



CORSO FULL STACK WEB DEVELOPER



LARAVEL CRUD

Convertiamoci al crudismo!



CRUD operations

Chiaramente non stiamo parlando di crudità...

CRUD è un acronimo e sta per:

- **CREATE**
- **READ**
- **UPDATE**
- **DELETE**

Questo è il set minimo di operazioni per qualsiasi tipo di applicazione.



Operazioni CRUD: un esempio pratico

Voglio creare un nuovo contatto nella rubrica

IPOTESI:

1 `www.sito.com/create?nome=fabio&telefono=123456789`

Non molto leggibile, e se volessi creare non solo contatti ma anche altre “entità” (ad es. azienda)?

2 `www.sito.com/rubrica?nome=fabio&telefono=12345678&azione=crea`

Nemmeno questa è molto leggibile



Ci serve uno standard chiaro, semplice e inequivocabile per le richieste CRUD



REST: REpresentational State Transfer

Un'architettura chiara, organizzata e condivisa

REST introduce il concetto di **risorse**, cioè fonti di informazioni a cui si può accedere tramite un **identificatore globale** (un **URI**).

Per utilizzare le risorse, client e server comunicano attraverso una interfaccia standard (HTTP) e si scambiano **rappresentazioni** di queste risorse.



REST: REpresentational State Transfer

Come funziona?

Il funzionamento prevede:

1. una **struttura degli URL ben definita e univoca** per ogni risorsa
2. l'utilizzo dei **metodi HTTP** per azioni specifiche sulle risorse:
 - **GET** per il recupero dei dati
 - **POST** per la creazione di nuovi dati
 - **PUT/PATCH** per l'aggiornamento dei dati esistenti
 - **DELETE** per la cancellazione di dati



L'architettura REST è applicata soprattutto alle **API**.
Laravel la utilizza anche nelle CRUD, grazie alle **Resources**.



Resource Controller

Abbiamo visto che con il comando artisan **make:controller** Laravel crea i controller con lo scheletro base.

Ma Laravel fa di più!

Può preparare per noi **i nomi delle funzioni** che ci servono per creare una **CRUD** base.

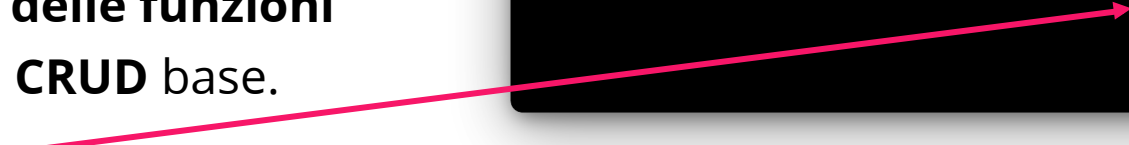
Basta aggiungere **--resource**



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



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



Possiamo utilizzare **--help** nei nostri comandi per vedere quali attributi possiamo aggiungere o semplicemente come scriverli.



Chiarimento necessario e indispensabile.

Come tutti i comandi di Laravel,
anche **le resources creano solo la struttura base richiesta**,
non scriveranno codice per noi!



Resource Route

Sappiamo già che la classe **Route** mette a disposizione i metodi `get()`, `post()`, ecc per definire le rotte.

Laravel ci aiuta anche con le rotte, grazie al metodo **`resource()`**, il quale genera per noi tutte le rotte necessarie per le operazioni CRUD.

Dobbiamo solo indicare qual è l'**URI di base** e il **Controller** che gestisce tutte le rotte.



```
1 1 Route::resource('users', UserController::class);
```



Route:list

Proviamo ora a eseguire il nostro comando **php artisan route:list**

Vediamo così come è la struttura automatica del nostro **resource controller**

```
GET|HEAD      users ..... users.index > UserController@index
POST          users ..... users.store > UserController@store
GET|HEAD      users/create ..... users.create > UserController@create
GET|HEAD      users/{user} ..... users.show > UserController@show
PUT|PATCH    users/{user} ..... users.update > UserController@update
DELETE        users/{user} ..... users.destroy > UserController@destroy
GET|HEAD      users/{user}/edit ..... users.edit > UserController@edit
```



RESTful CRUD - index

1 CRUD - Leggere i dati di tutti gli utenti

Analizziamo la rotta index

URI	Name	Action
GET /users	users.index	UserController@index

Metodo GET

URI /users

Nome della rotta

users.index

Azione - Controller e funzione da chiamare

UserController@index



Index - Controller

In questo metodo potremo semplicemente prendere dal DB tutti i record di un'entità e passarli ad una view.

Usiamo il **Model User** e il suo metodo `::all()` per prendere tutti i record di quella tabella.

```
1 1 use App\Models\User;
2
3 public function index() {
4     $users = User::all();
5
6     return view('users.index', compact('users'));
7 }
```



Index - View

Nella **view**

users/index.blade.php

Stampiamo una semplice tabella

Id	Username	Email	Mobile	Address
1	Mario Rossi	m.rossi@mail.it	123 46 78 90	Via Roma, 1
2	Luigi	l.bianchi@mail.it	46 789 120	Via Bari, 79/C
8	Anna Verdi	a.verdi@email.com	789 1234 56	Via Pisa, 143
9	Carla	f.carla@carfer.it	234 67 810	Via Milano, 5a
10	Alex Ricci	ricci.a@email.com	567 820	Via Potenza, 4
13	Lucia	lucia@lib.com	891 236 780	Via Torino, 13



RESTful CRUD - show

1 CRUD - Leggere i dati di uno specifico utente

Analizziamo la rotta show

URI	Name	Action
GET /users/{user}	users.show	UserController@show

Metodo GET

URI /users/{user}

Nome della rotta

users.show

Azione / Controller e funzione da chiamare

UserController@show



Show - Controller

Show() ci servirà per prendere un entità e vederla nel dettaglio.

Di default Laravel **usa come argomento un id**, che potremo usare per cercare il record nel nostro DB, sfruttando ad esempio il metodo **find()**.

```
1 1 public function show($id) {  
2     // Cerchiamo il nostro record tramite id  
3     $user = User::find($id);  
4  
5     // restituiamo la view con i dettagli dell'utente  
6     return view('users.show', compact('user'));  
7 }
```



Show - Controller

Dependency injection

Se al posto di \$id, come parametro della funzione show(), **passiamo un'istanza del Model**, in questo caso **User**, esso corrisponderà al record con id uguale a quello passato tramite la **URI**.

Basterà passare l'id alla rotta

e Laravel prenderà per noi il giusto record dal database.

In questo modo possiamo evitare di scrivere: **User::find(\$id)**, demandando il tutto a Laravel.

UserController.php

```
1 1 use App\Models\User;
2
3 public function show(User $user) {
4     return view('users.show', compact('user'));
5 }
```



Attenzione! Il nome della variabile deve combaciare con il nome del modello tutto minuscolo

index.blade.php

```
1 1 {{ route('users.show', $user->id) }}
```



LIVE CODING

Creiamo una Show



RESTful CRUD - create

2 CRUD - Creare un nuovo utente

Analizziamo la rotta create

URI	Name	Action
GET /users/create	users.create	UserController@create

Metodo GET

URI /users/create

Nome della rotta

users.create

Azione / Controller e funzione da chiamare

UserController@create



Create - Controller

Il metodo **create()** si occuperà di **restituire** semplicemente una view contenente **un form**.

Tramite questo form creeremo un nuovo record della nostra entità.

Home / Posts

Posts

ID

Title

Slug

Lead Photo

Lead Text

Search Reset

Create Post

No results found.



Create - Esempio di form

Dobbiamo inserire come valore dell'**attributo action** la rotta che avrà il compito di gestire i dati del form. Nel nostro caso si tratta della rotta **users.store**

Inoltre, l'attributo **method** deve essere uguale a **POST**.

Dobbiamo poi compilare correttamente gli **attributi name** di tutti gli input, in modo che combacino con i **nomi delle colonne** della tabella mappata con questa resource.

create.blade.php

```
1 1 <form action="{{ route('users.store') }}"  
   method="POST">  
2   @csrf  
3  
4   <label for="name">Nome</label>  
5   <input type="text" name="name" id="name">  
6  
7   <label for="lastname">Cognome</label>  
8   <input type="text" name="lastname" id="lastname">  
9  
10  <input type="submit" value="Invia">  
11 </form>
```



**Avete notato altri elementi
nuovi in questo form?**



@csrf token

@csrf

è un token che genera Laravel per assicurarsi che la chiamata post avvenga tramite un form del sito.

Dobbiamo inserirlo in ogni form.

<https://laravel.com/docs/9.x/csrf>

```
1 1 <form action="{{ route('users.store') }}"  
   method="POST">  
2   @csrf  
3  
4   <label for="name">Nome</label>  
5   <input type="text" name="name" id="name">  
6  
7   <label for="lastname">Cognome</label>  
8   <input type="text" name="lastname" id="lastname">  
9  
10  <input type="submit" value="Invia">  
11 </form>
```




RESTful CRUD - store

2 CRUD - Creare un nuovo utente

Analizziamo la rotta store

URI	Name	Action
POST /users	users.store	UserController@store

Metodo POST

URI /users

Nome della rotta

users.store

Azione / Controller e funzione da chiamare

UserController@store



Index vs Store

URI	Name	Action
GET /users	users.index	UserController@index
POST /users	users.store	UserController@store



Notate qualcosa di strano nelle URI?



Rotte + Methods



Possiamo usare la stessa URI con methods diversi.

Quando richiamiamo quell'indirizzo ci agganciamo ad una diversa funzione del Controller a seconda del method utilizzato.

URI	Name	Action
GET /users	users.index	UserController@index
POST /users	users.store	UserController@store



Store - Controller

In **store()** vengono passate le coppie **name-value** inviate dal form, tramite un'istanza della classe **Request \$request**.

Usiamo il suo metodo **all()** per ottenere tutte le coppie in un **array associativo**.

Per salvare i dati sfruttiamo il metodo **save()** del **Model**.

UserController.php

```
1 1 use App\Models\User;
2
3 public function store(Request $request) {
4     $data = $request->all();
5
6     $newUser = new User();
7     $newUser->name = $data['name'];
8     $newUser->lastname = $data['lastname'];
9     $newUser->save();
10
11     return redirect()->route('users.show', $newUser-
    >id);
12 }
```



LIVE CODING

Creiamo dei nuovi record nella nostra tabella