للمكون) مع الخصائص الممررة باستخدام دالة `merge`.

عرض المكون (Component View)

```
<div {{ $attributes->merge(['class' => 'alert alert-' . $type]) }}>
    {{ $message }}
</div>
```

الاستخدام (Usage)

```
<x-alert type="error" :message="$msg" class="mb-4" />
```

HTML الناتج (Resulting HTML)

```
<div class="alert alert-error mb-4">...</div>
```

NotebookLM

Slots: قلب المكونات النابض

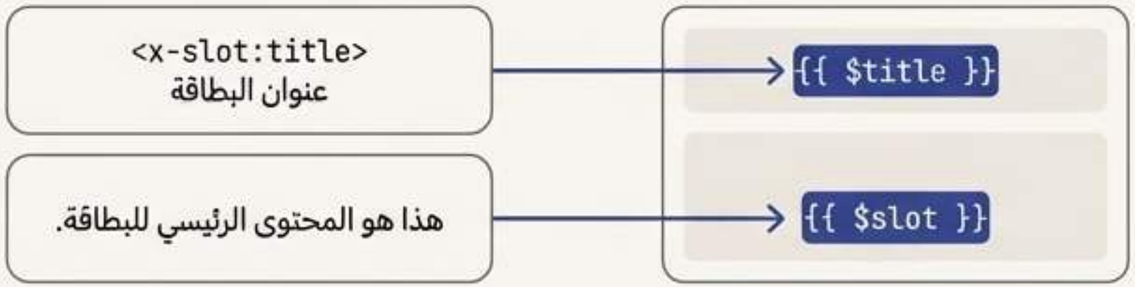
الـ "Slots" هي الطريقة التي تمرر بها محتوى HTML كبير ومعقد إلى المكون الخاص بك، مما يسمح بتركيب المكونات داخل بعضها البعض.

الـ Slots المسماة (Named Slots)

يمكنك تعريف عدة slots في المكون وتمرير المحتوى لكل واحد منها بشكل منفصل باستخدام وسم `<x-slot:name>`.

```
<x-card>
    <x-slot:title>عنوان البطاقة</x-slot>
    هذا هو المحتوى الرئيسي للبطاقة.
</x-card>
```

```
<div class="card">
    <h1 class="title">{{ $title }}</h1>
    <div class="content">{{ $slot }}</div>
</div>
```

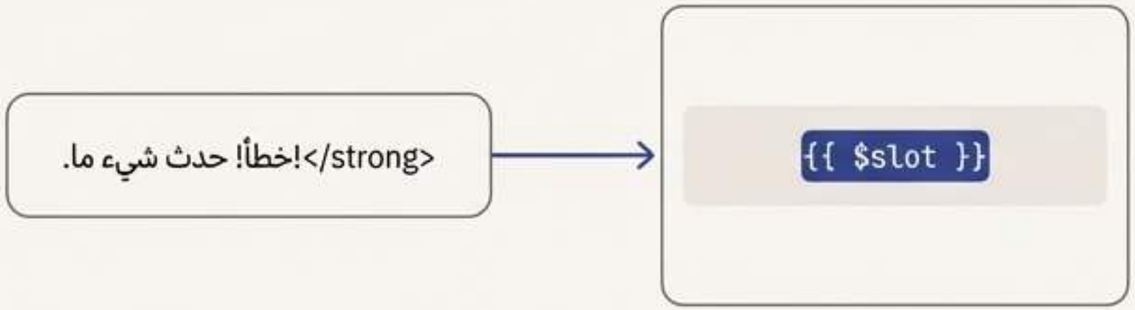


الـ Slot الافتراضي (Default Slot)

أي محتوى تضعه بين وسمي فتح وإغلاق المكون يتم تمريره تلقائياً إلى متغير `$slot`.

```
<x-alert>
    <strong>خطأ!</strong> حدث شيء ما.
</x-alert>
```

```
<div class="alert">{{ $slot }}</div>
```



NotebookLM

المكونات المجهولة (Anonymous): البساطة في أبهى صورها

المفهوم

هي مكونات لا تحتاج إلى ملف Class PHP.
هي مجرد ملف Blade view موجود في `resources/views/components`.
مثالية للمكونات التي تركز على العرض فقط ولا تحتوي على منطق معقد.

كيف تعمل؟

بدلاً من تعريف الخصائص في constructor، استخدم الأمر `@props` في بداية ملف الـ view لتحديد المتغيرات التي يتوقعها المكون. يمكنك أيضاً تحديد قيم افتراضية لها.

الملف: `resources/views/components/button.blade.php`

```
@props(['type' => 'button', 'color' => 'blue'])

<button type="{{ $type }}" class="btn btn-{{ $color }}"
    {{ $slot }}
</button>
```

الاستخدام (نفس طريقة الاستخدام)

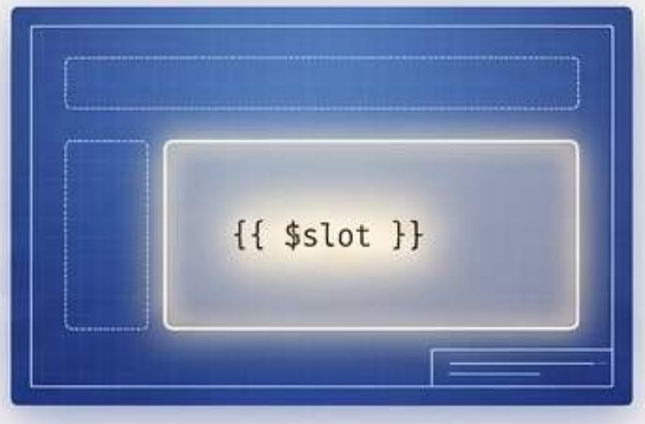
```
<x-button color="red">
    Delete
</x-button>
```

NotebookLM

بناء الصفحات: الطريقة الحديثة باستخدام المكونات

بدلاً من استخدام الوراثة (@extends)، يمكنك تعريف تخطيط صفحتك بالكامل كمكون واحد، ثم "تغليف" محتوى كل صفحة بهذا المكون. هذا الأسلوب أكثر وضوحاً ومرونة.

layout.blade.php



الخطوة 2: استخدام التخطيط في صفحة (posts.blade.php)

```
<x-layout>
    <x-slot:title>
        All Blog Posts
    </x-slot>

    <h1>Blog Posts</h1>
    <!-- Page content goes here -->
</x-layout>
```

الخطوة 1: إنشاء مكون التخطيط

resources/views/components/layout.blade.php

```
<html>
    <head>
        <title>{{ $title ?? 'My App' }}</title>
    </head>
    <body>
        <nav><!-- Navigation --></nav>
        <main>
            {{ $slot }}
        </main>
    </body>
</html>
```

الخطوة 2: استخدام التخطيط في صفحة

resources/views/posts.blade.php

```
<x-layout>
    <x-slot:title>
        All Blog Posts
    </x-slot>

    <h1>Blog Posts</h1>
    <!-- Page content goes here -->
</x-layout>
```

NotebookLM

وراثة القوالب: الطريقة التقليدية (@extends)

قبل المكونات، كانت هذه هي الطريقة الأساسية لبناء التخطيطات. لا تزال مدعومة بالكامل، وستجدها بكثرة في المشاريع القديمة.

كيف تعمل؟

@yield('name'): يحدد "مكاناً فارغاً" أو "مقبساً" في القالب الأب.

@extends('layout.name'): يحدد أن هذا الـ view 'يرث' من قالب آخر.

@section('name') ... @endsection: يملأ المحتوى في 'المكان الفارغ" المحدد بـ **@yield**.

المثال

layouts/app.blade.php

```
<html>
    <head><title>App Name - @yield('title')</title></head>
    <body>
        @yield('content')
    </body>
</html>
```

child.blade.php

```
@extends('layouts.app')
@section('title', 'Page Title')
@section('content')
    <p>This is my body content.</p>
@endsection
```

NotebookLM

أوامر مساعدة: النماذج (Forms) والمكدسات (Stacks)

أوامر النماذج (Form Directives)

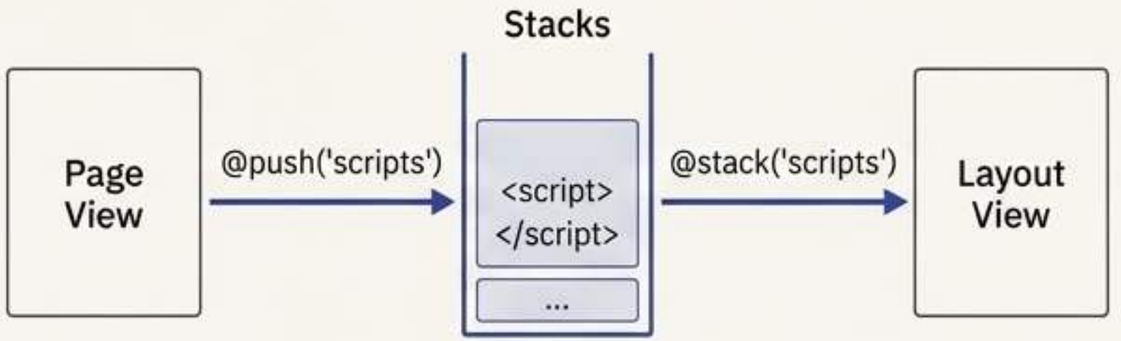
يوفر Blade أوامر لحماية نماذجك وتسهيل التعامل معها.

- **@csrf**
  يضيف حقل token مخفي للحماية من هجمات CSRF.
- **@method('PUT')**
  يستخدم لتزييف أفعال HTTP التي لا تدعمها نماذج HTML.
- **@error('field')**
  يتحقق من وجود أخطاء validation لحقل معين ويعرضها.

```
<form method="POST" action="/profile">
    @csrf
    @method('PUT')
    <input class="@error('title') is-invalid @enderror">
    @error('title')
        <div class="alert alert-danger">{{ $message }}</div>
    @enderror
</form>
```

المكدسات (Stacks)

تسمح لك بإضافة (push) محتوى إلى 'مكدس' مسمى من أي مكان، ثم عرضه في مكان آخر (عادة في ال. layout). مثالية لإدارة مثالية لإدارة ملفات JS/CSS الخاصة بكل صفحة.



Page View Code:

```
@push('scripts')
   <script src="/page-specific.js">
</script>
@endpush
```

Layout View Code:

```
<head>
    ...
    @stack('scripts')
</head>
```

NotebookLM

توسيع قدرات Blade: أوامر مخصصة

Blade قابل للتوسيع بشكل كامل. يمكنك إنشاء أوامر (directives) وشروط مخصصة لتناسب احتياجات تطبيقك وتبسيط الكود. يتم تعريف هذه الأوامر عادة في `AppServiceProvider`.

إنشاء أمر مخصص @datetime

ServiceProvider Code

```
Blade::directive('datetime', function (string $expression) {
    return "<?php echo ($expression)->format('m/d/Y H:i'); ?>";
});
```

Usage Code

```
@datetime($post->created_at)
```

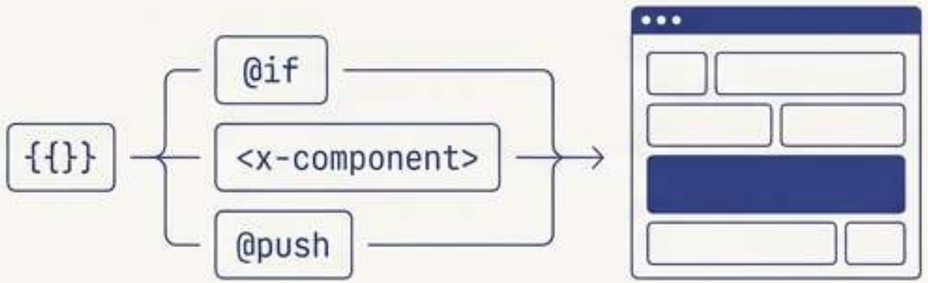
إنشاء شرط مخصص @disk

ServiceProvider Code

```
Blade::if('disk', function (string $value) {
    return config('filesystems.default') === $value;
});
```

Usage Code

```
@disk('local')
    <!-- The application is using the local disk... -->
@enddisk
```



الخاتمة

الآن لديك كل الأدوات لبناء واجهات مستخدم نظيفة وقوية وقابلة للتطوير باستخدام Laravel Blade. من عرض البيانات البسيط إلى بناء أنظمة تصميم كاملة بالمكونات، Blade هو شريكك في رحلة التطوير.

NotebookLM