Implementing Bootable Eloquent Model Traits

Before we actually leave this section, let's make our global scopes, beside reusable it also easy to implement. At this point, we can apply our global scopes (`SearchScope` and `FilterScope`) in a certain model by assign them in model's `booted` method like so:

```
protected static function booted()
{
    static::addGlobalScope(new SearchScope);
    static::addGlobalScope(new FilterScope);
}
```

Then define the `filterColumns` and `SearchColumns` properties in your model like so:

```
class Contact extends Model
{
    public $filterColumns = ['company_id'];
    public $searchColumns = ['first_name', 'last_name', 'email'];
    // ...
}
```

Now in this lesson we're going to learn another technique to make our global scopes much easier to implement into a model.

1. Put the global scopes in a Trait

To achieve that you can follow these steps.

1. Go to `Contact` model, then cut the model's `booted` method. Now your `Contact` model will look like this:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory; // This added in Laravel 8+

class Contact extends Model
{
    use HasFactory; // This added in Laravel 8+

    protected $fillable = ['first_name', 'last_name', 'email', 'phone', 'address',
'company_id'];
    public $filterColumns = ['company_id'];
```

```
    public function company()
    {
        return $this->belongsTo(Company::class);
    }

    public function scopeLatestFirst($query)
    {
        return $query->orderBy('id', 'desc');
    }
}
```

2. Go to `Scopes` folder, then create a new file called `FilterSearchScope.php`. In that file define a `FilterSearchScop` trait as follow.

```
<?php

namespace App\Scopes;

trait FilterSearchScope
{

}
```

3. Paste the `booted` method that you grabbed from the `Contact` model. Change the method name from `booted` to `bootFilterSearchScope`.

```
<?php

namespace App\Scopes;

trait FilterSearchScope
{
    protected static function bootFilterSearchScope()
    {
        static::addGlobalScope(new ContactSearchScope);
        static::addGlobalScope(new FilterScope);
    }
}
```

> Note that by convention all static method named `boot[TraitName]` that you defined on your `trait` will be executed by the `boot()` method on Eloquent model.

4. To make it consistent let's use the `SearchScope` instead of `ContactSearchScope`.

```php
<?php

namespace App\Scopes;

trait FilterSearchScope
{
    protected static function bootFilterSearchScope()
    {
        static::addGlobalScope(new SearchScope);
        static::addGlobalScope(new FilterScope);
    }
}
```

2. Use the `FilterSearchScope` trait in the model

Back to `Contact` model then use the `FilterSearchScope` trait that we've just created. Also since we assigned the `SearchScope` in the trait we also need to define the `searchColumns` on the model.

```php
<?php

namespace App\Models;

use App\Scopes\FilterSearchScope;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class Contact extends Model
{
    use HasFactory, FilterSearchScope;

    protected $fillable = ['first_name', 'last_name', 'email', 'phone', 'address',
'company_id'];
    public $searchColumns = ['first_name', 'last_name', 'email'];
    public $filterColumns = ['company_id'];

    public function company()
    {
        return $this->belongsTo(Company::class);
    }

    public function scopeLatestFirst($query)
    {
        return $query->orderBy('id', 'desc');
    }
}
```

So now, whenever you want to apply the `FilterScope` and `SearchScope` scopes together in your model, you only need to use the `FilterSearchScope` on your model, then specify the filter columns in

`filterColumns` and search columns in `searchColumns`. I think this way is much simpler, right?

Now, let's finally move on to the next lesson.