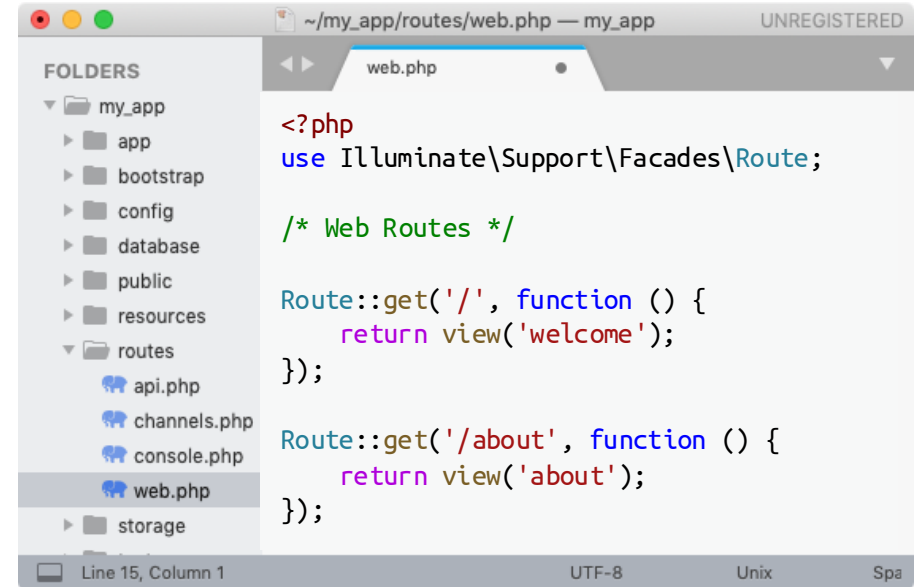


# Struttura di un app Laravel: *route*

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

Una **route** ("rotta") definisce allora la risposta dell'app a una data richiesta HTTP, ossia *instrada* la richiesta in arrivo verso il codice che produce la risposta  
Il paradigma di programmazione di Laravel è cioè *reattivo/orientato agli eventi*



The screenshot shows a code editor window titled '~my\_app/routes/web.php — my\_app'. On the left, a 'FOLDERS' sidebar lists the project structure: my\_app (containing app, bootstrap, config, database, public, resources, routes, and storage), routes (containing api.php, channels.php, console.php, and web.php), and storage. The main editor displays the content of web.php, which includes the PHP opening tag, the use of the Route facade, a comment for Web Routes, and two Route::get() calls. The first call is for the root path '/' returning a 'welcome' view. The second call is for the '/about' path returning an 'about' view. The editor status bar at the bottom indicates 'Line 15, Column 1', 'UTF-8' encoding, and 'Unix' line endings.

```
<?php
use Illuminate\Support\Facades\Route;

/* Web Routes */

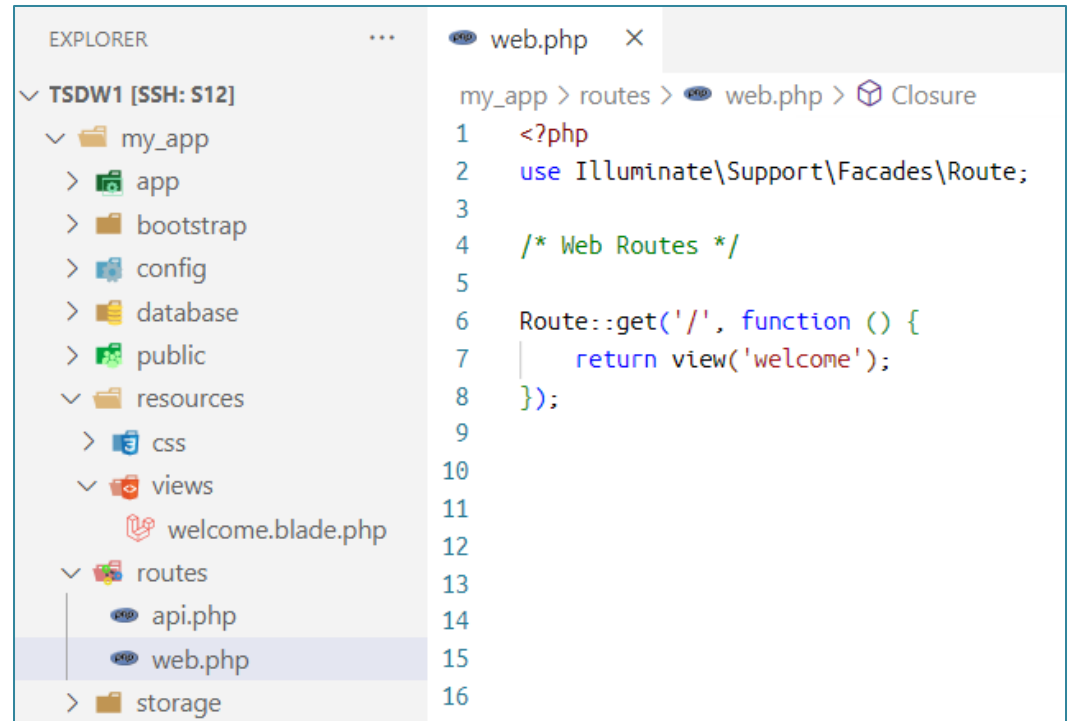
Route::get('/', function () {
    return view('welcome');
});

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

# Callback e view

Il file *web.php* qui a destra è (in forma lievemente semplificata) quello generato per default con una nuova app; in esso:

- Il callback restituisce l'output prodotto, a sua volta, dalla chiamata di funzione `view('welcome')`, la quale:



The screenshot shows an IDE with two panels. The left panel is the 'EXPLORER' view showing a file tree for a project named 'TSDW1 [SSH: S12]'. The tree includes folders like 'my\_app', 'app', 'bootstrap', 'config', 'database', 'public', 'resources', 'views', 'routes', and 'storage'. The 'resources' folder is expanded, showing 'css', 'welcome.blade.php', and 'routes'. The 'routes' folder is also expanded, showing 'api.php' and 'web.php'. The 'web.php' file is selected. The right panel shows the code for 'web.php'. It starts with a PHP tag, uses the 'Illuminate\Support\Facades\Route' facade, and defines a GET route for the root path ('/') that returns the 'welcome' view.

```
EXPLORER
TSDW1 [SSH: S12]
├── my_app
│   ├── app
│   ├── bootstrap
│   ├── config
│   ├── database
│   ├── public
│   ├── resources
│   │   ├── css
│   │   ├── welcome.blade.php
│   │   └── routes
│   │       ├── api.php
│   │       └── web.php
│   └── storage
└── ...

web.php
my_app > routes > web.php > Closure
1  <?php
2  use Illuminate\Support\Facades\Route;
3
4  /* Web Routes */
5
6  Route::get('/', function () {
7      return view('welcome');
8  });
9
10
11
12
13
14
15
16
```

- preleva una pagina HTML, detta appunto *view*, dal file '*welcome.blade.php*', posto in *resources/views*, e
- ne restituisce il contenuto, dopo un eventuale pre-processing da parte dell'engine *Blade* (v. oltre); è questo l'HTML inviato al cliente
- Così per qualsiasi argomento *stringa* di `view(...)` in un callback

# Callback: cosa restituisce

- Qui a destra un *web.php* semplificato: il callback restituisce testo HTML direttamente, anziché 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, se un callback restituisce un dato PHP, come un array, questo sarà automaticamente convertito in JSON
  - v. esempio qui a destra (N.B.: il path */gigi* )
- Infine, vedremo in seguito che più callback per *route* "omogenee" si possono accorpare come metodi di un componente detto *controller*, una classe (maggiori dettagli più avanti)

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

```
<?php
// Web Routes

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

# Semantica di una app Laravel: le rotte

Cos'è in effetti una **app** Laravel? All'avvio con `artisan serve`, l'esecuzione entra nella directory *public* e si comporta (grosso modo) come se si eseguisse: `php -S localhost:8000 public/index.php`

Qui, i meccanismi sfruttati dalla app Laravel si fanno complessi, ma è semplice descriverne il comportamento "percepito" in termini *event-driven* o *reattivi*: l'app ascolta richieste HTTP dai clienti e, quando ne riceve una (l'evento), reagisce così:

- individua in *web.php* la *rotta* corrispondente a metodo HTTP e *path* della richiesta pervenuta; p.es. a `GET /ciao` può corrispondere `Route::get('/ciao', function () {return 'ciao';});`
- a questo punto, l'app invoca il callback (`function () {...}`) della rotta individuata e ne invia l'output (es. `'ciao'`) al cliente, come risposta

Qui a destra un test in cui il cliente è *telnet*

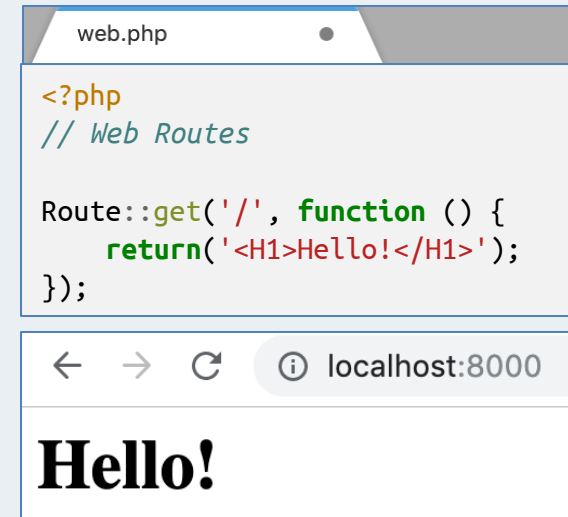
```
$ telnet localhost 8000
Trying... Connected to 127.0.0.1
GET /ciao

HTTP/1.1 200 OK
...

ciao
Connection closed by foreign host.
```

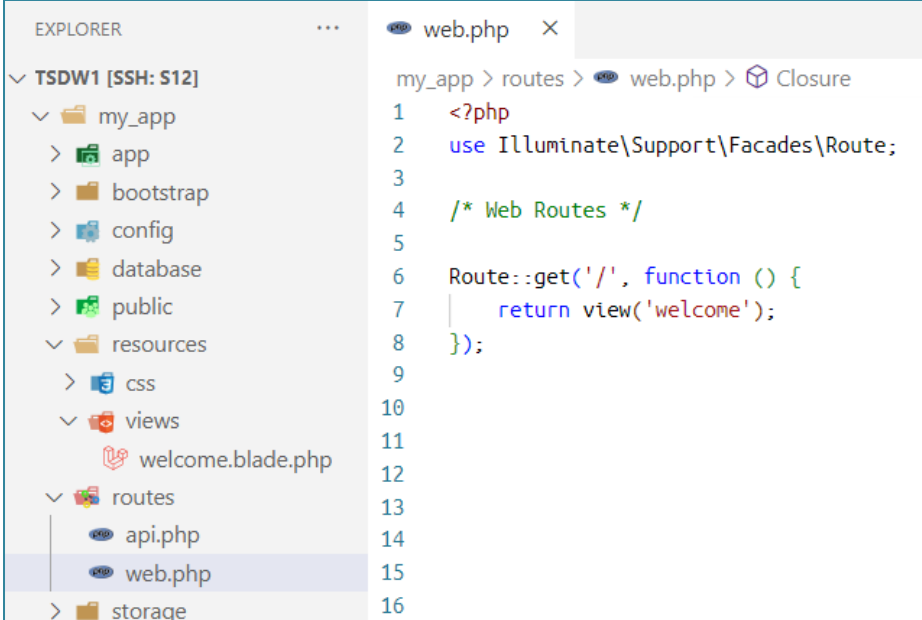
# Le route: aspetti tecnici

- Struttura e forma di *web.php* pongono delle questioni tecniche interessanti su Laravel
- Ne accenniamo tre, anche se le risposte per (1) e (2) sono al di là dei nostri scopi
  1. Tecnicamente `Route::get(...)`; è una chiamata di un metodo statico *get()* della classe *Route*, allora:
    - quando viene effettuata? ad opera di quale componente?
    - come si svolge la sua esecuzione? complicato rispondere... Meglio vederla quindi come una sorta di dichiarazione di rotta
  2. Quali meccanismi fanno sì che il callback venga invocato, in reazione a ciascuna richiesta sul path associato nella rotta?
  3. Dove sono definite le funzioni *get()* o *view()* e la classe *Route*?
    - ora alcuni brevi chiarimenti su quest'ultimo aspetto



# Una semplice app Laravel: la view *welcome*

- Per quanto detto, lo sviluppo di un'app dallo "scheletro" generato con *laravel new* può iniziare dal file delle rotte *routes/web.php*
- Qui a destra vediamo *web.php* di default, con una rotta che associa, alla richiesta 'GET /', la view *welcome.blade.php*
- Le view e gli altri "asset" di una app si trovano in *resources/*
- *blade* è un *engine* (processore, componente di Laravel) usato per generare pagine web/php, istanziando (e diversificando) una pagina *template* di base
- *blade* offre poi diversi costrutti per automatizzare in vari modi la generazione di HTML



The screenshot shows a code editor with two panes. The left pane, titled 'EXPLORER', displays the file structure of a Laravel application. The right pane shows the code for *web.php* in the *routes* directory.

**EXPLORER**

- TSDW1 [SSH: S12]
  - my\_app
    - app
    - bootstrap
    - config
    - database
    - public
    - resources
      - css
      - views
        - welcome.blade.php
    - routes
      - api.php
      - web.php
    - storage

**web.php**

```
1 <?php
2 use Illuminate\Support\Facades\Route;
3
4 /* Web Routes */
5
6 Route::get('/', function () {
7     return view('welcome');
8 });
9
10
11
12
13
14
15
16
```

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

```
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">

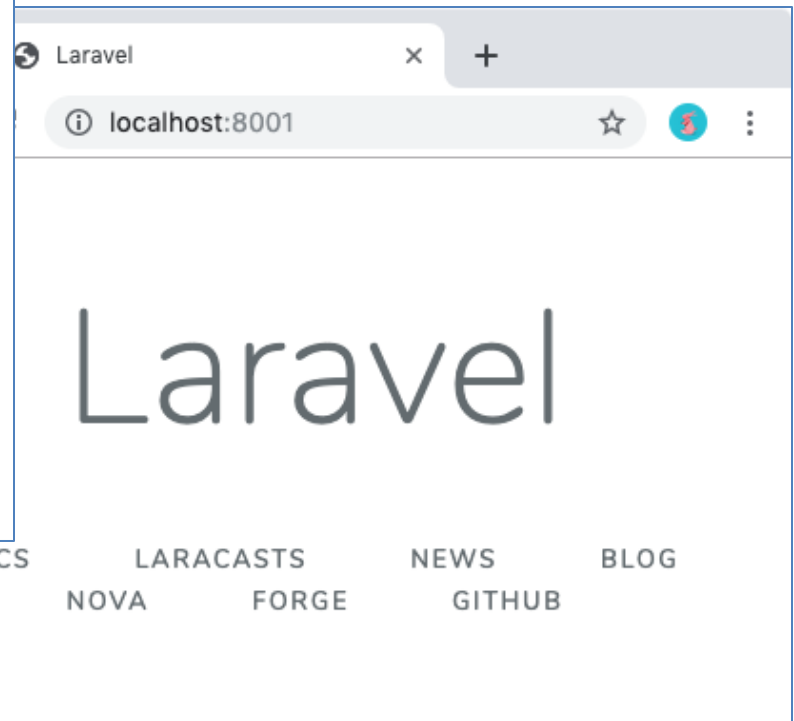
  <title>Laravel</title>

  . . .

  @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>
```



Iniziamo quindi a modificare il file *welcome.blade.php* ...

# La nuova view *Welcome* e il suo output



```
my_app
├── app
├── bootstrap
├── config
├── database
├── public
├── resources
│   ├── css
│   └── views
│       └── welcome.blade.php
├── routes
│   ├── api.php
│   └── web.php
```

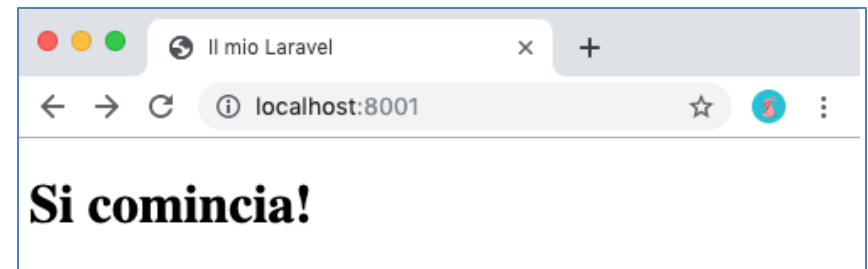
```
{{!-- welcome.blade.php --}}
```

```
<!DOCTYPE html>
<html>
<head>
<title>Il mio Laravel</title>
</head>
<body>
<h1>Si comincia!</h1>
</body>
</html>
```

NB: 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 incollato nel relativo file per sperimentare facilmente l'effetto (qui a destra)

NB: occhio a non incollare eventuali caratteri spuri!



Ora, aggiungiamo una route *contact* nel file *web.php*



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

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

```
<?php

// Web Routes

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

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

- Ed ecco l'errore: non c'è una view *contact*!
- Si crea allora, in *resources/views contact.blade.php*

InvalidArgumentException  
View [contact] not found.

Application frames (2) All frames (54)

53 InvalidArgumentException  
../vendor/laravel/framework/src/Illuminate/View/FileViewFinder.php:137

52 Illuminate\View\FileViewFinder findInPaths  
../vendor/laravel/framework/src/Illuminate/View/FileViewFinder.php:79

51 Illuminate\View\FileViewFinder find  
../vendor/laravel/framework/src/Illuminate/View/Factory.php:130

50 Illuminate\View\Factory make  
../vendor/laravel/framework/src/Illuminate/Foundation/helpers.php:968

1. "View [contact] not found."

Environment & details:

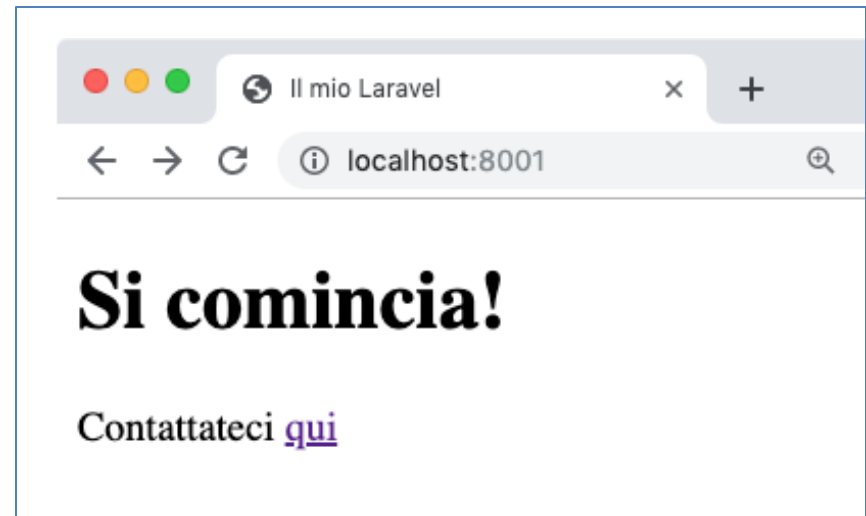
# La nuova route con la view *contact* e il suo output

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



Per renderla raggiungibile, aggiungiamo un link a */contact* sulla pagina "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*

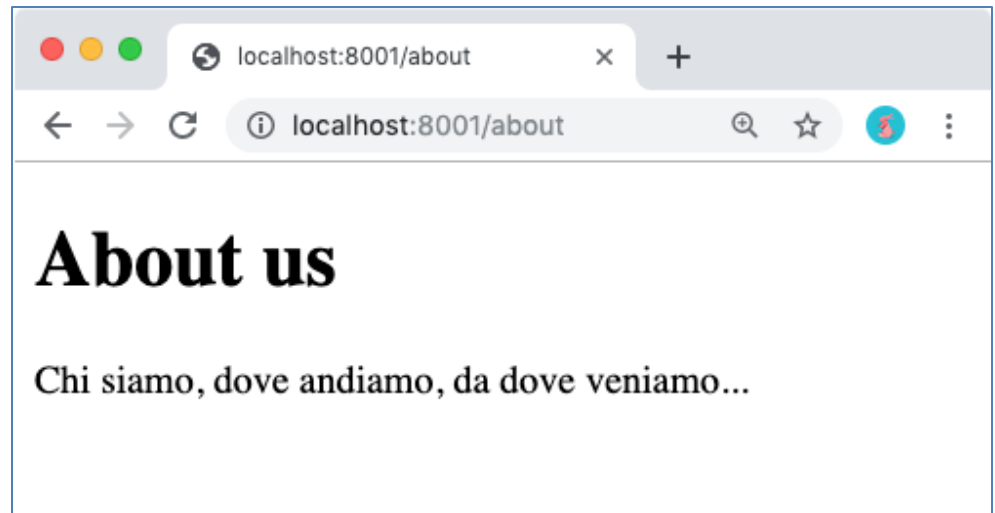
```
web.php x
<?php
...
Route::get('/', function () {
    return view('welcome');
});

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 ora di volere i link "About us" e "Contattateci qui" in ogni pagina/view e non solo nella home/root (*welcome*) ...
- Piuttosto che inserirli a mano in *ogni* pagina, presente e futura, si introduce una vista che li contiene e farà da "template" per tutte le altre...
- Iniziamo copiando (l'attuale) *welcome.blade.php* sul template *layout.blade.php*, così da avere in questo i 2 link (`<a href ...>`)
- Ora, si vuole rendere generica la parte specifica (`<h1>...</h1>`)...



```
layout.blade.php
<!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>
```

# Il template *layout.blade.php*

- Eliminiamo quindi dal template la parte specifica `<h1>...</h1>` del *body*, sostituendola con il costrutto blade `@yield` (*yield* vuol dire *genera*)
- Questa è la parte "variabile", che ogni view basata sul template *layout.blade.php* potrà rimpiazzare a piacimento



```
layout.blade.php x
<!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>
```

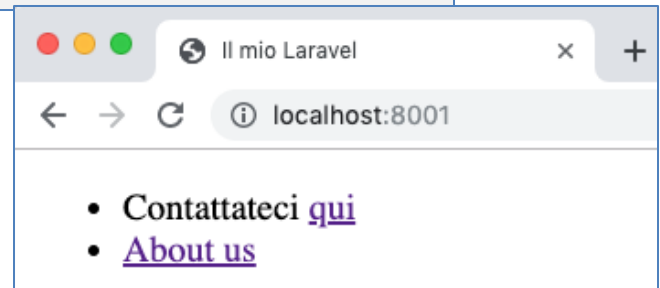
- A questo scopo, una view basata sul template *layout.blade.php* deve definire un blocco HTML, sia *B*, marcato come `@section('contenuto')`
- Quando Laravel serve al client una view basata sul template *layout.blade.php* invierà il template stesso, sostituendovi *B* a `@yield('contenuto')`
- Così, ogni view che sia basata sul template *layout.blade.php* lo riproduce, istanziandone e differenziandone la parte "variabile" e replicando il resto

# Istanza di un file di layout

- Con `@extends('layout')` si rende *welcome.blade.php* un'istanza del template *layout.blade.php*, con l'effetto mostrato qui a fianco
- Ciò si spiega riprendendo *layout.blade.php*, qui riprodotto per chiarezza
- NB: il costrutto `@yield('contenuto')` presente in *layout.blade.php* non produce alcun effetto nel rendering di *welcome.blade.php*
- Ciò in quanto *welcome.blade.php* non contiene, in questa versione, una corrispondente definizione di `@section('contenuto')`
- Vediamo quindi cosa succede introducendola...

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

```
@extends('layout')
```



```
{{-- 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 / 2

- Ecco quindi una versione di *welcome.blade.php* in cui compare `@section('contenuto')` che si accoppia all'annotazione `@yield('contenuto')` nel template *layout.blade.php*, con l'effetto mostrato qui a fianco
- Come si vede, nell'inviare la view *welcome.blade.php* al browser, Laravel, attraverso *blade*, invia il template *layout.blade.php*
  - sostituendo `@yield('contenuto')` nel template della view
  - con il testo di `@section('contenuto')` nella view *welcome.blade.php*
- Ciò illustra come *blade* sia prima di tutto un *template engine*

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

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



```
{{-- 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 / 3

- Ora, se si decide che i link devono precedere (anziché seguire) il contenuto, basta spostare `@yield('contenuto')` in fondo nel template
- Così le viste "istanza" del template cambieranno tutte, ma i rispettivi file sorgente sono invariati!
- Proseguire l'esercizio:
  - rendere tutte le view istanze del template `layout.blade.php`
  - aggiungere 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 --}}

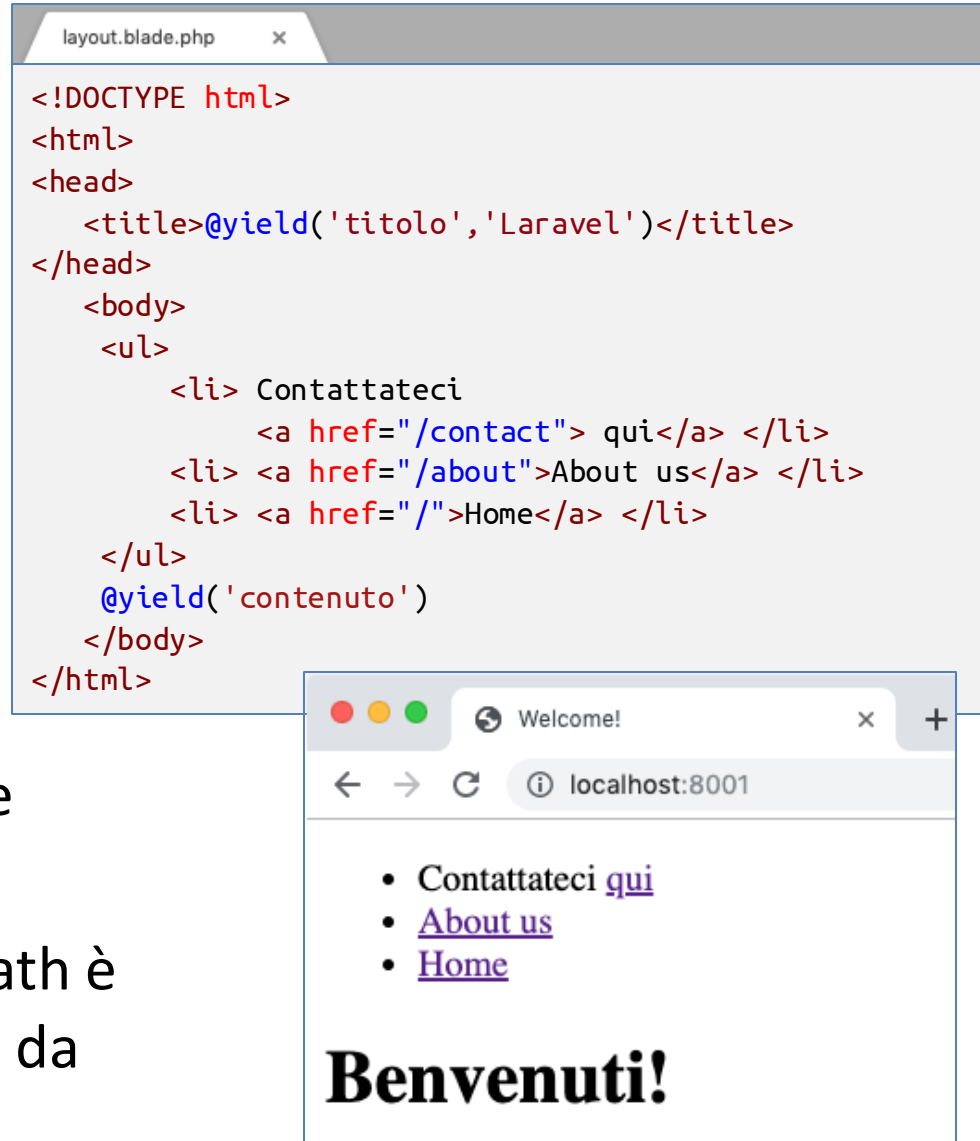
@extends('layout')

@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, si può anche definire un default per lo `yield`, così: `@yield('titolo','Laravel')`
- Infine, si aggiunge a `layout`, quindi a tutte le view che lo istanziano, un link *Home* che punta al path base ("/")
- Come si ricorderà, questo path è associato alla view *welcome* da una rotta del file *web.php*



# 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/Blade istanziando il layout

