



CORSO FULL STACK WEB DEVELOPER



LARAVEL

Model e Controller



ROUTE

Una chiamata base

Route::

facciamo riferimento alla classe Route

get

usiamo il metodo get in questo caso, ma avremo anche accesso a POST e a tutti gli altri metodi

'/'

il primo parametro della funzione get è l'url associato a questa rotta, in questo caso la home del nostro sito

function

il secondo parametro è una callback, in questo caso una funzione che ritorna una stringa, la quale viene stampata a schermo, come se stessimo stampando dell'html nella pagina

```
1 Route::get('/', function () {  
2     return 'La mia Homepage';  
3 });
```



Route:: ?

Cos'è questa stregoneria?

In ogni parte del nostro codice avremo accesso a delle **classi Laravel** che ci permetteranno di sfruttare al massimo le potenzialità del framework.

Tutte queste classi prendono il nome di Facades e vengono autocaricate ogni qualvolta viene chiamata la nostra applicazione.

Per prima cosa quindi chiameremo la **facade Route**, da cui possiamo accedere a tutti i suoi metodi per gestire le rotte della nostra applicazione.

La **Facade**, **fə'säd**, letteralmente **Facciata**, è un design pattern che prevede la creazione di un facade object che fa da **API semplificata** per una porzione di codice.

In Laravel gli oggetti facade servono come interfaccia per gli oggetti disponibili in **ServiceContainer**.

Questo container fa da collante tra i componenti.

"The Laravel service container is a powerful tool for managing class dependencies and performing dependency injection. Dependency injection is a fancy phrase that essentially means this: class dependencies are "injected" into the class via the constructor or, in some cases, "setter" methods."

<https://laravel.com/docs/7.x/container>



Il Controller

Il Controller è un file PHP che verrà chiamato dai file routes e si occuperà di

- rispondere ad una rotta
- recuperare e manipolare i dati
- restituire un risposta all'utente sotto forma di view o altro

È una vera e propria classe che estende il controller base di Laravel e che contiene al suo interno delle funzioni pubbliche (se chiamate direttamente da routes) e private.

Possiamo creare facilmente un controller con il comando:

```
1 php artisan make:controller NomeController
```

Per convenzione i nomi dei Controller sono al singolare in PascalCase.



Il Controller

Riprendendo l'esempio precedente abbiamo:

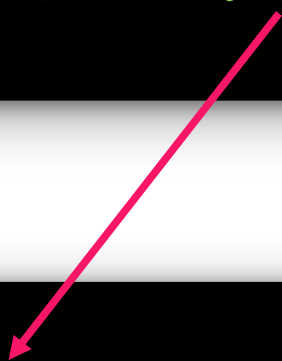
1. una rotta che avrà come url `/`
2. che viene gestita dal controller **HomeController**
3. più precisamente dalla public function **index()** all'interno di **HomeController**

web.php

```
1 Route::get('/', 'HomeController@index');
```

HomeController.php

```
1 public function index() {  
2     return view('homepage');  
3 }
```





Passare dati alla View

Array di dati

Possiamo ovviamente passare anche dei dati che ci serviranno nella view in diversi modi.

- 1 Creare un array che contiene tutti i dati da passare alla view e passarlo come secondo parametro dopo il path della view.

In questo modo, nella view saranno disponibili delle variabili che avranno come nome **le chiavi dell'array**.

UserController.php

```
1 public function index() {  
2     $data = [  
3         'name' => 'Fabio'  
4     ];  
5     return view('homepage', $data);  
6 }
```



Passare dati alla View

Compact

- 2 Usare il metodo **compact** passandogli come argomento il nome di una variabile precedentemente creata, senza il simbolo \$!

La variabile potrà contenere anche un array.

In questo modo, nella view sarà disponibile una variabile con lo **stesso nome** della variabile passata nel metodo compact.

UserController.php

```
1 public function index() {  
2     $name = 'Fabio';  
3     return view('homepage', compact('name'));  
4 }
```




LIVE CODING

Creiamo un Controller



ORM

Un nuovo modo di connettersi al Database

Abbiamo già visto in PHP come **connetterci** ad un **database MySQL** utilizzando **mysqli**. Con Laravel (e in generale con i framework) il processo di connessione, recupero dei dati e scrittura è più ad alto livello e sfrutta i vantaggi della programmazione ad oggetti.

Questa modalità prende il nome di **ORM (Object Relationship Mapping)** e serve ad ottenere, attraverso un'interfaccia orientata agli oggetti, tutti i servizi inerenti alla persistenza del dato.

Ogni **tabella** del database verrà quindi **mappata in una classe PHP** che faciliterà la gestione, la scrittura e la lettura dei dati, attraverso dei metodi forniti dall'ORM stesso.



ORM

Un nuovo modo di connettersi al Database

id	marca	modello	targa
1	Fiat	Punto	DA788CK
2	Ford	Fiesta	EF100LN
3	Opel	Zafira	BT967RS

```
1 class Auto {  
2     public $id;  
3     public $marca;  
4     public $modello;  
5     public $targa;  
6 }
```

```
1 $auto = new Auto();  
2 $auto->id // 1  
3 $auto->marca // Fiat  
4 $auto->modello // Punto  
5 $auto->targa // DA788CK
```



Eloquent

ORM di Laravel

Eloquent è l'ORM di Laravel, è potente e semplice, oltre che essere uno dei più famosi ORM basati sull'**active record**.

Lavora facilmente con diversi tipi di database, tra cui, ovviamente, MySQL.

Ogni tabella del nostro database sarà **mappata con un modello** Laravel che ci permetterà facilmente di leggerne il contenuto, crearlo, aggiornarlo o cancellarlo.

I termini:

ORM - Propria della programmazione ad oggetti, mette in comunicazione i paradigmi riguardanti i Database relazionali con il Linguaggio di Programmazione.

Ogni tabella come detto verrà gestita per essere utilizzata all'interno del codice.

Active record - È un tipo di architettura utilizzato per la relazione OOP e Database Relazionali.



Model

Le classi che mappano le tabelle del database prendono il nome di **Model** e rappresentano la struttura della tabella.

Una istanza di un Model rappresenta una riga della tabella corrispondente.

In Laravel, le classi Model possono essere create con un comando artisan.

Per creare un Model usiamo il comando:

```
1 php artisan make:model MyModel
```

Per convenzione i nomi dei Model sono al singolare in PascalCase.



Model

È tutto qui?

Sì, grazie a quell'**extends Model!**

```
1 use Illuminate\Database\Eloquent\Model;
2
3 class Flight extends Model
4 {
5     //
6 }
```



Quindi **Model** che cosa è?

```
1 use Illuminate\Database\Eloquent\Model;
2
3 class Flight extends Model
4 {
5     //
6 }
```



Connettiamoci al DB

modifichiamo .env

Per connetterci al DB dovremo modificare queste variabili nel file **.env**

I valori per la connessione al DB vengono presi proprio da qui, attraverso il **metodo env()**, il quale restituisce il valore della *key* passata come parametro.

```
1 DB_CONNECTION=mysql
2 DB_HOST=127.0.0.1
3 DB_PORT=3306
4 DB_DATABASE=nomeDB
5 DB_USERNAME=UserDB
6 DB_PASSWORD=PwDB
```




.env

All'interno della folder del progetto Laravel troviamo il file **.env-example** nel quale c'è la struttura base (con dati fittizi) delle variabili indispensabili per far funzionare il progetto. (debug, la connessione al DB, l'invio mail, etc.). Questo file **verrà pushato su Git ma non verrà utilizzato dal sistema**, essendo solo da esempio.

Quando scarichiamo un progetto Laravel è necessario creare il file .env copiando quello di esempio ed inserendo i dati corretti.

Da console nella cartella del progetto

MAC / Linux

```
1 mv .env-example .env
```

Windows PowerShell

```
1 Copy-Item -Path .env-example -Destination .env
```



La prima query

Attraverso la classe Model avremo accesso a tutto **il dataset** in maniera facile e veloce.

Nel nostro **Controller** potremmo ad esempio voler prendere tutti i film contenuti in database. Supponendo di avere una **tabella movies** e un model **Movie**, recuperiamo tutti i records

```
1 // prima della dichiarazione della classe Controller
2 use App\Movie;
3
4 ...
5
6 // all'interno di un metodo di un Controller
7 $movies = Movie::all();
```

App\Movie è il **model**,
usiamo il suo namespace per trovarlo

::all() è il **metodo** statico con il quale prendiamo
tutto ciò che si trova nella tabella movies.



LIVE CODING

Prendiamo un po' di dati dal DB



Ma possiamo filtrare i dati?



WHERE

Il metodo **where()** accetta due o tre parametri:

se ne usiamo due

il primo parametro è la colonna

il secondo è il valore

(viene sottinteso l'operatore uguaglianza)

se ne usiamo tre

il primo parametro è la colonna

il secondo è l'operatore

il terzo il valore



```
1 App\Movie::where('active', 1)->get(); // operatore =
```



```
1 App\Movie::where('active', '!=', 1)->get(); // operatore !=
```



get()

Se utilizziamo il metodo **where()** o **orderBy()** o qualsiasi altro metodo **dobbiamo utilizzare il metodo *get()*** per recuperare i risultati della query.

```
1 App\Movie::where('active', 1)->get();
```



OrderBy e Limit

Per ordinare i risultati delle query, si può utilizzare il metodo **orderBy()**, passando **due parametri**:

il primo parametro è la **colonna**

il secondo è la **direzione** (asc o desc)

Il metodo **limit()** invece accetta in ingresso

un solo parametro:

il **numero dei record** da prendere



```
1 $movies = App\Movie::where('active', 1)
2         ->orderBy('title', 'desc')->get();
```



```
1 $movies = App\Movie::where('active', 1)
2         ->orderBy('title', 'desc')
3         ->limit(10)->get();
```



First

First() si usa **al posto di get()**
e restituisce il primo record trovato.



```
1 $movies = App\Movie::where('active', 1)->first();
```




Find

find() cerca un record **per id**.

Per altri metodi:

<https://laravel.com/docs/7.x/eloquent>



```
1 $users = User::all();  
2  
3 $user = $users->find($id);
```



I metodi `all()` e `get()` restituiscono una **collection** di oggetti **Model**,

mentre i metodi `first()` e `find()` restituiscono direttamente un **oggetto Model**.



LIVE CODING



E per salvare un dato?

Nel Controller

```
1 $auto = new Auto();  
2 $auto->marca = 'Fiat';  
3 $auto->modello = 'Punto';  
4 $auto->targa = 'DA788CK';  
5  
6 $auto->save();
```







LIVE CODING



ESERCITAZIONE