Laravel 大型项目系列教程（三）

一、前言

上一节教程中完成了用户管理，这节教程将大概完成发表 Markdown 格式文章并展示的功能。

二、Let's go

1.数据库迁移

文章模块中我们需要建立 *articles*、*tags* 以及 *article_tag* 表，每篇文章会有一到多个标签，每个标签会有一到多篇文章，创建迁移文件：

```
$ php artisan migrate:make create_articles_table --create=articles
$ php artisan migrate:make create_tags_table --create=tags
$ php artisan migrate:make create_article_tag_table --create=article_tag
```

修改*_create_articles_table.php*：

```
Schema::create('articles', function(Blueprint $table)
{
    $table->increments('id');
    $table->string('title');
    $table->string('summary')->nullable();
    $table->text('content');
    $table->text('resolved_content');
    $table->integer('user_id');
    $table->softDeletes();
    $table->timestamps();
});
```

修改*_create_tags_table.php*：

```
Schema::create('tags', function(Blueprint $table)
{
    $table->increments('id');
    $table->string('name')->unique();
    $table->integer('count')->default(0);
    $table->softDeletes();
    $table->timestamps();
```

```
});
```

修改*_create_article_tag_table.php：

```
Schema::create('article_tag', function(Blueprint $table)
{
    $table->increments('id');
    $table->integer('article_id');
    $table->integer('tag_id');
});
```

执行迁移：

```
$ php artisan migrate
```

2.创建 Article 和 Tag 模型

创建 Article 和 Tag 模型：

```
$ php artisan generate:model article
$ php artisan generate:model tag
```

先在 User.php 中增加：

```
public function articles()
{
    return $this->hasMany('Article');
}
```

一个用户会有多篇文章。

修改 Article.php：

```
use Illuminate\Database\Eloquent\SoftDeletingTrait;

class Article extends \Eloquent {

    use SoftDeletingTrait;

    protected $fillable = ['title', 'content'];

    public function tags()
    {
        return $this->belongsToMany('Tag');
    }

    public function user()
    {
        return $this->belongsTo('User');
    }
}
```

一篇文章会有多个标签并属于一个用户。

修改 *Tag.php* ：

```
use Illuminate\Database\Eloquent\SoftDeletingTrait;

class Tag extends \Eloquent {

	use SoftDeletingTrait;

	protected $fillable = ['name'];

	public function articles()
	{
		return $this->belongsToMany('Article');
	}
}
```

一个标签会有多篇文章。

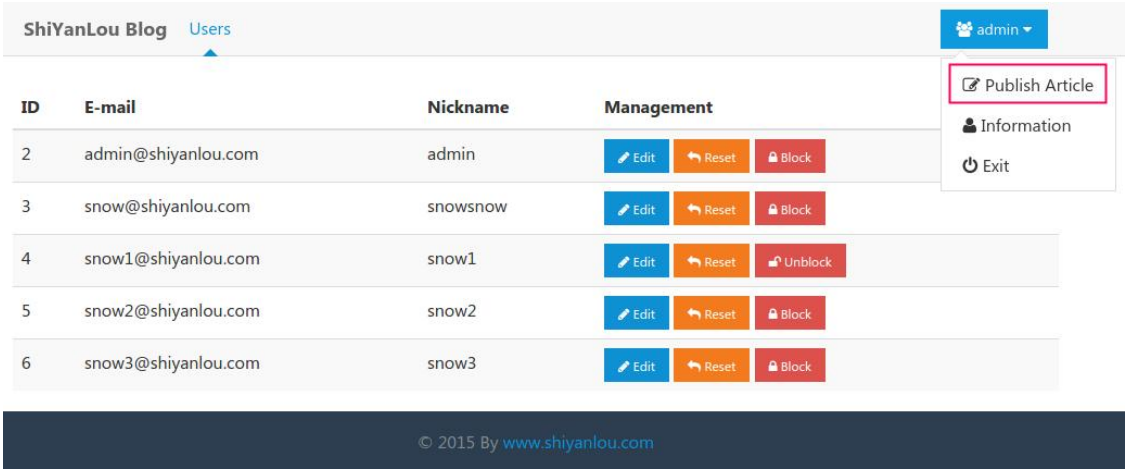
上面用到了软删除，*belongsToMany* 用于多对多关联。

3.发表文章视图

首先在导航条 *nav.blade.php* 中添加一个发表文章的选项：

```
<li><a href="{{ URL::to('article/create') }}"><span class="am-icon-edit"></span> Publish Article</a></li>
```

这时候登录会发现多了一个选项：



下面创建视图：

```
$ php artisan generate:view articles.create
```

修改 *articles/create.blade.php* ：

```
@extends('_layouts.default')

@section('main')
<div class="am-g am-g-fixed">
  <div class="am-u-sm-12">
```

```
        <h1>Publish Article</h1>
        <hr/>
        @if ($errors->has())
        <div class="am-alert am-alert-danger" data-am-alert>
          <p>{{ $errors->first() }}</p>
        </div>
        @endif
        {{ Form::open(array('url' => 'article', 'class' => 'am-form')) }}
            <div class="am-form-group">
              <label for="title">Title</label>
              <input id="title" name="title" type="text" value="{{ Input::old('title') }}"/>
            </div>
            <div class="am-form-group">
              <label for="content">Content</label>
              <textarea id="content" name="content"
rows="20">{{ Input::old('content') }}</textarea>
              <p class="am-form-help">
                  <button id="preview" type="button" class="am-btn am-btn-xs
am-btn-primary"><span class="am-icon-eye"></span> Preview</button>
              </p>
            </div>
            <div class="am-form-group">
              <label for="tags">Tags</label>
              <input id="tags" name="tags" type="text" value="{{ Input::old('tags') }}"/>
              <p class="am-form-help">Separate multiple tags with a comma ","</p>
            </div>
            <p><button type="submit" class="am-btn am-btn-success"><span
class="am-icon-send"></span> Publish</button></p>
        {{ Form::close() }}
    </div>
</div>

<div class="am-popup" id="preview-popup">
    <div class="am-popup-inner">
        <div class="am-popup-hd">
            <h4 class="am-popup-title"></h4>
            <span data-am-modal-close
                  class="am-close">&times;</span>
        </div>
        <div class="am-popup-bd">
        </div>
    </div>
</div>
<script>
    $(function() {
        $('#preview').on('click', function() {
            $('.am-popup-title').text($('#title').val());
            $.post('preview', {'content': $('#content').val()}, function(data, status) {
                $('.am-popup-bd').html(data);
            });
            $('#preview-popup').modal();
```

```
    });
  });
</script>
@stop
```

开始上一节中我们发现 *routes.php* 中的代码已经很零乱，那是因为我们把业务逻辑写在了这个文件中，对于文章模块我们把业务逻辑写在控制器中，首先创建一个文章控制器：

```
$ php artisan generate:controller ArticleController
```

我们会发现在 *app\controllers* 下多了一个 *ArticleController.php* 文件，它是一个资源控制器，**在 *routes.php* 中增加**：

```
Route::resource('article', 'ArticleController');
```

对应路由如下：

资源控制器对应的动作

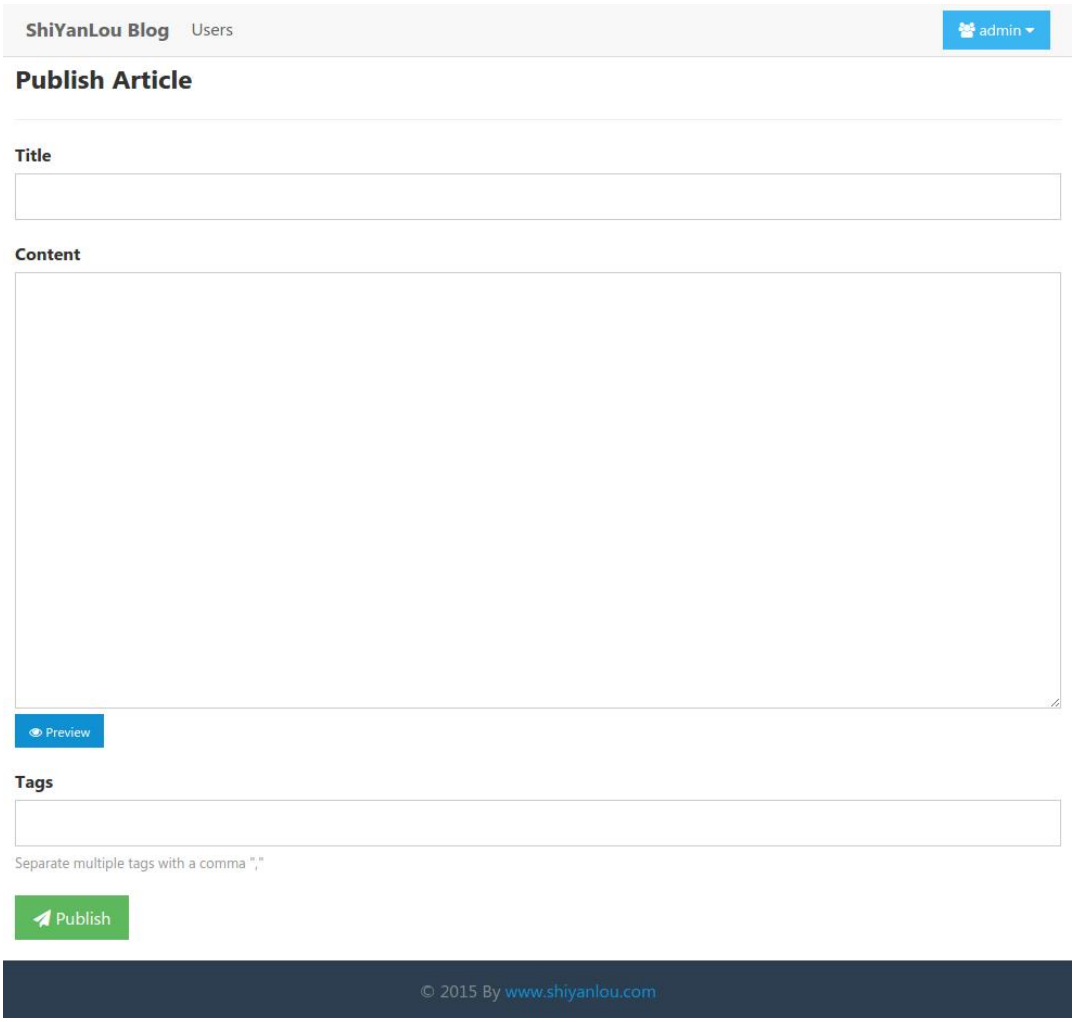
Verb	Path	Action	Route Name
GET	/resource	index	resource.index
GET	/resource/create	create	resource.create
POST	/resource	store	resource.store
GET	/resource/{resource}	show	resource.show
GET	/resource/{resource}/edit	edit	resource.edit
PUT/PATCH	/resource/{resource}	update	resource.update
DELETE	/resource/{resource}	destroy	resource.destroy

现在在 *ArticleController.php* 中增加过滤器并修改 *create* 方法：

```
public function __construct()
{
    $this->beforeFilter('auth', array('only' => array('create', 'store', 'edit', 'update', 'destroy')));
}

public function create()
{
    return View::make('articles.create');
}
```

这时在登录后点击 *Publish Article* 选项后，会出现发表文章的页面：



4.文章预览

这里我们将使用 *Markdown* 格式来编写文章，同时需要提供一个预览功能，先需要安装 *Markdown* 解析插件，在 *composer.json* 的 *require* 中增加：

"maxhoffmann/parsedown-laravel": "dev-master"

然后 *composer update* 安装，在 *config/app.php* 中增加：

```
'providers' => array(
    ...
    'MaxHoffmann\Parsedown\ParsedownServiceProvider'
),

'aliases' => array(
    ...
    'Markdown'          => 'MaxHoffmann\Parsedown\ParsedownFacade',
),
```

安装完后，**在 *ArticleController.php* 中增加：**

```
public function preview() {
```

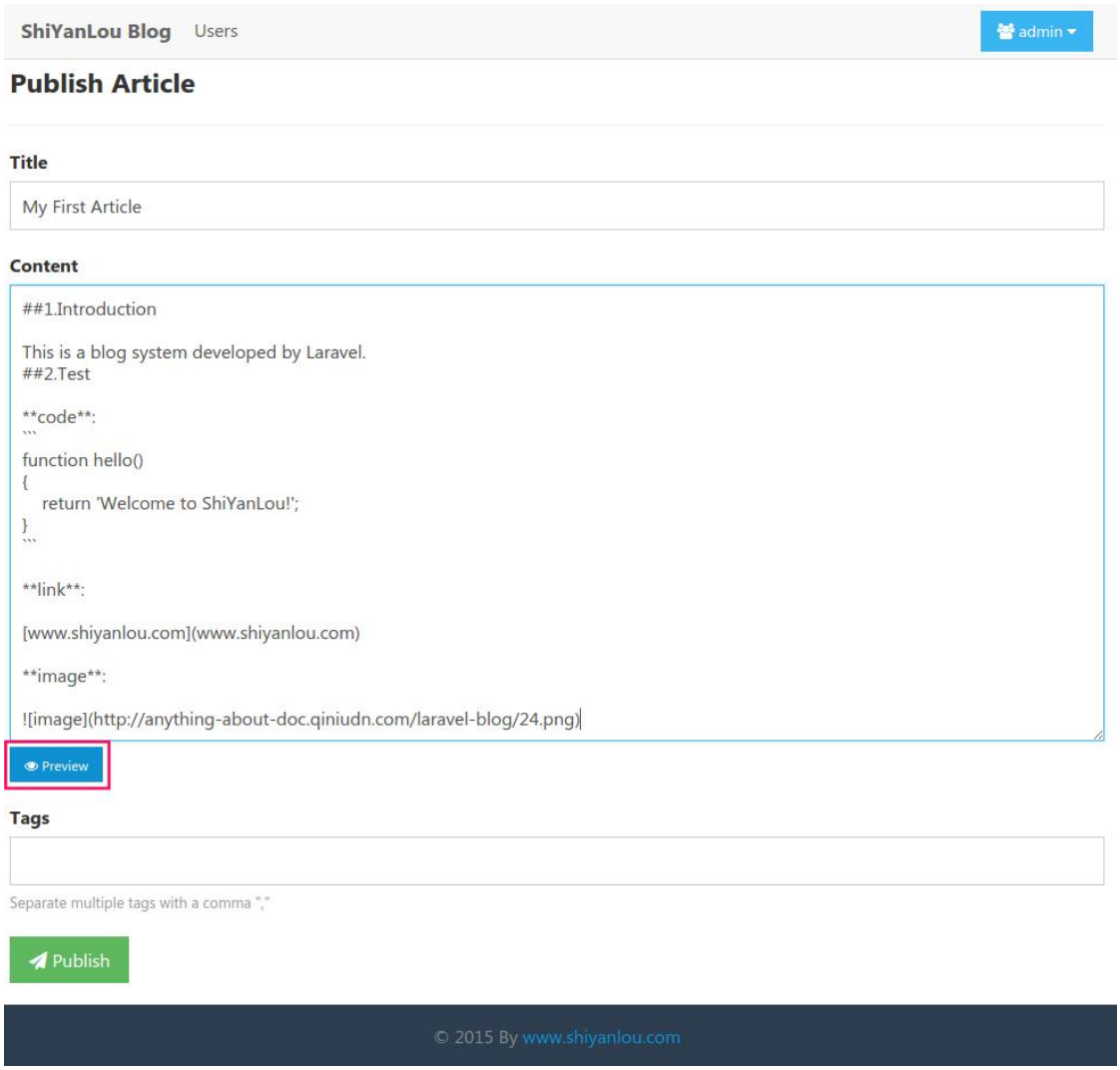
```
        return Markdown::parse(Input::get('content'));
}
```

在 *routes.php* 中增加：

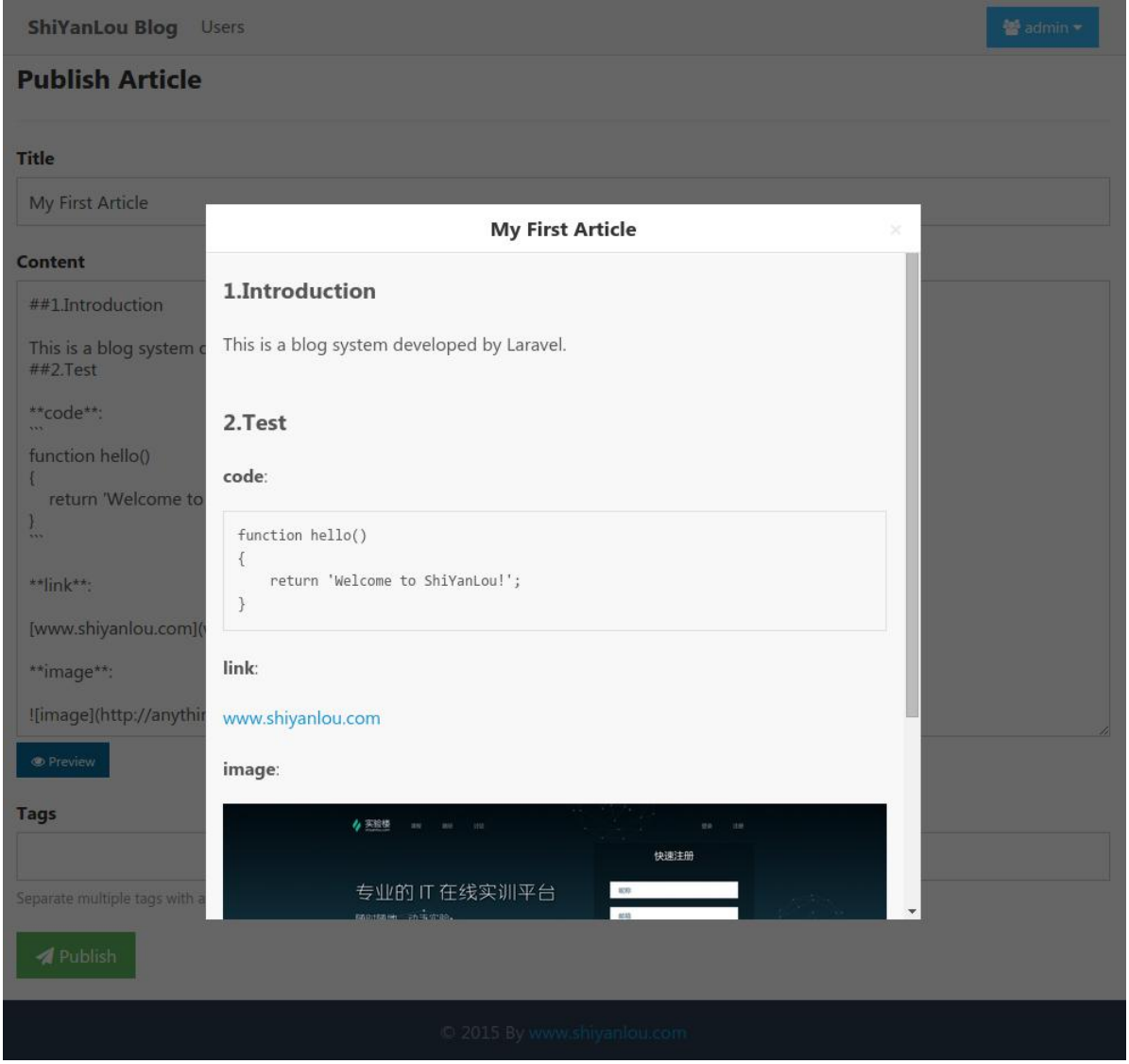
```
Route::post('article/preview', array('before' => 'auth', 'uses' =>
'ArticleController@preview'));
```

切记要添加在 *Route::resource('article', 'ArticleController');* 前面。

现在来试试我们的预览功能吧，点击 *Preview* 按钮：



出现预览界面：



5.添加文章

下面就要向数据库添加文章了，在 *ArticleController.php* 中修改：

```
public function store()
{
    $rules = [
        'title'    => 'required|max:100',
        'content' => 'required',
        'tags'     => array('required', 'regex:/^\w+$|^(\w+,)+\w+$/'),
    ];
    $validator = Validator::make(Input::all(), $rules);
    if ($validator->passes()) {
        $article = Article::create(Input::only('title', 'content'));
        $article->user_id = Auth::id();
        $resolved_content = Markdown::parse(Input::get('content'));
        $article->resolved_content = $resolved_content;
        $tags = explode(',', Input::get('tags'));
        if (str_contains($resolved_content, '<p>')) {
            $start = strpos($resolved_content, '<p>');
            $length = strpos($resolved_content, '</p>') - $start - 3;
            $article->summary = substr($resolved_content, $start + 3, $length);
        } else if (str_contains($resolved_content, '</h')) {
```

```
                $start = strpos($resolved_content, '<h');
                $length = strpos($resolved_content, '</h') - $start - 4;
                $article->summary = substr($resolved_content, $start + 4, $length);
            }
            $article->save();
            foreach ($tags as $tagName) {
                $tag = Tag::whereName($tagName)->first();
                if (!$tag) {
                    $tag = Tag::create(array('name' => $tagName));
                }
                $tag->count++;
                $article->tags()->save($tag);
            }
            return Redirect::route('article.show', $article->id);
        } else {
            return Redirect::route('article.create')->withInput()->withErrors($validator);
        }
    }

    public function show($id)
    {
        return View::make('articles.show')->with('article', Article::find($id));
    }
```

上面代码实现了保存文章和显示文章的业务逻辑，保存文章时验证 `tags` 用了 `regex` 正则表达式来验证标签是否用`,` 号分隔，有没有发现 `Article` 模型中有一个 `resolved_content` 字段，这个字段来保存解析后的内容，这样只需要在保存的时候解析一次，显示文章页面就显示 `resolved_content` 字段的内容，不需要再解析一次，这是空间换时间的做法，看个人喜好了，如果想要更好的体验，可以只在前台页面解析，保存的时候把前台解析的内容保存到 `resolved_content` 字段，前台解析 Markdown 有一个很好的工具 StackEdit。

现在就差个显示文章的视图了，创建：

```
$ php artisan generate:view articles.show
```

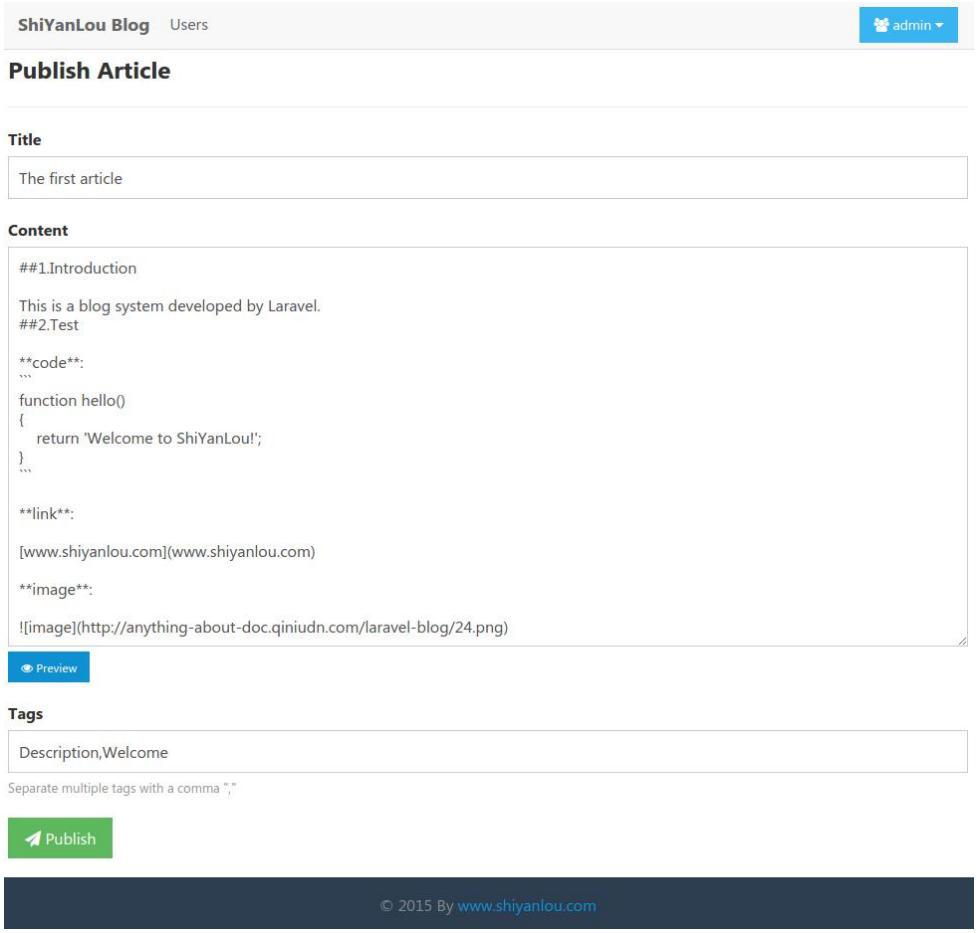
修改 `articles/show.blade.php`：

```
@extends('_layouts.default')

@section('main')
<article class="am-article">
  <div class="am-g am-g-fixed">
    <div class="am-u-sm-12">
      <br/>
      <div class="am-article-hd">
        <h1 class="am-article-title">{{{ $article->title }}}</h1>
        <p class="am-article-meta">Author: <a style="cursor:
pointer;">{{{ $article->user->nickname }}}</a> Datetime: {{ $article->updated_at }}</p>
```

```
        </div>
        <div class="am-article-bd">
          <blockquote>
            Tags:
            @foreach ($article->tags as $tag)
                <a class="am-badge am-badge-success am-radius">{{ $tag->name }}</a>
            @endforeach
          </blockquote>
        </p>
        <p>{{ $article->resolved_content }}</p>
      </div>
      <br/>
    </div>
  </div>
</article>
@stop
```

完成之后看看效果吧，先编辑文章：



发布后跳转到显示文章页面：

ShiYanLou Blog Users admin

The first article

Author: admin Datetime: 2015-01-23 10:24:17

Tags: Description Welcome

1.Introduction

This is a blog system developed by Laravel.

2.Test

code:

```
function hello()
{
    return 'Welcome to ShiYanLou!';
}
```

link:

www.shiyanlou.com

image:

6.小结

这节教程使用了控制器，完成了发布文章并展示的功能，但还是有很多瑕疵，在代码方面，现在你可以重构下 *User* 和验证登录的路由，把它们都变成控制器，在用户体验上你可以把发布文章编辑内容时由服务器端解析改成客户端解析，推荐 StackEdit，下一节教程将完成网站首页和用户主页展示文章列表和标签，并且让用户能够删除和修改文章。

本文详细出处：http://www.shiyanlou.com/courses/123