Laravel 4 : chapitre 21 : Un blog : utilisation de contrôleurs

Nous allons continuer l'application « blog » en essayant de mieux organiser le code. Dans la précédente étape nous avons utilisé les routes pour effectuer tout le traitement. Ça fonctionne correctement mais ce n'est pas très lisible. Alors je vous propose d'utiliser un contrôleur pour faire le traitement et ne conserver dans les routes que l'aiguillage de requêtes.

Les routes

Le fichier des routes ne conserve que ce pour quoi il est fait, le routage :

```
<?php
Route::get('/', array('uses' => 'HomeController@accueil', 'as' => 'accueil'));

Route::model('cat', 'Categorie');
Route::get('cat/{cat}', 'HomeController@categorie');

Route::model('art', 'Article');
Route::get('art/{cat_id}/{art}', 'HomeController@article');

Route::post('find', 'HomeController@find');
Route::post('comment', 'HomeController@comment');
```

J'ai prévu de nommer une seule route, pour l'accueil, je trouve que le code est ensuite plus lisible et ça permet de montrer comment on nomme une route vers un contrôleur par la même occasion. J'ai conservé l'injection d'instance des modèles Categorie et Article qui rend la syntaxe plus légère.

Le contrôleur

Le contrôleur HomeController est chargé de toutes les actions pour les articles :

```
<?php

class HomeController extends BaseController {

        public function __construct()
        {
                $this->beforeFilter('csrf', ['on' => 'post']);
        }
  /**
        * Génère la vue de l'accueil
        *
        * @param $articles
        * @return View
        */
        private function gen_accueil($articles)
        {
                return View::make('accueil',  array(
                        'categories' => Categorie::all(),
                        'articles' => $articles,
                        'actif' => 0
                ));
        }

  /**
        * Affiche la page d'accueil
        *
        * @return View
        */
        public function accueil()
        {
                   $articles = Article::select('id', 'title',
'intro_text')
                        ->orderBy('created_at', 'desc')
                        ->paginate(4);
                return $this->gen_accueil($articles);
        }
```

```
  /**
        * Affiche les articles d'une catégorie
        *
        * @param $categorie
        * @return View
        */
        public function categorie(Categorie $categorie)
        {
                                                $articles   =
$categorie->articles()->orderBy('created_at',
'desc')->paginate(4);
          return View::make('accueil', array('categories' =>
Categorie::all(), 'articles' => $articles, 'actif' =>
$categorie->id));
        }

  /**
        * Affiche un article
        *
        * @param $cat_id
        * @param $article
        * @return View
        */
        public function article($cat_id, Article $article)
        {
//////////////////////////////////////////////////////////
                // Log d'un utilisateur pour tester la saisie des
commentaires
                    $user = User::where('username', '=',
'admin')->first();
                Auth::login($user);
//////////////////////////////////////////////////////////
                                          $comments         =
$article->comments()->orderBy('comments.created_at', 'desc')
                                     ->join('users', 'users.id',
'=', 'comments.user_id')
                                      ->select('users.username',
'title', 'text', 'comments.created_at')
                                    ->get();
      return View::make('article', array('categories' =>
Categorie::all(), 'article' => $article, 'actif' => $cat_id,
'comments' => $comments));
        }
```

```
  /**
        * Traitement du formulaire de recherche
        *
        * @return View ou Redirect
        */
        public function find()
        {
                $match = Input::get('find');
    if($match) {
      $articles = Article::select('id', 'title', 'intro_text')
->orderBy('created_at', 'desc')
->where('intro_text', 'like', '%'.$match.'%')
->orwhere('full_text', 'like', '%'.$match.'%')
->get();
                        Session::flash('flash_notice', 'Résultats
pour la recherche du terme '.$match);
                       return $this->gen_accueil($articles);
    }
     return Redirect::to('/')->with('flash_error', 'Il faudrait
entrer un terme pour la recherche !');
        }

  /**
        * Traitement du formulaire d'ajout de commentaires
        *
        * @return Redirect
        */
        public function comment()
        {
                Comment::create(array(
                        'title' => Input::get('title'),
                        'user_id' => Auth::user()->id,
                        'article_id' => Input::get('id_art'),
                        'text' => Input::get('comment')
                ));
                            return  Redirect::to(url('art',
array(Input::get('id_cat'), Input::get('id_art'))));
        }

}
```

Il reprend le code que nous avons déjà vu précédemment. J’ai cette fois bien séparé les fonctions, en particulier entre

l'accueil et la recherche. Remarquez l'application dans le constructeur du filtre csrf pour la méthode POST. On pourrait aussi mettre ce filtre dans les routes en groupant les deux concernées.

Au niveau de l'affichage des articles j'ai prévu l'authentification artificielle d'un utilisateur pour avoir accès à la saisie des commentaires. On supprimera ce code dans un chapitre ultérieur consacré à l'authentification.

Les vues

Les vues on subi quelques améliorations que je vous livre sans commentaires.

Voici le template template.blade.php :

```
<!DOCTYPE html>
<html lang="fr">
        <head>
                <meta charset="utf-8">
                <title>
                        Mon joli blog
                </title>
                <meta name="viewport" content="width=device-width,
initial-scale=1.0">
                                                                {{
HTML::style('//netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstr
ap.min.css') }}
                                                                {{
HTML::style('//netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstr
ap-theme.min.css') }}
                {{ HTML::style('assets/css/main.css') }}
                                                                {{
HTML::style('http://fonts.googleapis.com/css?family=Imprima') }}
        </head>

        <body>
                <div class="container">

                  <header class="jumbotron" id="entete">
```

```
                    <h1>Mon joli blog !</h1>
                  </header>

                  <nav class="navbar navbar-default">
                    <div class="navbar-header">
                       <button type="button" class="navbar-toggle"
data-toggle="collapse" data-target=".navbar-collapse">
                         <span class="icon-bar"></span>
                         <span class="icon-bar"></span>
                         <span class="icon-bar"></span>
                       </button>
                        <a class="navbar-brand" href="#">Mon joli
blog</a>
                    </div>
                    <div class="collapse navbar-collapse">
                      <ul class="nav navbar-nav">
                        @yield('navigation')
                      </ul>
                         {{ Form::open(array('url' => 'find',
'method' => 'POST', 'class' => 'navbar-form navbar-left pull-
right')) }}
                                              {{ Form::text('find', '',
array('class' => 'form-group form-control', 'placeholder' =>
'Recherche')) }}
                                             {{ Form::close() }}
                    </div>
                  </nav>

                       <section class="row">
                               @yield('content')
                       </section>

                  <footer>
                    <em>© 2013</em>
                  </footer>

                </div>
                                                                    {{
HTML::script('//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery
.min.js'); }}
                                                                    {{
HTML::script('//netdna.bootstrapcdn.com/bootstrap/3.0.2/js/bootstr
ap.min.js'); }}
```

```
        </body>
</html>
```

La navigation navigation.blade.php :

```
@section('navigation')
  <li>{{ link_to_route('accueil', 'Accueil', null, ($actif == 0)?
array('class' => 'actif'): null) }}</li>

  @foreach ($categories as $categorie)
      <li>{{ link_to('cat/'.$categorie->id, $categorie->title,
($actif == $categorie->id)? array('class' => 'actif'): null)
}}</li>
  @endforeach
@stop
```

L'accueil accueil.blade.php :

```
@extends('template_blog')

@include('navigation')

@section('content')

    @if (Session::has('flash_notice'))
        <div class="col-md-12">
            <div class="alert alert-success">
                {{ Session::get('flash_notice') }}
            </div>
        </div>
    @endif
    @if (Session::has('flash_error'))
        <div class="col-md-12">
            <div class="alert alert-danger">
                {{ Session::get('flash_error') }}
            </div>
        </div>
    @endif

    @for ($i = 0; $i < count($articles); $i++)
        <div class="col-md-6">
            <h3>{{$articles[$i]->title}}</h3>
            <p>{{$articles[$i]->intro_text}}</p>
                    <a  class="btn  btn-default"  href="{{
```

```
url('art/'.$actif.'/'.$articles[$i]->id) }}">Lire la suite <span
class="glyphicon glyphicon-play"></span></a>
        </div>
    @endfor
    @if (method_exists($articles,'links'))
                {{$articles->links()}}
    @endif
@stop
```

Et enfin la vue pour les articles article.blade.php :

```
@extends('template_blog')

@include('navigation')

@section('content')

        <div class="col-md-12">
                <div class="well">
                        <em class="pull-right">
                                        Ecrit le {{date('d-m-
Y',strtotime($article->created_at))}}
                        </em>
                        <h3>
                                {{$article->title}}
                        </h3>
                        <p>
                                {{$article->full_text}}
                        </p>
                </div>
        </div>

@foreach ($comments as $comment)
        <div class="col-md-12">
                <div class="comment">
                        <em class="pull-right">
                                 Ecrit par {{$comment->username}}
le {{date('d-m-Y',strtotime($comment->created_at))}}
                        </em>
                        <h5>
                                {{$comment->title}}
                        </h5>
                        <p>
                                {{$comment->text}}
```

```
                        </p>
                </div>
        </div>
@endforeach

@if ($article->allow_comment and Auth::check())
<hr />
        <div class="col-md-12">
                <div class="well">
      {{ Form::open(array('url' => 'comment')) }}
                        {{ Form::hidden('id_art', $article->id) }}
                        {{ Form::hidden('id_cat', $actif) }}
                         {{ Form::label('title', 'Titre de votre
commentaire :') }}
                         {{ Form::text('title', '', $attributes =
array('class' => 'form-control')) }}
                            {{ Form::label('comment', 'Votre
commentaire :') }}
                            {{ Form::textarea('comment', '',
$attributes = array('class' => 'form-control')) }}
                         {{ Form::submit('Envoyer', array('class'
=> 'btn btn-default')) }}
                        {{ Form::close() }}
                </div>
        </div>
@endif

@stop
```

Conclusion

Le code est maintenant mieux organisé avec un fichier de routes léger et lisible et un contrôleur pour assurer la gestion de l'application. Le tout est-il maintenant satisfaisant ?

Pour le moment nous allons nous en satisfaire mais ce n'est pas une idée excellente de trouver le traitement des données dans le contrôleur. Dans l'idéal un contrôleur devrait juste recevoir des requêtes et leur apporter des réponses. Il existe un principe nommé « Repository Pattern » qui consiste à créer un intermédiaire entre le contrôleur et les modèles d'Eloquent. Nous verrons dans

des articles ultérieurs comment effectuer ce découplage.