



# *Modulo 5*

# Laravel

## *Capitolo 1*

# Cos'è Laravel

- Laravel è un Framework PHP con architettura MVC (Model View Controller)
- Il creatore si chiama Taylor Otwell (<https://twitter.com/taylorotwell/status/1182663227525734404>).
- Il progetto è completamente Open Source\* da giugno 2011 ed è disponibile al seguente link:  
<https://github.com/laravel/laravel>
- Attualmente conta piú di 800 Contributor che dedicano gratuitamente il loro tempo libero all'evoluzione del progetto
- Ovviamente esistono anche Contributor stipendiati (Laravel Core Team) che si occupano della manutenzione del codice, bug fixing e falle di sicurezza che necessitano di interventi repentinii.



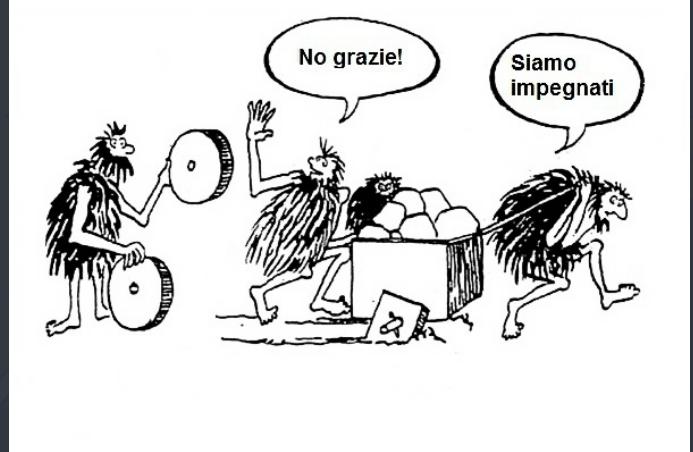
\*Open-source - [Approfondimento TedTalk](#)

# Cosa significa Framework

Un framework non è un linguaggio di programmazione, ma è un'architettura logica di supporto sulla quale un software può essere progettato e realizzato.

Bootstrap è un Framework CSS, non è un linguaggio di programmazione ma una serie di classi pre-costruite che ti permettono uno sviluppo rapido:

**Non bisogna reinventare la ruota ad ogni progetto.**

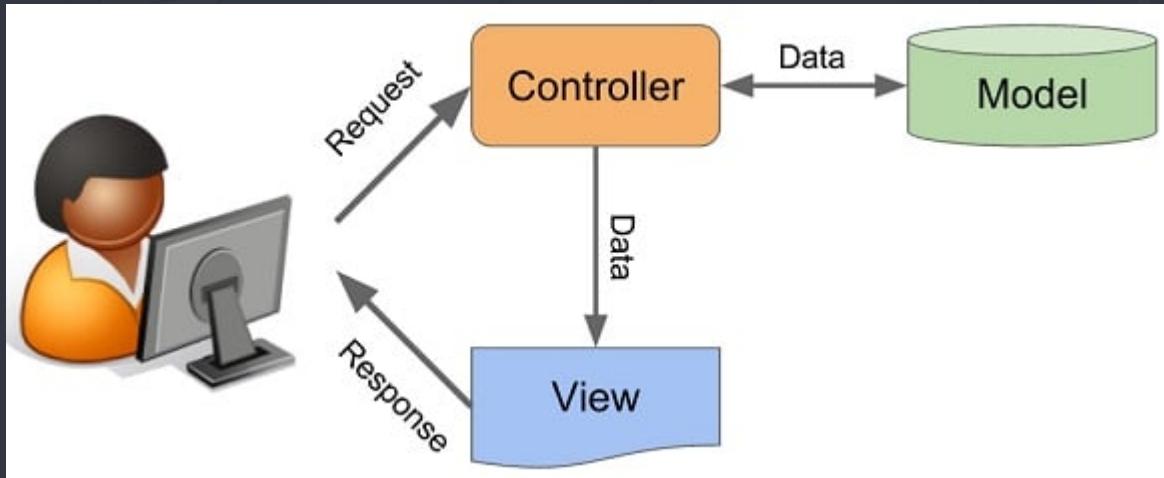


# Cosa significa MVC

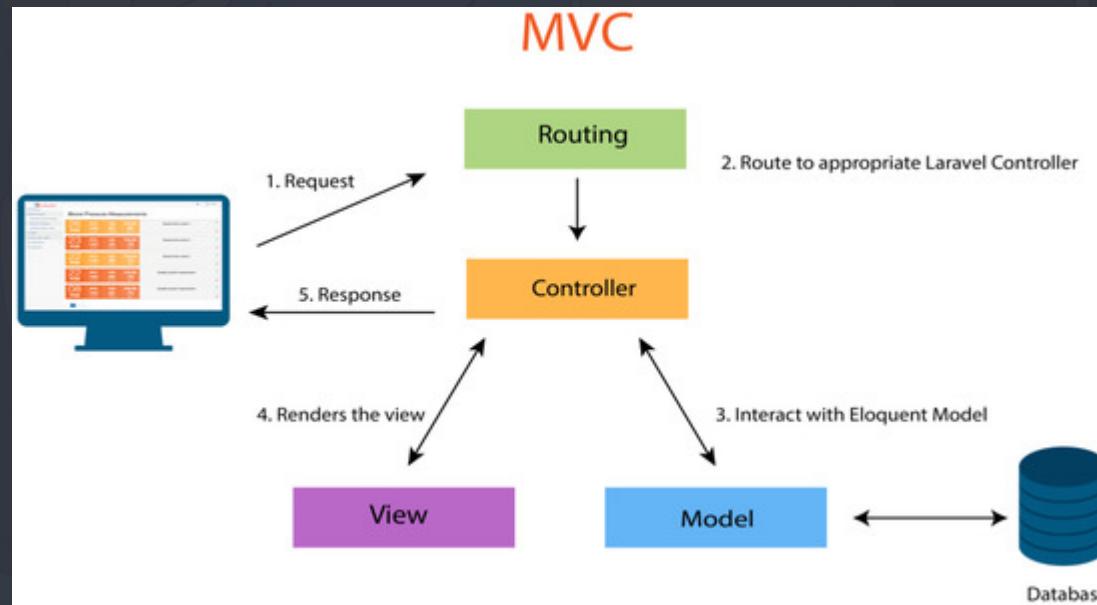
MVC — Model-View-Controller — è uno schema architettonale molto diffuso nello sviluppo di software, visto che si occupa della suddivisione del flusso logico in tre blocchi interconnessi tra loro.

- **Funzionalità di Business [Model]:** Il blocco che gestisce direttamente i dati, i principi logici e le regole dell'applicazione. Questo significa che lo stesso dato può avere un output diverso in base all'interfaccia grafica e al tipo di richiesta, il tutto senza modificare l'informazione sorgente.
- **Logica di Controllo [Controller]:** Il nucleo del sistema (Core) è il blocco che contiene materialmente il codice con tutte le istruzioni scritte dallo sviluppatore. Intercetta gli input dell'utente, li elabora, interroga i dati e genera un output non interpretato.
- **Logica di Presentazione [View]:** Interfaccia grafica (GUI) o Vista (View), è il blocco che si interfaccia direttamente con l'utente finale. Qui vengono visualizzati ed interpretati tutti gli output generati dal sistema.

# Cosa significa MVC

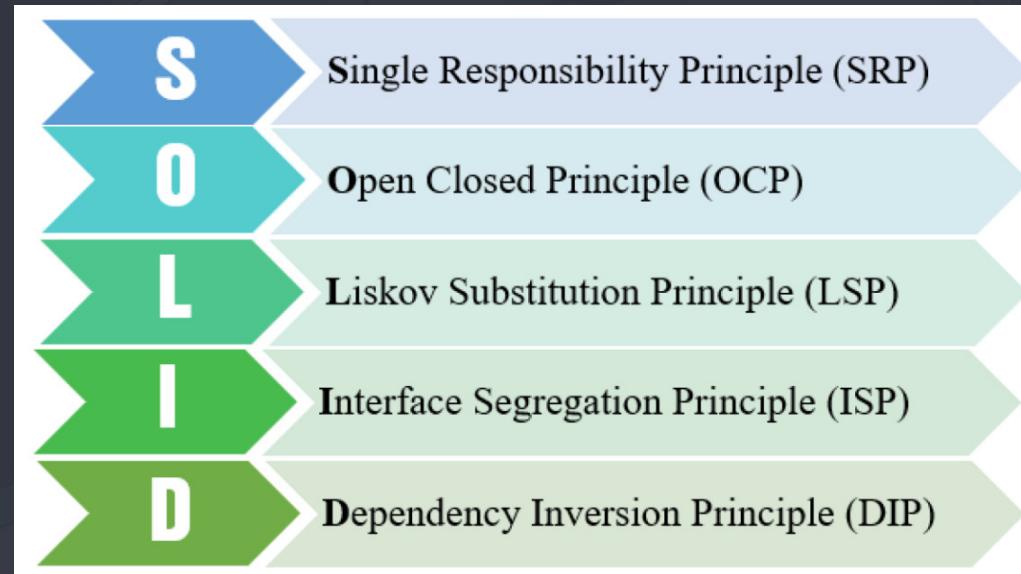


# Cosa significa MVC in Laravel?



# Principi S.O.L.I.D.

Di seguito i principi SOLID della programmazione ad oggetti.



# Principi S.O.L.I.D.

**[S] Single responsibility principle (o principio di singola responsabilità):** Ogni classe dovrebbe avere una ed una sola responsabilità. Se stai realizzando una logica complessa, separa tutto in piccole funzioni che restituiscono un risultato.

Ne gioverai sia per quanto riguarda la robustezza del codice, in caso di problemi potrai isolare meglio la parte incriminata e correggerla sia per il riuso del codice.

Fonte: <https://accesto.com/blog/solid-php-solid-principles-in-php/>

# Principi S.O.L.I.D.

**[O] Open/closed principle (o principio aperto/chiuso):** Ragiona sempre secondo il principio di scalabilità. Ovvero un'entità software dovrebbe essere aperta alle estensioni, ma chiusa alle modifiche.

Esempio: Hai creato una funzione per leggere un .csv contenente dei dati. E se il fornitore dovesse modificarlo con un .json? Estendere questa nuova funzionalità è possibile o dovrai mettere mano al codice sorgente e modificarlo?

Fonte: <https://accesto.com/blog/solid-php-solid-principles-in-php/>

# Principi S.O.L.I.D.

**[L] Liskov substitution principle (o principio di sostituzione di Liskov):** Questo principio ha lo scopo di guidare il processo di progettazione delle classi all'interno delle gerarchie con l'obiettivo principale di non creare classi figlie che hanno sempre meno in comune con il genitore, ma che richiedono solo alcuni metodi della classe genitore.

Ovvero, se estendiamo una classe dobbiamo assicurarci che questa estensione mantenga lo scopo e l'integrità della classe principale. In caso contrario dobbiamo creare una nuova classe.

Fonte: <https://accesto.com/blog/solid-php-solid-principles-in-php/>

# Principi S.O.L.I.D.

[I] Interface segregation principle (o principio di segregazione delle interfacce): Sorvoliamo

Fonte: <https://accesto.com/blog/solid-php-solid-principles-in-php/>

# Principi S.O.L.I.D.

**[D] Dependency inversion principle (o principio di inversione delle dipendenze):** Sviluppare processi logici troppo elaborati fuori dalla classe primaria non è mai buona prassi, in quanto quello stesso sviluppo potrebbe essere utile ad un'altra sottoclasse.

Fonte: <https://accesto.com/blog/solid-php-solid-principles-in-php/>

# Cenni: Eloquent ORM

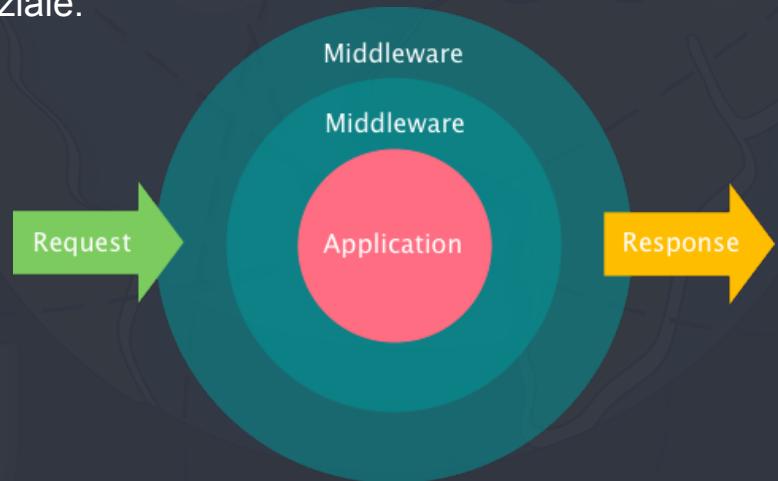
**ORM (Object-Relational Mapping)** è un'interfaccia che fornisce, tramite programmazione a oggetti, tutti i servizi inerenti alla persistenza dei dati astraendo le caratteristiche del DBMS usato. In parole poche **ce ne freghiamo del tipo di DB!**

Possiamo gestire le entità e le relazioni di un DBMS direttamente tramite Laravel con Eloquent in modo agevole scrivendo poche righe di codice e soprattutto chiare, interfacciarsi con un linguaggio piuttosto che un altro non ci interessa.

# Cenni: Middleware

Una applicazione web, in fin dei conti, non è altro che un'interfaccia grafica in grado di lanciare una serie di chiamate HTTP al server contenenti un input e, di tutta risposta, il server stesso si prodigherà nel comunicare l'output al browser/client.

Il Middleware (o più middleware) si pone in mezzo a questo scambio di informazioni, una specie di filtro tra il Client e il Server, in modo tale da evitare sovraccarichi inutili al server per chiamate non previste dal flow iniziale.



# Cenni: Facades

Le facciate (traduzione letteraria), non sono altro che un'interfaccia semplificata di un sottosistema più complesso.

Tramite l'assegnazione di un Aliases, sarà più facile richiamare e ricordare la classe prescelta.

Il concetto di Facade è forse l'argomento più amato e odiato al tempo stesso. Il codice è subito leggibile e facilmente interpretabile grazie alle facade. Ma la continua staticizzazione dei metodi per molti è un limite in termini di performance.



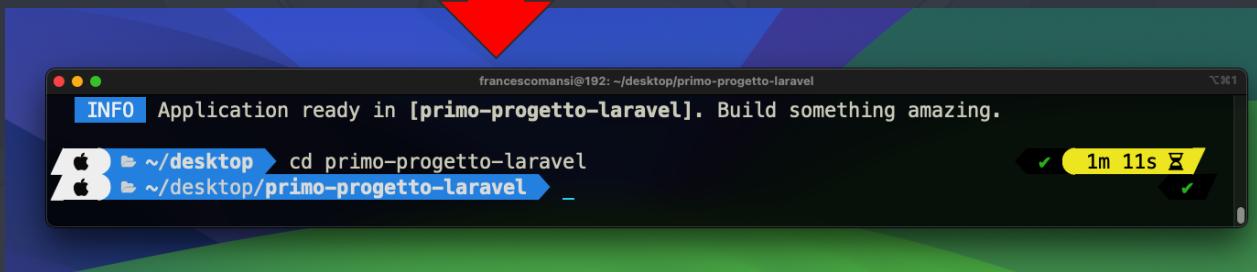
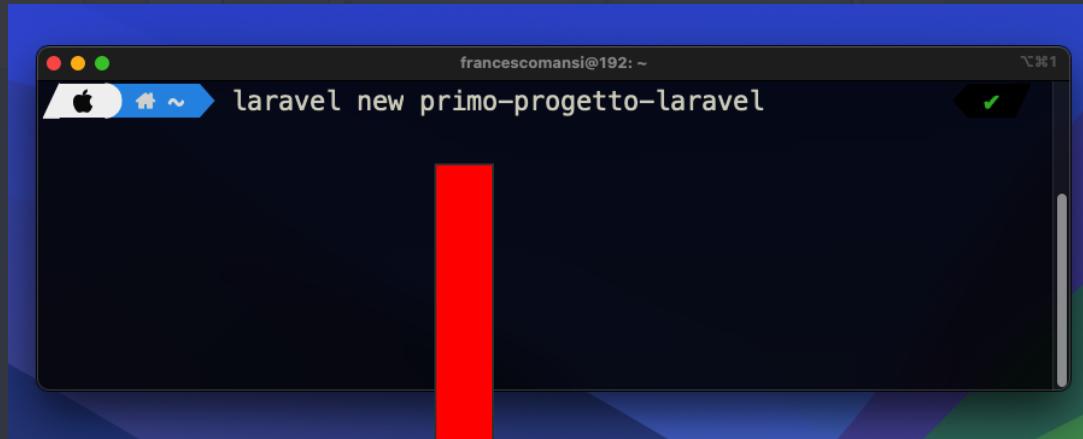
```
Test::print('ciao');
```

# Il primo Progetto Laravel

Ci serviranno:

- PHP (php -v)
- COMPOSER (composer -v)
- LARAVEL INSTALLER (laravel -v)

# Il primo Progetto Laravel 2

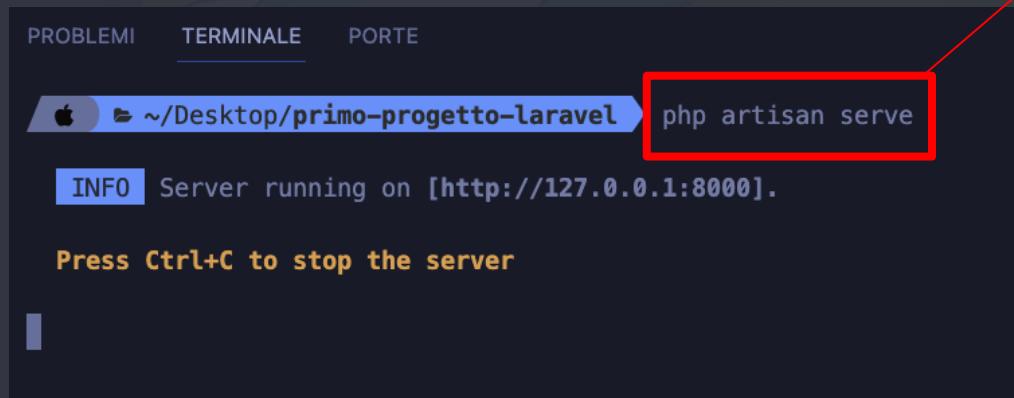


# Comandi Artisan

```
git ⚡ master ➔ php artisan
```

<https://laravel.com/docs/12.x/artisan#main-content>

# Lancia il server locale



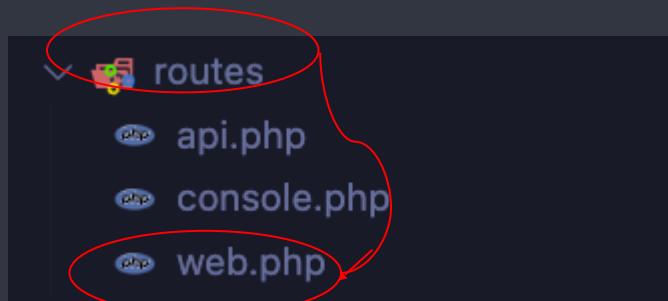
The image shows a screenshot of a terminal window. At the top, there are three tabs: 'PROBLEMI', 'TERMINALE' (which is underlined), and 'PORTE'. Below the tabs, the terminal prompt shows an Apple icon followed by the path `~/Desktop/primo-progetto-laravel`. To the right of the path, a red rectangular box highlights the command `php artisan serve`. Below the command, the terminal displays the output: `[INFO] Server running on [http://127.0.0.1:8000].` and `Press Ctrl+C to stop the server`.

```
PROBLEMI TERMINALE PORTE
~/Desktop/primo-progetto-laravel php artisan serve
[INFO] Server running on [http://127.0.0.1:8000].
Press Ctrl+C to stop the server
```

# Le Routes

Laravel implementa un fantastico sistema di routing, di facile utilizzo e comprensione.  
Non importa quanti percorsi avrai o vorrai creare, tutto è gestito dal file:

**/routes/web.php**



```
<?php  
  
use Illuminate\Support\Facades\Route;  
  
Route::get('/', function () {  
    return view('welcome');  
});
```

# Le Routes

Per definire una routes, la sintassi generale sarà:



```
Route::facades ($uri, $callback);
```

**Route:** è la classe principale;

**Facades:** è il metodo statico utilizzato per accedere alla risorsa (GET, POST, PUT, VIEW ecc..);

**\$uri:** l'indirizzo richiamato nel browser;

**\$callback:** la chiamata da associare alla route (Un controller, una funzione ecc..).

# Routes e Funzioni

Appena configurato troverai la seguente route abilitata:



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

# Pagina 404

Per gestire una pagina 404 occorre creare il file views in:

**resources/views/errors/404.blade.php**

# Le Routes - Parametri

Nulla ti vieterà di rendere le tue rotte dinamiche.

Potrai inserire tutti i parametri che desideri:



```
Route::get('/profile/{id}', function ($parametro_formale_id) {
    return $parametro_formale_id;
});
```

# Routes name

Una buona abitudine è quella di assegnare sempre un nome ad ogni Route creata.

Oltre che rispettare il principio di integrità all'interno del progetto (*trattando la route come una costante con un nome univoco, potrai cambiare tranquillamente l'URI in web.php senza problemi*) renderà il tuo progetto più chiaro e pulito.

Ad esempio, una route del tipo:

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

Potrà essere richiamata utilizzando semplicemente **homepage**.

```
● ● ●  
Route::get('/', function () {  
    return view('welcome');  
})->name('homepage');
```

# Richiamare routes name

- □ ×

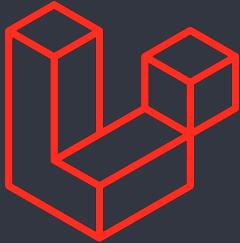
```
<a href="{{route('homepage')}}"> Link </a>  
<a href="{{route('profile', ['id' => 4])}}"> Link </a>
```

# Route List

Con il comando `artisan route:list` potrai avere sempre il polso della situazione delle route attive sul tuo progetto.



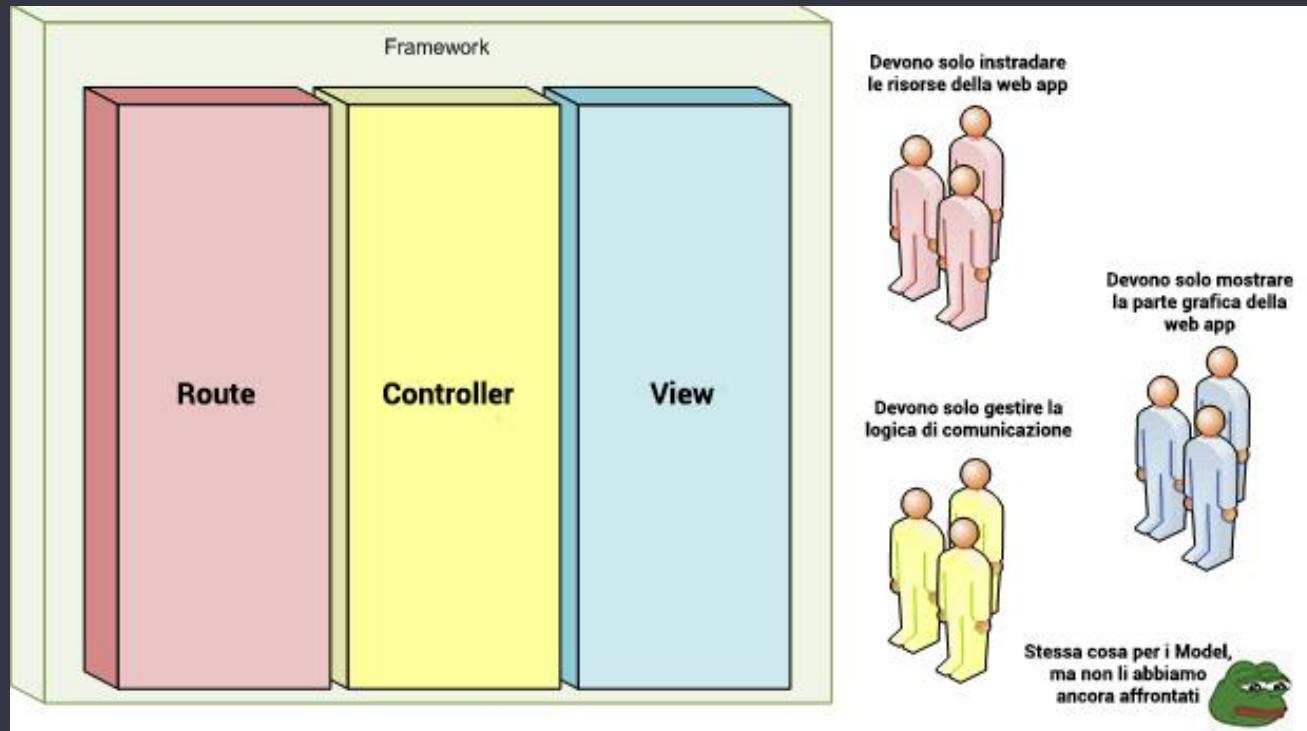
```
php artisan route:list
```



# Modulo Laravel Controllers

# SoC

La SoC (Separation of Concerns - Separazione delle Competenze) è un principio fondamentale nell'ingegneria del software che mira a suddividere un sistema complesso in parti più piccole e gestibili, dove ogni parte si concentra su un aspetto specifico del sistema, senza intaccare le altre parti



# I Controller

Nel precedente capitolo abbiamo inserito tutta la parte logica nelle routes: **Dimenticate tutto, non è una best practices.**

Utilizzare le routes per la parte logica cozza con il principio di singola responsabilità (S. di SOLID) visto nel primo capitolo.

Le routes nascono per instradare i controller e dovranno occuparsi solamente di quello.

I controller, invece, si occuperanno di tutta la logica di business della nostra applicazione web.

Potrai trovare tutti i controller predefiniti nella directory predefinita app/Http/Controllers .

Quando utilizzerai il comando Artisan, ogni controller che creerai verrà posizionato in quel percorso:

```
● ● ●  
php artisan make:controller NomeController
```

# I Controller

Inoltre, tutti i controller generati saranno un'estensione del controller base situato in  
app/Http/Controllers/Controller.php

Che a sua volta sarà un'astrazione del BaseController situato nel core di Laravel.  
Ciò significa che avrà accesso a tutti i metodi e Helper del core.

# I Controller

Il controller normale viene generato mediante comando:



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

E genererà un codice simile:



```
<?php  
  
namespace App\Http\Controllers;  
  
use App\Http\Controllers\Controller;  
  
class NomeController extends Controller  
{  
    //  
}
```

# Basic Controller



```
<?php  
  
namespace App\Http\Controllers;  
  
use App\Http\Controllers\Controller;  
  
class UserController extends Controller  
{  
  
    public function index()  
    {  
        return view('index');  
    }  
}
```



```
use App\Http\Controllers\UserController;  
  
Route::get('/index', [UserController::class, 'index']);
```

# Extra - dd()



# Extra - dd()

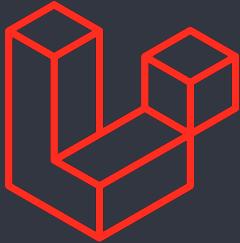
```
$variabile = 'Test';

//PHP puro

var_dump($variabile);
die();

//Laravel

dd($variabile);
```



# Corso Laravel

# View & Components

# Le views

La componente view ( o logica di presentazione) è la parte in cui l'utente finale visualizza i dati manipolati e interpretati dal controller.

In pochissime parole: la view è il codice html caricato dal browser.

# Blade

Blade è il template engine predefinito di Laravel.

Un file blade non è altro che una view scritta in php. Dovrai aggiungerci soltanto una pre-estensione  
view.blade.php

A differenza degli altri template engine, Blade non ha alcuna restrizione sul PHP. Potrai scrivere direttamente porzioni di codice nativo senza problemi.

Di default tutte le views dovrai inserirle nel percorso resources/views .

Da qui, verranno inizialmente compilate e, successivamente, memorizzate nella cache di sistema (storage/framework). Le views non verranno processate e compilate ogni volta, ma semplicemente caricate e lette in modo statico.

# Richiamare una View

```
● ● ●

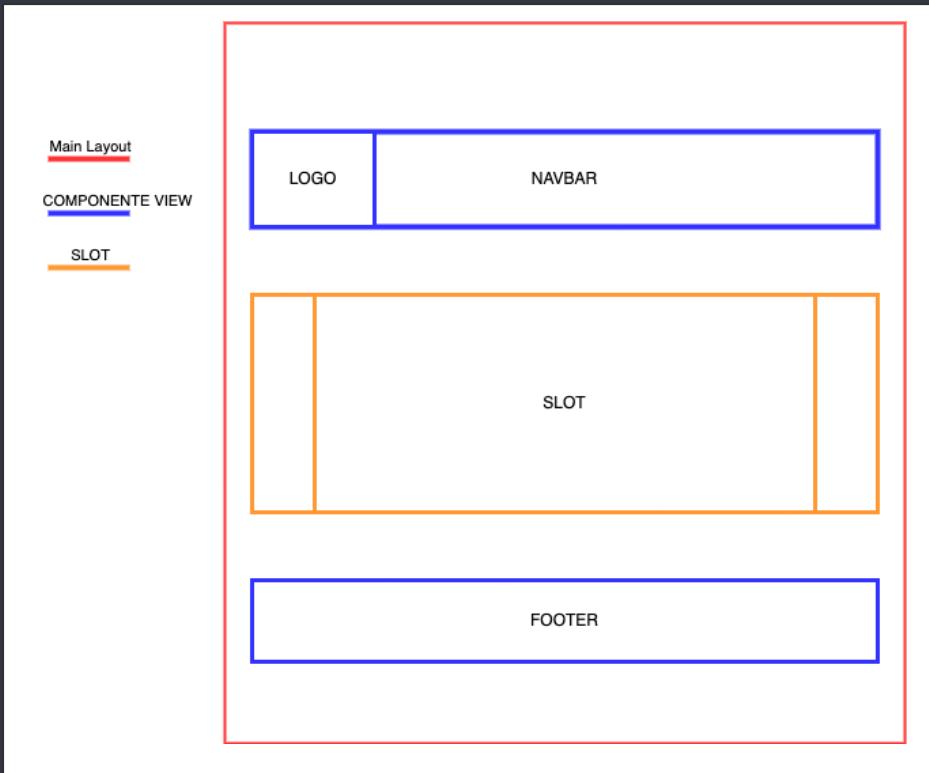
<?php
namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
use App\User;

class UserController extends Controller
{
    public function profile($id){
        return view('welcome');
    }
}
```

## Come passare i dati ad

```
public function homepage(){
    $data = [
        'name' => 'Francesco',
        'country' => 'ITA'
    ];
    return view('homepage', ['data' => $data]);
    return view('homepage')->with('data', $data);
    return view('homepage', compact('data'));
}
```

# A cosa serve un Layout



# Strutturare un Layout

Il comando appena lanciato andrà a creare la view in  
resources/views/components/main.blade.php

E andremo ad inserire la variabile \$slot in quanto quello sarà il nostro segnaposto che, come un portale (Avete mai visto la serie TV Stargate?), stamperà il contenuto ereditato.

```
● ● ●  
  
<body>  
<main>  
    {{$slot}}  
</main>  
<footer>Francesco Mansi</footer>  
</body>
```

# Strutturare un Layout

Ora, non ci resta che creare tutte le pagine che andranno ad estendere quel layout. Come? In questo modo:

```
● ● ●  
<x-main-layout>  
  <h1>Ciao</h1>  
</x-main-layout>
```

# Strutturare un Layout

Il risultato finale sarà questo:

```
//File finale - HTML
<body>
  <main>
    <h1>Ciao</h1>
  </main>
  <footer> Francesco Mansi </footer>
</body>
```

# A cosa serve un componente

The image displays three screenshots of a website illustrating the concept of component reuse:

- Homepage:** Shows a dark header with "Palestra Inizio da Gennaio". Below it is a large banner with the heading "Sempre Aperti dopo le festività". The main content area features a grid of cards labeled "A better way to start building." and "Corso disponibili". Three specific cards in the "Corso disponibili" grid are circled in red.
- Elenco Corsi:** Shows a list of courses. The first card in the list is circled in red. Below the list is a "New products, delivered to you." section.
- Dettaglio Corso:** A detailed view of a course. The top card is circled in red. Below it is a "New products, delivered to you." section.

**3 pagine diverse, stesso componente**

# Strutturare un Layout

Laravel questa volta copia a piene mani dai linguaggi frontend come Vue e React, appropriandosi della logica dei componenti.

Sempre rispettando il principio della segregazione delle interfacce e di singola responsabilità (SOLID) utilizzare una struttura a componenti ti permetterà di scrivere sempre meno codice e di qualità.



```
php artisan make:component Main --view
```

# Passare i dati nel componente

```
<x-card :data="$variabile_php" :altro="'Stringa da inserire'"/>
```

# Passare i dati nel componente

Short Syntax

```
<x-profile :userId :name />  
{{-- Equivale --}}  
  
<x-profile :user-id="$userId" :name="$name" />
```

# Slot principale

Ogni componente, di default, ha uno \$slot disponibile:

```
● ● ●  
  
<body>  
  <main>  
    {{$slot}}  
  </main>  
</body>
```

```
● ● ●  
  
<x-app-layout>  
  Contenuto da Stampare  
</x-app-layout>
```

# Slot Custom

Ogni componente, di default, ha uno \$slot disponibile:

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()→getLocale()) }}">
    <head>
        {{ $title }}
    </head>
    <body >
        <div>

            <main>
                {{ $slot }}
            </main>
        </div>
        {{ $error }}
    </body>
</html>
```

```
<x-app-layout>
    <x-slot name="header">
        Titolo
    </x-slot>

    <x-slot:error>
        Server Error
    </x-slot>
</x-app-layout>
```

# EXTRA: Comandi Blade

IF condizionale con @if, @else, @endif

Creare una struttura di controllo IF risulterà molto semplice. Ti basterà anteporre la chioccaia al metodo di controllo per utilizzarlo immediatamente.

```
● ● ●  
@if (count($var) === 1)  
    Risultato 1  
@elseif (count($var) > 1)  
    Maggiore di 1  
@else  
    Vuoto  
@endif
```

# EXTRA: Comandi Blade

Nullo o vuoto con @isset o @empty

Anche i famosissimi metodi PHP sono disponibili in Blade, potrai richiamarli in questo modo:

```
@isset($var)
    // $var è settato e non nullo
@endisset
@empty($var)
    // $var è vuoto
@endempty
```

# EXTRA: Comandi Blade

Gestire tutti i casi con @switch, @case, @break, @default

Potresti voler confrontare una variabile su più condizioni, valori diversi e con risultati diversi. La struttura switch fa al caso tuo, ecco come implementarla:

```
● ● ●  
@switch($age)  
    @case($age > 0 && $age < 19)  
        //Minorenne  
    @break  
    @case($age > 18 && $age < 100)  
        //Maggiorenne  
    @break  
    @default  
@endswitch
```

# EXTRA: Comandi Blade

## Uno sguardo ai Loop

Blade semplifica anche le strutture classiche di PHP, come i cicli foreach, for e while.  
Eccone un'assaggio:

```
● ● ●

@for ($i = 0; $i < 10; $i++)
    Il valore è {{ $i }}
@endfor

@foreach ($users as $user)
    <p>Ciao, {{ $user->name }}</p>
@endforeach

@forelse ($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>Nessun utente</p>
@endforelse

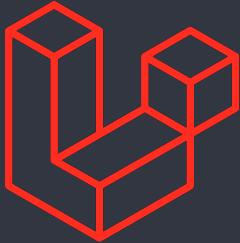
@while (true)
    <p>Verso l'infinito e oltre!</p>
@endwhile
```

# EXTRA: Comandi Blade

Dentro un ciclo foreach potrai accedere a tantissimi metodi utili:

- `'\$loop->index'`: L'indice corrente (inizia da 0);
- `'\$loop->iteration'`: L'iterazione del loop corrente (inizia da 1);
- `'\$loop->remaining'`: Quanti indici mancano alla fine;
- `'\$loop->count'`: Il numero totale degli elementi;
- `'\$loop->first'`: Se questa è la prima iterazione del ciclo;
- `'\$loop->last'`: Se questa è l'ultima iterazione del ciclo;
- `'\$loop->even'`: Se questa è un'iterazione pari;
- `'\$loop->odd'`: Se questa è un'iterazione dispari;
- `'\$loop->depth'`: Il livello di annidamento del loop corrente.
- `'\$loop->parent'`: Quando si trova in un ciclo nidificato, la variabile del ciclo del genitore.

```
● ● ●  
@foreach ($users as $user)  
  
    @if ($user->type == 1)  
        @continue  
    @endif  
    <li>{{ $user->name }}</li>  
    @if ($user->number == 5)  
        @break  
    @endif  
@endforeach
```



# Corso Laravel Assets & Building

# Installazione

Installare Node sul proprio pc. Verificare nel terminale che i comandi funzionino:

node -v

npm -v

# NPM

Per prima cosa occorrerà installare Tutti i modulo presenti in package.json:

```
"devDependencies": {  
    "axios": "^1.6.4",  
    "laravel-vite-plugin": "^1.0",  
    "vite": "^5.0"  
}
```

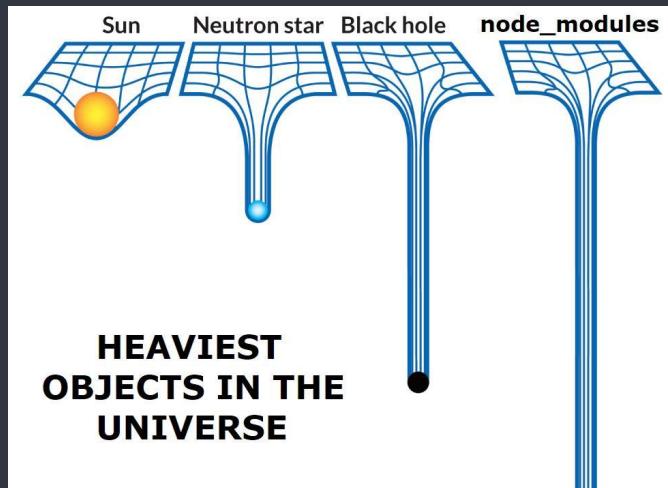
Quindi lancia nel terminale:



```
npm install
```

# NPM

Attraverso <https://www.npmjs.com/> sarà possibile scaricare tutti i package javascript esistenti.



# Bootstrap

Successivamente avrai bisogno di Bootstrap.

Da terminale, lanciare:

Install

```
> npm i bootstrap
```

# Importare le Librerie CSS

Apri il file:

resources/css/app.css

E importa i file di Bootstrap in questo modo:

```
1  
2  
3 @import 'bootstrap';  
4 |
```

# Importare le Librerie JS

Apri il file:

resources/js/app.js

E importa i file di Bootstrap in questo modo:

```
import 'bootstrap';
```

Senza chiocciola questa volta.  
Lascia intatto tutto il resto

# Richiamare Vite in Blade

Ultimo step è importare @Vite nel Main Layout principale del progetto:

```
1 <!doctype html>
2 <html lang="en">
3
4 <head>
5     <!-- Required meta tags -->
6     <meta charset="utf-8">
7     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
8
9     @vite(['resources/css/app.css', 'resources/js/app.js'])
10
11 <body>
```

# Comandi npm

Sono disponibili due comandi:

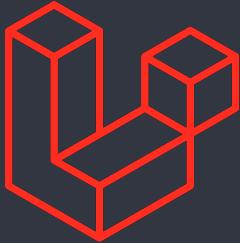
npm run dev

Oppure quando il progetto è pronto per l'ambiente produzione

npm run build

Oppure ancora

composer run dev



# Corso Laravel

Request & Form

# I Form

L'elemento form ha come parametri richiesti:

- action;
- method;
- Tutti gli input devono avere un name;
- I pulsanti devono essere di tipo submit.
- Il value avrà il valore dell'input
- Utilizzando {{old('name-value')}}
- recuperiamo il dato in sessione

```
● ● ●

<form action="" method="">>

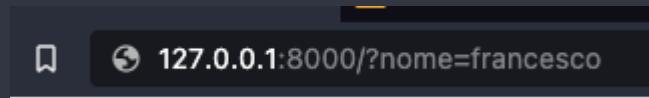
    <div class="mb-3">
        <label class="form-label">Nome</label>
        <input class="form-control" type="text" name="name" readonly
value=""
            placeholder="Nome Utente" />
    </div>

    <div class="d-grid">
        <button class="btn btn-primary btn-lg"
type="submit">Invia</button>
    </div>

</form>
```

# Invio dati via Get e Post

- Con il metodo GET i dati viaggiano nell'URL



- Con il metodo POST i dati viaggiano nel BODY



# Metodo POST e Requests

Come saprai, sfruttando il protocollo HTTP è possibile gestire gran parte delle operazioni di input, cookie, server e molto altro (dispositivo utilizzato, geolocalizzazione, browser ecc ..)

Per prima cosa dovrà importare la classe `Illuminate\Http\Request`.  
Ti sarà utile anche per gestire le variabili che passerai in POST.

```
use Illuminate\Http\Request;

Route::post('/', function (Request $request) {
    dd($request→all());
});
```

# Comandi Requests

## Dati del Client

- \$request->server()
- \$request->cookie()
- \$request->headers()
- \$request->path()
- \$request->url()
- \$request->fullUrl()
- \$request->isMethod('post')

## Gestione dei dati ricevuti in POST

- \$request->flash();
- \$request->all()
- \$request->input();
- \$request->input('name')
- \$request->input('name','francesco')
- \$request->has('name')
- \$request->query();
- \$request->query('search');
- \$request->has('name')
- \$request->merge(['votes' => 0]);
- \$request->mergeIfMissing(['votes' => 0]);

# Errore 419 e CSRF



```
<form method="POST" action="">  
    @csrf  
</form>
```

# Errori comuni

1. Crsf mancante, errore 419 e token in ispezione cosa genera
2. Name mancanti, request vuota
3. \$request senza dependency injection

# Metodo send



```
public function send(Request $request)
{
    $data = [
        "firstname" => $request->firstname,
        "lastname" => $request->lastname,
        "email" => $request->email,
    ];
}
```

# Validazione

Con le validation rules è possibile validare le request.

Al primo errore, il controller andrà in return in quanto  
TUTTE le regole devono essere rispettate.

<https://laravel.com/docs/11.x/validation#validation-quickstart>

```
● ● ●  
  
$request→validate([  
    'name' => 'required|max:255',  
    'email' => 'required|email',  
]);
```

# Visualizzare Errori di Validazione

Per visualizzare gli errori in Blade,  
utilizza questo piccolo snippet:

```
@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

# Creare Classe Mail



```
php artisan make:mail InfoMail
```

# Classe Mail



```
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Mail\Mailable;
use Illuminate\Mail\Mailables\Content;
use Illuminate\Mail\Mailables\Address;
use Illuminate\Mail\Mailables\Envelope;
use Illuminate\Queue\SerializesModels;

class InfoMail extends Mailable
{
    //Contenuto
}
```



```
<?php

use Queueable, SerializesModels;

public $content;

public function __construct($contenuto)
{
    $this->content = $contenuto;
}

public function envelope()
{
    return new Envelope(
        from: new Address('salando@email.it', 'Jeffrey Way'),
        subject: 'Ordine inoltrato',
    );
}

public function content()
{
    return new Content(
        view: 'emails.email',
    );
}

public function attachments()
{
    return [];
}
```

# Vista email



```
<p>
    Ti ha L'utente di nome {{$content['firstname']}}
    {{$content['lastname']}}
    con email {{$content['email']}}
</p>
```

# Aggiunta classe email



```
public function send(Request $request)
{
    $data = [
        'firstname' => $request->firstname,
        'lastname' => $request->lastname,
        'email' => $request->email
    ];

    Mail::to($request->email)->send(new ContactMail($data));
    return redirect()->route('homepage');
}
```

# Mailtrap



**mailtrap**  
by railsware

1. Andare su [mailtrap.io](https://mailtrap.io)
2. Nella Inboxes, copiare la configurazione per Laravel e incollarla in .env
3. Testare l'invio

# Return redirect



```
return redirect()→route('thank-you')
```

# HTTP Session - Flash data

Http, come detto è Stateless.

Ovvero non mantiene i dati tra una chiamata e l'altra. Ma Laravel (e PHP) si.

<https://laravel.com/docs/11.x/session>

# Flash data

Nome variabile  
sessione

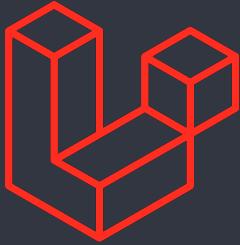
Contenuto variabile  
sessione

```
return redirect()->back()->with('success',  
    'Email Inviata');
```

# Flash data

Nome variabile  
da richiamare

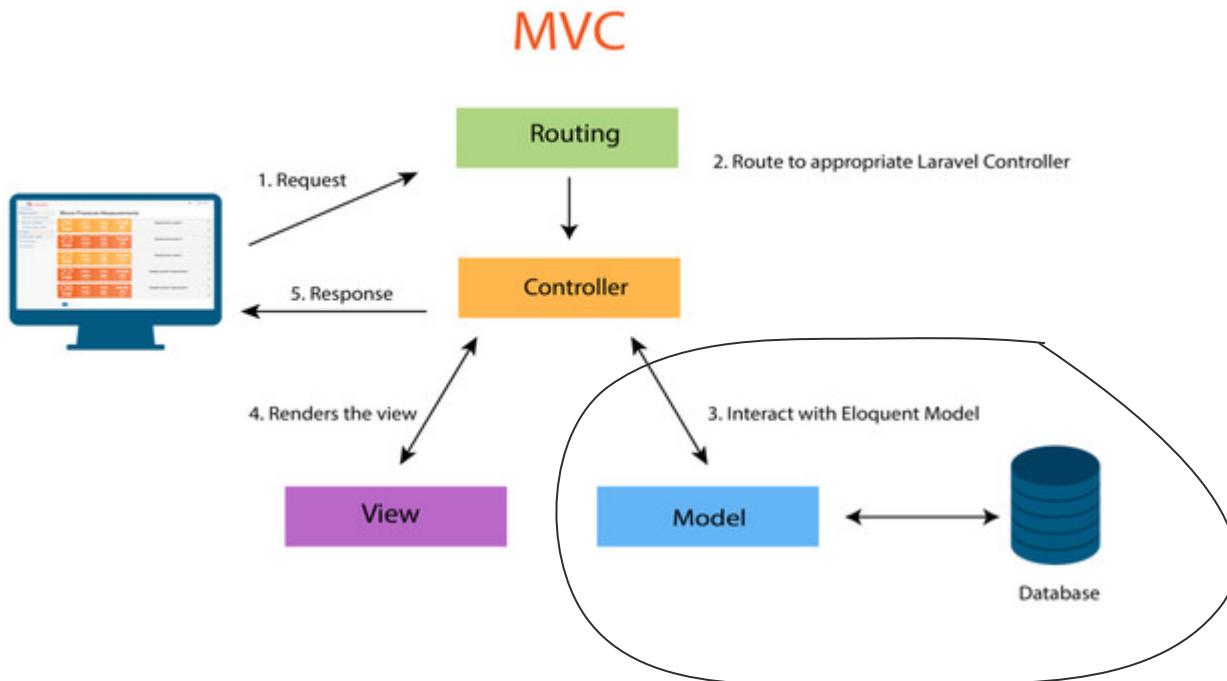
```
@if(session('success'))  
<div>  
    {{session('success')}}  
</div>  
@endif
```



# Corso Laravel

## Introduzione Database

# Model - Database

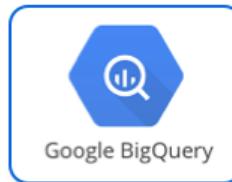


# Database

DATABASE (Base dati), abbreviato in DB, è un insieme di dati correlati tra di loro immagazzinati fisicamente su di un supporto fisico.

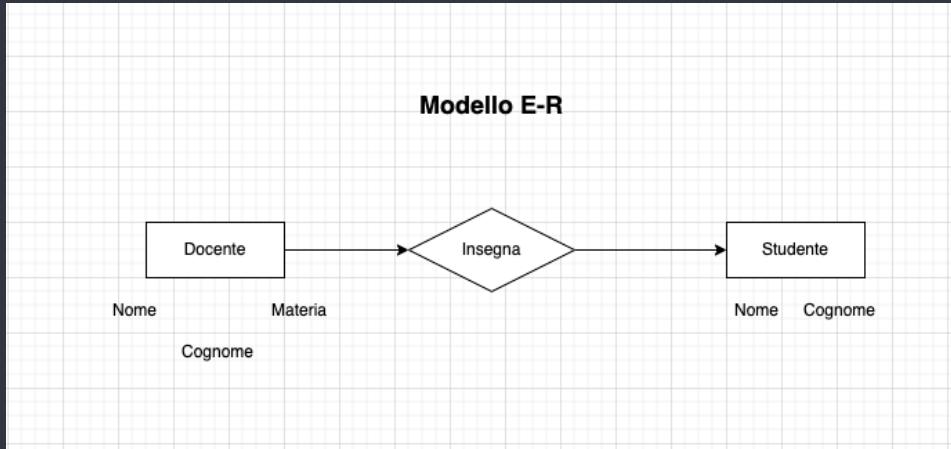
- Per "dati" si intendono dei fatti noti ed empirici, che possono essere memorizzati
- Es: nome, cognome, indirizzo e telefono di un abbonato telefonico

# Database



# Tipi di Database

**Relazionali:** Un database relazionale, come ci suggerisce il nome, è basato sul modello entità – relazione (E-R).



# Tipi di Database

**Non relazionali:** Un database non-relazionale, non utilizza la logica delle relazioni ma sono dei documenti strutturati sotto forma di elenco organizzato

```
● ● ●  
1  
2 {  
3   "Name" : "Sherlock Holmes",  
4   "Category" : "Action",  
5   "Length" : 123,  
6   "Actors" : [  
7     {  
8       "actorId" : 5,  
9       "Name" : "Robert",  
10      "Surname" : "Downey Jr"  
11    },  
12    {  
13      "actorId" : 6,  
14      "Name" : "Jude",  
15      "Surname" : "Law"  
16    },  
17  ]  
18 }  
19 }
```

# Un po di storia - flat file

Il primo database in assoluto si chiamava Flat File

Studenti			
<b>Nome</b>	<b>Cognome</b>	<b>Docente</b>	<b>Materia</b>
Paolo	Rossi	Professor Biagio	Educazione Fisica
Giovanni	Verdi	Professor Carlo	Storia
Antonio	Bianchi	Professor Biagio	Educazione Fisica

Docenti	
<b>Docente</b>	<b>Materia</b>
Professor Biagio	Educazione Fisica
Professor Carlo	Storia

Cosa succede se  
il Professor Biagio  
dovesse cambiare  
cattedra?

# Un po di storia - Flat File

Ridondanza e Inconsistenza

Studenti			
Nome	Cognome	Docente	Materia
Paolo	Rossi	Professor Biagio	Educazione Fisica
Giovanni	Verdi	Professor Alfredo	Storia
Antonio	Bianchi	Professor Biagio	Educazione Fisica

Docenti	
Docente	Materia
Professor Biagio	Nutrizionista
Professor Carlo	Storia

Il Professor Biagio ha cambiato cattedra

Chi è Alfredo?

# Un po di storia - Relazionale

Rimozione  
Ridondanza e  
Integrità Referenziale

Studenti			
Matricola	Nome	Cognome	Docente
S1	Paolo	Rossi	D1
S2	Giovanni	Verdi	D2
S3	Antonio	Bianchi	D1

Docenti		
Matricola	Docente	Materia
D1	Professor Biagio	Educazione Fisica
D2	Professor Carlo	Storia

# La Biblioteca.

## Un esempio di database nella vita reale

Ogni libro rappresenta un pezzo di informazione che vogliamo conservare, come una storia o delle informazioni su un argomento specifico.

Un database è come un sistema di scaffali e schedari all'interno della **biblioteca** che aiuta a organizzare tutti i libri. Ogni scaffale contiene diversi tipi di libri, ad esempio romanzi, libri di storia o libri di scienza. Ogni schedario rappresenta una specifica categoria di libri, come i libri per bambini o i libri di cucina.



# I Database nella vita reale

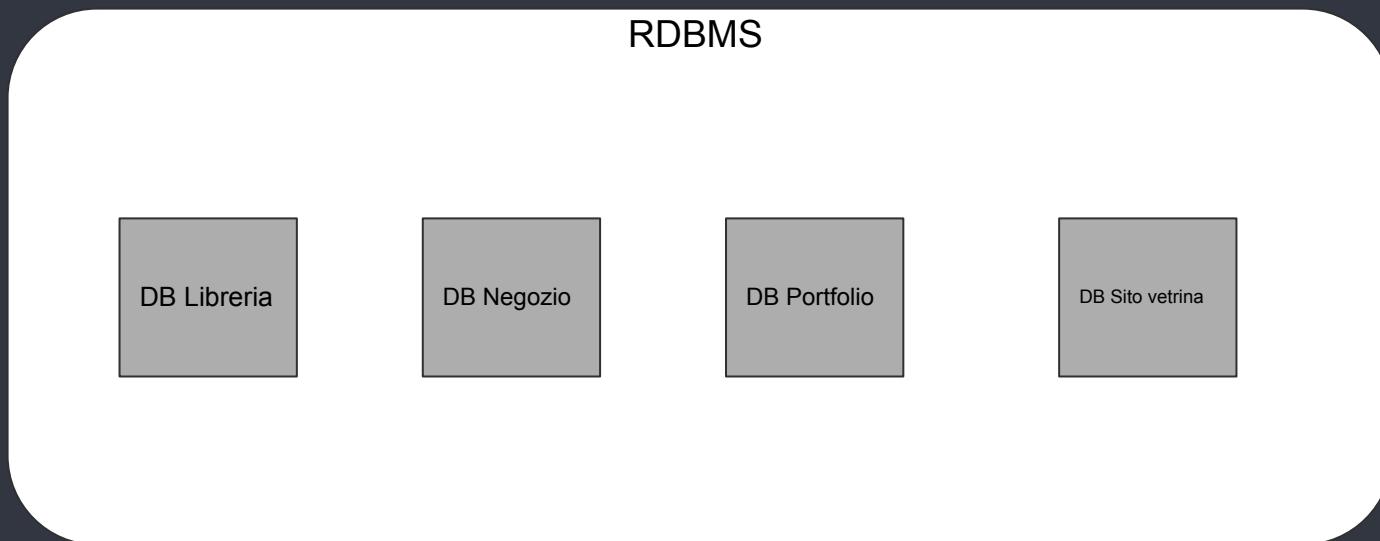
Un sistema di gestione di database, o **RDBMS**, è come un bibliotecario che gestisce tutto il sistema.

Essi si occupano di organizzare i libri, aiutare le persone a trovare i libri di cui hanno bisogno, che vengono salvati correttamente e che i libri siano restituiti correttamente.

# RDBMS

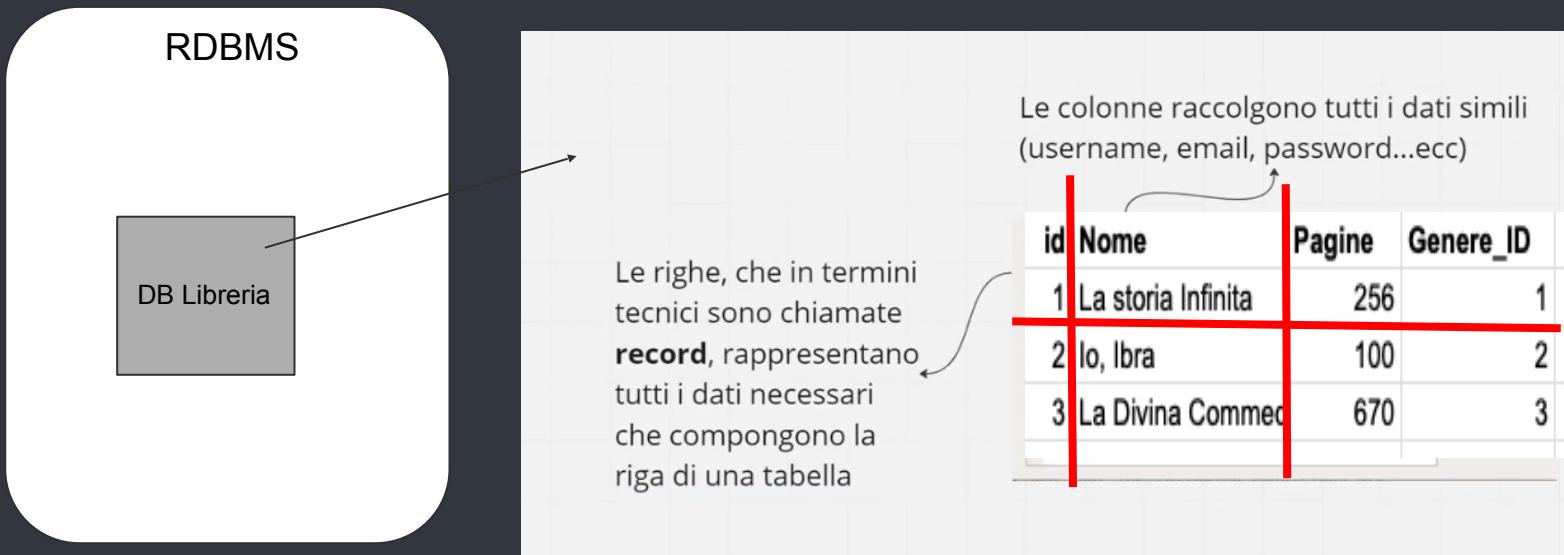
Relational database management system, ovvero è il sistema che si occupa di gestire i database.

Noi Useremo MySQL



# Tabelle

Ovviamente dentro un database i dati hanno una loro organizzazione, si utilizzano le tabelle



# Tabelle

Ogni tabella deve avere un id

<b>id</b>	<b>Nome</b>	<b>Pagine</b>	<b>Genere_ID</b>
1	La storia Infinita	256	1
2	Io, Ibra	100	2
3	La Divina Commed	670	3
4	Mio Libro	Tante	5

<b>Id</b>	<b>Nome</b>
1	Fantasy
2	Sportivo
3	Poema



# FOCUS

- Chiavi Primarie
- Chiavi Esterne

# SQL

Structured Query Language (SQL) è un linguaggio di interrogazione (query) utilizzato per creare, modificare e gestire i dati in un database relazionale.

A cosa ci serve? Per parlare con l'RDBMS (Il bibliotecario)

```
● ● ●  
SELECT *  
FROM tabella_libri  
WHERE nome = 'La storia Infinita'
```

# MySQL

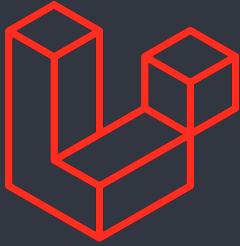
Una volta installato MySQL, digita questi comandi nel tuo terminale per accedere al RDBMS

```
#MAC  
mysql -u root -p  
  
#windows  
winpty mysql -u root -p
```

# Software Utili

Per gestire i database: <https://tableplus.com/>

Per creare i grafici: <https://dbdiagram.io/>



# Corso Laravel

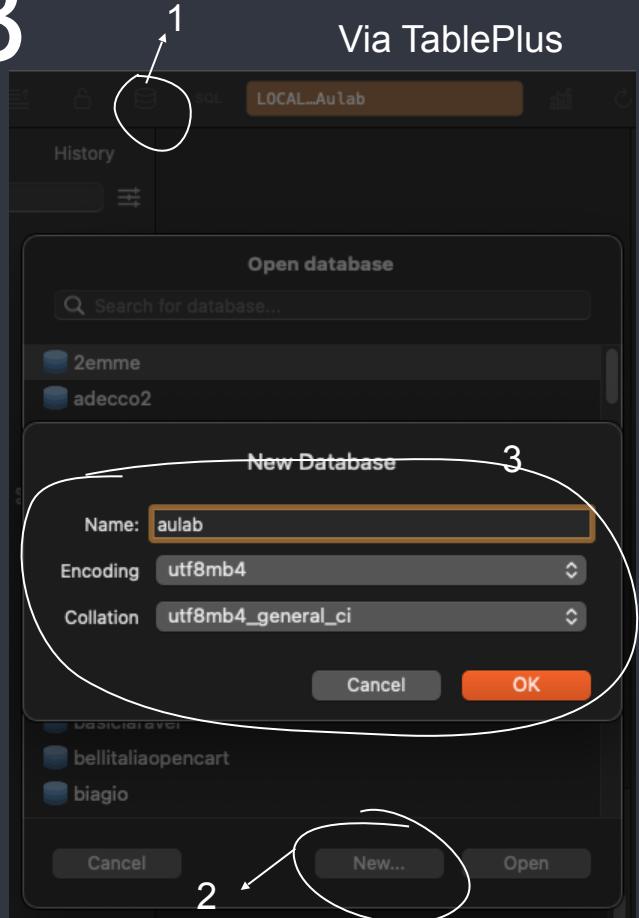
Model e Migration

# Create DB

Via TablePlus

Via terminale

```
mysql -u root -p  
Enter password:  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 14  
Server version: 5.7.24 MySQL Community Server (GPL)  
  
Copyright (c) 2000, 2023, Oracle and/or its affiliates.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
mysql> create database aulab;
```



# Collegare DB

Dentro il file .env andiamo ad impostare i dati precedentemente configurati in MySQL

```
DB_CONNECTION=mysql  
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=database  
DB_USERNAME=root  
DB_PASSWORD=root
```

# Schema Tabella

```
Table books{  
    id int [primary key]  
    name varchar(100)  
    pages int [default: null]  
    year int [default: null]  
}
```

books		
	id ↗	int
	name	varchar(100)
	pages ↙	int
	year ↙	int

# Comandi

```
● ● ●  
#MAC  
mysql -u root -p  
  
#windows  
winpty mysql -u root -p
```

# Migration per le tabelle

Utilizzare le Migrations nei tuoi progetti ti permetterà di tenere traccia, come succede in GIT, di tutte le modifiche effettuate al database da parte tua e dei tuoi colleghi.

Tutto ciò è possibile grazie alla **facade** Schema che, in modo agnostico, ti permetterà di creare e manipolare qualsivoglia tabella del Database.

Da console lancia il comando Artisan:

```
php artisan make:migration create_books_table
```

Ricordati in `database/migrations` potrai constatare tu stesso la comparsa di un nuovo file che avrà nel nome la data e l'ora di creazione concatenata al nome dichiarato nel comando, ad esempio:

AAAA\_GG\_MM\_hhmmss\_create\_books\_table.php

# Convenzioni dei nomi

Cosa	Come	Giusto	Sbagliato
Controller	singolare	ArticleController	<b>ArticlesController</b>
Route	plurale	articles/1	<b>article/1</b>
Route name	snake_case con notazione punto	users.show_active	<del>users.show-active, show-active-users</del>
Model	singolare	User	<b>Users</b>
<b>Tabelle</b>	plurale	articles	<b>article</b>

# Creare una tabella SQL

```
CREATE TABLE books (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    pages INT NOT NULL,
    year INT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

# Creare una tabella

```
● ● ●

public function up(): void
{
    Schema::create('books', function (Blueprint $table) {
        $table->id();
        $table->string('name', 100);
        $table->integer('pages');
        $table->integer('year')->nullable();
        $table->timestamps();
    });
}
```

# La nostra tabella

```
Schema::create('books', function (Blueprint $table) {
    $table->id();
    $table->string('name', 100);
    $table->integer('pages');
    $table->timestamps();
});
```

<b>id</b>	<b>name</b>	<b>pages</b>	<b>created_at</b>	<b>updated_at</b>
1	Divina Commedia	256	2023-0... ⏺	2023-05... ⏺

<https://laravel.com/docs/11.x/migrations#available-column-types>

# Up() & down()

**Migrate**: Una volta definite le tabelle lato migration dovrà lanciarle.

**Rollback**: Per ripristinare l'ultima operazione di migrazione, è possibile utilizzare il comando rollback.

DA COMPLETARE con differenza tra rollback rollout e immagiin

```
public function up(): void
{
    Schema::create('books', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}

public function down(): void
{
    Schema::dropIfExists('books');
}
```

# Modificare una tabella

Come per la creazione, per modificare delle colonne già esistenti dovrà utilizzare il metodo `table`. Il metodo appena citato accetta due argomenti: il nome della tabella e la Closure Blueprint:

```
php artisan make:migration add_to_books_table
```

# Model

Un Model non è altro che un file fisico .php in grado di astrarre il concetto di "tabella".

Scrivendo query in puro Eloquent potrai cambiare il Driver del database (MySQL, SQLite o PostgreSQL ) senza impattare sulle query.

Di default potrai trovare i Model all'interno della cartella app/Models .

Tutti i modelli Eloquent estendono la classe **Illuminate\Database\Eloquent\Model** , questo significa che avrai accesso ad un numero sconfinato di metodi.

# Model

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Book extends Model
{
    use HasFactory;

    protected $fillable = ['name', 'pages'];
}
```

# Model -> Controller

```
public function index()
{
    $books = Book::all();

    return view('index', ['books' => $books]);
}
```

# Store di un dato

Salvataggio PHP

```
<?php  
  
public function store(Request $request){  
  
    $user = new User;  
    $user->firstname = $request->firstname;  
    $user->lastname = $request->lastname;  
    $user->save();  
}  
}
```

Salvataggio Eloquent

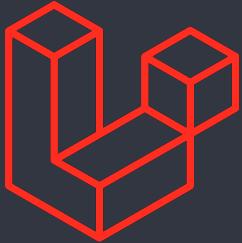
```
<?php  
  
public function store(Request $request){  
  
    //Primo Modo  
    Users::create($request->all());  
  
    //Secondo Modo  
    $request->validate([  
        'firstname'=> 'required',  
        'lastname'=> 'required',  
    ]);  
    Users::create( $request->validated() );  
  
    //Secondo Modo  
    Properties::create([  
        'nome'=> $request->firstname,  
        'cognome'=> $request->lastname,  
    ]);  
}
```

# Mass Assignment

```
protected $fillable = [  
    'name', 'pages'  
];
```

```
public function store(){  
    Books::create([  
        'name' => 'Libro 1',  
        'pages' => 45,  
        'gratis' => true  
    ])  
}
```

"gratis" verrà ignorato



# Corso Laravel

## Validation Rules

## URI

127.0.0.1:8000/libri

## Route

```
Route::get('/libri', [BookController::class, 'index'])->name('books.index');
```

## Controller

```
public function index()
{
    $books = Book::all();
    return view('index', compact('books'));
    //se chiave valore sono uguali, possiamo usare compact()
    //return view('index', ['books' => $books]);
}
```

## View

- La Divina Commedia - Dante Alighieri
- Le Avventure di Pinocchio - Collodi

# Recap MVC

## Tabella

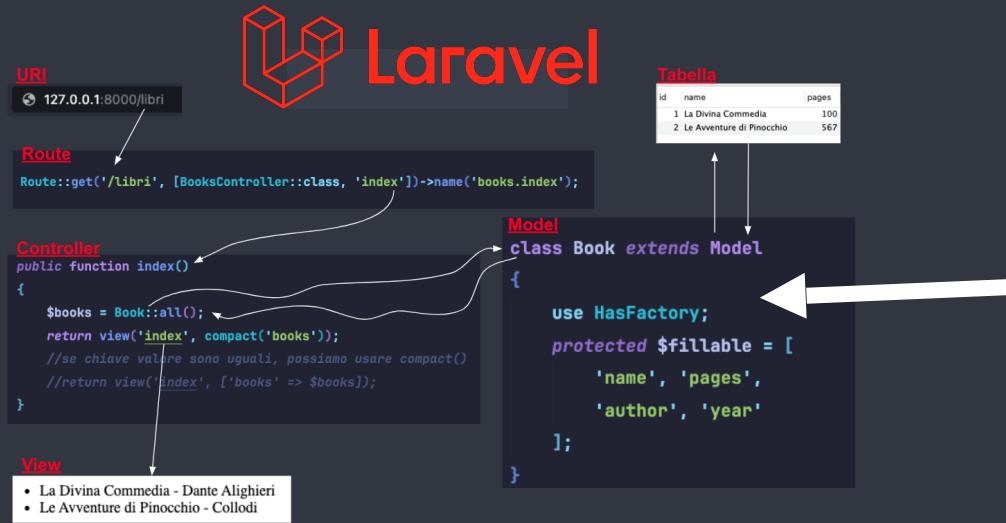
id	name	pages
1	La Divina Commedia	100
2	Le Avventure di Pinocchio	567

## Model

```
class Book extends Model
{
    use HasFactory;

    protected $fillable = [
        'name', 'pages',
        'author', 'year'
    ];
}
```

# Importanza dei Form



Name

Email Address

Country

Phone

Password

I accept terms & conditions

Create Account



# Recap sui Form

L'elemento form ha come parametri richiesti:

- action;
- method;
- Tutti gli input devono avere un name;
- I pulsanti devono essere di tipo submit.
- Il value avrà il valore dell'input
- Utilizzando {{old('name-value')}}
- recuperiamo il dato in sessione

```
● ● ●

<form action="" method="">

    <div class="mb-3">
        <label class="form-label">Nome</label>
        <input class="form-control" type="text" name="name" readonly
value=""
            placeholder="Nome Utente" />
    </div>

    <div class="d-grid">
        <button class="btn btn-primary btn-lg"
type="submit">Invia</button>
    </div>

</form>
```

# Validazione

Con le validation rules è possibile validare le request.

Al primo errore, il controller andrà in return in quanto  
TUTTE le regole devono essere rispettate.

<https://laravel.com/docs/12.x/validation#validation-quickstart>



```
$request→validate([
    'name' => 'required|max:255',
    'email' => 'required|email',
]);
```

# Visualizzare Errori di Validazione

Per visualizzare gli errori in Blade,  
utilizza questo piccolo snippet:

```
@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

# Classe request

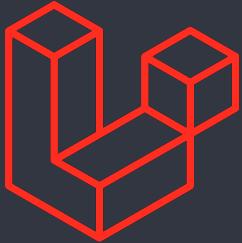


```
php artisan make:request BookRequest
```

```
public function authorize(): bool
{
    return true;
}

public function rules(): array
{
    return [
        'name' => 'required|string',
        'author' => 'required',
    ];
}

public function messages(): array
{
    return [
        'name.required' => 'Il nome è richiesto',
        'name.string' => 'Il nome deve essere una stringa',
        'author.required' => 'Autore Obbligatorio',
    ];
}
```



# Corso Laravel

## File Upload

# Migrazione

```
protected $fillable = ['name', 'author', 'pages', 'image'];
```



```
php artisan make:migration add_to_books_table
```

Per creare questa tabella, dovete lanciare il comando `php artisan migrate`

```
/**  
 * Run the migrations.  
 */  
  
public function up(): void  
{  
    Schema::table('books', function (Blueprint $table) {  
        $table->string('image')->after('author')->nullable();  
    });  
}  
  
/**  
 * Reverse the migrations.  
 */  
  
public function down(): void  
{  
    Schema::table('books', function (Blueprint $table) {  
        $table->dropColumn('image');  
    });  
}
```

# Input di tipo file



```
<input class="form-control" id="image" name="image" type="file">
```

# Form enctype



```
<form action="{{route('store')}}" method="POST" enctype="multipart/form-data">
```

# Salvare la request NEW

Utilizzando \$request possiamo accedere a tutti i metodi per manipolare le immagini e salvarle



```
$path_image = $request→file('image')→store('covers', 'public');  
//Oppure  
$name_file = $request→file('image')→getClientOriginalName();  
$path_image = $request→file('image')→storeAs('covers', $name_file, 'public');
```

# Salvare la request

```
Book::create([
    'name' => $request->name,
    'author' => $request->author,
    'pages' => $request->pages,
    'image' => $path_image
]);
```

# Validare immagine

```
$request->validate([
    "name" => "required|string",
    "pages" => "required|numeric",
    "author" => "required",
    "image" => 'mimes:jpg,jpeg,bmp,png'
]);
```

# Visualizzare immagine view

```
<div class="row gx-4 gx-lg-5 align-items-center">
    <div class="col-md-6">
        name }}>
    </div>
    <div class="col-md-6">
        <h1 class="display-5 fw-bolder">{{ $book->name }}</h1>
        <p>Autore: {{ $book->author }} </p>
        <p>Numero Pagine: {{ $book->pages }} </p>
    </div>
</div>
```

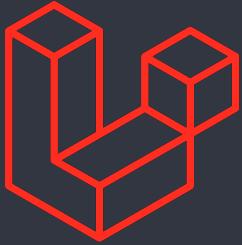
Problema!

# Storage::link & Storage::url()



```
php artisan storage:link
```

```
<div class="col-md-6">
    name}}">
</div>
```



# Corso Laravel

## File Upload

# Migrazione

```
protected $fillable = ['name', 'author', 'pages', 'image'];
```



```
php artisan make:migration add_to_books_table
```

Per creare questa tabella, dovete lanciare il comando `php artisan migrate`

```
/**  
 * Run the migrations.  
 */  
  
public function up(): void  
{  
    Schema::table('books', function (Blueprint $table) {  
        $table->string('image')->after('author')->nullable();  
    });  
}  
  
/**  
 * Reverse the migrations.  
 */  
  
public function down(): void  
{  
    Schema::table('books', function (Blueprint $table) {  
        $table->dropColumn('image');  
    });  
}
```

# Input di tipo file



```
<input class="form-control" id="image" name="image" type="file">
```

# Form enctype



```
<form action="{{route('store')}}" method="POST" enctype="multipart/form-data">
```

# Salvare la request

Utilizzando \$request possiamo accedere a tutti i metodi per manipolare le immagini e salvarle

```
→storeAs('percorso_di_salvataggio', 'nome_file.estensione');
```

```
$path_image = '';
if ($request->hasFile('image')) {
    $file_name = $request->file('image')->getClientOriginalName();
    $path_image = $request->file('image')->storeAs('public/images', $file_name);
}
```

# Salvare la request

```
Book::create([
    'name' => $request->name,
    'author' => $request->author,
    'pages' => $request->pages,
    'image' => $path_image
]);
```

# Validare immagine

```
$request->validate([
    "name" => "required|string",
    "pages" => "required|numeric",
    "author" => "required",
    "image" => 'mimes:jpg,jpeg,bmp,png'
]);
```

# Visualizzare immagine view

```
<div class="row gx-4 gx-lg-5 align-items-center">
    <div class="col-md-6">
        name }}>
    </div>
    <div class="col-md-6">
        <h1 class="display-5 fw-bolder">{{ $book->name }}</h1>
        <p>Autore: {{ $book->author }} </p>
        <p>Numero Pagine: {{ $book->pages }} </p>
    </div>
</div>
```

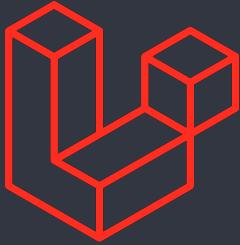
Problema!

# Storage::link & Storage::url()



```
php artisan storage:link
```

```
<div class="col-md-6">
    name}}">
</div>
```



# Corso Laravel

Fortify & Middleware

# Autenticazione con Fortify

Laravel Fortify è uno scaffolding di autenticazione senza una vera e propria interfaccia grafica.

La logica la scrive lui, la parte frontend la scrivi tu.

Documentazione ufficiale: <https://laravel.com/docs/11.x/fortify>

Repository Ufficiale: <https://github.com/laravel/fortify>



Laravel Fortify

# Installazione

## # Installation

To get started, install Fortify using the Composer package manager:

```
composer require laravel/fortify
```

Next, publish Fortify's resources using the `fortify:install` Artisan command:

```
php artisan fortify:install
```

This command will publish Fortify's actions to your `app/Actions` directory, which will be created if it does not exist. In addition, the `FortifyServiceProvider`, configuration file, and all necessary database migrations will be published.

Next, you should migrate your database:

```
php artisan migrate
```

<https://laravel.com/docs/11.x/fortify>

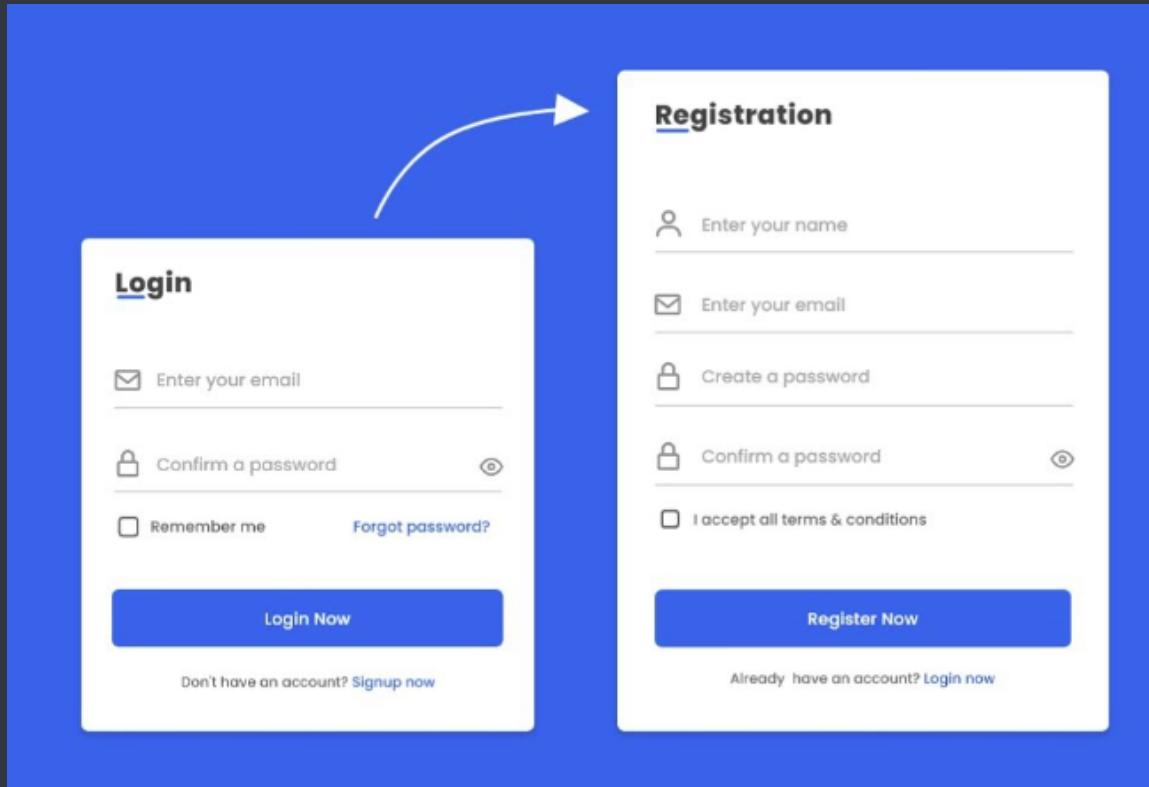
# Configurazione: config

Ad installazione completa, vai nel nuovo file config/fortify.php e cerca la chiave 'features' assicurandoti che Features::registration(); sia attivo e non commentato:



```
'features' => [
    Features::registration(),
    Features::resetPasswords(),
    Features::emailVerification(),
    Features::updateProfileInformation(),
    Features::updatePasswords(),
    Features::twoFactorAuthentication([
        'confirm' => true,
        'confirmPassword' => true,
        'window' => 0,
    ]),
],
```

# Logica Autenticazione



# Configurazione: provider

Adesso vai in app/Providers/FortifyServiceProvider.php e aggiungi le seguenti rotte nel metodo boot() in coda agli altri metodi:

```
public function boot(): void
{
    //... metodi presenti

    Fortify::loginView(function () {
        return view('auth.login');
    });

    Fortify::registerView(function () {
        return view('auth.register');
    });
}
```

# Registrazione

Come specificato nel provider, Fortify si aspetta una vista in:

resources/views/auth/register.blade.php

Ovviamente in questa vista andremo a posizionare un form con:

- Il metodo POST
- La action {{ route('register') }}
- i campi:
  - name;
  - email;
  - password;
  - password\_confirmation.

# Login

Come specificato nel provider, Fortify si aspetta una vista in:

resources/views/auth/login.blade.php

Ovviamente in questa vista andremo a posizionare un form con:

- Il metodo POST
- La action {{ route('login') }}
- i campi:
  - email;
  - password;

# Logout

Infine gestiamo il logout dell'utente.

Possiamo creare, utilizzando Blade, anche un sistema che in base all'utente se autenticato o no, ci mostri degli url piuttosto che altri.

```
@auth

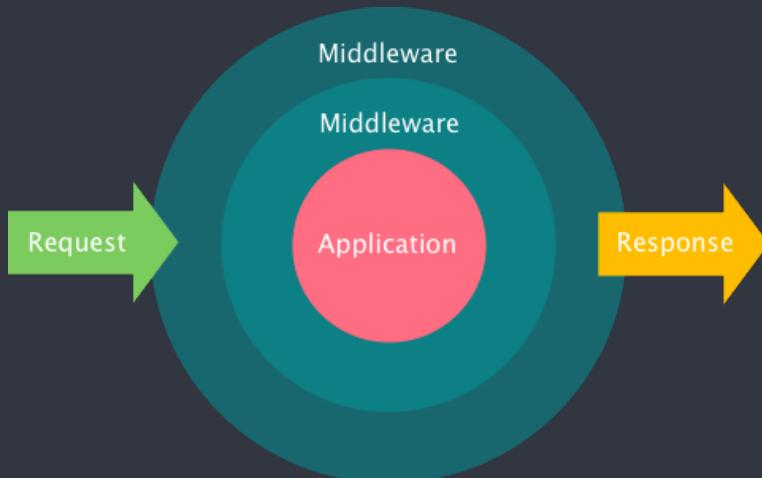

Ciao, {{Auth::user()->email}},


<form action="{{ route('logout') }}" method="POST">
    @csrf
    <button>
        Logout
    </button>
</form>
@else
<a href="{{ route('login') }}>Accedi</a>
<a href="{{ route('register') }}>Registrati</a>
@endauth
```

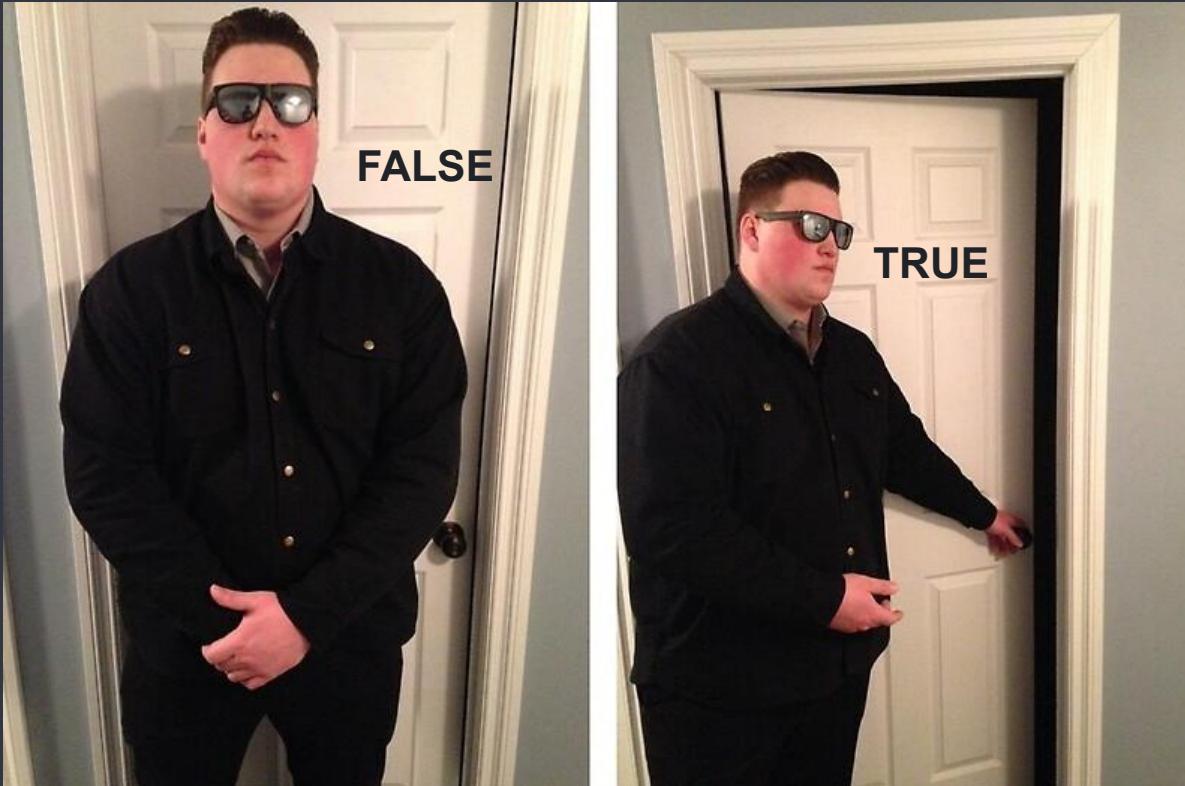
# Middleware

Il Middleware (o più middleware), come accennato nella prima lezione su Laravel, è un filtro tra il Client e il Server basato su una condizione di verifica:

Sono un admin? True puoi entrare, False ti rimbalzo indietro.



# Middleware



# Route Middleware

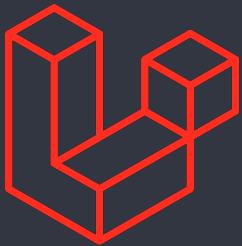
Applicando ->middleware('auth') alla rotte:

```
Route::get('/libri', [BookController::class, 'index'])->name('books.index')->middleware('auth');
Route::get('/libri/crea', [BookController::class, 'create'])->name('books.create')->middleware('auth');
Route::post('/libri/salva', [BookController::class, 'store'])->name('books.store')->middleware('auth');
Route::get('/libri/{book}/dettagli', [BookController::class, 'show'])->name('books.show')->middleware('auth');
```

# Route Group Middleware

Con una ::middleware(['auth'])->group(function) su tutte le rotte interessate:

```
Route::middleware(['auth'])->group(function () {
    Route::get('/libri', [BookController::class, 'index'])->name('books.index');
    Route::get('/libri/crea', [BookController::class, 'create'])->name('books.create');
    Route::post('/libri/salva', [BookController::class, 'store'])->name('books.store');
    Route::get('/libri/{book}/dettagli', [BookController::class, 'show'])->name('books.show');
});
```



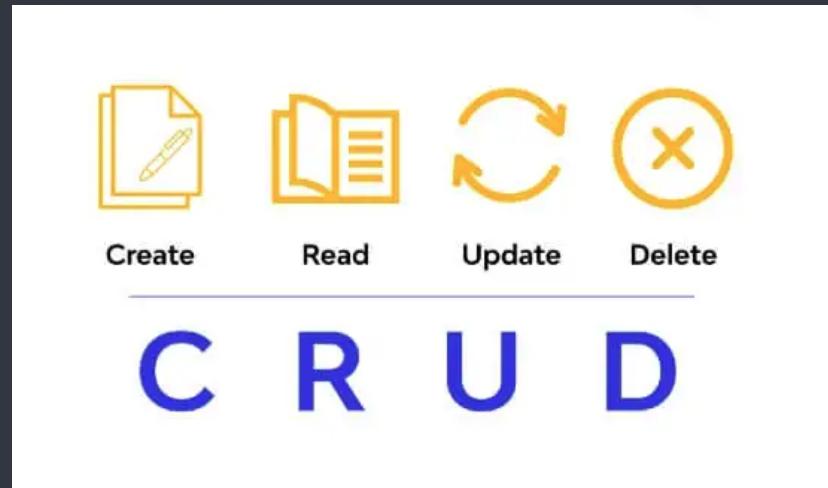
# Corso Laravel

## CRUD

# CRUD

Per gestire una risorsa abbiamo bisogno di:

- Una lista con tutti gli elementi (index)
- Una pagina di creazione che ospiterà un form per raccogliere i dati (create)
- Un link di "azione" che andrà ad immagazzinare la risorsa nel DB (store)
- Una pagina di dettaglio per visualizzare il singolo elemento con i relativi dettagli (show)
- Un link di modifica che ospiterà un from per mostrare i dati della risorsa esistente (edit)
- Una pagina di "azione" che andrà a modificare la risorsa nel DB (update)
- Un link di cancellazione per rimuovere i riferimenti della risorsa nel DB (delete)



# Metodi HTTP



# CRUD

```
Route::get('/libri', [BookController::class, 'index'])->name('books.index');
Route::get('/libri/crea', [BookController::class, 'create'])->name('books.create');
Route::post('/libri/salva', [BookController::class, 'store'])->name('books.store');
Route::get('/libri/{book}/dettagli', [BookController::class, 'show'])->name('books.show');
Route::get('/libri/{book}/modifica', [BookController::class, 'edit'])->name('books.edit');
Route::put('/libri/{book}', [BookController::class, 'update'])->name('books.update');
Route::delete('/libri/{book}', [BookController::class, 'destroy'])->name('books.destroy');
```

# Index

## Controller

```
public function index()
{
    $books = Book::all();
    return view('books.index', compact('books'));
}
```

## View

```
<body class="antialiased">
    <ul>
        @foreach ($books as $book)
            <li>{{$book['name']}} - {{$book['author']}}</li>
        @endforeach
    </ul>
</body>
```

# Create

## View

```
<body class="antialiased">
<form action="{{ 'books.store' }}" method="POST">
    @csrf
    @method('POST')
    <div class="form-floating mb-3">
        <input class="form-control" name="name" type="text" value="{{ old('name') }}"
            placeholder="Nome Libro">
        <label for="name">Libro</label>
    </div>
    <div class="d-grid gap-3">
        <button class="btn btn-primary btn-lg p-2" type="submit">Submit</button>
        <button class="btn btn-danger btn-lg p-2" type="reset">Cancella</button>
    </div>
</form>
</body>
```

## Controller

```
public function create()
{
    return view('create');
}
```

# Store

## Controller

```
public function store(BookRequest $request)
{
    $path_image = '';
    if ($request->hasFile('image') && $request->file('image')->isValid()) {
        $path_name = $request->file('image')->getClientOriginalName();
        $path_image = $request->file('image')->storeAs('public/images', $path_name);
    }
    Book::create([
        'name' => $request->input('name'),
        'author' => $request->author,
        'pages' => $request->pages,
        'image' => $path_image
    ]);

    return redirect()->route('books.index')
        ->with('success', 'Creazione avvenuta con successo!');
}
```



->storeAs('public/images', \$path\_name, 'public')

# Show

## Controller

```
public function show(Book $book)
{
    return view('books.show', compact('book'));
}
```

## View

```
<body class="antialiased">
    <h2>Dettagli</h2>
    <p>Nome: {{$book->name}}</p>
    <p>Autore: {{$book->author}}</p>
</body>
```

# Edit

## Controller

```
public function edit(Book $book)  
{  
    return view('books.edit', compact('book'));  
}
```

## View

```
<form action="{{route('books.update', ['book' => $book->id])}}" method="POST"  
      enctype="multipart/form-data"  
      @method('PUT')  
      @csrf  
      <div class="form-floating mb-3">  
          <input class="form-control" id="name" name="name" required type="text"  
                 value="{{ $book->name }}" placeholder="Inserisci titolo libro">  
          <label for="name">Nome Libro</label>  
          @error('name')  
              <span class="text-danger">  
                  {{ $message }}  
              </span>  
          @enderror  
      </div>  
      <div class="d-grid gap-3">  
          <button class="btn btn-primary btn-lg p-2" type="submit">Salva</button>  
      </div>  
  </form>
```

# Update

## Controller

```
public function update(BookRequest $request, Book $book)
{
    $path_image = $book->image;
    if ($request->hasFile('image') && $request->file('image')->isValid()) {
        $path_name = $request->file('image')->getClientOriginalName();
        $path_image = $request->file('image')->storeAs('public/images', $path_name);
    }
    $book->update([
        'name' => $request->input('name'),
        'author' => $request->author,
        'pages' => $request->pages,
        'image' => $path_image
    ]);
    return redirect()->route('books.index')
        ->with('success', 'Modifica avvenuta con successo!');
}
```



```
->storeAs('public/images', $path_name, 'public')
```

# Delete

## Controller

```
public function destroy(Book $book)
{
    $book->delete();
    return redirect()->route('books.index')
        ->with('success', 'Cancellazione avvenuta con successo!');
}
```

## View

```
<form action="{{route('books.delete', ['book' => $book['id']])}}" method="POST">
    @csrf
    @method('DELETE')
    <button class="btn btn-danger" type="submit">Elimina</button>
</form>
```

# Resource Controller

```
Route::resource('books', BookController::class);
```



```
php artisan make:controller BookController --resource
```

# Comando All in One



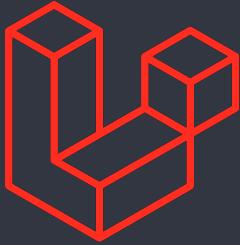
```
php artisan make:model Test -mcrR
```

Crea:

- Model
- Migration
- Controller di tipo Resource
- Request Form Class

Dobbiamo Creare solamente

- Rotte (Route::resource())
- view



# Corso Laravel

One to Many (1 a N )oppure (1 a Molti)

# Relazione 1 a N (o a Molti)

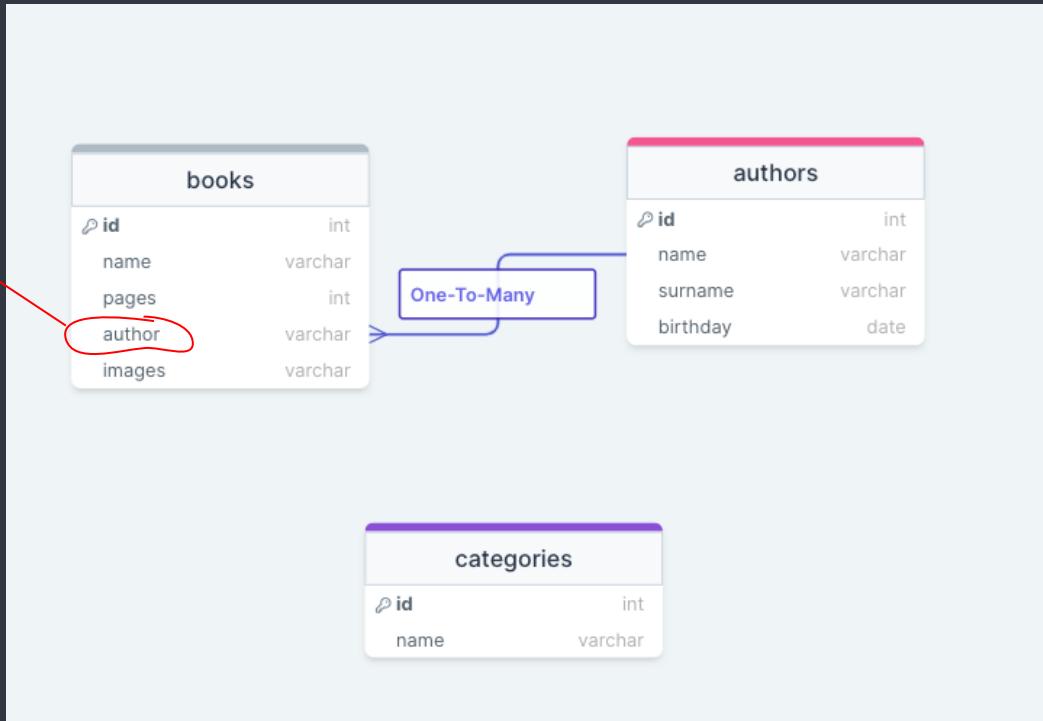
MySQL, il nostro RDBMS, è un Relational Database Management System ovvero utilizza le Relazioni per creare connessione tra tabelle.

Ad esempio, dobbiamo definire la relazione tra Libri e Autori.



# Integrità referenziale

Possibili  
problemi di  
integrità!



# Integrità referenziale 2

Ridondanza e Inconsistenza

Studenti			
Nome	Cognome	Docente	Materia
Paolo	Rossi	Professor Biagio	Educazione Fisica
Giovanni	Verdi	Professor Alfredo	Storia
Antonio	Bianchi	Professor Biagio	Educazione Fisica

Docenti	
Docente	Materia
Professor Biagio	Nutrizionista
Professor Carlo	Storia

Il Professor Biagio ha cambiato cattedra

Chi è Alfredo?

# Integrità referenziale 3

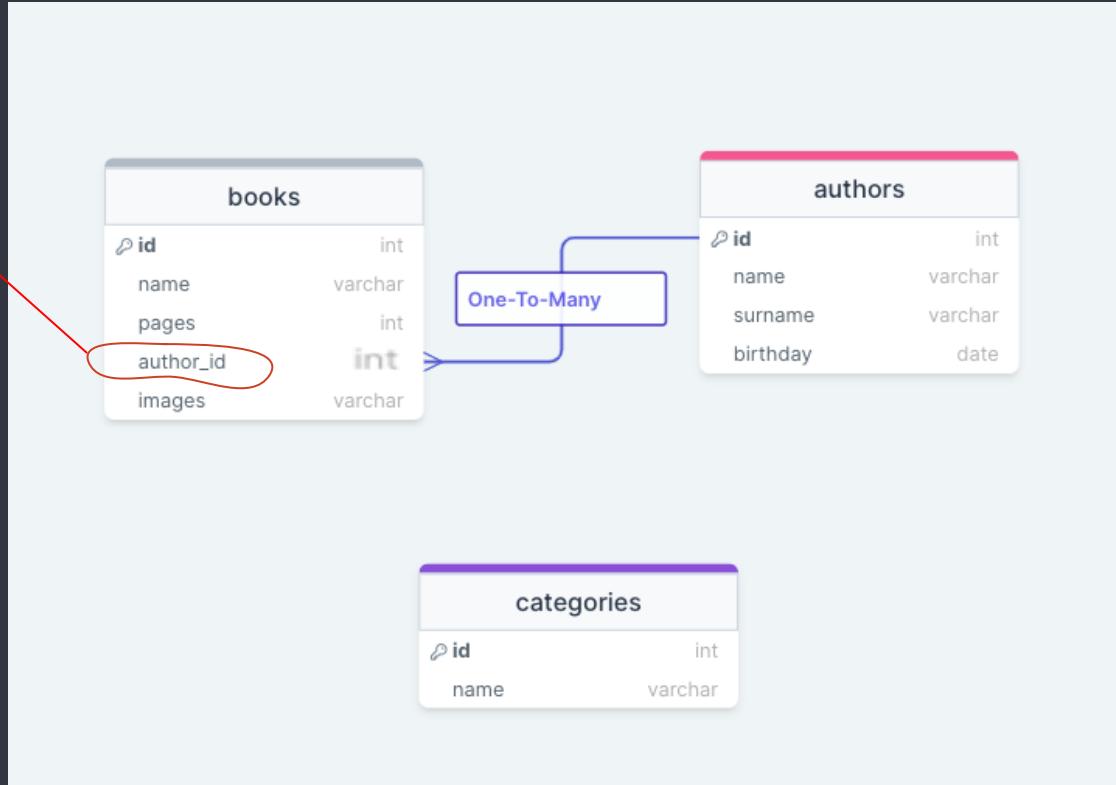
Rimozione  
Ridondanza e  
Integrità Referenziale

Studenti			
Matricola	Nome	Cognome	Docente
S1	Paolo	Rossi	D1
S2	Giovanni	Verdi	D2
S3	Antonio	Bianchi	D1

Docenti		
Matricola	Docente	Materia
D1	Professor Biagio	Educazione Fisica
D2	Professor Carlo	Storia

# Chiave Esterna

Utilizziamo un  
id come  
chiave  
esterna



# PK vs FK

## PK (Primary Key) Chiave Primaria:

Una chiave primaria è un identificatore univoco per una riga specifica in una tabella in un database.

Viene utilizzata per garantire l'integrità dei dati e per prevenire voci duplicate.

Viene utilizzato per cercare rapidamente e recuperare righe specifiche dalla tabella.

Le chiavi primarie devono contenere valori univoci e non possono contenere valori null.

# PK vs FK

## **FK (Foreign Key) Chiave Esterna:**

Una chiave esterna è una colonna in una tabella del database che viene utilizzato per identificare in modo univoco una riga in un'altra tabella.

Le chiavi esterne vengono utilizzate per stabilire e applicare le relazioni tra le tabelle in un database, come la relazione tra un libro e un autore o un articolo e un commento.

# PK vs FK

Chiave Primaria	Chiave Esterna
Identifica in maniera univoca un record nella stessa tabella	Identifica un record di un'altra tabella
Non possono esserci valori NULL	Eventualmente possono esserci valori NULL.
Nella stessa colonna non possono esserci duplicati, deve esistere un solo valore di chiave primaria.	Nella stessa colonna possono esserci duplicati

# Come funziona

books

ID	Name	pages	author_id
1	Libro	345	2
2	Libro 2	1	2
3	Harry Potter 2	500	1
4	La bibbia	1203	3

authors

ID	Name	Surname	Birthday
1	J.K.	Rowling	10/10/2000
2	Maccio	Capatonda	11/10/2000
3	Autori	Vari	14/10/2000



id	books.name	books.pages	authors.name	authors.surname	authors.birthday
3	Harry potter 2	500	J.K.	Rowling	10/10/2000

# Migrazione semplice

```
public function up(): void
{
    Schema::table('books', function (Blueprint $table) {
        $table->unsignedBigInteger('author_id');
        $table->foreign('author_id')->references('id')->on('authors');
        //Oppure
        $table->foreignId('author_id')->constrained();
    });
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::table('books', function (Blueprint $table) {
        $table->dropForeign(['author_id']);
        $table->dropColumn('author_id');
    });
}
```



# Migrazione a cascata

```
public function up(): void
{
    Schema::table('books', function (Blueprint $table) {
        $table->unsignedBigInteger('author_id')->nullable();
        $table->foreign('author_id')->references('id')->on('authors')->onDelete('cascade');
        //Oppure
        $table->foreignId('author_id')->nullable()->constrained()->onDelete('cascade');
    });
}
```

# Relazione nel Model Book (N)

Quanti autori può avere un libro? 1 quindi Singolare author

```
class Book extends Model
{
    use HasFactory;

    protected $fillable = ['name', 'author_id', 'pages', 'image'];

    public function author()
    {
        return $this->belongsTo(Author::class);
    }
}
```

# Relazione nel Model Author (1)

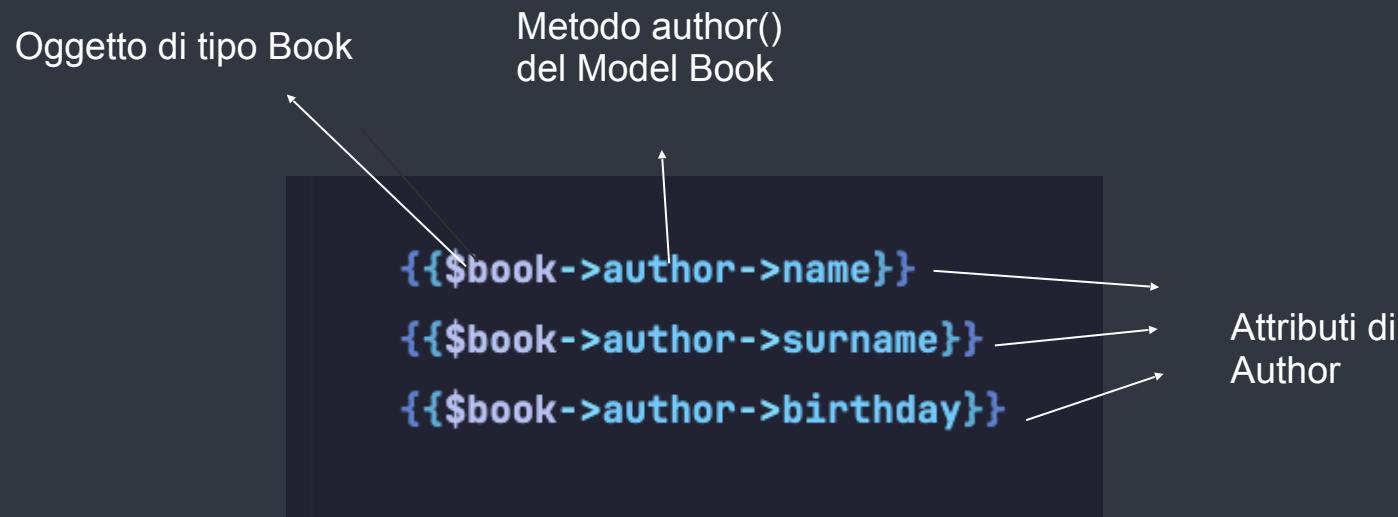
Quanti libri può avere un autore?  
N quindi Plurale  
**books**

```
class Author extends Model
{
    use HasFactory;
    protected $fillable = ['name', 'surname', 'birthday'];

    protected $casts = [
        'birthday' => 'datetime',
    ];

    public function books()
    {
        return $this->hasMany(Book::class);
    }
}
```

# Accedere ai dati su Book



# Accedere ai dati su Author

Oggetto di tipo Author

Metodo books()  
del Model Author

```
@forelse ($author->books as $book)
```

```
    {{$book->name}}
```

```
    {{$book->pages}}
```

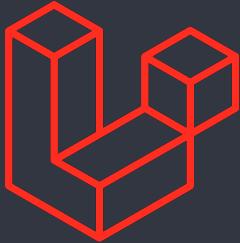
```
    {{$book->images}}
```

```
@empty
```

```
    Nessun Libro Aggiunto
```

```
@endforelse
```

Attributi di  
Author



# Corso Laravel

Many to Many (N a N ) oppure (Molti a Molti)

# Relazione N a N

MySQL, il nostro RDBMS, è un Relational Database Management System ovvero utilizza le Relazioni per creare connessione tra tabelle.

Ad esempio, dobbiamo definire la relazione tra Categorie e Libri.



# Relazione 1 a N

users

<b>id</b>	<b>name</b>	<b>menu_id</b>
1	Francesco	3
2	Nino	1
3	Jessica	2
4	Katia	4

menus

<b>id</b>	<b>name</b>
1	Popcorn
2	Patatine
3	Birra
4	Pizza

# Soluzioni Sbagliate: 1

Decido un massimo di pietanze ordinabili e creo le colonne relative.

<b>id</b>	<b>name</b>	<b>menu_id</b>	<b>menu_id_2</b>	<b>menu_id_3</b>
1	Francesco	3	null	null
2	Nino	1	2	3
3	Jessica	2	3	null
4	Katia	4	1	3

<b>id</b>	<b>name</b>
1	Popcorn
2	Patatine
3	Birra
4	Pizza

# Soluzioni Sbagliate: 2

Creo record per ogni pietanza associata al cliente

<b>id</b>	<b>name</b>	<b>menu_id</b>
1	Francesco	3
2	Nino	1
3	Jessica	2
4	Katia	4
5	Nino	2
6	Jessica	3

<b>id</b>	<b>name</b>
1	Popcorn
2	Patatine
3	Birra
4	Pizza

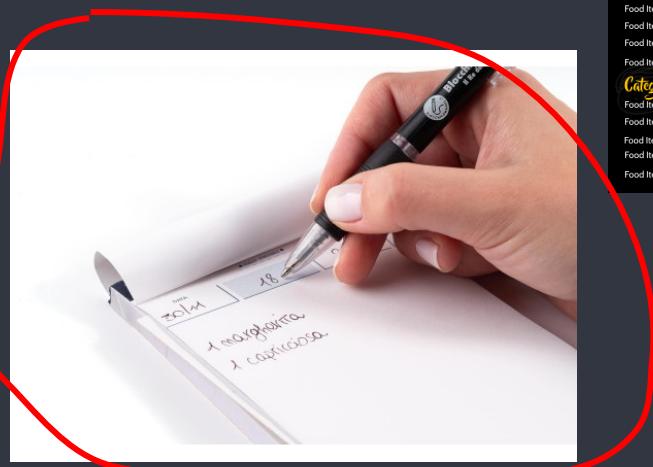
Ridondanza e  
Inconsistenza

# Tabella Pivot

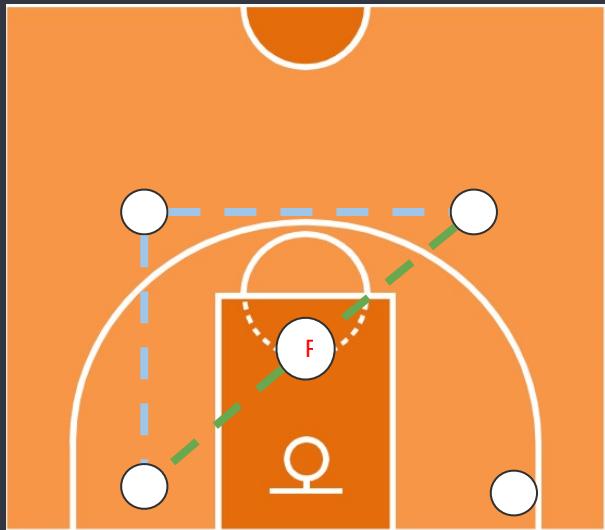


N

N

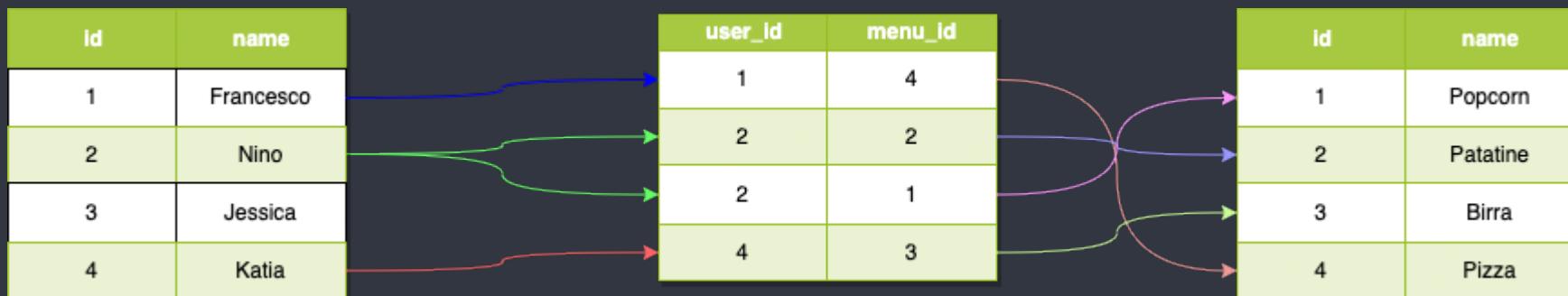


# Tabella Pivot



# Soluzione con Pivot

Utilizzo una tabella di Pivot per scrivere tutte le relazioni tra chiavi primarie



# Creare la tabella Pivot

Una tabella pivot è una tabella di sole chiavi esterne.

**DISCLAIMER: Per la tabella Pivot non serve creare il model apposito!**



```
php artisan make:migration create_book_category_table
```

Come scrivere il comando:

- `create_a_b_table` (inserirli in ordine alfabetico);
- inserire i nomi dei model tutto minuscolo;

# Migrazione Pivot

```
public function up(): void
{
    Schema::create('book_category', function (Blueprint $table) {
        $table->id();

        $table->unsignedBigInteger('book_id');
        $table->unsignedBigInteger('category_id');

        $table->foreign('book_id')->references('id')->on('books');
        $table->foreign('category_id')->references('id')->on('categories');

        $table->timestamps();
    });
}
```

# Relazione nel Model Book (N)

Quante categorie  
può avere un  
libro? Molte  
quindi plurale  
**categories**

```
class Book extends Model
{
    use HasFactory;

    public function categories()
    {
        return $this->belongsToMany(Category::class);
    }
}
```

# Relazione nel Model categorie (N)

Quanti libri può avere una categoria? N quindi Plurale books

```
class Category extends Model
{
    use HasFactory;

    protected $fillable = ['name'];

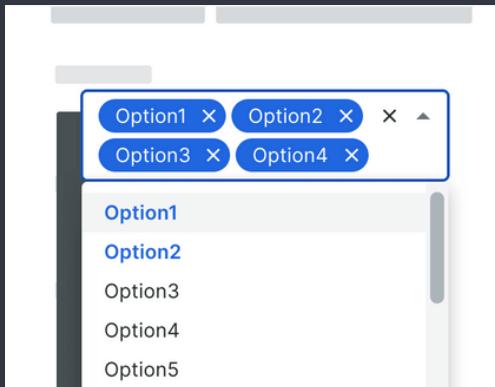
    public function books()
    {
        return $this->belongsToMany(Book::class);
    }
}
```

# 1 a N N a N

<b>One to Many (1 a N)</b>	<b>Many to Many (N a N)</b>
Devo creare tanti model quante sono le tabelle che ho creato	Non ho bisogno di ulteriori Model, devo solo creare la tabella PIVOT
Ho bisogno di una relazione con il nome singolare per belongsTo (appartiene a)	Ho bisogno di due relazione con il nome plurale per belongsToMany (appartiene a molti)
Ho bisogno di una relazione con il nome plurale per hasMany (a molti)	

# Interfaccia Grafica N a N

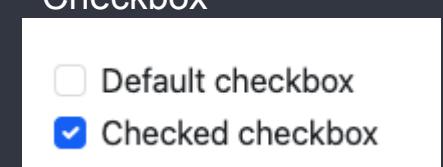
Multi Select



Multi Autocomplete



Checkbox



# Implementare checkbox View

```
@foreach ($categories as $category)
    <div class="form-check">
        <input class="form-check-input"
            value="{{ $category->id }}"
            type="checkbox"
            id="category_id"
            name="categories[]">
        <label class="form-check-label" for="category_id">
            {{ $category->name }}
        </label>
    </div>
@endforeach
```

# Implementare salvataggio

Devo inserire le  
tonde quando  
salvo  
`categories()`

*(Se devo  
accedere in  
lettura le  
parentesi non  
servono)*

```
$book = Book::create([  
    'name' => $request->input('name'),  
    'author_id' => $request->author_id,  
    'category_id' => $request->category_id,  
    'pages' => $request->pages,  
    'image' => $path_image,  
    'user_id' => Auth::user()->id  
]);
```

```
$book->categories()->attach($request->categories);
```

# Accedere ai dati in index

```
<ul>
    @foreach ($book->categories as $category)
        <li> {{$category->name}}</li>
    @endforeach
</ul>
```

# Accedere ai dati in edit

```
@foreach ($categories as $category)
    <div class="form-check">
        <input class="form-check-input"
            value="{{ $category->id }}"
            type="checkbox"
            id="category_id"
            name="categories[]"
            @if($book->categories->contains($category->id))
                checked
            @endif>
        <label class="form-check-label" for="category_id">
            {{ $category->name }}
        </label>
    </div>
@endforeach
```

Il metodo  
contains controlla  
se l'id è presente  
nell'array passato

# Accedere ai dati in edit da PRO

```
@foreach ($categories as $category)
    <div class="form-check">
        <input class="form-check-input"
            value="{{ $category->id }}"
            type="checkbox"
            id="category_id"
            name="categories[]"
            @checked($book->categories->contains($category->id)) >
        <label class="form-check-label" for="category_id">
            {{ $category->name }}
        </label>
    </div>
@endforeach
```

# Aggiornare i dati in Update

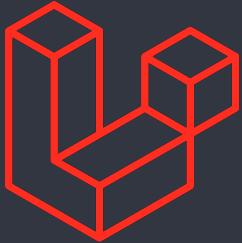
```
$book->categories()->detach();  
$book->categories()->attach($request->categories);
```

# Aggiornare i dati in Update da PRO

```
$book->categories()->sync($request->categories);
```

# Cancellare dati in Destroy

```
public function destroy(Book $book)
{
    $book->categories()->detach();
    $book->delete();
    return redirect()
        ->route('books.index')
        ->with('success', 'Cancellazione avvenuta con successo!');
}
```



# Modulo Laravel Factory & Seeder

# Panoramica Migration e Migrate

Migration



Definisce la struttura  
delle **TABELLE**,  
ovvero del  
contenitore che andrà  
ad ospitare i dati

`php artisan migrate`



Crea fisicamente  
le tabelle

# Panoramica Factory e Seeder

Factory



Definisce la struttura  
dei **DATI**

Seeder

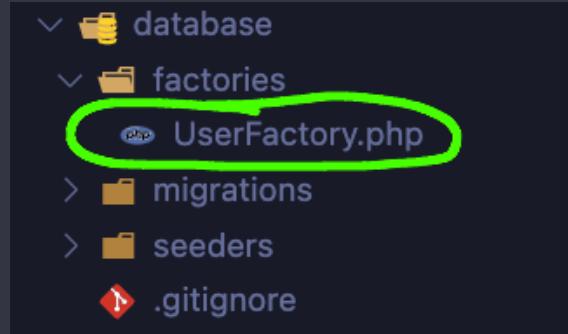


Crea fisicamente i record con  
`php artisan db:seed`

# Factory

Le factories sono una caratteristica di Laravel che semplifica la creazione di dati fintizi (fake data) da utilizzare durante lo sviluppo e il test delle applicazioni.

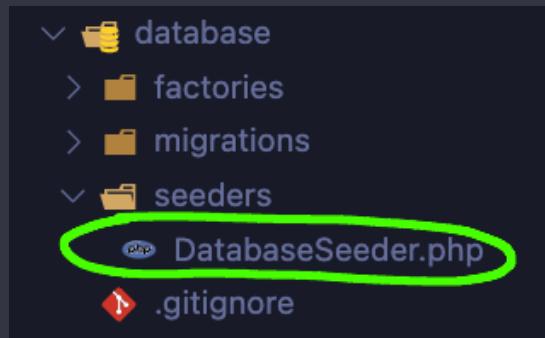
L'utilizzo di fake data e' cosi' comune che, di default, Laravel ha gia' al suo interno una classe specifica chiamata UserFactory.php che troviamo nella directory database/factories.



# Seeder

Il seeder in Laravel è l'azione di "seminare dei dati", un po come piantare semi per far crescere dati di esempio nel tuo database utilizzando la struttura precedentemente definita nelle factory.

Dopo aver creato una factory, utilizzi un seeder per "seminare" il tuo database con dati di esempio:



# Comando seeder



```
php artisan db:seed
```

# Nuova Factory

Andiamo a creare una nuova factory per gli autori e i libri



```
php artisan make:factory BookFactory
```



```
php artisan make:factory AuthorFactory
```

# Definition method

Definiamo gli elementi da creare in AuthorFactory e in BookFactory

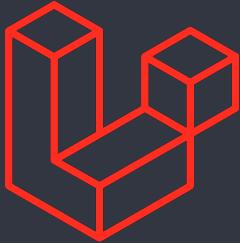
```
public function definition(): array
{
    return [
        'name' => fake()→name(),
        'surname' => fake()→lastName()
    ];
}
```

```
public function definition(): array
{
    return [
        'name' => fake()→name(),
        'pages' => fake()→randomDigit(1, 100),
        'years' => fake()→randomDigit(1900, 2024),
        'author_id' => Author::factory()
    ];
}
```

# Seeder

Infine specifichiamo il model da richiamare:

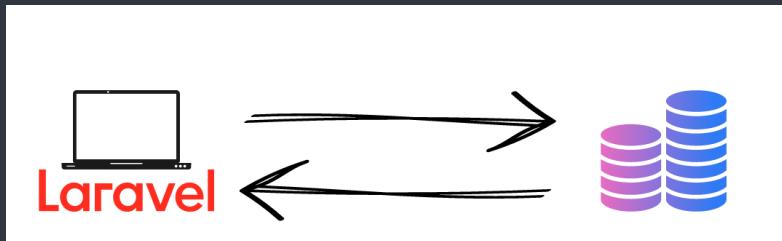
```
public function run(): void
{
    \App\Models\User::factory(100000)→create();
    \App\Models\Book::factory(100000)→hasAuthor()→create();
}
```



# Corso Laravel

# API e HTTP Client

# Cos'è un'API



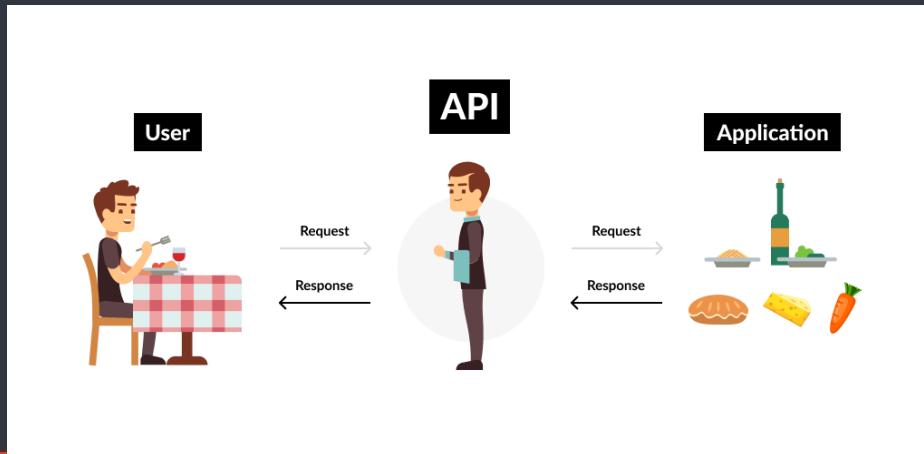
Eloquent è un Database API

Eloquent: <https://laravel.com/docs/11.x/eloquent>

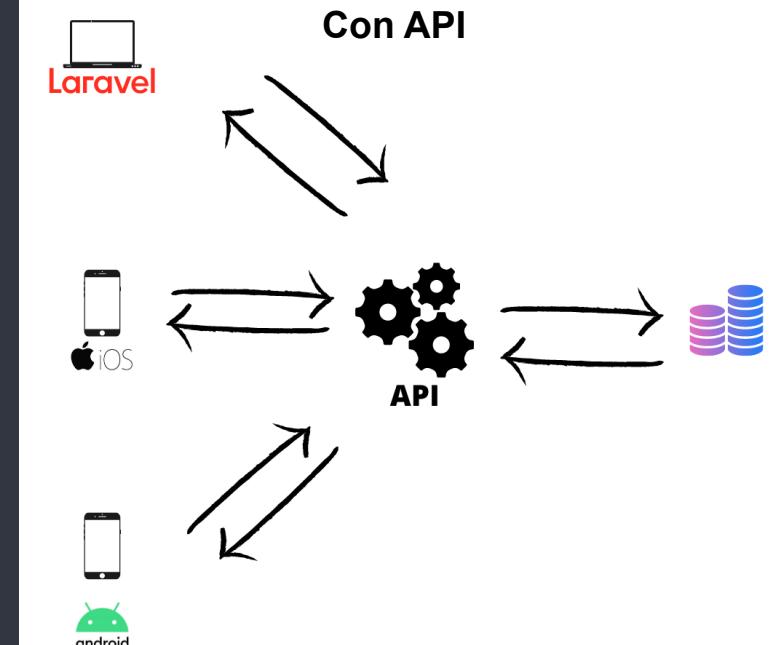
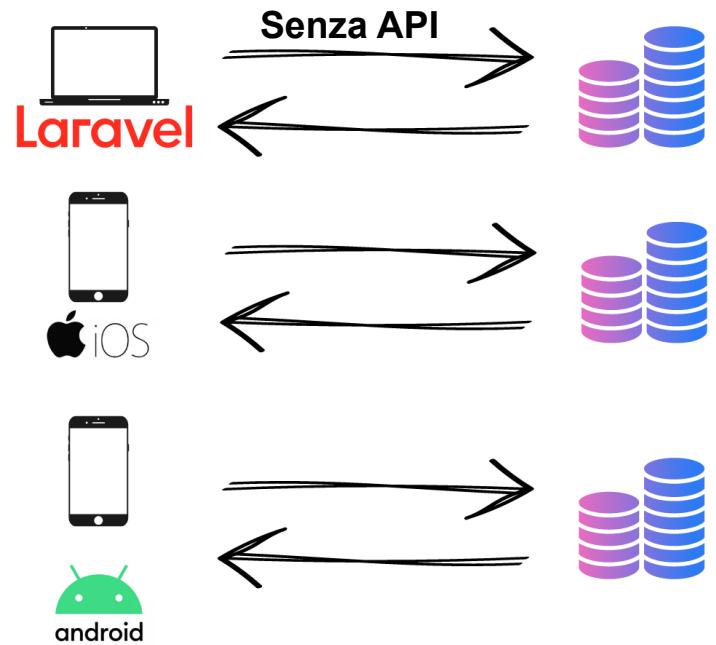
# Cos'è un'API

Application programming interface (API), in italiano "interfaccia di programmazione dell'applicazione", si indica un insieme di procedure (in genere raggruppate per strumenti specifici) atte a risolvere uno specifico problema di comunicazione tra diversi computer o tra diversi software o tra diversi componenti di software.

**"Un'interfaccia di standardizzazione"**



# Cos'è un'API



Eloquent: <https://laravel.com/docs/12.x/eloquent>

Fluent: <https://docs.vapor.codes/fluent/overview/>

Ktorm: <https://www.ktorm.org/>

# Laravel come consumatore di API

```
use Illuminate\Support\Facades\Http;
use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    $response = Http::get('https://api.jikan.moe/v4/genres/anime')→json();
    dd($response);
});
```

<https://laravel.com/docs/12.x/http-client>

<https://docs.api.jikan.moe/>

# API - jikan.moe

- Documentazione: <https://docs.api.jikan.moe/>
  - Lista Categorie Anime: <https://api.jikan.moe/v4/genres/anime>
  - Lista Anime di quella Categoria: [https://api.jikan.moe/v4/anime?genres={genre\\_id}](https://api.jikan.moe/v4/anime?genres={genre_id})
  - Dettaglio Anime: [https://api.jikan.moe/v4/anime/{anime\\_id}](https://api.jikan.moe/v4/anime/{anime_id})

# In caso di errori [Windows]

GuzzleHttp \ Exception \ RequestException

PHP 8.2.5 10.11.0

cURL error 60: SSL certificate problem: unable to get local issuer certificate (see <https://curl.haxx.se/libcurl/c/libcurl-errors.html>) for  
<https://pokeapi.co/api/v2/pokemon/>

- Scarica questo file <https://curl.se/ca/cacert.pem> e inseriscilo in c:\php\php\extras\ssl
- Successivamente apri il file dove hai salvato php (ad esempio c:\php\php\php.ini) e modifica queste due voci eliminando i punti e virgola nelle righe:

```
[curl]
curl.caInfo = "c:\php\php\extras\ssl\cacert.pem"
```

- Riavvia il server e dovrebbe andare

# Esempi di API/JSON

- Nazioni API Documentazione: <https://restcountries.com/#rest-countries>
  - index: <https://restcountries.com/v3.1/all>
  - show: <https://restcountries.com/v3.1/name/italy>
- Pokemon API Documentazione: <https://pokeapi.co/>
  - index: <https://pokeapi.co/api/v2/pokemon>
  - show: <https://pokeapi.co/api/v2/pokemon/bulbasaur>
- FakeStore API Documentazione: <https://fakestoreapi.com/>
  - index: <https://fakestoreapi.com/products>
  - show: <https://fakestoreapi.com/products/6>
- RickAndMorty API Documentazione: <https://rickandmortyapi.com/>
  - index: <https://rickandmortyapi.com/api/character>
  - show: <https://rickandmortyapi.com/api/character/2>
- Italia
  - Comuni: <https://axqvoqvbfjpaamphztgd.functions.supabase.co/comuni>
  - Province: <https://axqvoqvbfjpaamphztgd.functions.supabase.co/province>
  - Regioni: <https://axqvoqvbfjpaamphztgd.functions.supabase.co/regioni>

# Seeder

Un seeder è uno strumento utilizzato per popolare il database con dati di esempio o predefiniti.

È particolarmente utile per l'inizializzazione di un nuovo progetto, la creazione di dati di prova per lo sviluppo o per la definizione di valori di default per certe tabelle.

Ed e' proprio quello di cui abbiamo bisogno: l'idea e' che andremo a "trasferire" i dati dal json al database.

# Panoramica Migration e Migrate

Migration



Definisce la struttura  
delle **TABELLE**,  
ovvero del  
contenitore che andrà  
ad ospitare i dati

`php artisan migrate`



Crea fisicamente  
le tabelle

# Panoramica Factory e Seeder

Factory



Definisce la struttura  
dei DATI

Seeder



Crea un file fisicamente, legge  
la struttura dati e genera i  
record  
php artisan db:seed

# Factory

Le factories sono una caratteristica di Laravel che semplifica la creazione di dati fintizi (fake data) da utilizzare durante lo sviluppo e il test delle applicazioni.

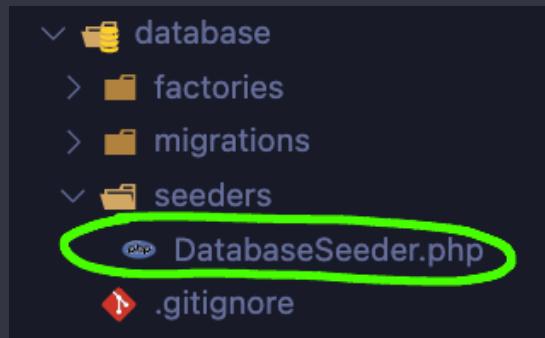
L'utilizzo di fake data e' cosi' comune che, di default, Laravel ha gia' al suo interno una classe specifica chiamata UserFactory.php che troviamo nella directory database/factories.



# Seeder

Il seeder in Laravel è l'azione di "seminare dei dati", un po come piantare semi per far crescere dati di esempio nel tuo database utilizzando la struttura precedentemente definita nelle factory.

Dopo aver creato una factory, utilizzi un seeder per "seminare" il tuo database con dati di esempio:



# Comando seeder



```
php artisan db:seed
```

# Laravel come provider di API



```
php artisan install:api
```

<https://laravel.com/docs/11.x/sanctum#main-content>

# Laravel come provider di API

```
Route::get('/esempio-di-api', function () {  
    return 'Test';  
})->name('esempio-api');
```



127.0.0.1:8000/api/esempio-di-api

# Ritorno di api.php

- Il return sarà sempre un JSON
- Verrà sempre anteposta la parola api a tutte le rotte (api/uri)
- Queste rotte ritorneranno sempre dei dati, mai una vista