# Laravel - Componentes

Componentes archivos blade y componentes como clase (POO)

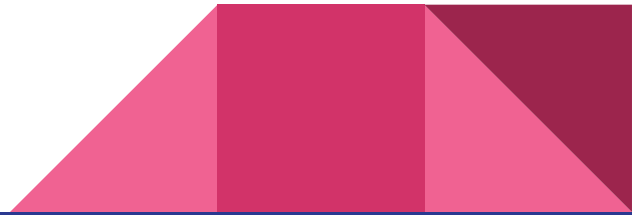
# Uso de componentes de Blade

✔ Cuando un **bloque de código se repite** en varias vistas.

✔ Cuando quieres hacer que el código sea más limpio y **modular**.

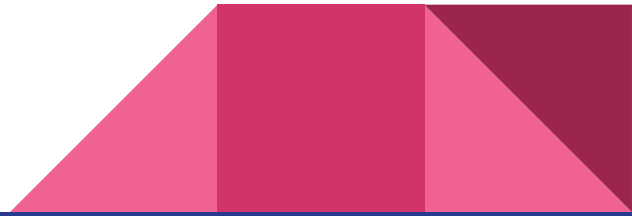
✔ Cuando necesitas **pasar variables a fragmentos HTML** sin escribir lógica en las vistas.

Ofrecen un beneficio similar a *@include()* de vistas parciales, pero con una estructura diferente.



# Tipos de Componentes

- Componentes **Basados en Archivos Blade (Anónimos)**
  - Un único fichero. Ej: resources/views/components
- Componentes **Basados en Clases** PHP
  - Clase del componente: app/View/Components/
  - Vista del componente: resources/views/components/



# Componentes Basados en Archivos Blade (Anónimos)

Son los componentes ya vistos: se denominan “anónimos” dado que consisten únicamente de una plantilla de Blade, sin clase asociada.

Para generar un componente anonimo, se puede emplear el comando:
***php artisan make:component [nombre_comp] --view***

Esto genera un fichero en **resources/views/components** denominado ***[nombre_comp].blade.php***

Este componente se puede mostrar en una vista con el tag:
***<x-[nombre_comp] />***

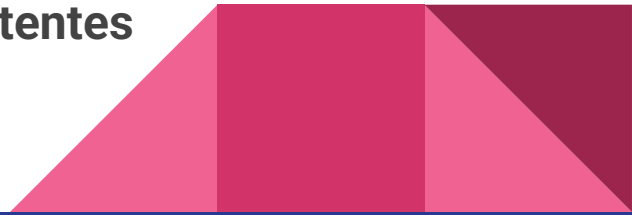
# Componentes Basados en Clases PHP

Se pueden generar con el comando: ***php artisan make:component [nombre_comp]***

Este comando genera dos ficheros:

- Una definición de clase para el componente, en la ruta: **app/View/Components/** denominado ***[nombre_comp]Component.php*** en formato camel-case
- Una plantilla de vista para el componente, en la ruta: r**esources/views/components/** denominado ***[nombre_comp].blade.php*** en formato kebab-case

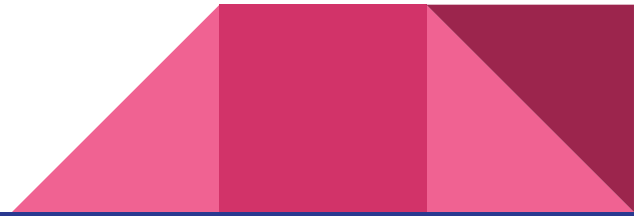
**Laravel detecta automáticamente los componentes existentes en estas carpetas.**



# Renderizar componentes

Como hemos visto, dentro de una plantilla de blade ( .blade.php ) se puede renderizar un componente mediante el tag “x” y su nombre en kebab-case.

Si se trata de un componente con clase PHP, es posible definir un método condicional para mostrarlo o no: shouldRender()



# Renderizar componente - método shouldRender()

```
use Illuminate\Support\Str;

public class [nombre_comp] extends Component{

    public $message;

    // Sí return es falso no muestra componente, si es verdadero, sí
    public function shouldRender(): bool{
        return Str::length($this->message) > 0;
    }

}
```

# Indexar componentes

Es posible agrupar componentes: cuando existen múltiples componentes dentro de una subcarpeta, y uno de estos tiene el mismo nombre que la subcarpeta, Blade asocia dicho componente como la raíz de la agrupación.

Por ejemplo, existen los componentes:

- App\Views\Components\Card\Card
- App\Views\Components\Card\Header
- App\Views\Components\Card\Body

**No** se requiere llamar a Card como <x-**card.card**>

# Indexar componentes (continuación)

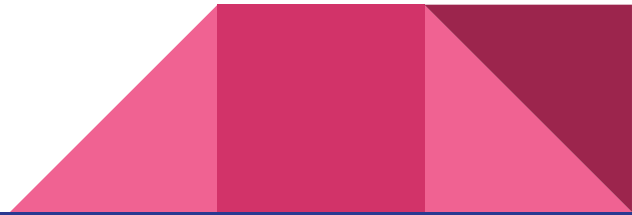
Para llamar a los componentes agrupados del ejemplo, bastaría con:

<x-**card**>

    <x-card.**header**>...</x-card.header>

    <x-card.**body**>...</x-card.body>

</x-card>

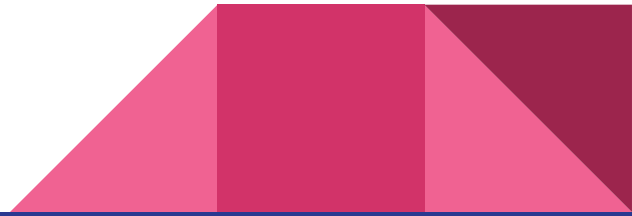


# Parsear datos a los componentes desde la vista

Es posible **pasar información de la vista a los componentes** al renderizarlos, desde el tag “x”:

- Pasar valores vía HTML (hardcodeado):
  <x-mi-componente **mensaje=”Un mensaje”** />

- Pasar valores vía PHP (variables) empleando : antes de declarar el dato:
  <x-mi-componente **:**mensaje=”**$message**” />



# Parsear datos a los componentes desde la vista

Recordatorio; los nombres de los atributos en el componente de clase estarán en camelCase, mientras que al invocarlos desde la vista se debe hacer empleando kebab-case.

**COMPONENTE CLASE**

```
public function __construct(
    public string $alertType
) {}
```

**VISTA RENDERIZA COMPONENTE**

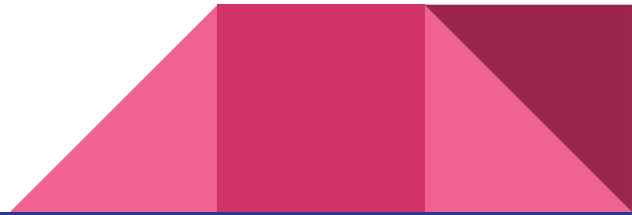
```
<x-alert alert-type="danger" />
```

# Parsear datos a los componentes desde la vista

Si los nombres de atributos a pasar coinciden en nombre con los valores en sus variables de PHP, se puede usar el formato corto.

{{-- La siguiente llamada a componente --}}
<x-profile **:$userId :$name** />

{{-- es equivalente a esta: --}}
<x-profile **:user-id="$userId" :name="$name"** />



# Parsear datos a los componentes clase

Si se usa un componente con clase de PHP, es recomendable definir como atributos todos los datos a parsear, e incluirlos en __construct().

De esta forma, **estarán disponibles en la vista del componente** automáticamente, y podrán ser invocados por su nombre de variable. Ojo, no confundir con vista que llama al componente.

**COMPONENTE CLASE**

```
public function __construct(
        public string $type,
        public string $message
) {}
```

**VISTA COMPONENTE (plantilla blade)**

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

# Llamar a métodos de los Componentes

Así como se puede trabajar con atributos públicos de una clase componente, **también es posible llamar sus métodos** desde la vista del componente.

En un componente de clase tenemos public $selected; y el siguiente método:

```
public function isSelected(string $option): bool {
    return $option === $this->selected;      // devuelve booleano
}
```

Desde la vista del componente, se puede **usar el método referenciando a su nombre como si fuera una variable**:

```
<option {{ $isSelected($value) ? 'selected' : '' }} value="{{ $value }}">
    {{ $label }}    {{-- Se marcará como seleccionado según valor de $value --}}
</option>
```

# Slots: Ranuras en Componentes - $slot

Si se requiere pasar contenido adicional a los componentes, se emplean “slots” o . Estas “ranuras” se muestran al incluir la variable **$slot**. Todo lo que se introduzca dentro del tag x se añadirá al contenido de la variable $slot.

Si tenemos un **componente** anónimo **alert**.blade.php con:

```
<div>
        {{ $slot }}
</div>
```

Si una **vista** de blade llama dicho componente tal que:

```
<x-alert>                                                    <!-- antes: <x-alert /> -->
        <strong>Whoops!</strong> ¡Se ha producido un fallo!
</x-alert>
```

El **resultado** final será:

```
<div>
        <strong>Whoops!</strong> ¡Se ha producido un fallo!
</div>
```

# Añadir ranuras a Componentes

Cuando se ha declarado la ranura $slot es posible añadir nuevas ranuras declarándose como nuevas variables.

En alert.blade.php:

```
<span>{{ $title }}</span>
 <div>
        {{ $slot }}
</div>
```

En este formato, $title se debe declarar obligatoriamente.

# Añadir ranuras a Componentes

En la vista que llama al componente alert:

```
<x-alert>
    <x-slot:title>
    Server Error
    </x-slot>

  <strong>Whoops!</strong> Something went wrong!

</x-alert>
```

Esto equivale a dar valor “Server Error” a $title.

# Añadir ranuras a Componentes

Resultado:

***Server Error***
* *
*Whoops! ¡Se ha producido un fallo!*
* *

# Añadir ranuras a Componentes

Para hacer el uso de la ranura $title **opcional**:

En alert.blade.php:

```
@isset($title)
        <span>{{ $title }}</span>
@endisset
 <div>
        {{ $slot }}
</div>
```