

Laravel-Angular中文文档

起步

[Github地址](#)

安装

为Laravel Angular Material 的初学者提供的步骤

=====

我应该使用哪个版本呢？3.4还是3.2？

如果你想要构建一个WebApp，那么推荐你使用3.4.x版本。

否则的话，我们推荐你坚持使用3.2.x版本

如果你是初学者，我推荐你看后面的友好安装方式

命令行安装

```
composer create-project jadjoubran/laravel5-angular-material-starter --prefer-dist
cd laravel5-angular-material-starter
npm install -g gulp bower
npm install
bower install
#记得修改根目录下.env中数据库的配置
php artisan migrate
gulp
```

监听静态资源文件的改动

```
gulp && gulp watch
```

预览站点

```
php artisan serve
```

 启动本地服务器然后用浏览器打开默认站点localhost:8000

初学者安装

依赖

1. node版本 ≥ 4 ，npm版本 ≥ 3 ，可以通过 `node -v` `npm -v` 查看
2. gulp(全局安装)，可以通过 `gulp -v` 来检查，如果没有安装，执

行 `[sudo] npm install -g gulp`

3. bower(全局安装), 可以通过 `bower -v` 来检查, 如果没有安装, 执

行 `[sudo] npm install -g bower`

安装

使用composer create-project命令创建默认项目

```
composer create-project jadjoubran/laravel5-angular-material-starter --prefer -dist
```

然后使用 `cd` 命令进入此目录

```
cd laravel5-angular-material-starter
```

然后安装局部的npm包和bower包依赖

根据你本地的数据库配置修改安装目录下的.env配置文件

```
php artisan migrate
```

编译静态文件并监听

```
gulp && gulp watch
```

现在不要关闭这个窗口, 打开一个新的窗口来进入当前目录启动本地服务器

```
php artisan serve
```

默认的启动端口是本地的8000端口即localhost:8000

你是在Windows下使用Vagrant的用户?

如果你计划这样安装, 这里已经有人提出过issue, 主要是使用npm, 也许可以帮助你[issue](#):)

概述

这个初始项目包的目的是提供一个利用Laravel5.3和Angular1.5来进行WebAPP最佳实践的起始点

为绝大多数用到的插件和包提供了合理配置

这个项目的目标并不包括包括尽可能多的包。打比方说, 项目里面没有任何默认安装的 `icon font` 库。这是因为尽管Google的**Material Design Icons**感觉是最适合的, 但是仍然有用户偏爱**Font Awesome**

为什么使用Angular Material

Angular Material与Angular的协作很好。

初始的包里没有jQuery, 如果你需要, 自行添加

引入的类库

这个库是一个启动包, 它建立在许多流行的开源框架之上。这些包已经被提前配置好以便他们之间不会发生冲突, 但是你还是需要熟悉这些类库以便你可以使用它们更好地工作。

引入的类库如下:

1. [Angular1.5](#)
2. [ui-router](#)
3. [Angular Material](#)
4. [Restangular](#)
5. [JWT Auth](#)
6. [Sateizer](#)
7. [Laravel cors](#)
8. [ngDescribe](#) for testing

3.4中有什么新特性

下面是3.4可用的新特性：

- * 移除了Laravel生成的dingo/api目录，现在提供了提供了一个分离的API路由文件
- * 工程中使用的Laravel已经更新至5.3

如果想看完整的新特性列表，请移步[Changelog](#)

如果你想要更新3.3的app，请按照[Upgrade guide](#)中的步骤进行

贡献

Laravel Angular Material Starter就是基于像你一样令人惊叹的开源贡献者。

你可以通过这些方式来推动此项目的发展，在Github提问，提issue，提议新特性，pull request和任何可能帮助到社区的方式。

文件结构

Laravel的文件结构自然是不变的。下面，你可以了解到更多关于Angular在Laravel Application中的信息

Angular

Angular被默认安装在安装目录下的/angular/目录下，在这个前提下，你可以在/angular/目录下自定义你自己的Angular文件结构。在/angular/目录下默认目录如下

- app
 - pages
 - components
- services
- directives
- filters
- config

- run
- dialogs: 来自定制对话框
- material: 通用的样式文件文件夹

Angular生成器

你可以用Angular生成器来生成一些指令，特性，从命令行的命令来生成它们可以很大程度上减轻你的工作量

应用入口

默认生成的默认页面的模板文件位于 `resources/views/index.blade.php`

在这里，你可以修改Livereload配置的自动刷新（本地环境）以及Angular app自动加载 (ng-app)

Elixir任务

`task` 目录中包含着使用elixir自定制的gulp任务，以下内容也可以这样来配置

- [Angular](#)
- [Bower](#)
- [ngHtml2.js](#)

你可以在这个文件夹下自定义你的任务甚至修改默认的任务

在线演示地址

升级指南

从3.3升级到3.4

首先，你要确认你希望构建一个[流行的Web App](#)

安装3.4版本，将你旧的app中的特有的文件迁移到3.4。在你升级之前务必要将下面的升级须知阅读完。

也可以选择只将Laravel从5.3升级到5.4

移除DINGO/API

Dingo/API已经被从初始项目中移除了，如果你还是需要，自行按照它官网的[文档](#)安装

3.4版本变更说明 推荐你仔细阅读[3.4版本变更的说明](#)，这样你可以快速了解到3.4中包含的新特性

入门

注意

在项目初始化的工程中**angular-animate**的版本是与Angular版本不兼容的，你可以参考github上的[issue](#)来进行修复

概述

这部分的内容是为了向你介绍Laravel Angular Material初始项目的一些不同的概念。

如果想了解更多的信息，那么你最好把所有的文档内容都阅读一遍

通过这个入门教程，我们将构建一个可以让我们post数据到数据库的页面

这个入门指南跳过了数据库的migration过程，如果你在Laravel的使用上还不是很熟练，推荐你去阅读以下它的[官方文档](#)

TDD

测试驱动开发

通过下面这个例子，我们定义了一个测试方法来测试我们的新特性中的一些行为

CreatePostTest.php

```
<?php

class CreatePostTest extends TestCase
{

    public function testStoresPostSuccessfully()
    {

        $post = factory(App\Post::class)->make();

        $this->post('/api/posts', [
            'name' => $post->name,
            'topic' => $post->topic,
        ]->seeApiSuccess()
        ->seeJsonObject('post')
        ->seeJson([
            'name' => $post->name,
            'topic' => $post->topic,
        ]));

        $this->seeInDatabase('posts', [
            'name' => $post->name,
            'topic' => $post->topic,
        ]);
    }

}
```

API 路由

我们可以通过下面的命令来生成我们的控制器

```
php artisan make:controller CreatePostController
```

并在Laravel中添加相关的路由

```
<?php

Route::post('posts', 'CreatePostController@create');
```

Laravel 控制器

现在我们可以控制器中编写需要被测试的create方法

CreatePostController.php

```
<?php

namespace App\Http\Controllers;

use App\Post;
use App\Http\Requests;
use Illuminate\Http\Request;

class CreatePostController extends Controller
{
    public function create(Request $request)
    {
        $this->validate($request, [
            'name' => 'required',
            'topic' => 'required',
        ]);

        $post = new Post;
        $post->name = $request->input('name');
        $post->topic = $request->input('topic');
        $post->save();

        return response()->success(compact('post'));
    }
}
```

这篇入门教程默认你会自己通过Laravel创建模型和这里需要的posts表，如果你对此不了解，可以通过Laravel官方教程来学习

Angular 组件

现在我们需要创建用户用来填充提交post请求的表单组件

```
php artisan ng:component create_post_form
```

组件思想鼓励代码重用

为每个功能创建一个组件的目的是我们接下来可以很方便地重用它。

使用编辑器打

开 `angular/app/components/create_post_form/create_post_form.component.html` 并在里面添加相关的HTML表单代码

create_post_form.component.html

```

<form ng-submit="vm.submit()">

  <md-input-container>
    <label>Name</label>
    <input type="text" ng-model="vm.name">
  </md-input-container>

  <md-input-container>
    <label>Topic</label>
    <input type="text" ng-model="vm.topic">
  </md-input-container>

  <md-button type="submit" class="md-primary md-raised">Create post</md-button>

</form>

```

然后修改 `create_post_form.component.js` 中的代码来增加控制器逻辑

`create_post_form.component.js`

```

class CreatePostFormController{
  constructor(API, ToastService){
    'ngInject';

    this.API = API;
    this.ToastService = ToastService;
  }

  submit(){
    var data = {
      name: this.name,
      topic: this.topic,
    };

    this.API.all('posts').post(data).then((response) => {
      this.ToastService.show('Post added successfully');
    });
  }
}

export const CreatePostFormComponent = {
  templateUrl: './views/app/components/create_post_form/create_post_form.component.html',
  controller: CreatePostFormController,
  controllerAs: 'vm',
  bindings: {}
}

```


编辑 `angular/index.components.js` 来引入这个组件（在artisan时已经自动生成）

JavaScript

```
import {CreatePostFormComponent} from './app/components/create_post_form/create_post_form.component';

angular.module('app.components')
  .component('createPostForm', CreatePostFormComponent);
```

EcmaScript 6 这里我们使用了ES 6的语法，如果你还在使用ES5，这里有一个很好的学习[EcmaScript 6](#)的资源

Angular 页面

现在我们已经有了需要的页面组件，我们可以使用它来创建页面。页面的主要功能是将组件像搭积木一样整合在一起，每一个组件就像一个积木块，你的页面可以包括一个或者多个组件。

```
php artisan ng:page create_post
```

在 `angular/app/pages/create_post.page.html` 中我们完成对 `createPostForm` 组件的调用

create_post.html

```
<h1>Create Post</h1>

<create-post-form></create-post-form>
```

这看起来有些重复，，但当我们重用它时可以从获益。我们可以添加其他相关的页面标题，甚至在一些复杂的例子中我们可以调用其他的指令。

前端路由

我们仍然需要前端路由来把之前的内容连接起来

让我们打开 `angular/config/routes.config.js` 文件来为state provider增加新的入口，States是通过[ui-router](#)组件来管理的,它是一个很强大的Angular路由库

JavaScript

```
$stateProvider.state('app.create_post', {
  url: '/create-post',
  views: {
    'main@': {
      templateUrl: getView('create_post')
    }
  }
})
```

现在你可以通过URL查看这个页面，或者使用 `$state.go('app.create_post')`，也可以直接在浏览器中手动输入localhost:8000/#/create-post

Angular 生成器

简介

Angular生成器

在初始化的项目中包含一系列的Angular生成器来方便你生成前端代码文件模板。
这个功能是通过使用 [LaravelAngular/generators recipe](#) 提供的功能实现的。

默认的目录/文件扩展名 你可以通过修改 `config/generators.php` 来自定制自动生成的文件名以及后缀名

生成器

使用 `-` 作为分隔符

不要使用word作为服务(Service)、组件(Component)或者指令(Directive)的名称，他们会为你自动生成

自动导入(Auto Import) 所有的生成器都会在对应的文件自动更新引入的文件和模块。对于组件，`index.component.js` 会自动更新。你也可以在 `config/generators.php` 中设置取消这一行为，也可以在命令后增加 `--no-import` 标示

下面展示可用的生成器列表

artisan ng:page {settings}

通过生成以下文件来在 `/angular/app/pages` 中创建一个新的页面

- angular/app/pages/settings.page.html
- angular/app/pages/settings.less

artisan ng:component {user-profile}

通过生成以下文件来在 `/angular/components` 中创建一个新的组件

- `angular/directives/components/user-profile.component.html`
- `angular/directives/components/user-profile.component.js`
- `angular/directives/components/user-profile.less`
还包括一个 `ngDescribe test` 文件
- `tests/angular/app/components/user-profile.spec.js`
 - > 组件 vs 指令
 - > 建议阅读Angular官方文档中对components（组件）和directives（指令）不同点的说明
 - > 在大多数场景中，你应该使用component来构建页面中的某一块

artisan ng:directive {is-admin}

通过生成以下文件来在 `/angular/directives` 中创建一个新的指令

- `angular/directives/is-admin/is-admin.directive.js`
还包括一个 `ngDescribe test` 文件
- `tests/angular/app/directive/is-admin.spec.js`

artisan ng:dialog {login}

通过生成以下文件来在 `/angular/dialogs` 中创建一个新的模态框

- `angular/dialog/login.html`
- `angular/dialog/login.dialog.js`
- `angular/dialog/login.less`

artisan ng:service {cache}

通过生成以下文件来在 `/angular/service` 中创建一个新的服务

- `angular/service/cache.service.js` 还包括一个 `ngDescribe test` 文件
- `tests/angular/services/cache.spec.js`

artisan ng:filter {ucfirst}

通过生成以下文件来在 `/angular/filters` 中创建一个新的服务

- `angular/filters/ucfirst.filter.js`

artisan ng:config {http}

通过生成以下文件来在 `/angular/config` 中创建一个新的配置

- `angular/config/http.config.js`

Progressive WEB APP

概述

受益于最新的技术，Progressive WEB APP可以带给用户最好的移动站点和native app。并且是可靠、迅速的，因此吸引了许多开发者。——[Google Web Fundamentals](#)

DEMO

你可以从浏览器（Chrome、Firefox、或Opera）打开flipkart.com并将它添加到桌面。progressive web app利用了浏览器最新的功能（因此叫Progressive，先进），将它添加到桌面之后，它可以从桌面启动，离线使用，甚至可以推送消息。

Web App Manifest

web app manifest将以一个文档的形式展现给用户关于它的信息（像名字、作者、图标、简介等等）它的初级目标是创建一个先进的web app：web app无需用户通过App store就可以被安装至系统桌面（此外还可以离线使用，app内容更改、升级时还可以推送消息）——[Mozilla Developer Network](#)

php artisan pwa:manifest

这个生成器将帮助你创建你自己的 `manifest.json`

你可以跳过它的问题来自己直接修改内容

App Shell

App Shell 是用来搭建用户界面和确保app高性能以及可靠性的组件之一的最小化的HTML、CSS和JavaScript，它在第一次使用的时候就会迅速加载，并且会被缓存起来。这意味着它不需要每次都被重新加载一次，app只需要加载必要的内容即可。[Google Web Fundamentals](#)

App Shell in index.blade.php

app shell 已经在 `resource/views/index.blade.php` 中为你配置完毕

你可以在index.blade.php中看看app shell以是怎样的形式在页面中出现的。

critical.scss

Critical CSS将为你的App Shell服务，这是app shell用来展示所需的最基本的CSS文件。它是通过sass预处理的，所以你可以使用变量。它们被排除在 `final.css` 之外所以不会重复（这里你只需要关心App Shell本身即可）

sw precache

一个用来生成可以缓存指定资源的服务对应代码的nodejs模块——[GoogleChrome/sw-precache](#)

sw precache & gulp

sw-precache已经在你的gulpfile.js中配置完毕，它会从 `precache-config.json` 读取配置，当你改动其中的配置后无需重新执行 `gulp watch` 命令

默认情况下，我们会缓存app shell的文件结构，字体样式，以及你最新的资源文件（CSS & JavaScript）

运行缓存

运行缓存允许你自己进行配置[sw-toolbox]

sw toolbox

一个用来处理运行时请求的服务工具集合——[GoogleChrome/sw-toolbox](#)

precache-config.json

你可以在 `precache-config.json` 中的runruntimeCaching对象中自定制你的缓存策略。默认情况下我们为Github的button和网页字体指定了 `cacheFirst` 的策略。

REST API

概述

在现代的web app中REST API是一个被频繁使用的功能。

这也是为什么我们专注于使这个过程保持一致。

这可以使你在你的API中更加方便地开发新的接口。

打比方来说，在RestAngular中验证错误会自动展示。

Laravel Angular Material Starter可以帮助你更好地规范API返回的数据格式

错误响应对应着一个具体的格式，成功的响应对应另一个。

前端也会受益于这种一致性，通过配置响应拦截器，在接受到错误的返回后它可以自动弹出一个验证错误的弹出层

所有这一切都提供了可选的[Json Web Token Authentication](#)支持和有用的[API test helpers](#)，这将使集成测试更容易。

响应宏（Response Macros）

[Response Macros](#)本来是Laravel框架中的一个特性

在上面我们提到了REST APIs需要保持一致性，所以我们提供了两个默认的响应宏来从你的接口返回成功、失败时的数据

PHP

```
<?php

class PostsController
{
    public function get()
    {
        $posts = App\Post::all();

        return response()->success('posts', $posts);
    }
}
```

PHP

```
<?php

class SettingsController
{
    public function update()
    {
        if ( !\Auth::user()->is_verified ){
            return response()->error('Not Authorized', 401);
        }
    }
}
```

注意即使验证错误通过了 `$this->validate($request, [])` 之后返回了与 `response()->error()` 相同的错误格式，但是会携带422的错误状态码（不可处理实体）

这意味着你期望所有的正确数据返回同一种格式的数据，错误数据也是一样（包括验证）得益于这种一致性，你可以在RestAngular中配置选择在Toast中展示验证错误信息。当然，在这样的前提下，API test helper也可以很容易地断言被用来返回的正确响应。例子：`->seeApiSuccess()`，`->seeValidationError()`，`->seeApiError(401)`。

验证响应

如果你计划修改`error`的响应宏，你需要在基础控制器文件 `App\Http\Controllers\controller.php` 中进行应用和修复。

RestAngular

RestAngular 是一个用来简化常用的GET、POST、DELETE、UPDATE请求的AngularJS服务，这样你可以使用最少的客户端代码来实现它，对于通过REST API来处理数据的web app来说，这是最适合的。

这使得它与这个库可以完美地配合起来。

RestAngular 是 `$http` 上的一层

默认配置

这个库提供给你一个叫做API的服务(文件路径：`angular/services/api.service.js`)，它配置了如下内容：

- Content negotiation (header)
- Content type (header)
- Base URL: 这意味着你在调用API端口时不需要写 `/api/` 作为前缀
- error interceptor（异常拦截器），它会将第一条错误信息展示在红色的弹出层中
- JWT 支持

所有这些配置都与我们期望的API（也内联到dingo / api的配置）是一致的。

你可以浏览他们提供的可选的其他选项——[their repository](#)

避免使用Restangular service,通过API来调用

推荐你避免使用Restangular服务

使用继承了Restangular的服务（如API）可以让你为其他API提供其他的配置，可以更加灵活。

虽然RestAngular对于初学者来说并不是那么容易掌握，但是这是非常值得的，一旦你掌握了它，你可以用非常流畅的语法来调用你的接口。

JWT Authenticated routes

在 `config/api.php` 中，JWT被配置作为 `dingo/api` 的一个验证提供者。

它允许你使用 `api.auth` 中间件为需要验证的路由提供验证服务

如果用户没有登录，或者验证头丢失，他们将无法访问接口。
如果你想增加更复杂的内容，你可以创建自己的中间件。

JWT AUTH

基本配置

当你使用 `composer create-project` 来安装时，下面的命令也将执行

```
php artisan jwt:generate
```

它将生成JWT运行所需的 `JWT_SECRET`

JSON Web Token是无状态的，这也是为什么我们从不把它储存在数据库的原因

JWT认证的样本代码在 `LoginController` 中有提供

你的API知道，如果用户发送了 `Authorization: Bearer {token}` 的头，这个用户是经过验证的。
这已经在API Service中为你自动配置

推荐你将 `config/jwt` 中生成的token移至 `.env` 中，这个问题将在下一个版本中修复

如果你想修改默认的验证模型，确保你更新了 `config/jwt.php` 中的内容来匹配你做的改动。特别是你想要更新 `user` & `identifier` 的时候。

认证路由

参考上一章中的JWT Authenticated routes部分

多租户

应用中往往需要多个用户角色，对于这种问题你可以创建两个路由组(比方说老师和学生)

- 一个只允许学生（验证用户的学生身份）
 - 一个只允许老师（验证用户的教师身份）
- 为他们各自创建一个中间件
当然，也可以创建一个带参数的中间件

PHP


```
<?php

class TeacherAuthMiddleware
{
    public function handle($request, Closure $next)
    {
        $user = \Auth::user();

        if (!$user || $user->type !== 'teacher') {
            return response()->error('Not Authorized', 401);
        }

        return $next($request);
    }
}
```

PHP

```
<?php

class StudentAuthMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param \Closure $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        $user = \Auth::user();

        if (!$user || $user->type !== 'student') {
            return response()->error('Not Authorized', 401);
        }

        return $next($request);
    }
}
```

不要忘记将这些中间件添加至你的Http Kernel中
接下来，你可以在**routes.php**中使用它们

PHP

```
<?php

//Public endpoints
$api->group([], function ($api) {

});

//Protected endpoints (Mentor or Mentee)
$api->group(['middleware' => 'api.auth'], function ($api) {

});

//Protected: Teacher only
$api->group(['middleware' => ['api.auth', 'auth.teacher']], function ($api) {

});

//Protected: Student only
$api->group(['middleware' => ['api.auth', 'auth.student']], function ($api) {

});
```

测试

参考下一章节中的JWT Auth Test部分

注册 & 登录

Satellizer

Laravel & Angular material starter引入了[satellizer](#)来管理token

你可以在 `angular/config/satellizer.config.js` 中自定义satellizer的配置

注册功能

Create an account

Name

Email

Password

REGISTER

注册功能是预置的功能

`/register` 路由的配置文件为 `routes.config.js`

你有一个包括 `register-form` 组件的页面文件 `register.page.html`

`register-form` 组件通过使用satellizer和 `Auth\AuthController.php` 文件来实现用户注册功能

如果你想添加新的字段，那么你应该同时编辑view，component和AuthController

登录功能

Log in to your account

Email

Password

LOG IN

登录功能也是预置功能。

登录的路由 `/login` 是在 `route.config.js` 中配置的。

在 `login.page.html` 中包含着 `login-form` 组件。

`login-form` 组件通过使用 `satellizer` 和 `Auth\AuthController.php` 实现用户的认证功能

我们可以在 `Auth\AuthController.php` 中检验登录功能的逻辑

重置密码

忘记密码

Forgot your password?

SUBMIT

忘记密码功能是预置的。

忘记密码的路由 `/forget-password` 的配置文件被配置在 `routes.config.js` 中
`forget_password.page.html` 中包含着 `forget-password` 组件

这个组件会发送一个带有重置密码链接的电子邮件。

你可以在 `Auth\PasswordResetController.php` 文件中检查它的逻辑

重置链接

默认的连接前缀是 `localhost:8000`，你需要在 `.env` 增加对应的变量并合理地管理它。

Email的发送地址

在使用这一新特性之前，你需要替换 `Auth\PasswordResetController.php` 为你自己使用的邮箱名

重置密码

Reset Password

点击邮件中的重置密码链接会转到 `/reset-password/:email/:token` 的路由
当加载图案旋转时，`reset-password` 组件将会立刻检测token的有效性
一旦数据库验证通过，重置密码的表单就会出现，提交表单就可以成功重置密码了。

验证路由（前端）

如果你打开 `angular/config/routes.config.js` 文件，你会注意到我们有一个空的参数 `data: {}`，每个state都继承了这一属性因为它在抽象路由 `app` 被定义。

尽管这是一个可选的参数，我们可以在 `angular/run/routes.run.js` 中自定义我们自己所需的逻辑，然后通过检查 `data` 对象中的值并据此完成认证逻辑。

如果你设置成下面的样子

```
data: {  
  auth: true  
}
```

这些路由就会被保护起来（他们需要认证权限）

这一行为可以在单独的路由中启用，也可以通过创建抽象的路由来定义一个路由群组来启用。

我们通过设置一个字符串值来继承这一行为，比如 `auth: 'admin'`，然后你可以在 `routes.run.js` 中定义你自己的逻辑

测试

API test helpers

这个库的目的是让你可以容易测试REST API

下面是一些提供给你的集成测试方法

test helper	用法
seeApiSuccess()	断言返回的响应使用了 <code>success</code> 响应宏（即返回了成功数据）
seeValidationError()	断言返回码为422，并且返回了错误（当你 <code>return response()->error('message', 422);</code> 时Laravel会抛出一个验证错误，这里断言的就是接收到此内容）
seeApiError(\$status_code)	断言具体的状态码错误，比方说401
seeJsonKey(\$key)	等价于 <code>->see('".\$key.'"')</code>
seeJsonValue(\$value)	等价于 <code>->see('".\$value.'"')</code>
seeJsonArray(\$entity)	等价于 <code>->see('".\$entity.':[')'</code> 当你想要得到一个数组时非常有用
seeJsonObject(\$entity)	等价于 <code>->see('".\$entity.':{')'</code> 当你想得到单个数据比方说一个订单的时候非常有用

避免在你的模型工厂中使用bcrypt()。因为这件减慢你的测试，只有当你需要知道user model创建的password时再使用它。（比方说：login,reset password的测试中）

下面是一些示例控制器和它们对应的集成测试方法

Routes.php

```
<?php

$api->group([], function ($api) {

    $api->post('posts', 'PostsController@create');

});
```

PostsController.php

```

<?php

use App\Post;

class PostsController{

    public function create(Request $request)
    {
        $this->validate($request, [
            'name' => 'required',
        ]);

        $post = Post::create([
            'name' => $request->input('name'),
        ]);

        return response()->success(compact('post'));
    }

}

```

CreatePostTest.php

```

<?php

class CreatePostTest extends TestCase
{
    public function testCreatingNewPostSuccessfully()
    {
        $post = factory(App\Post::class)->make();//we're not saving it in the database

        $this->post('/api/posts', ['name' => $post->name])
            ->seeApiResponse()
            ->seeJsonObject('post')
            ->seeJson(['name' => $post->name]);

        $this->seeInDatabase('posts', [
            'name' => $post->name,
        ]);
    }
}

```

不要忘了在你的测试方法中路由头部的 `/api`

JWT Auth Tests

当做测试驱动开发时，我们的大部分测试都会覆盖到被保护的接口。这些接口都需要认证。这个库提供了一些用来测试被保护的接口的方法，而无需再手动发送token

可用方法

以下方法与Laravel的test helper使用了同样的签名(get, post, put, delete, call)。

authUserGet()	GET (验证通过情况)
authUserPost()	POST (验证通过情况)
authUserPut()	PUT (验证通过情况)
authUserDelete()	DELETE (验证通过情况)
authUserCall()	自定义的方法、cookie、files、server等等 (验证通过情况)

你可以通过使用 `$this->getAuthUser()` 来访问当前已经认证的用户，使用 `$this->getAuthUserToken()`，他们在 `/tests/TestCase.php` 中是可用的。

示例

Routes.php

```
<?php

$api->group(['middleware' => 'api.auth'], function ($api) {

    $api->post('posts', 'PostsController@create');

});
```

FavoritePostsTest

```
<?php

class FavoritePostsTest extends TestCase
{
    public function testCreatingNewPost()
    {
        $post = factory(App\Post::class)->make();

        $this->authUserPost('/api/posts', ['name' => $post->name])
            ->seeApiSuccess()
            ->seeJsonObject('post')
            ->seeJsonKeyValueString('name', $post->name);
    }
}
```

多租户应用的测试方法

如果你的应用和平台上有多种用户，你希望有下面的test helper吗？

`->StudentGet()`，`->StudentPost()`，`->TeacherPost()`，`->TeacherGet()`，等等 如果你打开了 `TestCase.php`，你可以看到我们可以通过调用 `$this->getAuthUserToken()` 手动登录一个用户。这是我们一个创建User的工厂。

你的方法应该是找到所有的认证用户方法和变量，并且将其重命名为对应的租户名，然后修改 `Set[租户名]` 方法来获得该类型用户的工厂。

```
$mentor = factory(App\User::class)->create(['type' => '租户名']);
```

其他租户同理

Angular tests

[ng-describe](#)是预置的类库，它可以使你为你的angular app编写流畅的测试代码。

ng-describe使测试**components**，**services**，**inject dependencies**和**mock ajax calls**变得非常简单，你可以浏览它为注册登录组件编写的测试方法。

Angular 生成器

components，**directives**和**service**都自动生成了ng-describe spec 文件，你只需要实现测试用例即可。

forget_password.spec.js

```
ngDescribe({
  name: 'Test forgot-password component',
  modules: 'app',
  inject: ['$http'],
  element: '<forgot-password></forgot-password>',
  http: {
    post: {
      '/api/auth/password/email': {
        data: true
      }
    }
  },
  tests: function(deps) {

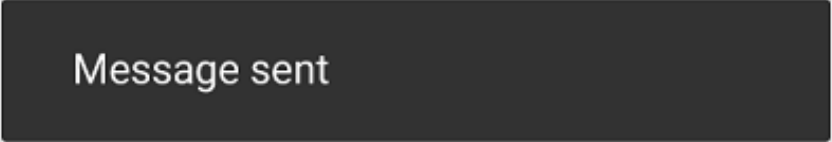
    it('should request email verification successfully', () => {
      var component = deps.element.isolateScope().vm;

      component.email = 'email@localhost.com';
      component.submit();

      deps.http.flush();
    });
  }
});
```

附录（补充）

Toasts



Message sent

[Toast](#)是Angular Material中可用的组件

由于大部分的实现都需要显示一个成功或者失败的Toast，所以我们的初始项目中预置了ToastService，你简单地调整便可以适应你的需要

这个服务允许你打开一个成功或失败的Toast

当API返回错误数据（认证失败）时，失败的Toast会自动展示（使用RestAngular的[异常拦截器](#)）

当这个服务有一个默认的延迟、定位和动作文本时，你可以在 `angular/services/toast.service.js` 中改动，你也可以增加一些自定义方法

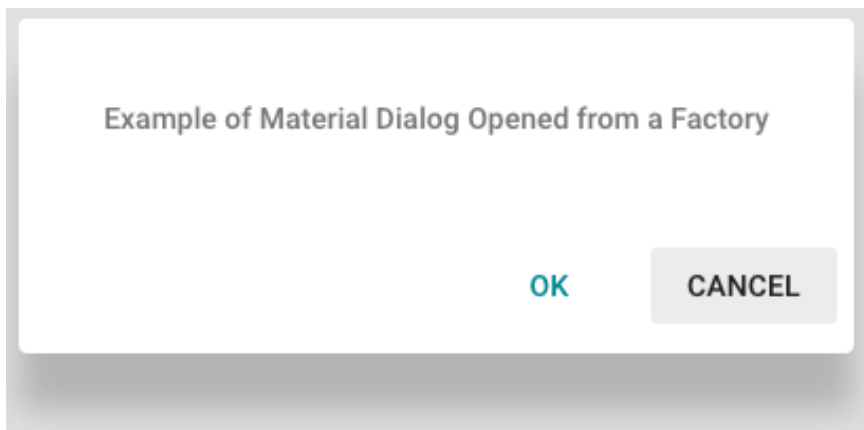
Toast Sample

```
export class PostsController(){
  constructor(ToastService){
    'ngInject';
    this.ToastService = ToastService;
  }

  success(){
    this.ToastService.show('Post added successfully!');
  }

  error(){
    this.ToastService.error('Connection interrupted!');
  }
}
```

Dialogs



[Dialogs](#)是Angular Material中的一个组件

这个库提供给你基于 Angular Material's dialog 服务的一个抽象层。

这个定制的 `DialogService` 允许你通过使用预定义的默认值（你可以在 `angular/services/dialog.service.js` 中进行改动）轻松地弹出一个alert或者confirm

DialogService example

```

export class LandingController() {
  constructor(DialogService){
    'ngInject';

    this.DialogService = DialogService;
    this.confirmMessage = '';
  }

  sampleAlert(){
    this.DialogService.alert('This is an alert title', 'You can specify some description text in here.');
```

```
  }
```

```
  sampleConfirm(){
```

```
    this.DialogService.confirm('This is a confirm title', 'Are you sure you want to do delete this file?').then(() => {
      this.confirmMessage = 'Success callback';
    }, () => {
      this.confirmMessage = 'Cancel callback';
    })
  }
}
```

```
}
```

定制Dialog

另一个有用的特性是你可以调用你定制的dialog
首先生成定制dialog的模板

```
php artisan ng:dialog login
```

这条命令会在angular/dialog目录下新建一个带有样例controller和模板的login文件夹
然后你可以通过下面的样例代码来轻松地实现dialog的调用

Custom Dialog

```
(function() {  
  "use strict";  
  
  angular.module('app.controllers').controller('LoginController', LoginController);  
  
  function LoginController(DialogService) {  
  
    var vm = this;  
  
    vm.customDialog = customDialog;  
  
    var customDialog = function() {  
      DialogService.fromTemplate('login');  
    };  
  
  });  
  
})();
```

这个样例是可以直接使用的

Elixir, live-reload, DialogService & Artisan generators可以一起工作来实现这一效果，不会出现冲突。

不支持的浏览器

Angular Material 的支持对象是目前还保持活力、主流的浏览器，这也是为什么它不支持IE10以及IE10以下的

Please update your browser

You are using an old version of Internet Explorer
Please update it or try one of these options.



它是怎么工作的呢

这段条件注释可以在 `index.blade.php` 中找到

```
<!--[if lte IE 10]>
    <script type="text/javascript">document.location.href = '/unsupported-browser'</script>
<![endif]-->
```

你可以根据自己的设计需要来改变这个页面，这个页面的模板路径为 `resources/views/unsupported_browser.blade.php`

HTTP访问控制（CORS）

[Laravel-cors](#)在项目中是预置的

如果你不需要访问控制，可以移除它。

如果你的应用因为CORS Header的错误出现了异常，你可以尝试创建下面的中间件来替换掉它。

CorsMiddleware.php

```

<?php

namespace App\Http\Middleware;

use Closure;

class CorsMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        return $next($request);
    }

    public function terminate($request, $response)
    {
        return $response->header('Access-Control-Allow-Origin', '*')
            ->header('Access-Control-Allow-Methods', 'POST, GET, OPTIONS, PUT, DELETE')
            ->header('Access-Control-Allow-Headers', 'Content-Type, Accept, Authorizati
on, X-Requested-With');
    }
}

```

主题

[Theming](#)是 Angular Material 预先提供的，你可以给你的用户界面指定三种主色。

这对所有的app来说都是常见的任务。所以我们将它写入了配置文

件 `angular/config/theme.config.js`

它看起来像下面这样：

JavaScript


```
export function ThemeConfig($mdThemingProvider) {
  'ngInject';
  /* For more info, visit https://material.angularjs.org/#/Theming/01_introduction
  */
  $mdThemingProvider.theme('default')
    .primaryPalette('indigo')
    .accentPalette('grey')
    .warnPalette('red');
}
```

更多的内容请参照它的[官方文档](#)

加载动画

[Angular Loading Bar](#)通过监听 `$http` 的事件工作。它为App的开发提供了极大的便利。当你请求一个新页面，请求一个API时，加载进度条就会自动在顶部出现
在RestAngular在 `$http` 上层工作时，它也可以正常工作。

额外的配置被写入了 `angular/config/loading_bar.config.js`

Angular 过滤器

我们提供了一些开箱即用的过滤器给你，这些过滤器都放置在 `angular/filters/` 文件夹中

Filter	用例
capitalize	大写第一个字母，其余小写
humanReadable	将token输出为人类易读的形式，比方说 <code>{{ 'contact_us' humanReadable }}</code> 会输出为Contact Us
truncateCharacters	根据指定的长度截断字符
truncateWords	根据指定的字数截断字符
trustHtml	<code>\$sce.trustAsHtml</code> 的别名，当你理解它的时候再去使用
ucfirst	等价于PHP的 <code>ucfirst</code>

Elixir

Elixir已经默认配置来处理下面的事物：

- 拉取 `bower.json` 中你主要的bower文件并生成 `vender.js` 和 `vender.css`
- 通过eslint来审查你的代码，生成sourcemap，为你的依赖生成注释，将一切在 `app.js` 中连接起来

(ES6支持)

- 编译less文件，生成 `app.css`
- 部分在自动生成时通过使用ngHtml2Js注入（减少了网络请求的消耗，提高了性能）
- 改动时实时重载

在使用 `bower install` 时一定要加上 `--save`，这样它才能被Elixir识别

覆盖bower的主文件

一些包在安装时并没有被正确加载，所以要在 `bower.json` 中进行定义

这是你用到[main-bower-files' overrides](#)特性的场景

比方说，你需要加入bootstrap的css文件，那么你可以在 `bower.json` 中加入这些

```
"overrides": {
  "bootstrap": {
    "main": [
      "dist/css/bootstrap.css",
      "dist/js/bootstrap.js"
    ]
  }
}
```

ESLint 异常/警告 你看到的ESLint抛出的异常或者警告并不会停止代码的运行，它只是为了提高代码质量。

Disable eslint on watch 你可以在 `tasks/angular.task.js` 中禁用eslint

你可以在 `/task` 中自定义你自己的任务

Recipes

到现在，你有一个很棒的Laravel & Angular (material)启动器而且已经对它有所了解。

但是我们不可能包括所有你可能用到的包。

[This repository](#)里面有着社区分享的一些可用包，它们都是社区里乐于分享和回报的人做的。