****

Web应用开发

实 验 报 告

|  |  |
| --- | --- |
| **学 院** | 网络空间安全学院 |
| **专 业** | 网络工程 |
| **班 级** | 19272401 |
| **学 号** | 19270717 |
| **学生姓名** | Qzt |
| **教师姓名** | 胡伟通 |
| **完成日期** | 2021/6/29 |
| **成 绩** |  |

**作业4 RESTful API的实现**

1. **实验目的和要求**
2. 实验目的：基于Laravel框架和RESTful设计准则，按照前后端分离的开发形式来实现简单的用户注册和登录。
3. **实验要求**：（1）实现Email形式的注册功能和相应的登录功能，注册部分具备邮件激活功能（使用Laravel的邮件发送机制或第三方组件）；（2）实现忘记密码时通过重置密码邮件设置新密码（使用Laravel的邮件发送机制或第三方组件）；（3）包含对某个物品（自己选择）的操作，以RESTful API风格进行；（4）如果使用JWT认证（JSON Web Tokens），后面会酌情加分；（5）不要求提供前端方面的实现，即不需要做网站或APP，此外，对API的实际使用在Postman上进行，并进行截图。

注意：要求1和要求2，不能按照作业3的方式来做。

1. **实验内容**

本次实验通过RESTful API实现Email形式的注册功能和相应的登录功能，实现忘记密码时通过重置密码邮件设置新密码，包含对某个物品（自己选择）的操作，以RESTful API风格进行，使用JWT认证，然后在postman上实现。

1. **程序设计思路**

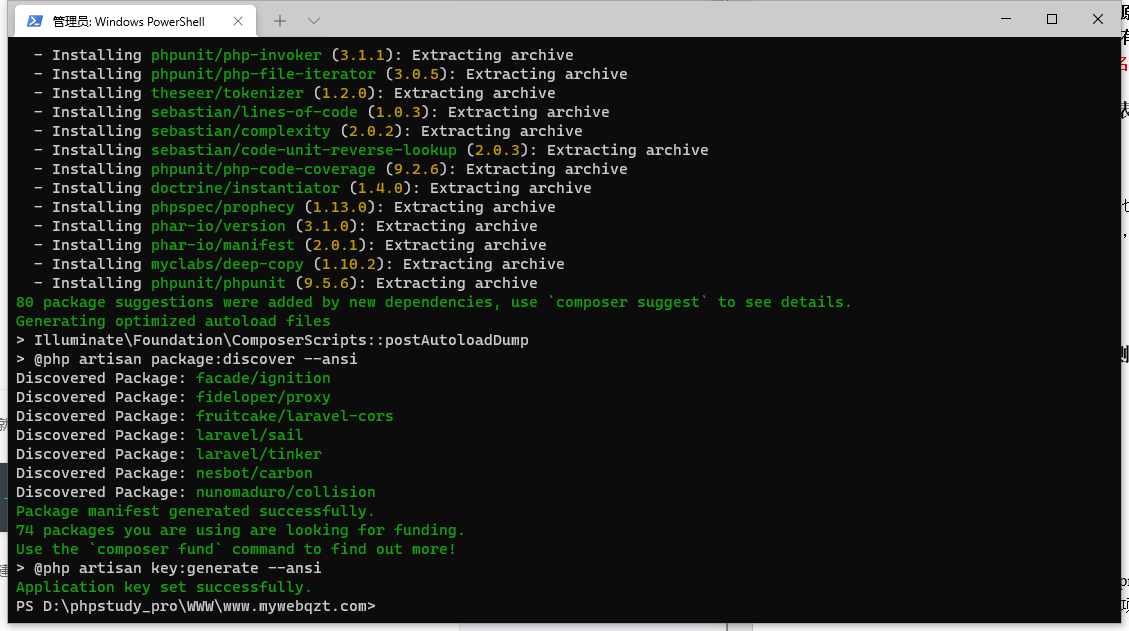
为了实现注册登录、对产品的操作等，需要在routes/api.php里添加路由，然后ApiController.php文件里添加注册登录等操作的代码，然后对产品的操作则要再ProductController.php文件里添加对产品操作的相关代码。

1. **程序源代码**
2. **源代码（包含数据库表设计代码）**

源代码在压缩包里面

1. **程序测试方法及测试结果记录**

**1.首先通过composer create-project --prefer-dist laravel/laravel jwt 命令，创建新的Laravel项目，这个命名会在jwt目录下创建一个新的laravel项目**



**2.安装tymon/jwt-auth 扩展包**

运行一下命令获取dev-develop版本的JWT包

**3.发布配置文件**



该命令会生成 config/jwt.php 配置文件

<?php

return [

    'secret' => env('JWT\_SECRET'),

    'keys' => [

        'public' => env('JWT\_PUBLIC\_KEY'),

        'private' => env('JWT\_PRIVATE\_KEY'),

        'passphrase' => env('JWT\_PASSPHRASE'),

    ],

    'ttl' => env('JWT\_TTL', 60),

    'refresh\_ttl' => env('JWT\_REFRESH\_TTL', 20160),

    'algo' => env('JWT\_ALGO', 'HS256'),

    'required\_claims' => [

        'iss',

        'iat',

        'exp',

        'nbf',

        'sub',

        'jti',

    ],

    'persistent\_claims' => [

        // 'foo',

        // 'bar',

    ],

    'lock\_subject' => true,

    'leeway' => env('JWT\_LEEWAY', 0),

    'blacklist\_enabled' => env('JWT\_BLACKLIST\_ENABLED', true),

    'blacklist\_grace\_period' => env('JWT\_BLACKLIST\_GRACE\_PERIOD', 0),

    'decrypt\_cookies' => false,

    'providers' => [

        'jwt' => Tymon\JWTAuth\Providers\JWT\Lcobucci::class,

        'auth' => Tymon\JWTAuth\Providers\Auth\Illuminate::class,

        'storage' => Tymon\JWTAuth\Providers\Storage\Illuminate::class,

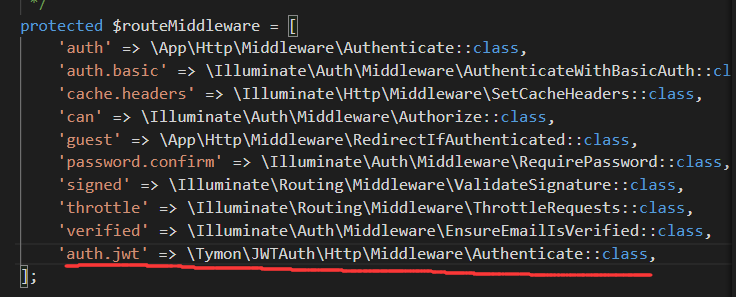
    ],

];

**4.通过运行下面的命令来生成密钥以便于签发令牌**

**php artisan jwt:secret**

**5. 在 app/Http/Kernel.php 中注册 auth.jwt 中间件，在图中所示位置添加代码**



**6.设置路由为routes/api.php添加以下代码**

Route::post('login', 'ApiController@login');

Route::post('register', 'ApiController@register');

Route::group(['middleware' => 'auth.jwt'], function () {

    Route::get('logout', 'ApiController@logout');

    Route::get('user', 'ApiController@getAuthUser');

    Route::get('products', 'ProductController@index');

    Route::get('products/{id}', 'ProductController@show');

    Route::post('products', 'ProductController@store');

    Route::put('products/{id}', 'ProductController@update');

    Route::delete('products/{id}', 'ProductController@destroy');

});

更新User模型，JWT 需要User模型中实现 JWTSubject 接口。这个接口要实现两个方法  getJWTIdentifier 和 getJWTCustomClaims。使用以下代码更新 app/User.php 。

<?php

namespace App\Models;

use Illuminate\Contracts\Auth\MustVerifyEmail;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Foundation\Auth\User as Authenticatable;

use Illuminate\Notifications\Notifiable;

use JWTAuth\Contracts\JWTSubject;

class User extends Authenticatable implements JWTSubject

{

    use HasFactory, Notifiable;

    /\*\*

     \* The attributes that are mass assignable.

     \*

     \* @var array

     \*/

    protected $fillable = [

        'name',

        'email',

        'password',

    ];

    /\*\*

     \* The attributes that should be hidden for arrays.

     \*

     \* @var array

     \*/

    public function getJWTIdentifier()

    {

        return $this->getKey();

    }

    /\*\*

     \* Return a key value array, containing any custom claims to be added to the JWT.

     \*

     \* @return array

     \*/

    public function getJWTCustomClaims()

    {

        return [];

    }

}

}

**7.创建一个表单请求来验证数据，通过以下命令创建RegisterAuthRequest 的表单请求**

**php artisan make:request RegisterAuthRequest**

**这段代码将在 app/Http/Requests 目录下创建 RegisterAuthRequest.php 文件。修改这个文件如下**

<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class RegisterAuthRequest extends FormRequest

{

    /\*\*

     \* Determine if the user is authorized to make this request.

     \*

     \* @return bool

     \*/

    public function authorize()

    {

        return true;

    }

    /\*\*

     \* Get the validation rules that apply to the request.

     \*

     \* @return array

     \*/

    public function rules()

    {

        return [

            'name' => 'required|string',

            'email' => 'required|email|unique:users',

            'password' => 'required|string|min:6|max:10'

        ];

    }

}

**8.运行以下命令创建一个新的ApiController**

**php artisan make:controller ApiController**

**9. 这段代码会在 app/Http/Controllers 目录下创建 ApiController.php 文件。修改以下代码如下**

<?php

namespace App\Http\Controllers;

use App\Http\Requests\RegisterAuthRequest;

use App\User;

use Illuminate\Http\Request;

use JWTAuth;

use Tymon\JWTAuth\Exceptions\JWTException;

class ApiController extends Controller

{

    public $loginAfterSignUp = true;

    public function register(RegisterAuthRequest $request)

    {

        $user = new User();

        $user->name = $request->name;

        $user->email = $request->email;

        $user->password = bcrypt($request->password);

        $user->save();

        if ($this->loginAfterSignUp) {

            return $this->login($request);

        }

        return response()->json([

            'success' => true,

            'data' => $user

        ], 200);

    }

    public function login(Request $request)

    {

        $input = $request->only('email', 'password');

        $jwt\_token = null;

        if (!$jwt\_token = JWTAuth::attempt($input)) {

            return response()->json([

                'success' => false,

                'message' => 'Invalid Email or Password',

            ], 401);

        }

        return response()->json([

            'success' => true,

            'token' => $jwt\_token,

        ]);

    }

    public function logout(Request $request)

    {

        $this->validate($request, [

            'token' => 'required'

        ]);

        try {

            JWTAuth::invalidate($request->token);

            return response()->json([

                'success' => true,

                'message' => 'User logged out successfully'

            ]);

        } catch (JWTException $exception) {

            return response()->json([

                'success' => false,

                'message' => 'Sorry, the user cannot be logged out'

            ], 500);

        }

    }

    public function getAuthUser(Request $request)

    {

        $this->validate($request, [

            'token' => 'required'

        ]);

        $user = JWTAuth::authenticate($request->token);

        return response()->json(['user' => $user]);

    }

}

**10.创建产品部分，需要Product模型，控制器与迁移文件。运行如下命令创建Product模型，控制器与迁移文件。**

**php artisan make:model Product -mc**

**这段代码会在database/migrations 目录下创建一个新的数据库迁移文件 create\_products\_table.php，现在更改以下里面的 up 方法。**

public function up()

    {

        Schema::create('products', function (Blueprint $table) {

            $table->increments('id');

            $table->integer('user\_id');

            $table->string('name');

            $table->integer('price');

            $table->integer('quantity');

            $table->timestamps();

            $table->foreign('user\_id')

                ->references('id')

                ->on('users')

                ->onDelete('cascade');

        });

**}**

**11.在Product模型中添加fillable属性。在app目录下打开Product.php文件并添加属性。**

class Product extends Model

{

    protected $fillable = [

        'name', 'price', 'quantity'

    ];

}

**12.在.env文件中设置数据库凭证，通过运行以下命令来迁移数据库。**

**php artisan migrate**

**13.在user模型中添加一个关系来检索相关的产品。在 app/User.php 中添加以下方法。**

public function products()

    {

        return $this->hasMany(Product::class);

    }

**14. 在 app/Http/Controllers 目录下打开 ProductController.php 文件。在文件开头添加 use 指令覆盖上一个。**

use App\Product;

use Illuminate\Http\Request;

use JWTAuth;

**15.添加5个方法**

① index, 为经过身份认证的用户获取所有产品列表

② show, 根据 ID 获取特定的产品

③ store, 将新产品存储到产品列表中

④ update, 根据 ID 更新产品详情

⑤ destroy, 根据 ID 从列表中删除产品

添加一个构造函数来获取经过身份认证的用户，并将其保存在 user 属性中。

protected $user;

public function \_\_construct()

{

    $this->user = JWTAuth::parseToken()->authenticate();

}

添加index方法

public function index()

{

    return $this->user

        ->products()

        ->get(['name', 'price', 'quantity'])

        ->toArray();

}

添加show方法

public function show($id)

{

    $product = $this->user->products()->find($id);

    if (!$product) {

        return response()->json([

            'success' => false,

            'message' => 'Sorry, product with id ' . $id . ' cannot be found'

        ], 400);

    }

    return $product;

}

添加store方法

public function store(Request $request)

{

    $this->validate($request, [

        'name' => 'required',

        'price' => 'required|integer',

        'quantity' => 'required|integer'

    ]);

    $product = new Product();

    $product->name = $request->name;

    $product->price = $request->price;

    $product->quantity = $request->quantity;

    if ($this->user->products()->save($product))

        return response()->json([

            'success' => true,

            'product' => $product

        ]);

    else

        return response()->json([

            'success' => false,

            'message' => 'Sorry, product could not be added'

        ], 500);

}

添加update方法

public function update(Request $request, $id)

{

    $product = $this->user->products()->find($id);

    if (!$product) {

        return response()->json([

            'success' => false,

            'message' => 'Sorry, product with id ' . $id . ' cannot be found'

        ], 400);

    }

    $updated = $product->fill($request->all())

        ->save();

    if ($updated) {

        return response()->json([

            'success' => true

        ]);

    } else {

        return response()->json([

            'success' => false,

            'message' => 'Sorry, product could not be updated'

        ], 500);

    }

}

添加destroy方法

public function destroy($id)

{

    $product = $this->user->products()->find($id);

    if (!$product) {

        return response()->json([

            'success' => false,

            'message' => 'Sorry, product with id ' . $id . ' cannot be found'

        ], 400);

    }

    if ($product->delete()) {

        return response()->json([

            'success' => true

        ]);

    } else {

        return response()->json([

            'success' => false,

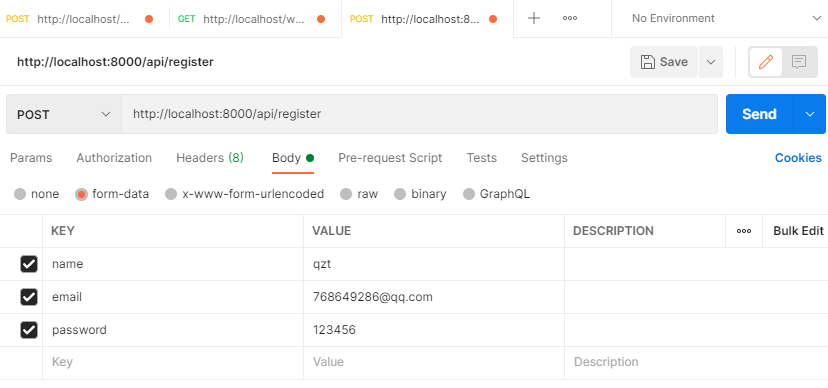
            'message' => 'Product could not be deleted'

        ], 500);

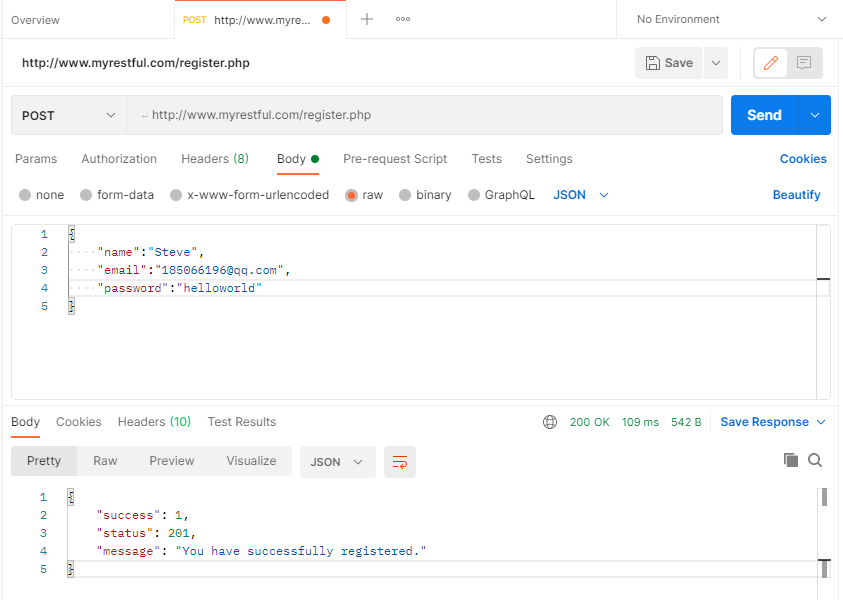
    }

}

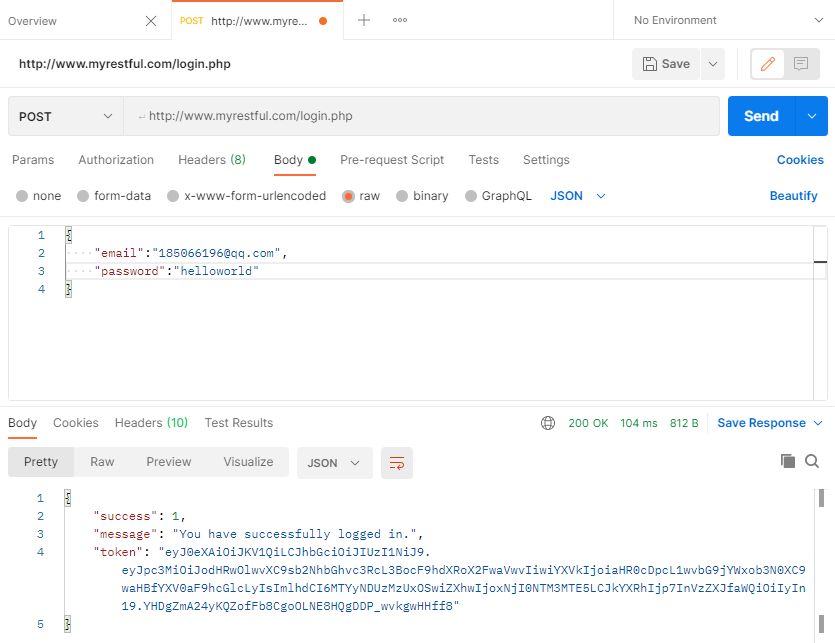
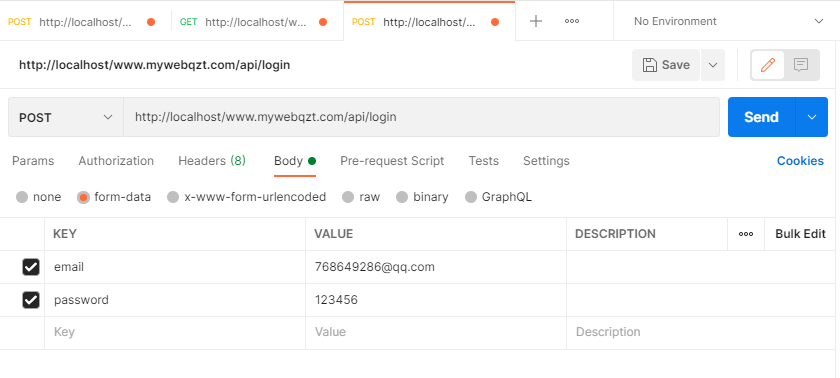
**16.打开postman开始测试**



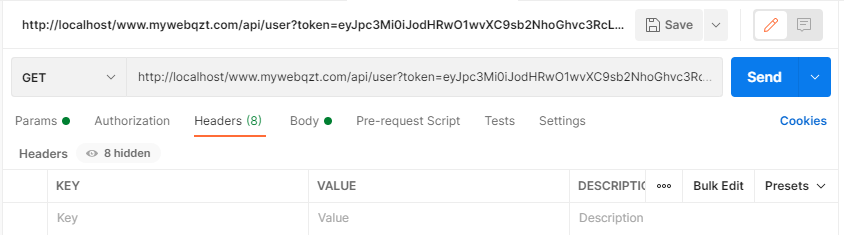
填入姓名，邮件，密码，然后点击send ，显示注册成功

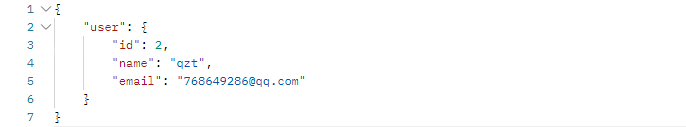


测试登录login路由，输入邮箱和密码

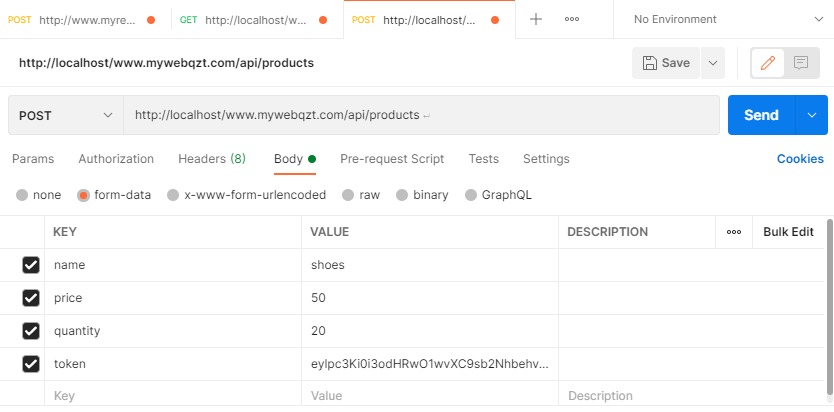


接下来通过token来获取用户的详细信息，此时输入刚刚生成的token

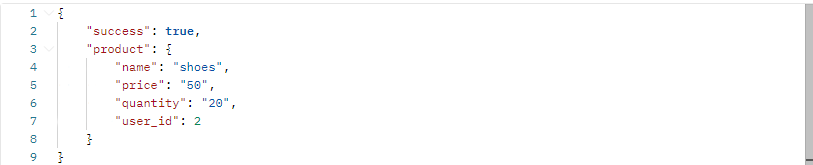




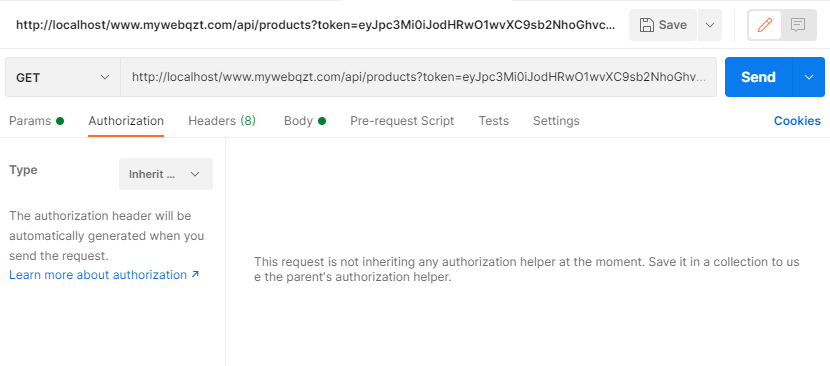
测试产品功能，首先创建一个产品

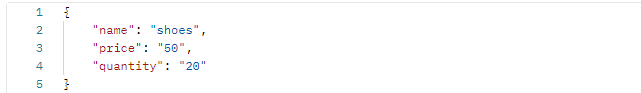


得到结果如下



然后通过index方法获取产品





1. **实验分析总结及心得**

通过此次实验，我实现了用PHP restful api登录和注册网站，在实验中也遇到了一些问题，我所编写的文件放在了网址根目录下的一个文件夹里，导致使用postman进行测试的时候调用不了我所编写的登录注册php文件，解决办法就是直接把我所编写的文件放到网址根目录下，这样用postman就可以测试成功了。