Dot Notation

Per richiamare **views** e altri **path**, Laravel utilizza nelle sue funzioni la **Dot Notation**.

Quindi se la nostra views è contenuta in
resources/views/home/index.blade.php

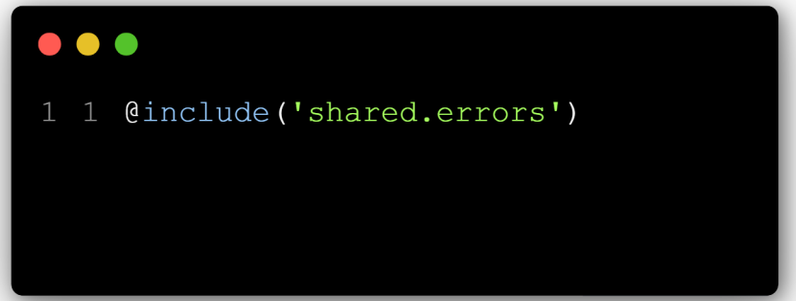
Scriveremo:
view('home.index')

Omettendo l'estensione (come facevamo in SASS), Laravel cercherà da solo i file con estensione blade.php

Include

Se vogliamo includere una porzione separata di html possiamo usare il metodo include di Blade.

Pensiamo ad esempio se avessimo una porzione di codice che **utilizzeremo in diverse pagine**, potremmo creare l'html una sola volta ed inserirlo dentro una view e poi includerla qualora ci servisse.



Creiamo un layout

Una delle feature fondamentali di **Blade** è la possibilità di creare dei **layout** che possiamo estendere all'interno di tutte le nostre pagine, per non dover copiare e incollare il nostro codice ogni qualvolta dobbiamo creare una nuova pagina.

Pensiamo ad esempio la dichiarazione dell'html, o i nostri file css, o jquery ecc.
In ogni pagina dovremmo copiare e incollare html, head, meta, style e script.

Non sarebbe comodo creare un bel timbro e riutilizzare sempre quello?

Creiamo un layout

Abbiamo il nostro layout con style, title, script e la struttura base dell'html.

Il metodo **@yield()** ci permette di mettere un **segnaposto** in cui andremo ad inserire il nostro codice per ogni nostra pagina.

```
1  <html>
2      <head>
3          <title>MyTitle</title>
4          // css bootstrap
5          // css style.css
6      </head>
7      <body>
8          <div class="container">
9              @yield('content')
10         </div>
11         // script vue
12         // script main.js
13     </body>
14 </html>
```

Creiamo un layout

Per estendere quindi per tutte le nostre pagine quel layout

```
1  1  @extends('layouts.app') // estende il layout
   2
   3  @section('content') // aggiunge un <p> all'interno del segnaposto @yield('content')
   4     <p>This is my body content.</p>
   5  @endsection
```

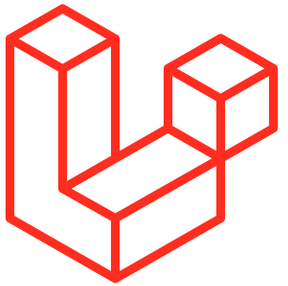
LIVE CODING

Creiamo un layout base e estendiamolo nella nostra home

Compilazione Assets con Vite

Come possiamo organizzare al meglio lo stile della nostra web-app?

Dalla versione 9.x Laravel utilizza Vite per la compilazione degli assets.
Inoltre, utilizza PostCSS al posto di SASS e Tailwind al posto di Bootstrap.



Questo non significa che siamo obbligati ad utilizzare questi tool, anzi!
Possiamo configurare Laravel e Vite per utilizzare gli strumenti a noi piú congeniali e familiari:

- Bootstrap
- SASS

Docs Laravel+Vite: https://laravel.com/docs/9.x/vite#loading-your-scripts-and-styles

Docs Bootstrap+Vite: https://getbootstrap.com/docs/5.2/getting-started/vite/

Compilazione Assets con Vite

Utilizziamo gli strumenti a noi piú familiari - Sass #1

A progetto appena creato eseguiamo questi passaggi:

1. rimuoviamo PostCSS
2. installiamo tutti i pacchetti di npm
3. installiamo SASS
4. modifichiamo files e cartelle
5. modifichiamo file vite.config.js

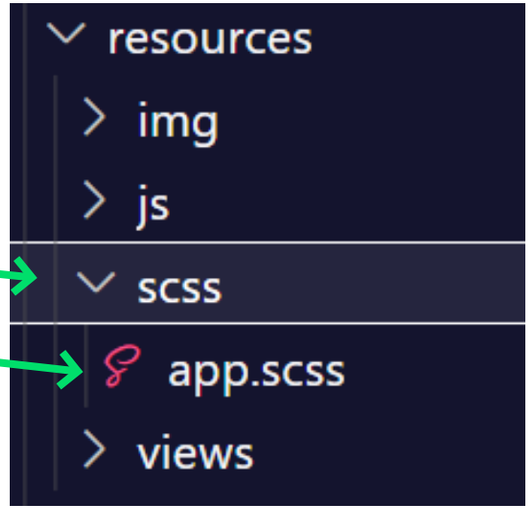
```
1 1 npm remove postcss // 1. Rimuoviamo PostCss
  2 npm i // 2. Istalliamo tutti i pacchetti di npm
  3 npm i --save-dev sass // 3. istalliamo sass
```

Compilazione Assets con Vite

Utilizziamo gli strumenti a noi piú familiari - Sass #2

Rinominiamo la cartella ~~css~~ in **scss**

ed il file ~~app.css~~ in **app.scss**



Compilazione Assets con Vite

Utilizziamo gli strumenti a noi piú familiari - Sass #3

Aggiorniamo il file **vite.config.js**

- modifichiamo il percorso del css
- aggiungiamo un alias

```
1   1 // vite.config.js
    2 export default defineConfig({
    3     plugins: [
    4         laravel({
    5             input: [
    6              // Modifichiamo il percorso del css usando sass
    7             'resources/scss/app.scss',
    8             'resources/js/app.js'
    9             ],
   10             refresh: true,
   11         }),
   12     ],
   13     //Aggiungiamo un alias per la cartella /resources/
   14     resolve: {
   15         alias: {
   16             '~resources': '/resources/'
   17         }
   18     },
   19 });
```

Compilazione Assets con Vite

Importiamo gli assets scss

Usiamo l'alias creato nel file vite.config.js per importare tramite JavaScript il nostro scss

```
1  1  // resources/js/app.js
   2
   3  // Import our custom CSS
   4  import '~resources/scss/app.scss'
```

Per includere nell'head del nostro HTML gli assets compilati da Vite, usiamo la direttiva **@vite()** alla quale possiamo passare una stringa o un array, che indica la posizione dei nostri assets.

Dato che abbiamo importato il nostro stile direttamente nel file app.js, ci basterà includere quest'ultimo e anche il css verrà processato automaticamente.

```
1  1  <head>
   2      <!-- Includiamo gli assets con la direttiva @vite -->
   3      @vite('resources/js/app.js')
   4  </head>
```

Compilazione Assets Vite/Blade

Istruiamo Vite e Blade affinchè processino correttamente i nostri assets

Quando costruiamo un layout abbiamo bisogno spesso di fare due cose: usare un'immagine nel markup o nel foglio di stile

```
1  1  // resources/js/app.js
   2
   3  import.meta.glob([
   4      '../img/**'
   5  ])
```

Per farlo dobbiamo dire a Vite di processare i percorsi affinchè processi tutte le immagini presenti nella cartella **resources/img/**

https://laravel.com/docs/9.x/vite#url-processing

Compilazione Assets Vite/Blade

Istruiamo Vite e Blade affinchè processino correttamente i nostri assets

Affinche blade processi i nostri assets statici usiamo il metodo asset di Vite **{{Vite::asset()}}**

```
1 1 <img src="{{ Vite::asset('resources/img/logo.png') }}" alt="">
```

Per far riferimento agli assets dal foglio di stile, usiamo semplicemente i percorsi relativi e Vite farà il resto.

```
1 1 // app.scss
  2 body {
  3     background-image: url('../img/page_bg.jpg');
  4     background-size: cover;
  5 }
```

Questi assets verranno poi versionati quando faremo la build con il comando `npm run build`

https://laravel.com/docs/9.x/vite#blade-processing-static-assets

Compilazione Assets Vite/Blade

E per far funzionare Bootstrap?

Installiamo **bootstrap** con npm

```
1 1 npm i --save bootstrap @popperjs/core
```

Aggiungiamo la costante **path** che verrà usata per creare degli alias alle cartelle di sistema

```
1 1 //file: vite.config.js
  2 const path = require('path')
```

Aggiungiamo un'**alias** `~bootstrap` nell'oggetto alias che punti alla cartella `node_modules/bootstrap`

```
1 1 // vite.config.js
  2
  3 //..
  4 alias {
  5   '~bootstrap': path.resolve(__dirname, 'node_modules/bootstrap'),
  6 }
```

Compilazione Assets Vite/Blade

E per far funzionare Bootstrap?

Nel file `/resources/scss/app.scss` **importiamo il css di bootstrap** usando la direttiva `@import` di sass con l'alias `~bootstrap` creato precedentemente nel file `vite.config.js`

Infine nel file `resources/js/app.js`, **importiamo Bootstrap** subito dopo la dichiarazione di import del nostro custom scss.

```
1  1  /*resources/scss/app.scss*/
   2  @import "~bootstrap/scss/bootstrap";
   3
   4  body {
   5      background-color: red;
   6  }
```

```
1  1  // resoures/js/app.js
   2
   3  /* Import Bootstrap 5 */
   4
   5  // Import our custom CSS
   6  import '~resources/scss/app.scss'
   7
   8  // Import all of Bootstrap's JS
   9  import * as bootstrap from 'bootstrap'
```

A proposito di Sass

Come possiamo organizzare al meglio lo stile della nostra web-app?

Conosciamo già la funzione **@import** di Sass per includere altri file.

Usiamola nel file che abbiamo creato **app.scss**.

Useremo questo file come **file padre** che incorpora tanti **file figli**, i quali corrispondono a piccole **porzioni del nostro sito**.

Questi file vengono detti **partials** e si distinguono nel nome per il simbolo _ (*underscore*) all'inizio del nome del file.