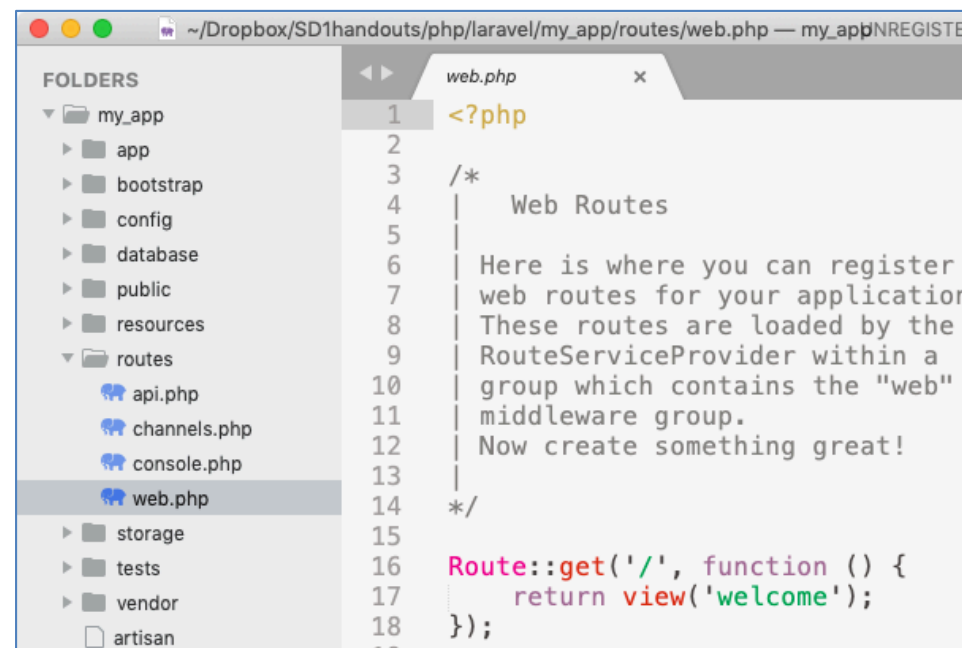


# Struttura di un app Laravel: *route*

- Partiamo da *routes/web.php*, il componente Laravel di base (router)
- Vi troviamo clausole del tipo `Route::get('/path/...', callback)` che specifica che:
  - se dal web arriva alla app una richiesta HTTP *GET* */path/...* ("*/path/...*" è ciò che, nella URL del browser che genera la richiesta, segue il nome del server)
  - la app reagisce invocando *callback*
- *callback* è una funzione, che restituisce testo (tipicamente HTML), che verrà inviato come *risposta* al browser da cui proviene la *richiesta*
- nel *web.php* qui sopra, *callback* è una *function* anonima detta *closure*



The screenshot shows a file explorer on the left with the following structure:

- my\_app
  - app
  - bootstrap
  - config
  - database
  - public
  - resources
    - api.php
    - channels.php
    - console.php
    - web.php
  - storage
  - tests
  - vendor
  - artisan

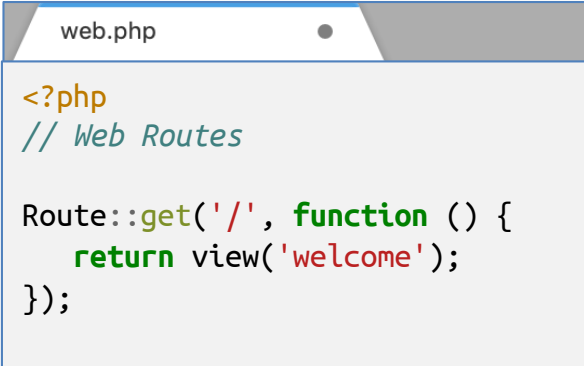
The main editor window shows the content of *web.php*:

```
1 <?php
2
3 /*
4  | Web Routes
5  |
6  | Here is where you can register
7  | web routes for your application
8  | These routes are loaded by the
9  | RouteServiceProvider within a
10 | group which contains the "web"
11 | middleware group.
12 | Now create something great!
13 |
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });
```

Una **route** definisce quindi la risposta dell'app a una data richiesta HTTP, ossia instrada ("*ruota*") la richiesta in arrivo verso il codice che genera la risposta

# Callback e view

- Qui a destra un *web.php* più conciso, ma equivalente al precedente (che era generato dal tool *laravel new*):
- Il callback restituisce il valore prodotto, a sua volta, dalla chiamata di funzione *view('welcome')*
- Questa funzione preleva una pagina HTML, detta appunto *view*, dal file di nome '*welcome.blade.html*', e la restituisce dopo eventuale elaborazione dell'engine *Blade* (Di Blade si dirà di nel seguito).

A screenshot of a code editor window titled 'web.php'. The code is as follows:

```
<?php
// Web Routes

Route::get('/', function () {
    return view('welcome');
});
```

# Callback: opzioni

- Qui a destra un *web.php* semplificato: il callback restituisce direttamente testo HTML anziché farlo attraverso la funzione *view()* e un file/view
- Si può restituire anche altro tipo di testo, es. JSON
  - HTML si addice a una Web app fruita via browser
  - JSON si addice a una API REST fruita da un cliente "programmatico" (es. javascript)
- Per facilitare questa seconda modalità, i callback di Laravel restituiscono i dati PHP, come gli array convertendoli automaticamente in notazione JSON!
- Infine, vedremo che più callback (per *route* omogenee) possono accorparsi in un componente, detto *controller*, che ha i callback come metodi
- Maggiori dettagli verranno forniti più avanti!

The screenshot shows a web browser window with the address bar at 'localhost:8000'. The page content displays 'Hello!' in a large, bold, black serif font. Above the browser window, a code editor shows the following PHP code for 'web.php':

```
<?php
// Web Routes

Route::get('/', function () {
    // return view('welcome');
    return('<H1>Hello!</H1>');
});
```

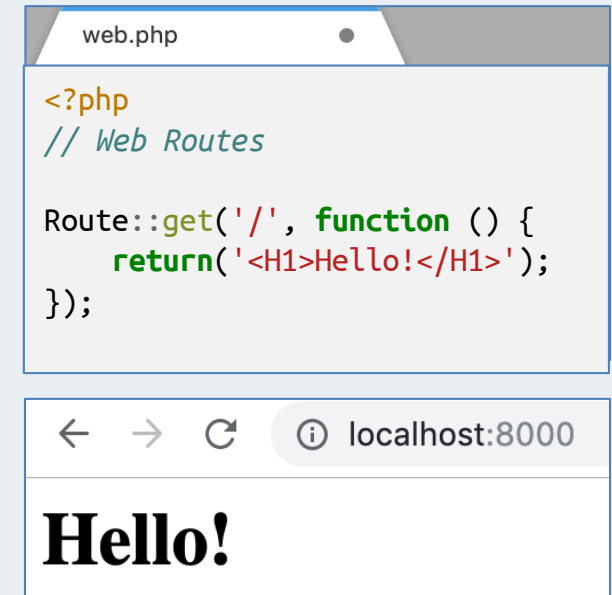
The screenshot shows a web browser window with the address bar at 'localhost:8000'. The page content displays a JSON object: { "nome": "Gigi", "cognome": "Riva" }. Above the browser window, a code editor shows the following PHP code for 'web.php':

```
<?php
// Web Routes

Route::get('/', function () {
    return([ 'nome'=>'Gigi',
             'cognome'=>'Riva' ]);
});
```

# Le route: aspetti tecnici

- Struttura e forma del router pongono alcune questioni tecniche interessanti su Laravel
- Ne accenniamo tre, anche se le risposte per (1) e (2) sono al di là dei nostri scopi
  1. Quale componente individua le clausole per le varie rotte e come le elabora?
    - Ovvero: *Route::get(...)* è solo una dichiarazione? O una chiamata a *get()* che sarà materialmente effettuata?
  2. Quali meccanismi portano all'invocazione del callback, come reazione a una richiesta sulla rotta associata?
  3. Dove sono definite funzioni come *get()* o *view()* e classi come *Route*?
    - ora alcuni brevi chiarimenti su quest'ultimo aspetto



# Le route: aspetti tecnici / 2

- All'avvio di una app Laravel, appositi meccanismi (piuttosto complessi) danno luogo a:
  - *autoloading* di codice: classi, librerie e *helper functions* (globali)
  - *definizioni* di *namespaces*, *alias*, etc. (maggiori dettagli dopo)
- Uno dei file eseguiti all'avvio è `.../config/app.php`, che, tra l'altro:
  - carica la classe `App\Providers\RouteServiceProvider`
  - definisce: `'Route'` come alias per la classe `Illuminate\Support\Facades\Route`
- Ora è possibile spiegare che, nelle rotte come:

```
Route::get('/', function () {  
    return view('welcome');  
});
```

  - `view()` è una helper function, definita nel file `vendor/laravel/framework/src/Illuminate/Foundation/helpers.php`
  - `Route` è l'alias di cui si è detto, per la classe Façade che dà accesso alla classe `vendor/laravel/framework/src/Illuminate/Routing/Router`
  - `get()` è un metodo di quest'ultima classe

# Le route: *RouteServiceProvider*

- Come detto, all'avvio di una app Laravel, *.../config/app.php* fa sì che sia caricata la classe *App\Providers\RouteServiceProvider*
- Vale la pena di riportarne qualche frammento, a destra
- La classe legge il file *web.php*, ne memorizza le route nelle strutture dati Laravel appropriate, e le collega al *middleware* 'web'
- Il middleware fa da filtro tra: la richiesta HTTP in ingresso e la generazione della risposta
- Lo stesso viene fatto per le route API (usate per interazione con cliente app anziché browser)

```
<?php
namespace App\Providers;
use Illuminate\Support\Facades\Route;
// consente i riferimenti Route:: sotto

...
class RouteServiceProvider extends ServiceProvider
{
    protected $namespace = 'App\Http\Controllers';
    // consente di omettere questo prefisso nelle route (in web.php)
    // con callback basati sui controller (vedi)

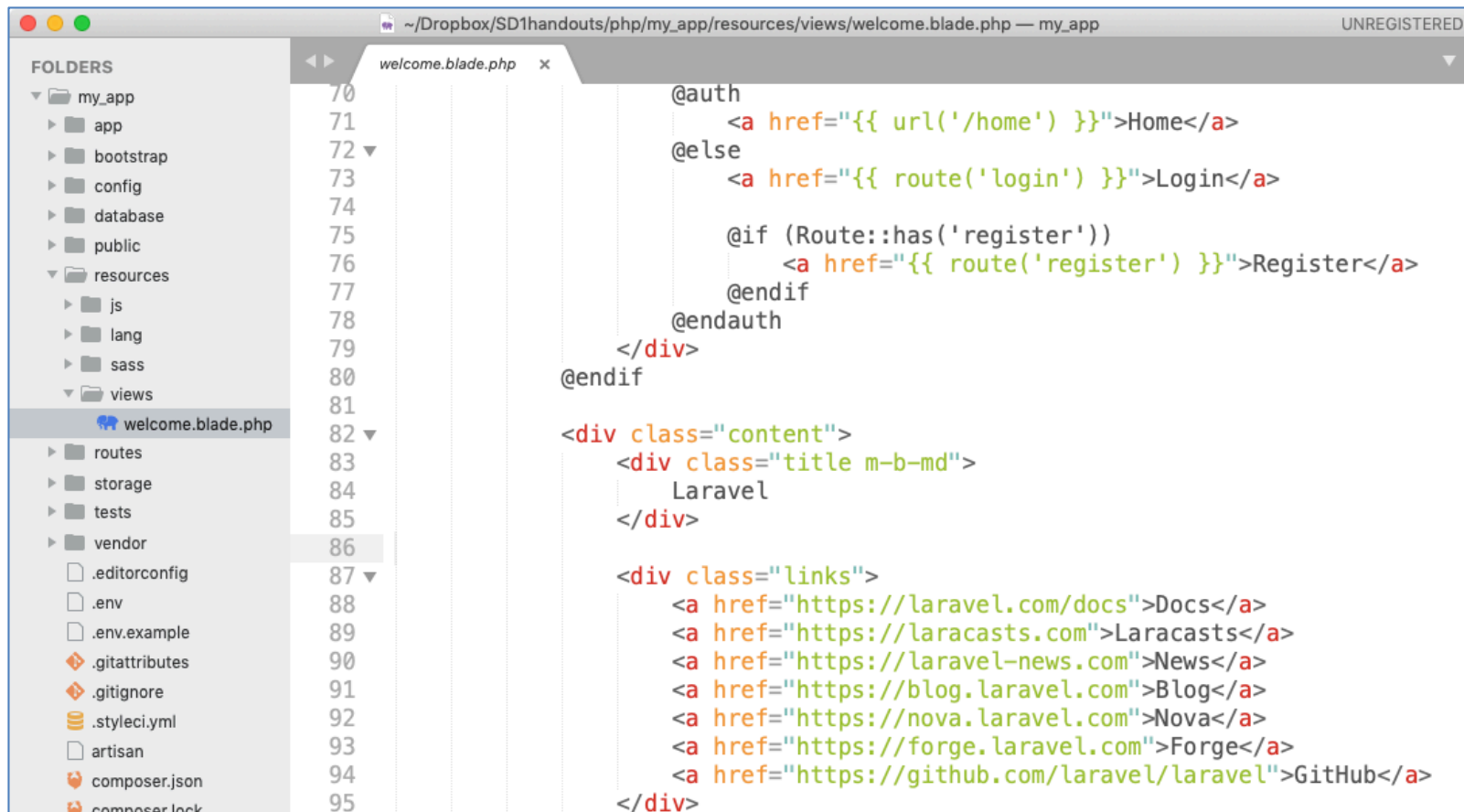
    public function map() // costruisce le route per l'app e per l'API
    {
        $this->mapApiRoutes();
        $this->mapWebRoutes();
    }

    // Legge web.php per definire le "web" routes dell'applicazione
    // il middleware 'web' gestisce la sessione, la protezione CSRF, etc.
    protected function mapWebRoutes()
    {
        Route::middleware('web')
            ->namespace($this->namespace)
            ->group(base_path('routes/web.php'));
    }

    /* mapApiRoutes() definisce le routes "api" per l'applicazione */
    ...
}
```

# La view *welcome*

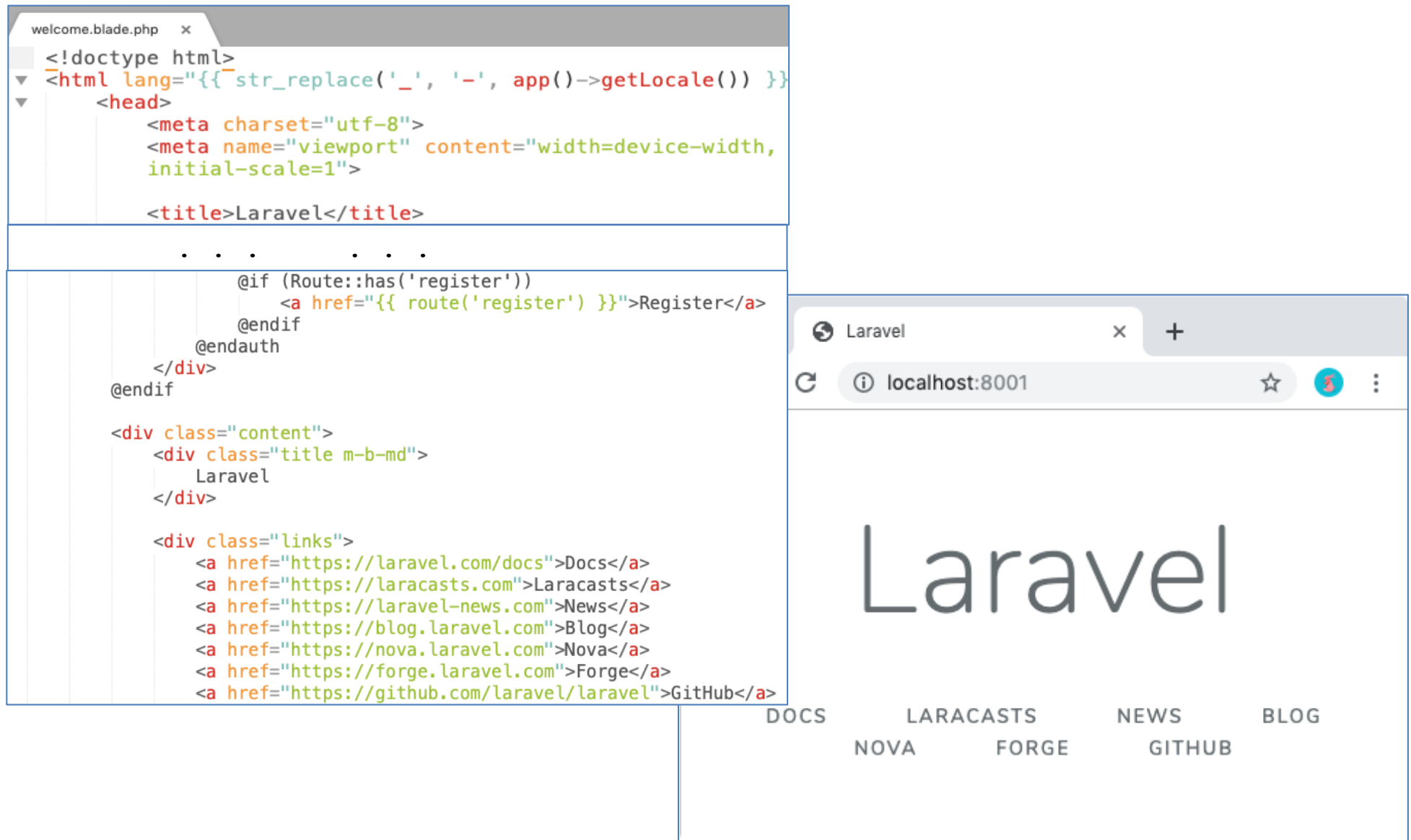
- Le view (*welcome.blade.php*) e gli altri "asset" di una app si trovano in *resources/*
- *blade* è un *engine* (componente di Laravel) usato per generare pagine web/php, ciascuna delle quali istanzia (e diversifica) una pagina *template* di base



```
70 @auth
71     <a href="{{ url('/home') }}">Home</a>
72 @else
73     <a href="{{ route('login') }}">Login</a>
74
75     @if (Route::has('register'))
76         <a href="{{ route('register') }}">Register</a>
77     @endif
78 @endauth
79 </div>
80 @endif
81
82 <div class="content">
83     <div class="title m-b-md">
84         Laravel
85     </div>
86
87     <div class="links">
88         <a href="https://laravel.com/docs">Docs</a>
89         <a href="https://laracasts.com">Laracasts</a>
90         <a href="https://laravel-news.com">News</a>
91         <a href="https://blog.laravel.com">Blog</a>
92         <a href="https://nova.laravel.com">Nova</a>
93         <a href="https://forge.laravel.com">Forge</a>
94         <a href="https://github.com/laravel/laravel">GitHub</a>
95     </div>
```

- Sopra la view *welcome* di default, da confrontare con la pagina resa dal browser

# La view *welcome* di default e il suo output



The image displays the default Laravel welcome view and its rendered output. On the left, a code editor shows the `welcome.blade.php` file. The code includes a DOCTYPE declaration, an HTML lang attribute using a Blade directive to get the locale, a head section with meta tags for charset, viewport, and initial-scale, and a title tag. The main content area contains a navigation bar with a 'Register' link (if available), a 'Laravel' title, and a list of links to various resources: Docs, Laracasts, News, Blog, Nova, Forge, and GitHub. On the right, a browser window shows the rendered output of this view at `localhost:8001`. The browser displays the 'Laravel' logo in a large font, with the same navigation links arranged in two rows below it.

```
welcome.blade.php x
<!doctype html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Laravel</title>
  . . .
  @if (Route::has('register'))
    <a href="{{ route('register') }}">Register</a>
  @endif
  @endauth
</div>
@endif
<div class="content">
  <div class="title m-b-md">
    Laravel
  </div>
  <div class="links">
    <a href="https://laravel.com/docs">Docs</a>
    <a href="https://laracasts.com">Laracasts</a>
    <a href="https://laravel-news.com">News</a>
    <a href="https://blog.laravel.com">Blog</a>
    <a href="https://nova.laravel.com">Nova</a>
    <a href="https://forge.laravel.com">Forge</a>
    <a href="https://github.com/laravel/laravel">GitHub</a>
  </div>
</div>
```

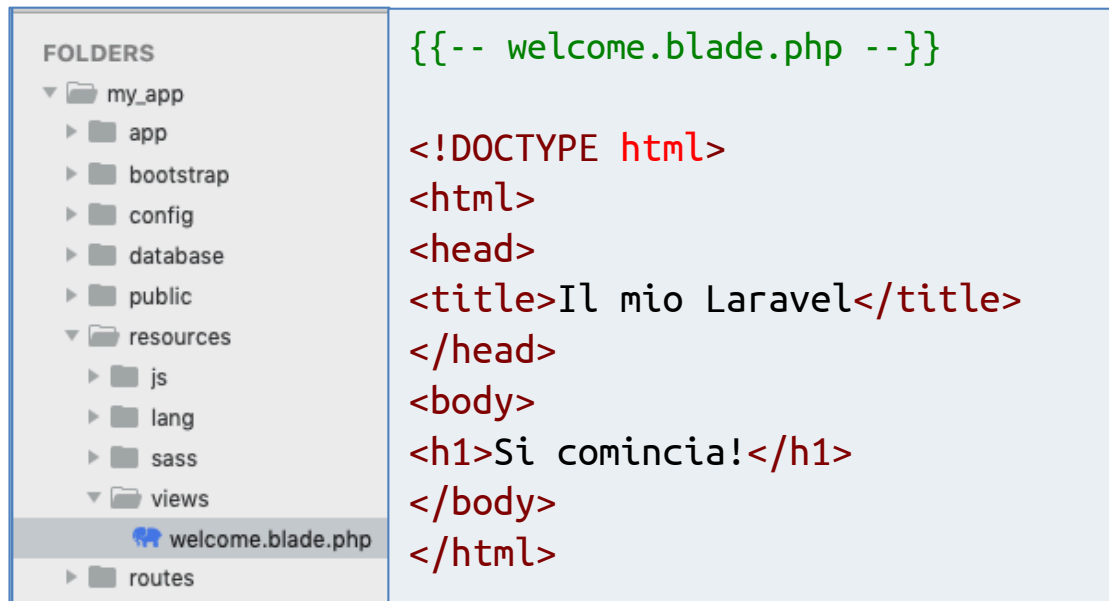
Laravel

DOCS LARACASTS NEWS BLOG  
NOVA FORGE GITHUB

- Possiamo provare a cambiare il file *welcome.blade.php* ...

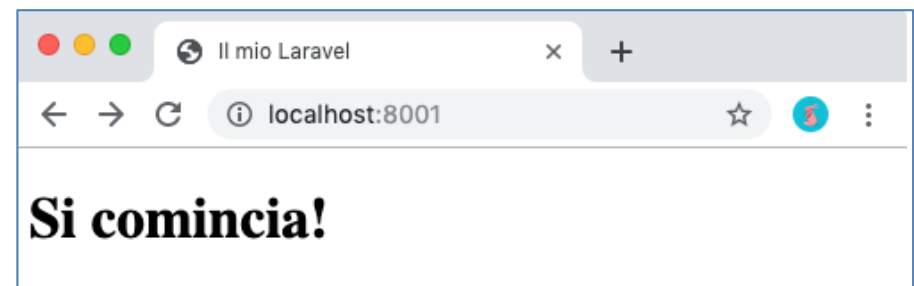


# La nuova view *Welcome* e il suo output



Il nome del file `welcome.blade.php` è qui mostrato in un commento `{{...}}` del linguaggio dell'engine blade

NB: il testo qui sopra (e quelli che seguiranno) può essere copiato nel relativo file per sperimentare facilmente l'effetto (qui a destra)



- Ora, aggiungiamo una route *contact* in *web.php*

# La nuova route senza view *contact* e il suo output

```
web.php x
<?php

// Web Routes

Route::get('/', function () {
    return view('welcome');
});

Route::get('/contact', function () { // nuova route
    return view('contact');          // nuova view
});
```

- Aggiungiamo una route per la URL **'/contact'**
- Senza però introdurre la view *contact.blade.php*
- ...

- Ecco l'errore: non c'è una view *contact*!
- Aggiungiamo allora la view come file *contact.blade.php*

The screenshot shows a web browser window with a dark theme. The address bar shows a path to a Laravel application. The main content area displays a red error message: "InvalidArgumentException View [contact] not found." Below the message is a "COPY" button. The developer console is open, showing a stack trace with the following frames:

- 53 InvalidArgumentException .../vendor/laravel/framework/src/Illuminate/View/FileNotFoundException.php:137
- 52 Illuminate\View\��iewFinder findInPaths .../vendor/laravel/framework/src/Illuminate/View/��iewFinder.php:79
- 51 Illuminate\View\��iewFinder find .../vendor/laravel/framework/src/Illuminate/View/Factory.php:130
- 50 Illuminate\View\��iewFactory make .../vendor/laravel/framework/src/Illuminate/Foundation/helpers.php:968

The right side of the console shows the source code of the `findInPaths` method in `FileNotFoundException.php`, highlighting the line that throws the `InvalidArgumentException` when a view is not found.

Arguments:

- "View [contact] not found."

Environment & details:

# La nuova route con la view *contact* e il suo output

```
{{-- contact.blade.php --}}  
  
<!DOCTYPE html>  
<html>  
<head>  
<title>Contatti</title>  
</head>  
<body>  
<h1>Modulo nostri contatti</h1>  
</body>  
</html>
```



- Aggiungiamo un link a */contact* sulla home:

```
{{-- welcome.blade.php --}}  
  
<!DOCTYPE html>  
<html>  
<head>  
<title>Il mio Laravel</title>  
</head>  
<body>  
<h1>Si comincia!</h1>  
Contattateci <a href="/contact">  
qui</a>  
</body>  
</html>
```



# Una terza route/view: *about*

```
web.php x
<?php

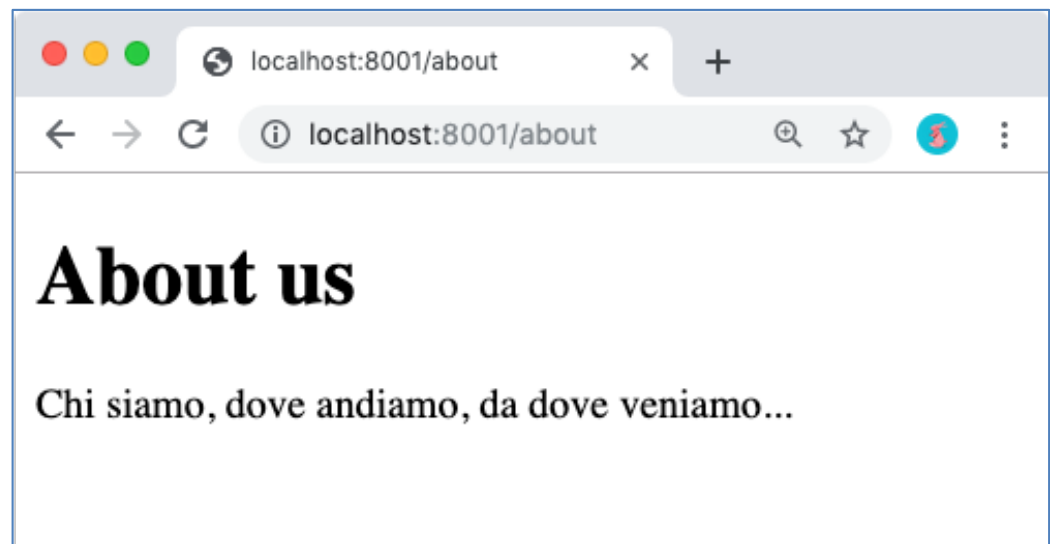
Route::get('/', function () {
    return view('welcome');
});

Route::get('/contact', function () {
    return view('contact');
});

Route::get('/about', function () {
    return view('about');
});
```

```
welcome.blade.php x
<!DOCTYPE html>
<html>
<head><title>Il mio Laravel</title>
</head>
<body>
    <h1>Si comincia!</h1>
    <ul>
        <li> Contattateci
            <a href="/contact"> qui</a> </li>
        <li> <a href="/about">About us</a> </li>
    </ul>
</body>
</html>
```

```
about.blade.php
<!DOCTYPE html>
<html>
<head><title></title>
</head>
<body>
    <h1>About us</h1>
    <p>Chi siamo, dove andiamo,
        da dove veniamo... </p>
</body>
</html>
```



# Un file di layout: motivazione

- Immaginiamo di volere i link "About us" e "Contatateci qui" in ogni pagina/view e non solo nella root (*welcome*) ...
- Piuttosto che inserirli in *ogni* pagina, presente e futura, si introduce una vista, qui chiamata *layout.blade.php*, che farà da "template" per le altre...
- Iniziamo copiando (l'ex) *welcome.blade.php* su *layout.blade.php*, in modo che il template contenga i due link (`<a href...>`)



```
layout.blade.php
<!DOCTYPE html>
<html>
<head>
  <title>Il mio Laravel</title>
</head>
<body>
  <h1>Si comincia!</h1>
  <ul>
    <li> Contatateci
      <a href="/contact"> qui</a> </li>
    <li> <a href="/about">About us</a> </li>
  </ul>
</body>
</html>
```

# Un file di layout: costruzione

- Eliminiamo dal template la parte specifica `<h1>...</h1>` del *body*, e lasciamo quella generica, che vogliamo si ripeta su tutte le view
- Introduciamo ora la parte parametrica, con l'annotazione `@yield` (cioè "genera") e diamole il nome, 'contenuto', con cui verrà richiamata, e istanziata in modo specifico, nelle varie pagine basate su *layout*

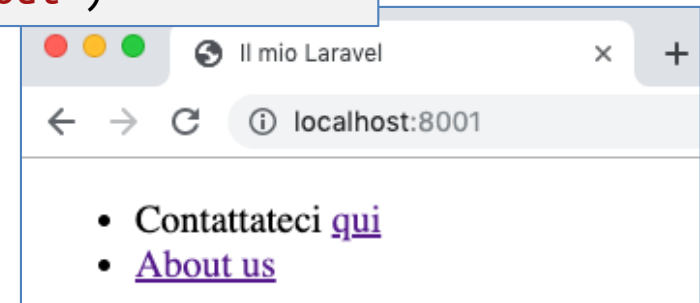
```
layout.blade.php
<!DOCTYPE html>
<html>
<head><title>Il mio Laravel</title>
</head>
<body>
  <ul>
    <li> Contattateci
      <a href="/contact"> qui</a> </li>
    <li> <a href="/about">About us</a> </li>
  </ul>
</body>
</html>
```

```
layout.blade.php
<!DOCTYPE html>
<html>
<head><title>Il mio Laravel</title>
</head>
<body>
  @yield('contenuto')
  <ul>
    <li> Contattateci
      <a href="/contact"> qui</a> </li>
    <li> <a href="/about">About us</a> </li>
  </ul>
</body>
</html>
```

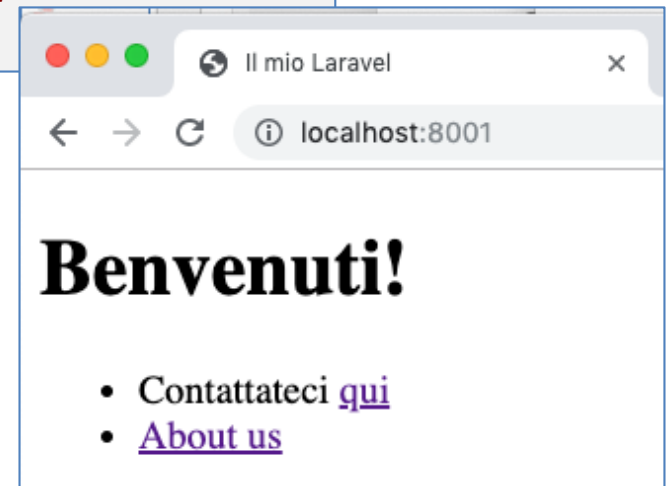
# Istanza di un file di layout

- Con `@extends('layout')` si rende `welcome.blade.php` un'istanza di `layout.blade.php`
- Introduciamo ora in `welcome.blade.php` una specifica `@section('contenuto')` corrispondente all'annotazione `@yield('contenuto')` in `layout.blade.php`
- Il contenuto di `@section` **istanzia** il corrispondente `@yield` del template
- Il componente `blade` di Laravel è essenzialmente un *template engine*

```
{{-- welcome.blade.php --}}  
  
@extends('layout')
```



```
{{-- welcome.blade.php --}}  
  
@extends('layout')  
  
@section('contenuto')  
<h1>Benvenuti!</h1>  
@endsection
```



# Istanza di un file di layout / 2

- Ora, se si decide che i link devono precedere il contenuto, basta spostare `@yield('contenuto')` alla fine del layout
- Così le viste "istanza" del layout cambiano tutte, ma i rispettivi file sorgente sono invariati!
- Proseguire l'esercizio:
  - rendiamo tutte le view istanze di *layout.blade.php*
  - aggiungiamo un link *Home* in tutte le viste



```
layout.blade.php x
<!DOCTYPE html>
<html>
<head>
  <title>Il mio Laravel</title>
</head>
<body>
  <ul>
    <li> Contattateci
      <a href="/contact"> qui</a> </li>
    <li> <a href="/about">About us</a> </li>
  </ul>
  @yield('contenuto')
</body>
</html>
```





# Layout con doppio yield

- Nel file di layout si possono introdurre più *@yield ...*
  - p.es. per rendere generico il titolo

```
layout.blade.php x
<!DOCTYPE html>
<html>
<head>
  <title>@yield('titolo')</title>
</head>
<body>
  <ul>
    <li> Contattateci
      <a href="/contact"> qui</a> </li>
    <li> <a href="/about">About us</a> </li>
  </ul>
  @yield('contenuto')
</body>
</html>
```

- Se una *@section*, come quella corrispondente allo yield *'titolo'*, è breve, la si può definire *inline*, anzichè come blocco

```
{{-- welcome.blade.php --}}

@extends('layout')

@section('titolo','Welcome')

@section('contenuto')
  <h1>Benvenuti!</h1>
@endsection
```

# Per concludere l'esempio su layout e Blade...

- Per evitare che una view che non istanzia lo `@yield('titolo')` resti con `<title>` vuoto, in *layout* si può anche introdurre un default: `@yield('titolo','Laravel')`
- Infine, si aggiunge a *layout*, quindi a tutte le view che lo istanziano, un link *Home* che punta alla URL-route base ("/")
- Come si ricorderà, questa route viene mappata sulla view *welcome*

```
layout.blade.php x
<!DOCTYPE html>
<html>
<head>
  <title>@yield('titolo','Laravel')</title>
</head>
<body>
  <ul>
    <li> Contattateci
      <a href="/contact"> qui</a> </li>
    <li> <a href="/about">About us</a> </li>
    <li> <a href="/">Home</a> </li>
  </ul>
  @yield('contenuto')
</body>
</html>
```

```
{{-- welcome.blade.php --}}

@extends('layout')

@section('titolo','Welcome')

@section('contenuto')
  <h1>Benvenuti!</h1>
@endsection
```

# Per concludere l'esempio su layout e Blade...

- Rinfreschiamo le pagine corrispondenti a ciascuna delle viste realizzate e ispezioniamo nel browser il relativo codice sorgente HTML, generato per ciascuno, da Laravel/PHP istanziando il layout

