Intermediate + Advanced

Model Events + Observers

Intermediate + Advanced

Sidrit Trandafili

@sidis405

@strandafili

Front-end

​VueJS

Vanilla JS

TailwindCSS

Sass

Devops + Sysops da 13 anni

Sviluppatore PHP 15+ anni

Laravel da 4 anni

Back-end

php

java

node

python

Intermediate + Advanced

Model Events + Observers

Intermediate + Advanced

*Model Events + Observers internamente “ Ogni volta che un evento. compiamo È possibile un azione ascoltare su un modello, quando questi Laravel eventi emette*

*vengono emessi e reagire in modo adeguato*

Gli eventi più comuni che Laravel emette per un model sono:

saving - Il modello sta per essere salvato - Prima della creazione/aggiornamento

saved - Il modello e' stato salvato - Dopo la creazione/aggiornamento

created - Il modello e' stato creato - Dopo la creazione

updated - Il modello e' stato aggiornato - Dopo l'aggiornamento

deleting - Il modello sta per essere cancellato - Prima della cancellazione

deleted - Il modello e' stato aggiornato - Dopo la cancellazione

Intermediate + Advanced

Come ascoltare e reagire agli eventi

Ipotizziamo un'applicazione dove nel listato degli utenti deve essere associato anche il titolo e data dell'ultimo post.

class User extends Model

{

//Tutti i post per questo User public function posts() {

return $this->hasMany(Post::class); } }

Model Events + Observers

Ulteriormente possiamo prendere l'ultimo post in una relazione tutta sua

class User extends Model

{

//L'ultimo post per questo User public function lastPost() {

return $this->posts()->latest()->take(1); } }

Intermediate + Advanced

Model Events + Observers

Come ascoltare e reagire agli eventi

$users = User::with('lastPost')->get();

Il problema in questo caso è che la

query non è efficace. Prima deve fare

un join di tutti gli utenti con tutti i loro

post, poi li deve ordinare in ordine

temporale discendente, e dunque

prenderne solo uno. Possiamo invece

creare un campo last\_post\_id nella

tabella users dove

programmaticamente mettere l'id

dell'ultimo post. Un primo approccio

sarebbe farlo dove si crea il post model

//Aggiorniamo la relation public function lastPost() {

return $this ->belongsTo('Post', 'last\_post\_id'); }

public function store() {

//creiamo il post e lo associamo a questo utente $post = auth()->user()->posts()

->create(request()->only('title', 'body'); //aggiorniamo l'id dell'ultimo post nel record user auth()->user()->last\_post\_id = $post->id(); auth()->user()->save(); }

Intermediate + Advanced

Come ascoltare e reagire agli eventi

E se dovessimo fare la stessa cosa per l'aggiornamento? È se un utente moderatore cambia l'autore

del post? Dovremmo in questo caso ripetere la questa funzione in più posti rendendo abbassando

la qualità del codice rendendolo difficile da mantenere. Vediamo due modi per risolvere questa

situazione

Model Events + Observers

Model Boot

class Post extends Model

{

public static function boot() {

parent::boot();

static::created(function ($post) {

$post->user->last\_post\_id = $post->id; $post->user->save(); }); } }

La funzione boot() viene richiamata per ogni istanza

del modello. Osserviamo created.

Questo evento viene emesso appena creato un

record e contiene una copia di quel record. In questo

modo il model è autosufficiente nello svolgere il

compito.

Intermediate + Advanced

Model Events + Observers

Come ascoltare e reagire agli eventi

I problemi di questa soluzione sono 3: in primo luogo se dovessimo supportare gli altri casi,

dovremmo scrivere tutto il codice nel metodo boot(). Secondo, è difficile da testare e terzo, tutte le

eventuali modifiche dovrebbero avvenire sempre su questo file.

Model Observer

<?php

Creiamo la classe PostObserver che verrà usata per reagire ai vari namespace App\Observers;

eventi model. use App\Post; class PostObserver

Definiamo una funzione created(), che come nell'esempio precedente {

/\*\*

\* Listen to the Post created event.

riceverà da Laravel un istanza del record creato. Qua possiamo in

\* @param \App\Post $post \*/

autonomia svolgere le elaborazioni.

public {

function created(Post $post)

Per supportare static::updated, basta creare un nuovo metodo // } }

Intermediate + Advanced

Model Events + Observers

Come ascoltare e reagire agli eventi

Adesso dobbiamo dire a Laravel, che ogni volta che un'operazione viene svolta sul modello Post, deve richiamare la nostra classe Observer.

Per questo usiamo il metodo ::observe nel AppServiceProvider boot().

[...] class AppServiceProvider extends ServiceProvider {

public function boot() {

Post::observe(PostObserver::class); } } [...]