

校园失物招领平台开发

——基于laravel框架构建最小内容管理系统

摘要

针对目前大学校园人口密度大、人群活动频繁、师生学习生活等物品容易遗失的基本现状，在分析传统失物招领过程中的工作效率低下、找回率低、保密性差、管理分散等问题和不足的基础上，提出了WEB模式的失物招领信息管理平台。该平台主要通过失物信息发布和失物领取功能，较好的解决了传统失物招领管理过程中的信息孤岛的缺陷，提高了失物招领的工作效率，减少了师生的直接经济损失，方便了广大师生的日常生活。本系统采用了LAMP（ubuntu+Apache+MySQL+php）作为开发环境，后端php框架使用了目前流行的laravel框架，完全遵循MVC的设计模式。选用国产开源的响应式HTML5开发框架AmazeUI作为前端视图框架，可以很好地适应移动端页面布局。

关键词：校园；失物招领；信息发布；内容管理系统；php；laravel；MySQL；AmazeUI

目录

一、引言

- 1.课题研究背景及意义
- 2.系统可行性分析

二、系统需求分析

- 1.业务流程分析
- 2.用户体验分析

三、系统模块设计

- 1.功能模块
- 2.用户界面

四、核心功能的技术实现

- 1.开发环境

- 2.技术框架
- 3.数据库设计
- 4.MVC设计模式
- 5.核心代码实现
- 6.测试及操作说明

五、总结

六、参考文献

引言

1. 课题研究背景及意义

大学生由于携带的东西较多，活动多，而且经常来往于不同的地方，因此很容易发生物品丢失的情况，而目前我们学校又缺乏一套行之有效的可以帮助学生寻找失物的系统。考虑到大学生普遍上网，而且校园网络覆盖率高，那么就通过网络这一便捷高效的方式来实现一个失物招领系统，优化失物招领业务，使得失物招领管理清晰化、透明化，便于操作，易于管理。通过本系统，拾主可以通过这个平台发布最新的招领启事以通知大家，而失主可以通过这个平台寻找和联系拾主。并且只要是网站的注册用户，就可以发布管理文章、留言，这样不仅财产上的损失避免了，还加强了人与人之间的交流。所以说开发这个失物招领平台，定会大大便利校园内广大师生的生活，不必再为丢失寻找物品这类琐事烦扰，真正地让技术服务于生活。

2.系统可行性分析

- 技术可行性

利用稳定的ubuntu linux作为服务器环境，Apache处理客户端与服务端的通信，MySQL数据库存放用户信息和发布的文章、留言，php作为后端语言，实现动态页面处理，再加上简洁优雅的响应式前端开发框架，可以开发出符合要求的管理系统。

- 经济可行性

网站平台的开发及后期的运营维护所需的人力物力很少，并且可以部署在学校的服务器上，由网络服务中心的工作人员统一管理。

- 社会可行性

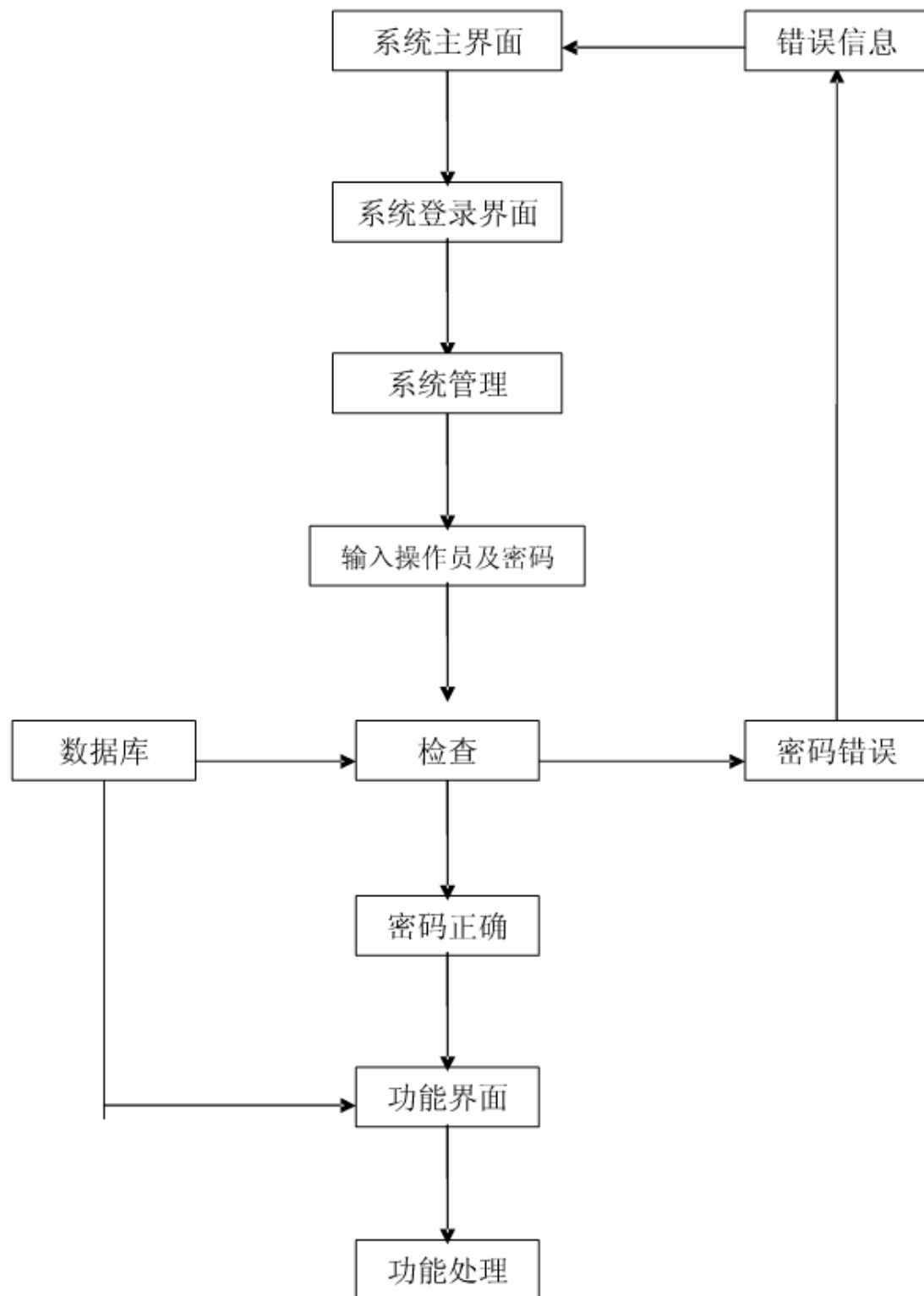
在校园里，我们经常见到这样一幕，一位同学在热水房门口一遍一遍寻找，然后口里说着，怎么又不见了呢？，一边不甘心的再找一遍。又或许是这么一幕，“寻物启事——宿舍号，丢失物品，最后再诅咒一下那些捡拾物品不还的人。”现有失物招领处工作繁琐且效率低，因此开发失物招领管理系统是非常必要的。

系统需求分析

1、业务流程

业务流程分析可以帮助开发者了解该业务处理过程，发现和处理系统调查工作中的错误和疏漏。业务流程分析是通过业务流程图来进行，即用一些规定的符号及连线来表示某个具体业务处理过程。

本系统具体的业务流程如下图所示:



2、用户体验分析

用户体验是现今产品开发过程中非常重视的一个方面，我们开发出一款产品或者提供一项服务，其最终面向的是不重视或不在乎技术层面的广大用户，因此在界面友好性、直观性、易于操作性方面必须加以考虑。本失物招领系统按照下述原则进行：

- 实用性：系统以用户需求为目标，以方便用户使用为原则，充分考虑实际操作的各项细节，使得普通用户在打开页面的第一眼即能对操作流程有清晰的理解。真正构建起一个为师生服务的平台，为用户的在线失物招领查询发布提供方便。
 - 高安全性：在设计中，将充分考虑网络软、硬件方面的各种安全措施，保障用户数据信息安全。比如用户存放在数据库的账号密码采用php的哈希加密，即使数据库泄露，看到的也将是一堆乱码。
 - 可维护性：网站的设计要求方便维护，文件目录及代码结构清晰。
 - 可扩展性：网站的设计以方便未来的扩展和系统扩充为目标，系统要求能够方便升级，方便添加功能模块。

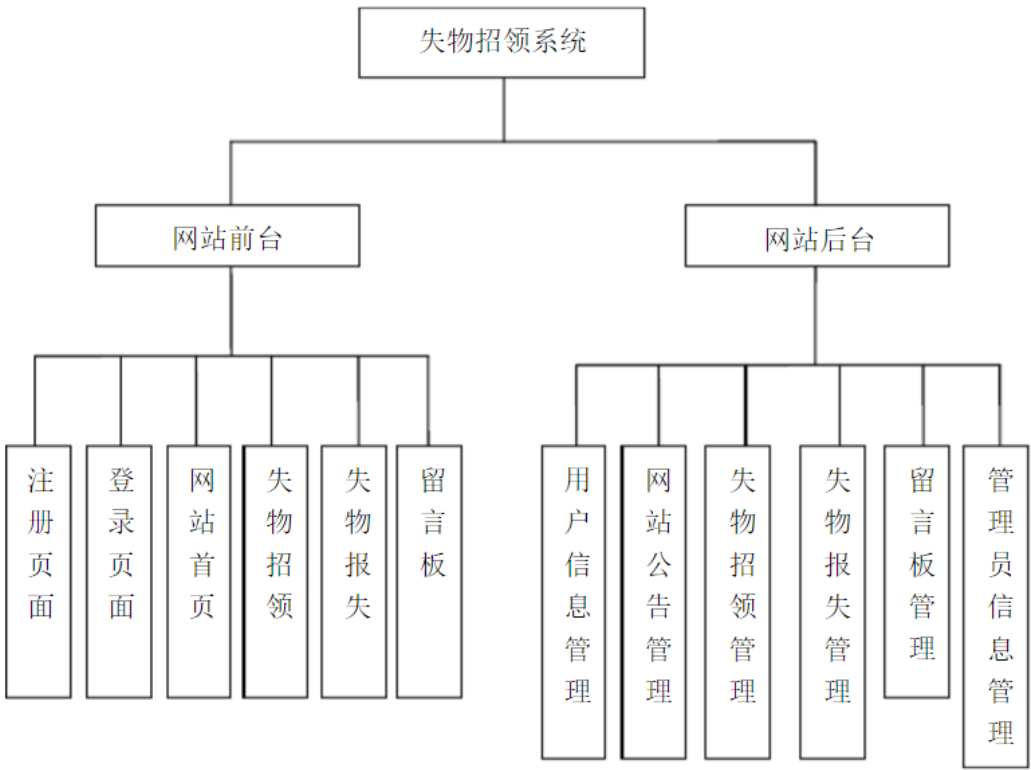
系统模块设计

1.功能模块

本管理系统主要能够实现用户注册、用户（管理员）登录、首页展示、失物招领的文章发布、留言板、后台管理，以及标签管理等几大模块的功能：

- 用户注册：本功能主要是实现对用户信息的注册管理
- 用户（管理员）登录：本功能主要实现用户（管理员）登录的功能
- 首页模块：本模块主要是对失物招领文章的展示功能、添加功能、查看详情功能及各模块入口
- 留言板模块：主要是实现用户（管理员）留言功能，以供用户反馈
- 后台管理：用户信息管理、失物招领文章管理、留言板留言管理、管理员信息管理、标签管理

本系统主要功能结构如下图所示：



2、用户界面

- 注册页面：本界面主要采集注册用户的信息，然后存入系统数据库
- 登录页面：本页面根据登录用户的信息和类型进行验证登录
- 首页：本页面展示用户所发布的失物招领的文章，以及文章的标签
- 留言板页面：本页面显示历史留言和添加留言的版块
- 用户信息页面：本页面显示当前登录的用户的信息及发布的文章管理
- 管理页面：本页面是管理员对本网站各个版块进行管理的可视化操作页面

核心功能的技术实现

1、开发环境

网站的本地开发环境使用lamp（即Linux+Apache+MySQL+PHP）。lamp是一组常用来搭建动态网站或者服务器的开源软件，本身都是各自独立的程序，但是因为常被放在一起使用，拥有了越来越高的兼容度，共同组成了一个强大的Web应用程序平台。lamp的所有开发工具都是开源软件，随着开源潮流的蓬勃发展，可以预见lamp会是未来web开发的主流，并且由于其零成本、学习资料多，自然成为我的首选开发环境。

2、技术框架

后端的php框架选用了国外流行的开源框架——laravel，也是号称“最简洁、优雅的php web开发框架”，基于此可以快速、高效地构建一个web APP，Laravel的目标是给开发者创造一个愉快的开发过程，并且不牺牲应用的功能性。刚开始学习php开发没多久，对php的一些框架了解甚少，在csdn上看到一篇文章《php开发框架流行度排名：laravel居首》，才知道有laravel这么一个框架，并且其文件目录、代码结构清晰，基于MVC的设计模式，对初学者较友好，故选用了此框架，本篇课程设计很大程度上也算是对laravel框架的学习实践吧。

php的应用框架众多，如国产的thinkPHP框架在国内也使用者甚广，前期曾尝试使用thinkPHP来开发，其结构代码简单直接，易于上手，但在代码规范性方面毁誉参半，不适合初学者养成良好的编码习惯，故弃之。

下面就对laravel框架体系结构作简要介绍。

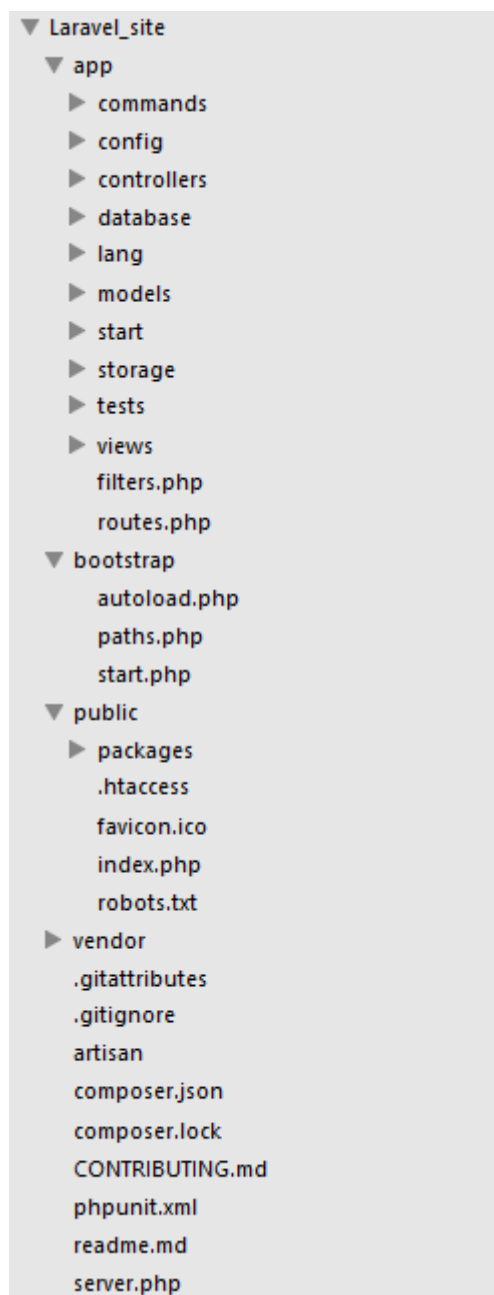
Laravel被称为“全栈”式框架，因为它能够处理从网络服务到数据库管理、HTML生成的一切事情，垂直集成的web开发环境给开发者提供了更好的体验。开发人员可以通过命令行工具,生成和管理Laravel项目环境。

Laravel带有一个名为Artisan的优秀的命令行工具，可以用它来生成框架代码和数据库架构，Artisan能够处理从数据库架构迁移到资源和配置管理的一切事情。

laravel项目使用composer来创建（Composer是PHP中用来管理依赖（dependency）关系的工具。你可以在自己的项目中声明所依赖的外部工具库（libraries），Composer会帮你安装这些依赖的库文件）。在linux终端中执行：

```
$ composer create-project laravel/laravel --prefer-dist web 5.1
```

就会在/home目录下创建一个名为web的项目文件夹，指定的laravel版本为5.1。其目录结构如下图所示：



下面是各个文件夹和文件的基本介绍：

顶级文件夹	作用
app	包含了站点的 controller （控制器）， models （模型）， views （视图）和 assets （资源）。这些事网站运行的主要代码，你将会花费大部分的时间在这些上面。
bootstrap	用来存放系统启动时需要的文件，这些文件会被如 index.php 这样的文件调用。
	这个文件夹是外界唯一可以看到的，是必须指向你 web 服务器的目录。它含有 laravel 框架核心的引导文件 index.php ，这个目录也可以用来存放任何可以公开的静态资源，如 css ， JavaScript ， images 等。
vendor	用来存放所有的第三方代码，在一个典型的 laravel 应用程序，这包括 laraceny 源代码及其相关，并含有额外的预包装功能的插件。

如上所述，/app是其核心部分，/app文件夹的详细信息如下：

	名称	修改日期
app		
commands	commands	2014-03-03 18:07
config	config	2014-03-03 18:07
packages	controllers	2014-03-03 18:07
testing	database	2014-03-03 18:07
controllers	lang	2014-03-03 18:07
database	models	2014-03-03 18:07
migrations	start	2014-03-03 18:07
seeds	storage	2014-03-03 18:07
lang	tests	2014-03-03 18:07
en	views	2014-03-03 18:07
models	filters.php	2014-03-03 18:07
start	routes.php	2014-03-03 18:07
storage		
cache		
logs		
meta		
sessions		
views		
tests		
views		
emails		
bootstrap		
public		
vendor		

下面是详细介绍：

文件及文件夹	作用
/app/config/	配置应用程序的运行时规则、数据库、 session 等等。包含大量的用来更改框架的各个方面的配置文件。大部分的配置文件中返回的选项关联PHP数组。
/app/config/app.php	各种应用程序级设置，即时区、区域设置（语言环境）、调试模式和独特的加密密钥。
/app/config/auth.php	控制在应用程序中如何进行身份验证，即身份验证驱动程序。
/app/config/cache.php	如果应用程序利用缓存来加快响应时间，要在此配置该功能。
/app/config/compile.php	在此处可以指定一些额外类，去包含由‘artisan optimize’命令声称的编译文件。这些应该是被包括在基本上每个请求到应用程序中的类。
/app/config/database.php	包含数据库的相关配置信息，即默认数据库引擎和连接信息。
/app/config/mail.php	为电子邮件发件引擎的配置文件，即 SMTP 服务器。
/app/config/session.php	控制Laravel怎样管理用户 sessions ,即 session driver , session lifetime 。
/app/config/view.php	模板系统的杂项配置。
/app/controllers	包含用于提供基本的逻辑、数据模型交互以及加载应用程序的视图文件的控制器类。
/app/database/migrations/	包含一些 PHP 类，允许 Laravel 更新当前数据库的架构并同时保持所有版本的数据库的同步。迁移文件是使用 Artisan 工具生成的。
/app/database/seeds/	包含允许 Artisan 工具用关系数据来填充数据库表的 PHP 文件。
/app/lang/	PHP 文件，其中包含使应用程序易于本地化的字符串的数组。默认情况下目录包含英语语言的分页和表单验证的语言行。
/app/models/	模型是代表应用程序的信息（数据）和操作数据的规则的一些类。在大多数情况下，数据库中的每个表将对应应用中的一个模型。应用程序业务逻辑的大部分将集中在模型中。
/app/start/	包含与 Artisan 工具以及全球和本地上下文相关的自定义设置。
/app/storage/	该目录存储Laravel各种服务的临时文件，如 session , cache , compiled view templates 。这个目录在 web 服务器上必须是可以写入的。该目录由Laravel维护，我们可以不关心。
/app/tests/	该文件夹给你提供了一个方便的位置，用来做单元测试。如果你使用 PHPUnit ，你可以使用 Artisan 工具一次执行所有的测试。
/app/views/	该文件夹包含了控制器或者路由使用的 HTML 模版。请注意，这个文件夹下你只能放置模版文件。其他的静态资源文件如 css , javascript 和 images 文件应该放在/ public 文件夹下。
/app/filters.php	此文件包含各种应用程序和路由筛选方法，用来改变您的应用程序的结果。 Laravel 具有访问控制和 XSS 保护的一些预定义筛选器。

文件及文件夹	作用
/app/routes.php	这是您的应用程序的路由文件，其中包含路由规则，告诉 Laravel 如何将传入的请求连接到路由处理的闭包函数、控制器和操作。该文件还包含几个事件声明，包括错误页的，可以用于定义视图的composers。

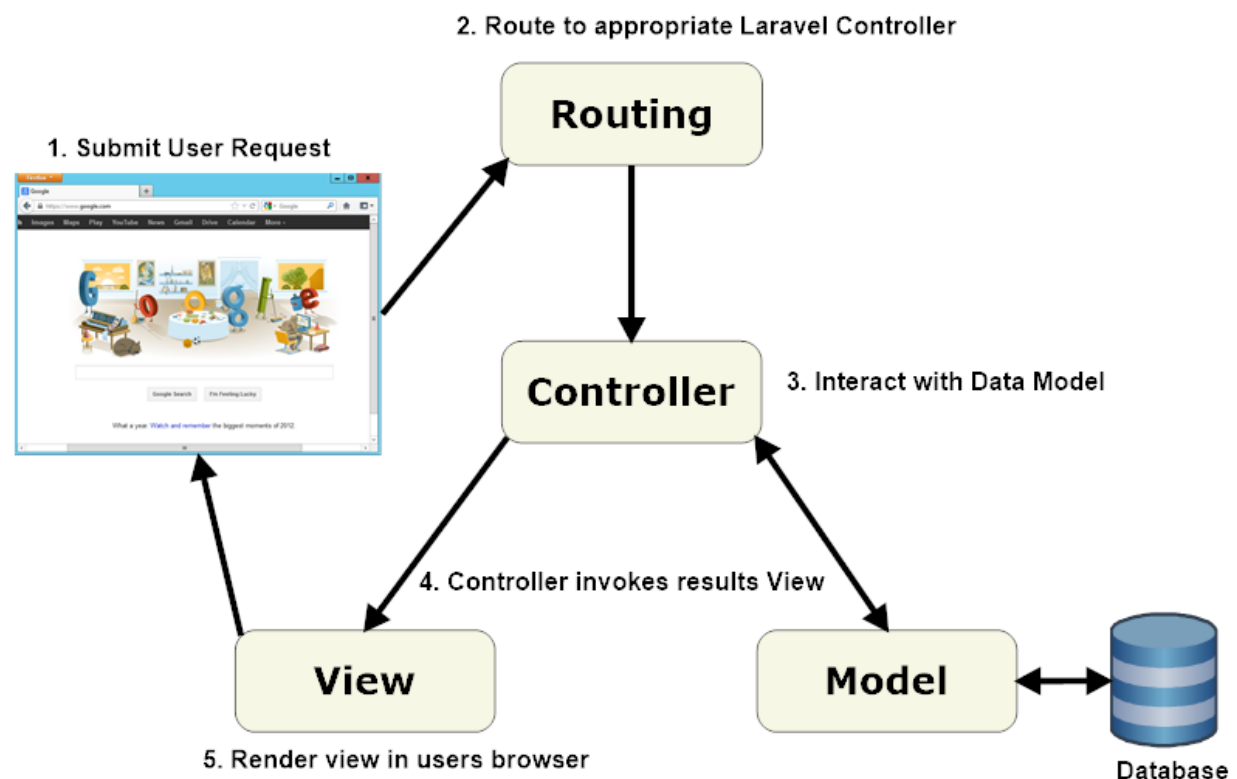
3、模型-视图-控制器（MVC）

/app文件夹下有三个子目录：**models/**、**views/**和**controllers/**。这说明**laravel**遵循MVC架构模式。这就是强制将输入到展示逻辑关系的“业务逻辑”与图形用户界面（GUI）分开。就**laravel web**应用而言，业务逻辑通常由像用户，文章这样的数据模型组成。GUI只是浏览器中的网页而已。MVC设计模式在web开发领域中很流行。

MVC模式包括三个组件：

- 模型（model）
- 视图（view）
- 控制器（controller）

一个典型的**laravel**应用程序包含上面提到的MVC组件，如下图：



laravel的响应流程

当与**Laravel**交互时，浏览器发送一个请求，web服务器接收到请求并且传给**Laravel**路由引擎。**Laravel**路由接收到请求，然后重定向给基于路由的URL模式的合适的控制器类方法。然后控制器类接管。在某种情况下，控制器会立即呈现出一个视图，它是一个被转换成HTML并送回浏览器的模版。更常见的动态网站，控制器与模型交互，这是一个PHP对象，它表示应用程序（如用户、博客文章）中的一个元素，并负责与数据库进行通信的。调用模型后，控制器则呈现最终视图（HTML，CSS和图像），并返回完

整的网页到用户的浏览器。**Laravel**促进了这样的概念——模型、视图和控制器，应通过存储这些元素在不同的目录中的单独的代码文件中来保持相当的独立性。这就是**Laravel**目录结构发挥了作用。

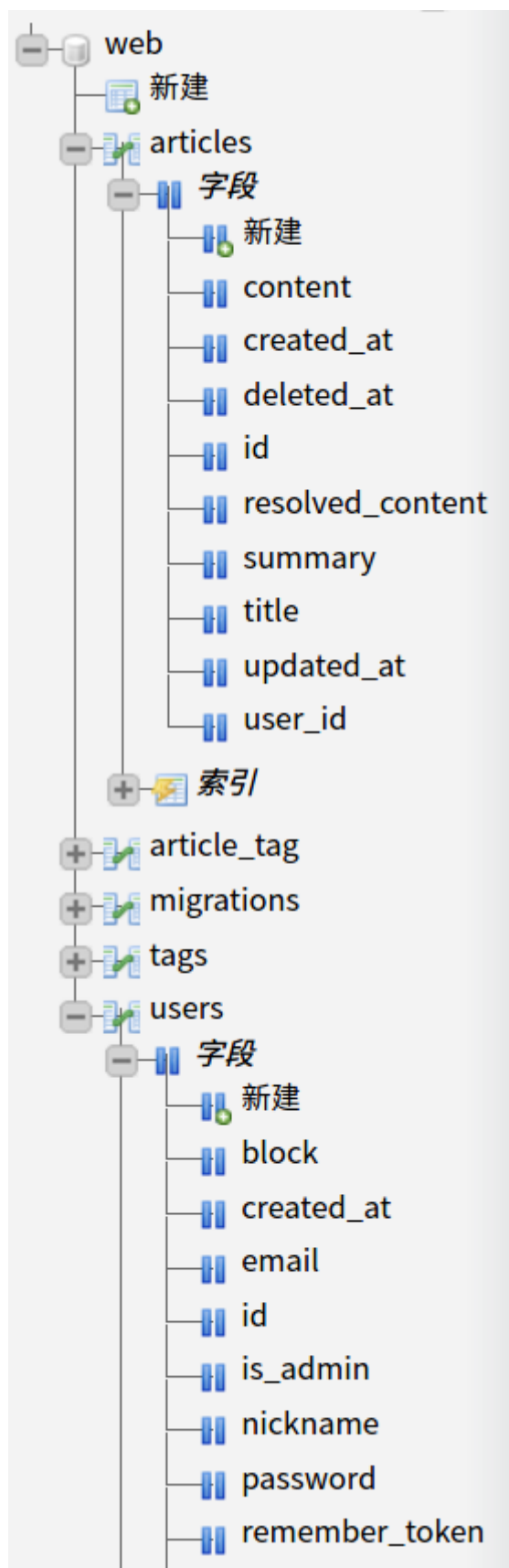
——laravel文档

4、数据库设计

使用phpmyadmin来对MySQL数据库进行可视化操作，在MySQL中先创建名为web的数据库，然后配置laravel的数据库配置文件/config/database.php如下：

```
'mysql' => [
    'driver'   => 'mysql',
    'host'     => env('DB_HOST', 'localhost'),
    'database' => env('DB_DATABASE', 'web'),
    'username' => env('DB_USERNAME', 'root'),
    'password' => env('DB_PASSWORD', 'sheng'),
    'charset'  => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix'   => '',
    'strict'   => false,
],
```

MySQL数据库中存在的表如下图所示：



部分表的说明：

articles：存放用户发布的失物招领文章

article_tag：发布的文章的标签

migrations: php做迁移数据时产生，与核心功能无关

tags: 用户标签

users: 存放用户账户信息












5、核心代码实现

web程序的代码一般都较多，如果把所有的代码均放到论文里，事无巨细，一一说明，是不现实的，故只会选择核心的业务逻辑部分代码，配以必要的解释。在前端方面，使用了AmazeUI响应式开发框架和jQuery的JavaScript库，与流行的bootstrap类似，故对前端的div，css等样式也不做过多说明。

着重解释的主要包括以下内容：

- 路由管理
- 用户管理，如用户注册、修改信息、锁定用户等
- 文章管理，如发表文章、修改文章等
- 标签管理，文章会有一到多个标签
- 数据库管理，如迁移、填充数据等
- Web表单验证
- Blade模版引擎
- 分页处理
- 安全处理

laravel使用blade模板引擎，故视图文件均须以xxx.blade.php方式命名，web/resources/views/文件夹下的目录结构如图所示：

	admin	3 项	文件夹	9月14日
	articles	4 项	文件夹	9月14日
	errors	3 项	文件夹	9月14日
	layouts	3 项	文件夹	9月14日
	tags	2 项	文件夹	9月14日
	users	2 项	文件夹	9月13日
	vendor	0 项	文件夹	2015年6月9日
	home.blade.php	3.1 KB	程序	9月14日
	index.blade.php	3.0 KB	程序	9月14日
	login.blade.php	1.5 KB	程序	9月13日
	welcome.blade.php	1.2 KB	程序	2015年6月9日

网站入口文件首先会加载的视图文件是layouts/defalut.blade.php:

```
<!DOCTYPE html>
```

```

<meta charset="UTF-8"/>
<title>校园失物招领平台</title>
<meta http-equiv="X-UA-Compatible" content="IE=edge"/>
<meta name="viewport"
      content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-
scale=1.0, user-scalable=no">
<meta name="format-detection" content="telephone=no"/>
<meta name="renderer" content="webkit"/>
<meta http-equiv="Cache-Control" content="no-siteapp"/>
<link rel="alternate icon" type="image/x-icon" href="{{ asset('img/favicon.ico') }}" />
<link rel="stylesheet" href="http://cdn.amazeui.org/amazeui/2.7.1/css/amazeui.min.css"/>
<link rel="stylesheet" href="{{asset('css/custom.css')}}"/>
<script src="//cdn.bootcss.com/jquery/3.1.0/jquery.min.js"></script>

```

```

<div class="am-container">
    <h1 class="am-topbar-brand">
        <a href="/">校园失物招领平台</a>
    </h1>
    @include('layouts.nav')
</div>

```

@yield('main')

@include('layouts.footer')

其中amazeui前端框架文件以及jQuery文件均是存放在cdn网络上，无需在本地加载，精简了网站文件夹的结构。

- asset('img/favicon.ico') 会生成 <http://localhost:8000/img/favicon.ico>
- asset('css/custom.css') 会生成，其中的img和css文件夹是放在public目录下的，public目录是项目的资源文件夹
- @include('layouts.nav') 会包含 app/views/layouts/nav.blade.php 文件
- @yield('main') 用于模版继承

大部分的视图文件都会继承default.blade.php的模板框架，例如index.blade.php:

@extends('layouts.default')

@section('main')

```

<div class="am-g am-g-fixed">
    <div class="am-u-md-8">
        <!-- 循环输出文章 -->
        @foreach ($articles as $article)
            <article class="blog-main">
                <h3 class="am-article-title blog-title">
                    <a href="{ { URL::route('article.show', $article->id) } }">{{
$article->title }}</a>
                </h3>
                <h4 class="am-article-meta blog-meta">
                    由 <a href="{ { URL::to('user/' . $article->user->id . '/articles')
}}">{{ $article->user->nickname }}</a> 发布于 {{ $article->created_at->format('Y/m/d
H:i') }} 标签:
                    <!-- 输出标签 -->
                    @foreach ($article->tags as $tag)
                        <a href="{ { URL::to('tag/' . $tag->id . '/articles') } }">{{
$tag->name }}</a>
                    @endforeach
                </h4>
                <div class="am-g">
                    <div class="am-u-sm-12">
                        @if ($article->summary)
                            <p>{!! $article->summary !!}</p>
                        @endif
                        <hr class="am-article-divider"/>
                    </div>
                </div>
            </article>
        @endforeach
    </div>

```

- @extends('layouts.default') 会继承 app/views/layouts/default.blade.php 文件
- @yield('main')对应 @section('main') 并填充为其中的内容

用户登录表单（在login.blade.php文件中）如下：

```

<form action="login" method="post" accept-charset="utf-8" class="am-form">
    <!-- 添加 token 值 -->
    <input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
    <label for="email">邮箱:
        <input type="email" name="email" value="{{Input::old('email')}}" placeholder="">
    </label>
    <br>
    <label for="password">密码:
        <input type="password" name="password" value="" placeholder="">
    </label>
    <br>
    <label for="remember_me">
        <input id="remember_me" name="remember_me" type="checkbox" value="1">
        记住我
    </label>
    <br>
    <div class="am-cf">
        <input type="submit" name="submit" value="登录" class="am-btn am-btn-primary am-btn-sm am-f1">
    </div>
</form>

```

laravel及大多数php框架使用路由（route）来生成URL，处理http请求，用户登录数据的验证也是放在了路由文件里（web/app/route.php）：

```

//post登陆数据
Route::post('login', function()

```



```

{
    //数据验证规则
    $rules = array(
        'email'      => 'required|email',
        'password'   => 'required|min:6',
        'remember_me' => 'boolean',
    );
    $validator = Validator::make(Request::all(), $rules);
    //验证通过
    if ($validator->passes())
    {
        if (Auth::attempt([
            'email'      => Request::input('email'),
            'password' => Request::input('password'),
            'block'      => 0],
            (boolean) Request::input('remember_me')))
        {
            return Redirect::to('home');
        }
        //账号或密码错误
        else {
            return Redirect::to('login')->withInput()->with('message', array('type' =>
            'danger', 'content' => 'E-mail or password error'));
        }
    }
    //数据格式错误
    else {
        return Redirect::to('login')->withInput()->withErrors($validator);
    }
});

```

//访问主页

```

Route::get('home', ['middleware' => 'auth', function()
{
    return view('home');
}]);

```

注册操作路由:

```

Route::post('register', function()
{
    $rules = [

```

```

        'email' => 'required|email|unique:users,email',
        'nickname' => 'required|min:4|unique:users,nickname',
        'password' => 'required|min:6|confirmed',

```

```

];
validator = Validator :: make(Request :: all(),rules);
if ($validator->passes())
{

    $user = new App\User();
    $user->email = Request::input('email');
    $user->nickname = Request::input('nickname');
    $user->password = Hash::make(Request::input('password'));
    if ($user->save())
    {
        return Redirect::to('login')->with('message', array('type' => 'success', 'content' =>
        'Register successfully, please login'));
    } else {
        return Redirect::to('register')->withInput()->with('message', array('type' =>
        'danger', 'content' => 'Register failed'));
    }

} else {

    return Redirect::to('register')->withInput()->withErrors($validator);

}
});

```

上面表单验证规则的**unique:users,email**能确保 **users** 表中的 **email** 字段是唯一的,例如当输入已存在的 **email** 时,会出现错误提示:

校园失物招领平台

The email has already been taken.

邮箱:

用户名:

密码:

确认密码:

注册

若注册成功就会跳转到登录页面，并给出成功的提示：

校园失物招领平台

注册成功，请登录

邮箱:

xzswolfly@outlook.com

密码:

.....

☐ 记住我

登录

用户发布失物招领启事，即文章发布模块，在数据库中对应该三张表：`articles`、`tags` 以及 `article_tag`，每篇文章会有一到多个标签，每个标签会有一到多篇文章。模型文件`/app/Article.php`和`Tag.php`、`User.php`的核心代码如下：

`User.php`:

```
<?php
namespace App;

use Illuminate\Auth\Authenticatable;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Auth\Passwords\CanResetPassword;
use Illuminate\Contracts\Auth\Authenticatable as AuthenticatableContract;
use Illuminate\Contracts\Auth\CanResetPassword as CanResetPasswordContract;

class User extends Model implements AuthenticatableContract, CanResetPasswordContract
{
    use Authenticatable, CanResetPassword;

    /**
     * The database table used by the model.
     *
     * @var string
     */
    protected $table = 'users';

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = ['name', 'email', 'password'];

    /**
     * The attributes excluded from the model's JSON form.
     *
     * @var array
     */
    protected $hidden = ['password', 'remember_token'];
    //模型关联
    public function articles()
    {
        return $this->hasMany('App\Article');
    }
}
```

一个用户会有多篇文章。

Article.php:

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes; //使用软删除 trait

class Article extends Model
{
    use SoftDeletes;

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

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

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

一篇文章会有多个标签并属于一个用户。

Tag.php:

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Tag extends Model
{
    use SoftDeletes;

    protected $fillable = ['name'];

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

一个标签会有多篇文章。

此文章编辑器使用了markdown编辑器，markdown简洁优雅的排版格式可以使文章样式更美观（可能也需要一定的学习成本）

向数据库中添加文章，要用到MVC中的control了。./web/app/http/Controllers/ArticleController.php的核心代码如下：

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
```

```

use App\Http\Requests;
use App\Http\Controllers\Controller;
use Markdown;
use Validator;
use App\Article;
use Auth;
use App\Tag;
use Redirect;

class ArticleController extends Controller

    *****
    *****

    //保存文章
    public function store(Request $request)
    {
        $rules = [
            'title' => 'required|max:100',
            'content' => 'required',
            'tags' => ['required', 'regex:/^\w+$|^\w+(\w+,)+\w+$/'],
        ];
        //数据校验
        $validator = Validator::make($request->all(), $rules);
        if ($validator->passes()) {
            $article = Article::create($request->only('title', 'content'));
            $article->user_id = Auth::id();
            $resolved_content = Markdown::parse($request->input('content'));
            $article->resolved_content = $resolved_content;
            $tags = explode(',', $request->input('tags'));
            //添加 summary
            if (str_contains($resolved_content, '<p>')) {
                $start = strpos($resolved_content, '<p>');
                $length = strpos($resolved_content, '</p>') - $start - 4;
                $article->summary = substr($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);
            }
            $article->save();
            //处理标签
            foreach ($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);
        }
    }

```

```

    }
}
//展示文章详情
public function show($id)
{
    return view('articles.show')->with('article', Article::find($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逗号分隔。

用户修改已经发布的文章的核心代码如下：

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

```
@section('main')
```

```

    <h1>Edit Article</h1>
    <hr/>
    @if ($errors->has())
    <div class="am-alert am-alert-danger" data-am-alert>
        <p>{{ $errors->first() }}</p>
    </div>
    @endif

```



```

<input type="hidden" name="_token" id="token" value="<?php echo csrf_token(); ?>">
<div class="am-form-group">
  <label for="title">Title:</label>
  <input type="text" name="title" id="title" value="{{ $article->title }}" placeholder="">
</div>
<div class="am-form-group">
  <label for="content">Content:</label>
  <textarea name="content" id="content" rows="20" >{{ $article->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:
    <input type="text" name="tags" value="{{ $article->tags }}" placeholder="">
  </label>
  <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-pencil"></span> Modify</button>
</p>

```

```

<div class="am-popup-hd">
  <h4 class="am-popup-title"></h4>
  <span data-am-modal-close
    class="am-close">×</span>
</div>
<div class="am-popup-bd">
</div>

```

@endsection

标签是为了给用户发布的文章分类，便于查找相关信息，其核心代码实现为：

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

```
@section('main')
```

```

<table class="am-table am-table-hover am-table-striped ">
<thead>
<tr>
    <th>TagName</th>
    <th>ArticleCount</th>
    <th>CreateDateTime</th>
    <th>Managment</th>
</tr>
</thead>
<tbody>
    @foreach ($tags as $tag)
    <tr>
        <td>{{{ $tag->name }}}</td>
        <td>{{ $tag->count }}</td>
        <td>{{ $tag->created_at->format('Y-m-d H:i') }}</td>
        <td>
            <a href="{{ URL::to('tag/'. $tag->id . '/edit') }}" class="am-btn am-btn-xs
am-btn-primary"><span class="am-icon-pencil"></span> Edit</a>
            <form action="{{ URL::to('tag/'.$tag->id.'/delete')}}" method="get" accept-
charset="utf-8" style="display: inline;">
                <button type="button" class="am-btn am-btn-xs am-btn-danger" id="delete{{
$tag->id }}">
                    <span class="am-icon-remove"></span> Delete
                </button>
            </form>
        </td>
    </tr>
    @endforeach
</tbody>
</table>

```

```

<div class="am-modal-bd">
</div>
<div class="am-modal-footer">
    <span class="am-modal-btn" data-am-modal-cancel>No</span>
    <span class="am-modal-btn" data-am-modal-confirm>Yes</span>
</div>

```

@endsection

错误处理:

如果用户访问的URL不存在或者服务器存在错误时, 我们不希望返回一个默认的错误页面, 而想返回一个友好提示的页面, 在 **Laravel** 中可以很轻松地实现, **Laravel** 有很简单的错误和日志处理, 当服务器端存在错误时, `app\Exceptions\Handler.php` 里默认有一个报告所有异常的程序:

```
/**
 * Report or log an exception.
 *
 * This is a great spot to send exceptions to Sentry, Bugsnag, etc.
 *
 * @param \Exception $e
 * @return void
 */

public function report(Exception $e)
{

    return parent::report($e);

}
```

当访问的URL不存在时, 服务器会抛出一个 **404** 错误, **laravel** 对 **HTTP** 异常有特别的处理方式:

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

@section('main')

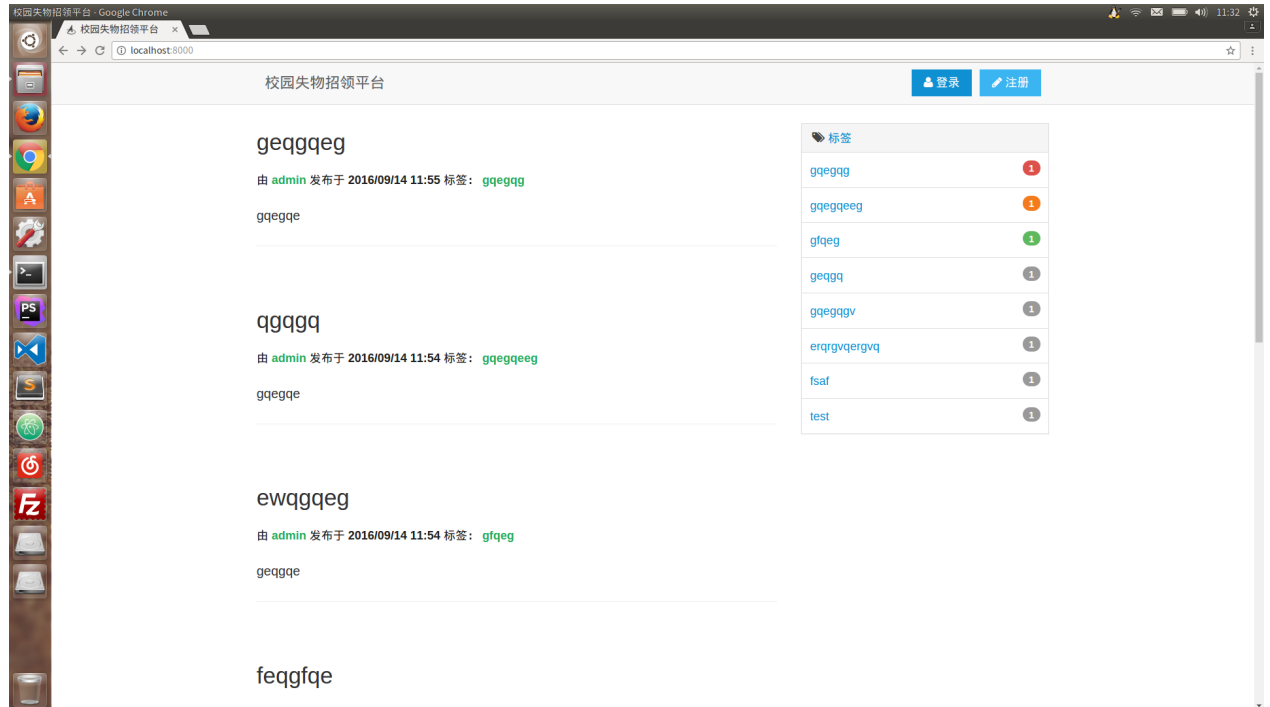
    <h1 style="text-align:center;">啊哦, 你访问的页面不存在!</h1>
    <h2 style="text-align:center;">返回 <a href="/">首页</a></h2>

@endsection
```

现在当你访问的 **URL** 不存在时, **laravel** 会自动到模板目录去寻找状态码为 **404** 的错误模板页面 `404.blade.php`

6、测试及操作说明

主页展示：



整体页面做的比较简洁，因为我侧重在功能实现，故对界面设计方面没有花太多时间。

用户登录：



这里的登录注册使用了laravel框架的regex正则表达式匹配。

登录之后，会出现文章管理界面：

校园失物招领平台

用户

文章

标签

我的文章

admin

标题	标签	管理
geqqqeg	gqegqg	<div>编辑</div> <div>删除</div>
qgqqq	gqegqeeg	<div>编辑</div> <div>删除</div>
ewqqqeg	gfqeg	<div>编辑</div> <div>删除</div>
feqgfqe	geqqq	<div>编辑</div> <div>删除</div>
eqwgfqegvqe	gqegqgv	<div>编辑</div> <div>删除</div>
fwevrewgwer	erqrgvqergvq	<div>编辑</div> <div>删除</div>
scvsvsevvw	fsaf	<div>编辑</div> <div>删除</div>
校园卡	test	<div>编辑</div> <div>删除</div>

© 2016 安徽工程大学 校园失物招领平台

这里随便添加了一些测试数据。

删除文章：

校园失物招领平台

用户

文章

标签

我的文章

标题	标签	管理
geqqqeg	gqegqg	<div>编辑</div> <div>删除</div>
qgqqq	gqegqeeg	<div>编辑</div> <div>删除</div>
ewqqqeg	gfqeg	<div>编辑</div> <div>删除</div>
feqgfqe	geqqq	<div>编辑</div> <div>删除</div>
eqwgfqegvqe	gqegqgv	<div>编辑</div> <div>删除</div>
fwevrewgwer	erqrgvqergvq	<div>编辑</div> <div>删除</div>
scvsvsevvw	你确定要删除吗？	
校园卡	否	是

© 2016 安徽工程大学 校园失物招领平台

点击删除文章会调用一段js代码，实现模态弹出框。

发布文章：

发布信息

标题

校园卡

内容

```
# 丢失一张校园卡
###本人x月x日在三食堂丢失一张校园卡
姓名: xxx
班级: xxx
学号: xxx
望拾到者与我联系
电话: xxxxxxxxx|
```

预览

标签

多个标签请用逗号隔开 "

这个编辑器使用了markdown语法来编辑文字，可能对于普通用户来说有比较高的门槛，本想用富文本编辑器的，但是由于时间较为紧张，而markdown插件易于使用，故在beta版本中以此来暂时代替。

点击预览：



这里也调用了js解析markdown，生成html。

发布之后：



这样一个简单的失物招领启事发布管理系统就完成了，测试各功能正常。

总结

通过开发这个校园失物招领平台，其实就完成了一个最小内容管理系统，一个完整的内容管理系统包括这几个核心的模块：

- 用户管理

- 文章管理
- 权限管理
- 标签管理

90%的网站功能开发都可以归纳为CRUD（即增删改查）操作，可能对于一个这样功能简单的管理系统来说，没有必要去使用重型的laravel框架，看起来有点过度设计了，但是我作为一个php初学者来说，也是想通过这个机会来学习一下这个优秀的框架，学习框架的过程也加深了对php语言的理解，对于以后开发更大型的网站能够积累一点相关经验。

这个系统目前还存在很多问题，比如界面不太符合失物招领的常规设计，有很多当初的设想也没能实现，markdown的编辑器不可能用在面向普通用户的网页中，入口首页和留言板功能由于时间关系没能加上，颇为遗憾。我希望以后能有时间去逐步地完善它，改造为我的个人博客，或者以此为基础，实现我一直以来的一个想法——搭建一个安工程学生门户，当然这个工程量就非常大了。

回顾一下这整个的学习开发过程，深感不易。从最基本的前端html+css+js学起，到php的基本语法，再到laravel框架的学习，期间查阅了大量的资料，观看了100+小时的在线mooc视频，才完成了这份课程设计。感谢那些技术博客博主的无私分享，前人的经验与见解避免后人少走了多少弯路，也由此深感开源分享精神之重要，正是开源运动才使得现今的互联网行业获得如此蓬勃的发展。所以我想在以后的学习过程中，也应当时时勿忘总结个人的经验，并且要分享出来，让自己的弯路成为别人的桥梁。

也感谢我的小伙伴们的鼓励和付出，正是一个团队的合作才使得这份作品能够如期完成。

参考文献

1. 《php与MySQL web程序设计》
2. 《html+css+js 网页设计》
3. 众多博客及慕课网、网易云课堂相关教程