SEARCH

Creiamo una ricerca ***fulltext***, dove un sistema indicizza le parole significative di un testo e crea un ***indice inverso*** dove ogni parola viene affiancata all'id dell'articolo nel quale e' contenuta.

Per farlo dobbiamo installare ***Scout***, un pacchetto che serve per integrare un engine di ricerca *fulltext* per Laravel e indicizzare i modelli Eloquent.
Esso utilizza i driver di Algolia.

- Installazione

Nella bash:

```
composer require laravel/scout
php artisan vendor:publish --provider="Laravel\Scout\ScoutserviceProvider"
```

Abbiamo installato il pacchetto e i *driver*.
Adesso scarichiamo un *driver* dedicato a Laravel di ***TntSearch***.

Link Driver TntSearch -> installation

```
composer require teamtnt/laravel-scout-tntsearch-driver
```

In *Config\app.php* ***se non sono già presenti*** inseriamo le seguenti classi per indicare a Laravel di far partire questi servizi all'avvio:

```
//Application Service Providers...
...
Laravel\Scout\ScoutServiceProvider::class,
TeamTNT\Scout\TNTSearchScoutServiceProvider::class,
```

x---------
In *Config\scout.php* modifichiamo in:

```
'driver' => env('SCOUT_DRIVER', 'tntsearch'),
```

oppure

Aggiungo al file *.env*:

```
SCOUT_DRIVER=tntsearch
```

oppure

In *Config\scout.php* modifichiamo in e non tocchiamo il *.env*:

```
'driver' => 'tntsearch',
```

---------x

In *Config\scout.php* aggiungiamo alla fine del file:

```
'tntsearch' => [
        'storage' => storage_path(),
        'fuzziness' => true,
        'fuzzy' => [
                'prefix_length' => 3,
                'max_expansions' => 50,
                'distance' => 3
        ],
        'asYouType' => false,
        'searchBoolean' => true,
],
```

fuzziness . Con *true* permettiamo alla ricerca di essere effettuata anche con degli errori di battitura

fuzzy . I valori all'interno determinano quanto morbida deve essere l'accettazione degli errori

- Con le modifiche apportate stiamo dicendo che vogliamo effettuare questo tipo di modifiche per tutti gli enviroment in cui lavoriamo.

In *.gitignore* aggiungiamo:

```
/storage/*.index
```

- Impostazione di Scout

Scout intercetta tutte le modifiche e cancellazioni di un modello, contatta il driver di indicizzazione fulltext e comunica le modifiche per quindi aggiornarle.

Per indicare che un determinato modello e' soggetto ad una indicizzazione fulltext, aggiungiamo al *Model:*

```
use Laravel\Scout\Searchable;

class Brewery extends Model{

        use Searchable;

public function toSearchableArray(){

        $birre = $this->beers->pluck('name')->join(', ');
        $array = [
                'id' => $this->id,
                'name' => $this->name,
                'description' => $this->description,
                'altro' => 'birrerie birra',
                'birre' => $birre
        ];

        return $array;
}
```

toSearchableArray . Indichiamo nell'array quali campi voglio che siano indicizzabili.
$array . Specifichiamo i campi della tabella che vogliamo siano indicizzati.
'altro' . Indichiamo delle parole chiave che porteranno a visualizzare *tutti* i record.
$birre . Diamo *$this* birreria come valore alla variabile, e diciamogli di catturare tutti i ***name*** delle ***beers*** che ha collegate.
pluck() . Metodo che ritorna un array di una determinata colonna di una tabella.
join() . Metodo che ritorna in stringa.

Per approfondire i metodi sulla Documentazione cerchiamo *Available Collections Methods*.

- Importiamo in Scout

Nella bash diamo il comando per *indicizzare* il *Model*:

```
php artisan scout:import "App\Model\Brewery"
```

Questo crea un file *brewery.index* in ***SQL LiteFormat***.
Con il programma ***DBbrowser*** posso esplorare questo file che contiene le tabelle con i collegamenti delle indicizzazioni.

Nel ***Tinker*** possiamo sperimentare le ricerche:

```
App\Brewery::search('Peroni -Moretti')->get();
```

In questo modo troveremo tutte le birrerie che hanno una Peroni MA non hanno anche una Moretti.

- Interfaccia

Nell'HTML creo il *Form* di ricerca ***get***:

```
<form action="{{route('search')}}" method="GET">
      <input type="text" name="q" placeholder="Search" />
      <button class="btn btn-danger" type="submit">Ricerca</button>
</form>
```

Creo una *rotta* search:

```
Route::get('/search', [HomeController::class , 'search'])->name('search);
```

Nel Controller:

```
public function search(Request $request){

      $q = $request->input('q');
      $breweries = Brewery::search($q)->where('visible', true)->get();

      return view('search_results', compact('q','breweries'));
```

where() . Specifichiamo di visualizzare soltanto le birrerie visibili (*revisor*)

- Creo la view

Creo una ***view*** per mostrare i risultati di ricerca ciclando all'interno le *collection* che ricevo dalla ricerca.

- Coda sul database

Quando Laravel comunica a Scout una **modifica** di un *Model*, in caso di Model molto complessi, essa può richiedere del tempo.
Per accorciare questi tempi e rendere più agile questo scambio di informazioni, *Scout* attiva un processo chiamato ***Job*** che permette di mettere in ***coda*** i processi e sganciare Laravel dall'attesa della risposta.

Questi processi possono essere memorizzati sul *Database*, su *Redis* (database in memoria), o in altre modalità più avanzate.

Nella bash diamo il comando per creare le migrazioni dedicate alle code:

```
php artisan queue:table
php artisan migrate
yes
```

Questo crea la tabella *Jobs* nel nostro Database.

Nel *.env*:

```
QUEUE_CONNECTION=database
```

x----------

In *app\Config\Scout.php* modifico e abilito la *queue*:

```
'queue' => env('SCOUT_QUEUE', true),
```

oppure

Nel *.env*:

```
SCOUT_QUEUE=true
```

----------x

Diamo il comando per permettere il passaggio del dato al ***worker*** che si occuperà di elaborarlo:

```
php artisan queue:work
```