

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="">

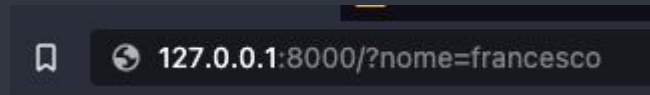
  <div class="mb-3">
    <label class="form-label">Nome</label>
    <input class="form-control" type="text" name="name" readonly
value=""
      placeholder="Nome Utente" />
  </div>

  <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 dovrai 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
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.

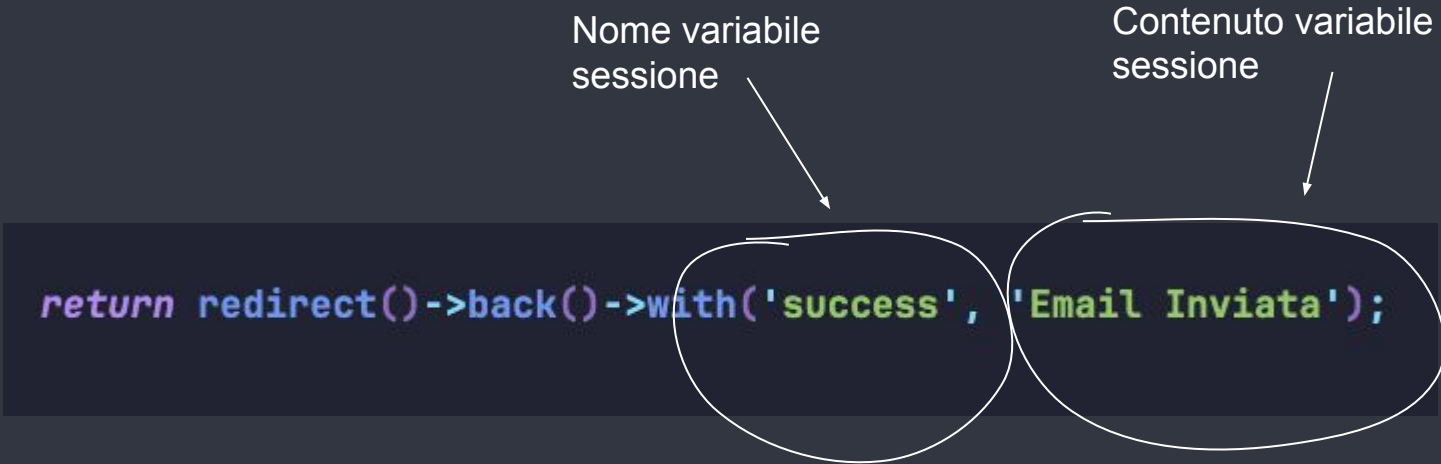
Ovvero non mantiene i dati tra una chiamata e l'altra. Ma Laravel (e PHP) si.

<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
```

The diagram illustrates the concept of flash data in a templating engine. It shows a code snippet where the variable name 'success' is highlighted with a white circle. An arrow points from the text 'Nome variabile da richiamare' (Variable name to be called) to this circle, indicating that 'success' is the variable name used to retrieve the flash data.