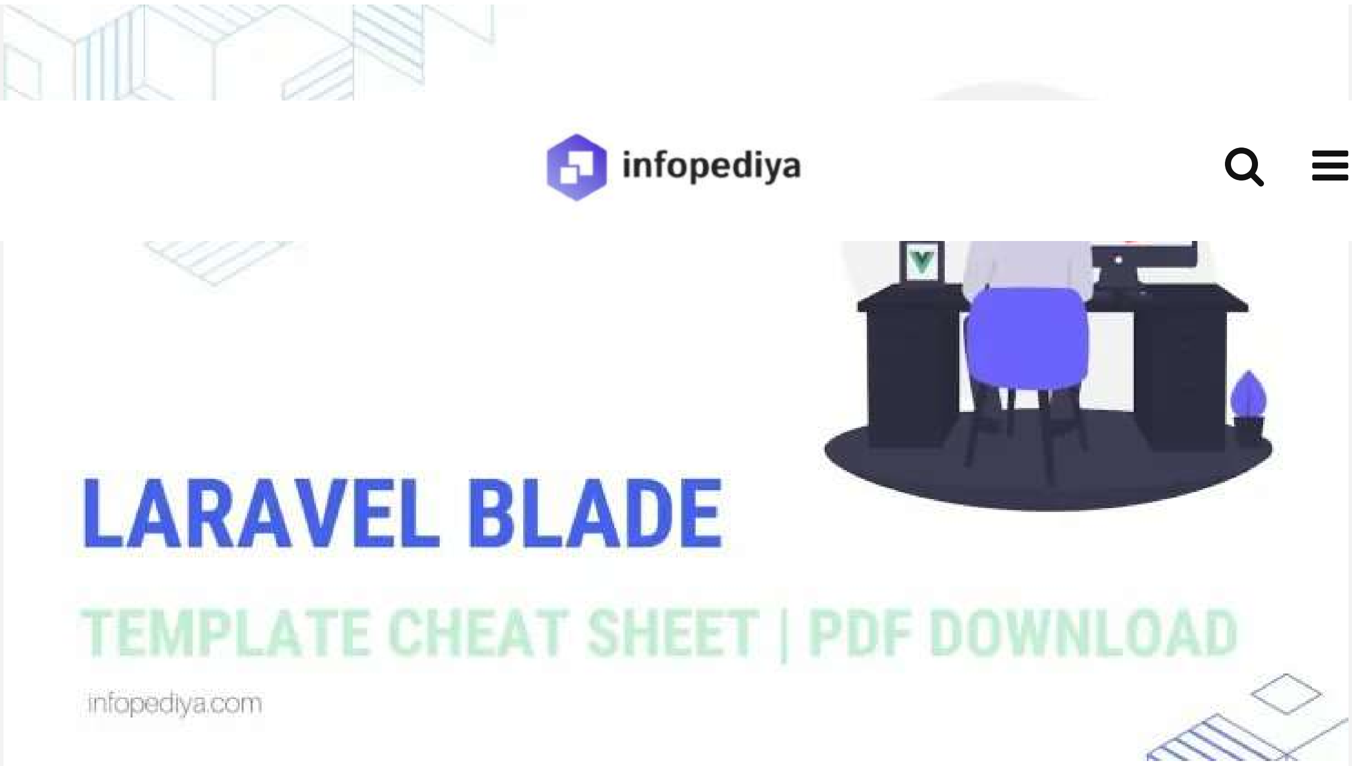

CHEAT SHEETS PHP

Laravel 9 Blade Template cheat sheet | PDF Download

3 seconds ago 4 min read

Laravel has a simple but powerful templating engine called Blade. You can read or download the Laravel 9 blade template cheat sheet for free in this article.

Unlike several PHP templating engines, Blade allows you to use plain PHP code in your templates. Blade template files are generally located in the **resources/views** directory and have the file extension `.blade.php`

Read More:

- Git Commands cheat sheet PDF download.
- Get geolocation in PHP? (country, zip, latitude, longitude, etc.)
- Best PHP frameworks for web development

Table of Contents

Sending Data to Blade Views

There are several ways to send data to views:

```
return view('view-file', ['variable-name' => 'value']);
```

```
return view('view-file')
            ->with('variable1', 'value')
            ->with('variable2', 'value');
```

You can pass an array of variables using the PHP `compact()` method

```
return view('view-file', compact(['var1','var2',....'varN']));
```

Displaying Data

The contents of the variables can be displayed as follows:

```
{{ $variable }}.
```

Any PHP function's output can also be displayed. In fact, you may include any PHP code you want within a Blade echo statement:

```
{{ time() }}.
```

Blade and JavaScript Frameworks syntax conflict

Some JavaScript frameworks also use "curly" braces to display data, you may use the @ symbol to tell the Blade rendering engine that an expression should be ignored. As an example:

```
<h1>Laravel</h1>

Hello, @{{ name }}.
```

Blade directives can also be escaped using the @ symbol:

```
{{-- Blade template --}}
@@if()

<!-- HTML output -->
@if()
```

If you're displaying many JavaScript variables in your template, you can use the

`@verbatim` directive to avoid having to preface each Blade `echo` statement with a @ symbol:

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

Rendering JSON

The most recent versions of the Laravel application offer a Js facade, which allows you to easily rendering it as JSON in order to initialize a JavaScript variable:

```
<script>
    var app = {{ Js::from($array) }};
</script>
```

Syntax of If Statements

```
@if (count($user) === 1)
    single user
@elseif (count($user) > 1)
    many users!
@else
    no user found!
@endif
```

Conditional statement – unless

```
@unless (Auth::check())
    You are not signed in.
@endunless
```

Conditional statements – isset and empty

```
@isset($records)
    // $records is defined and is not null...
@endisset

@empty($records)
    // $records is "empty"...
@endempty
```

Authentication Directives

```
@auth
    // The user is authenticated...
@endauth

@guest
    // The user is not authenticated...
@endguest
```

You can provide the authentication guard that should be verified if necessary:

```
@auth('admin')
```

```
    // The user is authenticated...
@endauth

@guest('admin')
    // The user is not authenticated...
@endguest
```

Environment Directives

```
@production
    // application is running in a production
@endproduction
```

```
@env('staging')
    // application is running in staging
@endenv

@env(['staging', 'production'])
    // application is running in staging or production
@endenv
```

Switch statement

```
@switch($i)
    @case(1)
        First case...
        @break

    @case(2)
        Second case...
        @break
    @default
        Default case...
@endswitch
```

Loops in blade

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach

@forelse ($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>No users</p>
@endforelse

@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

The Loop Variable

```
@foreach ($users as $user)
    @if ($loop->first)
        This is the first iteration.
    @endif

    @if ($loop->last)
        This is the last iteration.
    @endif
    <p>This is user {{ $user->id }}</p>
@endforeach
```

Conditional Classes

```
@php
    $isActive = false;
    $hasError = true;
@endphp

<span @class([
    'p-4',
    'font-bold' => $isActive,
    'text-gray-500' => ! $isActive,
    'bg-red' => $hasError,
])></span>

<span class="p-4 text-gray-500 bg-red"></span>
```

Including Subviews

```
@include('shared.errors')
```

you can pass an array of data

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

If you want to include a view that might or might not be present, use the

`@includeIf` directive:

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

If you want to `@include` a view based on whether a specified boolean expression evaluates to true or false, you can use the `@includeWhen` and `@includeIf`

```
@includeWhen($boolean, 'view.name', ['status' => 'complete'])

@includeUnless($boolean, 'view.name', ['status' => 'complete'])
```

You may use the `@includeFirst` directive to include the first view in a supplied array of views:

```
@includeFirst(['custom.admin', 'admin'], ['status' => 'complete'])
```

The `@each` directive

```
@each('view.name', $jobs, 'job')
```

```
@each('view.name', $jobs, 'job', 'view.empty')
```

The `@once` Directive

The `@once` directive specifies a section of the template that will be evaluated only once per rendering cycle.

```
@once
// This code will render once
@endonce
```

This is suitable to push some javascript code:

```
@once
    @push('scripts')
        <script>
            // Your JavaScript code
        </script>
    @endpush
@endonce
```

You can also use `@pushOnce`

```
@pushOnce('scripts')
    <script>
        // Your custom JavaScript...
    </script>
@endPushOnce
```

Writing Raw PHP

```
@php
    $var = "some value";
@endphp
```

Comments in Blade Template

```
{{-- This is an example of a comment in Blade template --}}
```

Conclusion

Laravel 5.1 introduces the concept of using Blade, a templating engine. Blade is distinct from other templating engines in the following ways:

- It does not preclude the developer from writing simple PHP code in views.
- The blade views created in this manner are compiled and cached until they are updated.

Best PHP frameworks for web development

You may also like

BACKEND DEVELOPMENT PHP

Best PHP frameworks for web development

we will give you a quick review of the top five to help you pick the best PHP frameworks for web development for your PHP based project

5 min read