Event Loop

Event -> Listener -> Job -> Command - 1

Sidrit Trandafili

@sidis405

@strandafili

- Sviluppatore PHP 15+ anni
- Laravel da 4 anni
- Back-end
  - php
  - java
  - node
  - python
- Front-end
  - VueJS
  - Vanilla JS
  - TailwindCSS
  - Sass
- Devops + Sysops da 13 anni

Events, Listener, Job, Command - 1

Events, Listener, Job, Command - 1

“ L'Event Loop di Laravel è una semplice implementazione dell'Observer Pattern, e ci permette di emettere i nostri eventi custom in un qualsiasi punto dell'applicazione e tramite i Listeners per poi ascoltare in un altro punto disconnesso dell'applicazione. Un evento può avere più Listeners perché con molta probabilità dopo un certo evento devono succedere più cose. Questa metodologia ci permette di tenere il codice separato e isolato (SOLID) e altamente testabile con TDD

“ Dato che i Listeners sono classi a parte, è possibile far partire dei Job (che non sono altro che classi da eseguire) sia in modo sincrono che asincrono che a loro volta quando vengono gestiti dalla coda di elaborazione possono eseguire comandi artisan creati a misura da noi per l'applicazione

Events, Listener, Job, Command - 1

Ipotizziamo un'applicazione dove appena modificato un Post da un utente, gli utenti admin devono ricevere notifica via email per poterlo moderare.

Creiamo un evento PostWasUpdated

```
php artisan make:event PostWasUpdated // app/Events/PostWasUpdated.php
```

Dopo aver modificato il post, emettiamo il nostro evento, passandogli l'oggetto post appena creato

```
[...]
use App\Events\PostWasUpdated;

   class PostsController extends Controller
    {
        public function posts(Post $post)
        {
            //prima fare la validation e authorization
            $post = Post::update(request('title', 'body'));

            event(new PostWasUpdated($post));

            return view(...);
        }
    }
```

Events, Listener, Job, Command - 1

Al momento non abbiamo un listener per rispondere a tale evento

Creiamo un listener PostUpdateListener

```
php artisan make:listener PostUpdateListener // app/Listeners/PostUpdateListener.php
```

Immettiamo il Post nell'evento

```
// app/Events/PostWasUpdated.php
[..]

use App\Post;

class PostWasUpdated
{
    [..]

    public $post;


    [...]
    public function __construct(Post $post)
    {
        $this->post = $post;
    }
```

Aggiorniamo il Listener

```
// app/Listeners/PostUpdateListener.php

class PostUpdateListener
{
    [...]

    public function handle(PostWasUpdated $ev
    {
        $post = $event->post;

    // $admins = User::whereRole('admin')...
    // Invio email

    }
}
```

Events, Listener, Job, Command - 1

Notifichiamo a Laravel che ogni volta che viene emesso l'evento PostWasUpdated, deve richiamare PostUpdateListener e passare i dati dell'evento. Il binding degli eventi con i listener si fa in EventServiceProvider

```
namespace App\Providers;

[...]

class EventServiceProvider extends ServiceProvider
{
    protected $listen = [
        'App\Events\PostWasUpdated' => [
            'App\Listeners\PostUpdateListener',
//              'App\Listeners\SecondoListenerPerAlraAzione',
//              'App\Listeners\TerzoListenerPerAlraAzione',
        ],
    ];
```