VIKI53

Zeste de savoir

Les scopes locaux dans Laravel/Eloquent

12 mai 2021

Table des matières

Créer un blog dans Laravel est assez facile grâce aux outils intégrés au framework. L'un d'eux s'est d'ailleurs montré très pratique pour gérer par exemple les états des articles et commentaires, ou les catégories à afficher…

Filtrer les articles publiés est chose courante sur le blog. Ce filtre est utilisé pour afficher des articles à plusieurs endroits, il est donc important d'éviter la duplication de code.

1. Récupérer les articles par état de publication

Ayant également besoin de filtrer les articles privés (accessibles uniquement via un lien direct, pratique pour demander une relecture) et les brouillons (uniquement accessibles par moi-même), notamment dans l'interface de gestion, je me suis créé des filtres assez simples:

```
1  public function scopePublished(Builder $query)
2  {
3      return $query->where($this->table . '.status',
       'public')->where($this->table . '.published_at', '<=',
       Carbon::now())->orderBy('published_at', 'DESC');
4  }
5  public function scopePrivate(Builder $query)
6  {
7      return $query->where($this->table . '.status',
       'private')->orWhere(function (Builder $query) {
8          $query->where($this->table . '.status',
       'public')->where($this->table . '.published_at', '>',
       Carbon::now());
9      })->orderBy('published_at', 'DESC')->orderBy('created_at',
       'DESC');
10 }
11 public function scopeDraft(Builder $query)
12 {
```

```
13      return $query->where($this->table . '.status',
        'draft')->orderBy('created_at', 'DESC');
14  }
```

Les articles sont encore considérés comme privés s'ils sont marqués comme publiés mais que la date de publication n'est pas encore passée.

2. Utiliser les portées

Une fois le *scope* défini il s'utilise comme une méthode classique, sans préciser `scope` dans le nom:

```
1  $posts =
       BlogPost::published()->withCount('comments')->paginate(10);
```

3. Utiliser un scope sur une relation

? C'est bien pratique tout ça, mais pour les relations ça marche comment?

Dans l'exemple ci-dessus vous avez vu que je chargeais le nombre de commentaires… sans filtrer leur état. Et si on comptait uniquement les commentaires publiés pour afficher un nombre qui correspond à ce qui est visible?

```
1  $posts = BlogPost::published()->withCount(['comments' => function
       ($query) {
2      return $query->published();
3  }])->paginate(10);
```

C'est là que l'utilisation de `$this->table` dans la définition des scopes prend tout son sens: ça évite les conflits entre deux tables jointes dont les champs filtrés ont le même nom.

3.1. Cumuler les filtres sur une relation

Ce dernier était pratique pour les visiteurs non connectés… mais si je veux afficher le nombre de commentaires en attente de validation pour l'auteur (par exemple dans l'interface de gestion)? Eh bien je peux utiliser plusieurs fois la relation `comments` en la renommant:

```
$posts = BlogPost::withCount([
    'comments as published_comments_count' => function(Builder
    $query) {
        return $query->published();
     },
     'comments as pending_comments_count' => function(Builder
    $query) {
        return $query->pending();
     },
]);
```

3.2. Les scopes de relations pré-définis

Eloquent propose aussi des filtres pré-définis pour les relations, par exemple pour travailler sur l'existence d'une relation. Pratique pour n'afficher que les catégories qui contiennent des articles publiés:

```
$categories = BlogCategory::whereHas('posts', function(Builder
    $query) {
    $query->published();
})->orderBy('title', 'ASC')->get();
```

En utilisant `whereHas` je filtre les catégories qui ont des articles liés, mais en fournissant une *closure* je peux appliquer un filtre supplémentaire sur l'état des articles. C'est comme ça que fonctionne la liste dans le menu du blog. 🍊

4. Bonus : les accesseurs pour calculer l'état d'un article

Une fois un article sorti de la base de donnée, utiliser un *scope* pour savoir s'il est publié ou non a très peu de sens. Heureusement ou peut définir un **accesseur** pour connaître son état.

```
public function getIsPublicAttribute()
{
    return $this->status === 'public' &&
    !empty($this->published_at) && $this->published_at->isPast();
}
public function getIsPrivateAttribute()
{
    return $this->status === 'private' || ($this->status ===
    'public' && (empty($this->published_at) ||
    $this->published_at->isFuture()));
```

4. Bonus: les accesseurs pour calculer l'état d'un article

```
}
public function getIsDraftAttribute()
{
    return $this->status === 'draft';
}
```

L'attribut `published_at` étant une date (gérée via la propriété `$casts` du modèle), on peut utiliser `isPast()` et `isFuture()` de la librairie Carbon .

Vous pouvez retrouver l'article d'origine sur mon blog