



ISO 9001:2008

# **BÁO CÁO CHUYÊN ĐỀ WEB 2**

**| GVHD: Phan Thanh Nhuận**

**| Khoa Công Nghệ Thông Tin – Khoá 2016**

**| Nhóm A:**

Nguyễn Linh Chân

Nguyễn Hoài Phong

Trương Diệu My

Võ Ngọc Phú

**Thành phố Hồ Chí Minh**

**Tháng 4 – 2019**

---

## MỤC LỤC

NỘI DUNG.....	1
GIỚI THIỆU.....	1
CHUẨN BỊ.....	2
CÀI ĐẶT.....	4
ĐẶC TẢ YÊU CẦU.....	8
TEST API.....	14
HƯỚNG DẪN TẠO 1 PACKAGE TRONG LARAVEL.....	28
HƯỚNG DẪN PUBLISH 1 PACKAGE LÊN PACKAGIST.....	41
HƯỚNG DẪN SỬ DỤNG PACKAGE.....	44
TÀI LIỆU THAM KHẢO.....	47

---

## | NỘI DUNG

- Chuẩn bị.
- Cài đặt.
- Đặc tả yêu cầu.
- Test API
- Hướng dẫn tạo package trong Laravel.
- Hướng dẫn publish 1 package lên packagist.
- Hướng dẫn sử dụng package.

## | GIỚI THIỆU

Tài liệu do nhóm A – khoá 2016 biên soạn dưới sự hướng dẫn của giảng viên giảng dạy môn chuyên đề web 2. Với mục đích tìm hiểu và hiện thực các phương thức CRUD của RESTful API, tìm hiểu cách tạo ra 1 package, cài đặt và sử dụng package trong framework “Laravel” đã đưa lên packagist. Nhóm đã tìm hiểu và hoàn thành các chức năng cơ bản để có thể viết ra file hướng dẫn này và có thể giúp cho các bạn sinh viên khoá sau có thể sử dụng tài liệu này cho mục đích học tập môn chuyên đề web 2.

Tuy nhiên, trong quá trình biên soạn, nhóm cũng đang trong quá trình thực hiện, nên không thể tránh khỏi những thiếu sót cũng như gặp phải những lỗi không mong muốn. Nhóm sẽ cố gắng khắc phục những lỗi cũng như những hạn chế không mong muốn.

## | CHUẨN BỊ

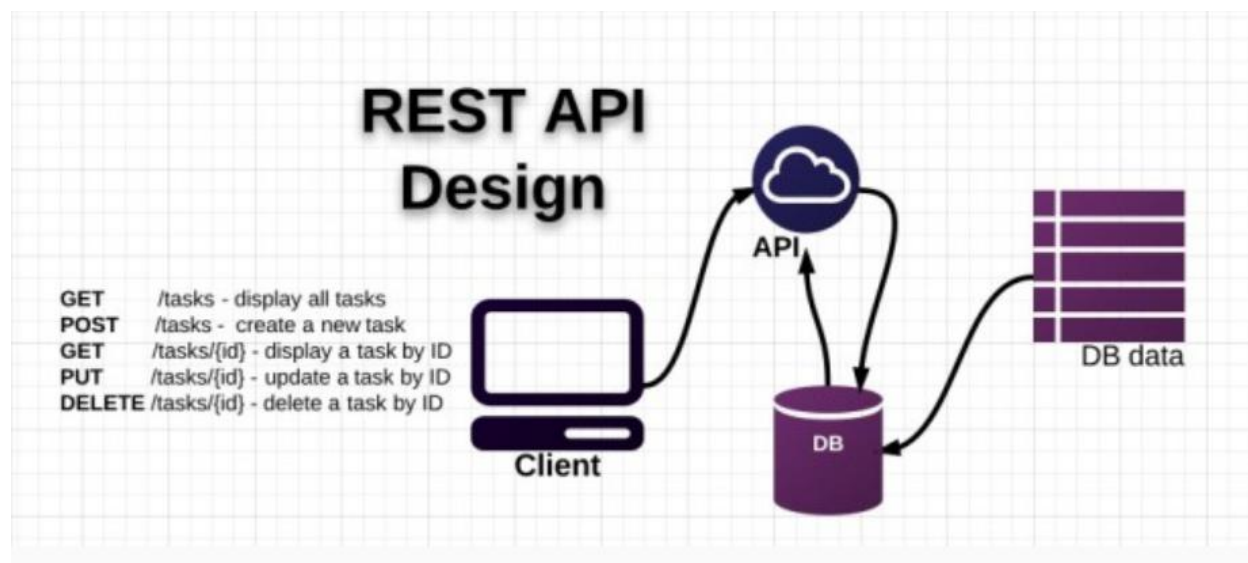
Để có thể bắt đầu, trước hết chúng ta cần hiểu về khái niệm RESTful API là gì.

REST là viết tắt của Representational State Transfer. REST là một chuẩn web dựa vào các kiến trúc cơ bản sử dụng giao thức HTTP. Nó xử lý tài nguyên, nơi mà mỗi thành phần là một tài nguyên và nguồn tài nguyên này có thể được truy cập qua các giao diện chung bởi sử dụng các phương thức HTTP chuẩn.

API (Application Programming Interface) là bộ các phương thức để kết nối máy tính với hệ thống và kết quả trả về theo dạng Json hay XML.

Một website thường sử dụng API để tương tác dữ liệu từ hệ thống. Phần lớn các website lớn như Facebook, Google đều sử dụng API. Website chủ yếu được thực hiện với 4 phương thức chính là CRUD: Create, Read, Update, Delete. Các lập trình viên khác nhau sẽ xây dựng website khác nhau từ 4 phương thức CRUD.

Khi người dùng (web browser) gửi 1 request thì thông qua url đó, thì phía server sẽ xử lý và trả về 1 trang HTML. Dữ liệu được truyền qua lại là 1 JSON. JSON là 1 dạng dữ liệu mà hầu như các ngôn lập trình ngày nay đều có thể đọc được.



Hình 1. Sơ đồ tổng quát RESTful API

## **Phương thức HTTP được sử dụng trong REST**

Có 4 phương thức sử dụng phổ biến trong kiến trúc REST:

**GET:** Dùng để đọc dữ liệu về từ các nguồn tài nguyên.

**POST:** Dùng để cập nhật hoặc thêm mới dữ liệu các nguồn tài nguyên.

**PUT:** Dùng để cập nhật dữ liệu từ các nguồn tài nguyên.

**DELETE:** Dùng để xóa dữ liệu từ các nguồn tài nguyên.

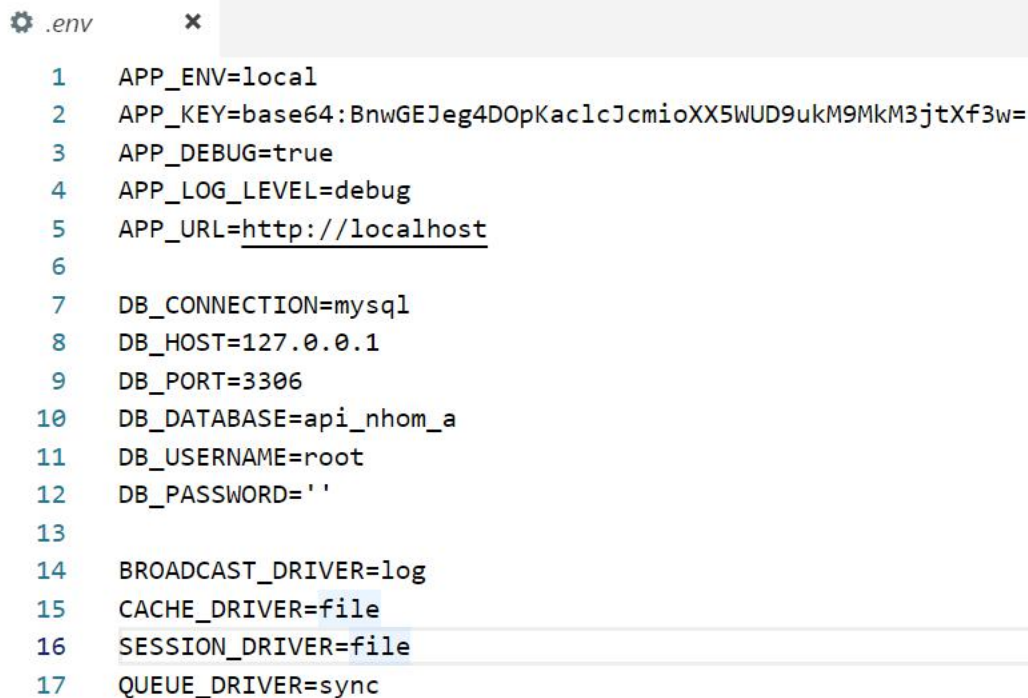
Bên cạnh đó, chúng ta phải có kiến thức cơ bản về MySQL, Laravel framework để giúp cho chúng ta có thể dễ dàng trong các bước thực hiện ở các phần sau.

Tiếp theo, chúng ta cần cấu hình laravel, kết nối database và tạo dữ liệu mẫu.

## | CÀI ĐẶT

Đầu tiên, chúng ta cần có 1 source Laravel (bao gồm vendor và file .env) và tên cơ sở dữ liệu. Hãy kết nối chúng lại với nhau.

Vào file cấu hình `.env` và thêm các dòng sau:



```
1 APP_ENV=local
2 APP_KEY=base64:BnwGEJeg4D0pKac1cJcmioXX5WUD9ukM9MkM3jtXf3w=
3 APP_DEBUG=true
4 APP_LOG_LEVEL=debug
5 APP_URL=http://localhost
6
7 DB_CONNECTION=mysql
8 DB_HOST=127.0.0.1
9 DB_PORT=3306
10 DB_DATABASE=api_nhom_a
11 DB_USERNAME=root
12 DB_PASSWORD= ''
13
14 BROADCAST_DRIVER=log
15 CACHE_DRIVER=file
16 SESSION_DRIVER=file
17 QUEUE_DRIVER=sync
```

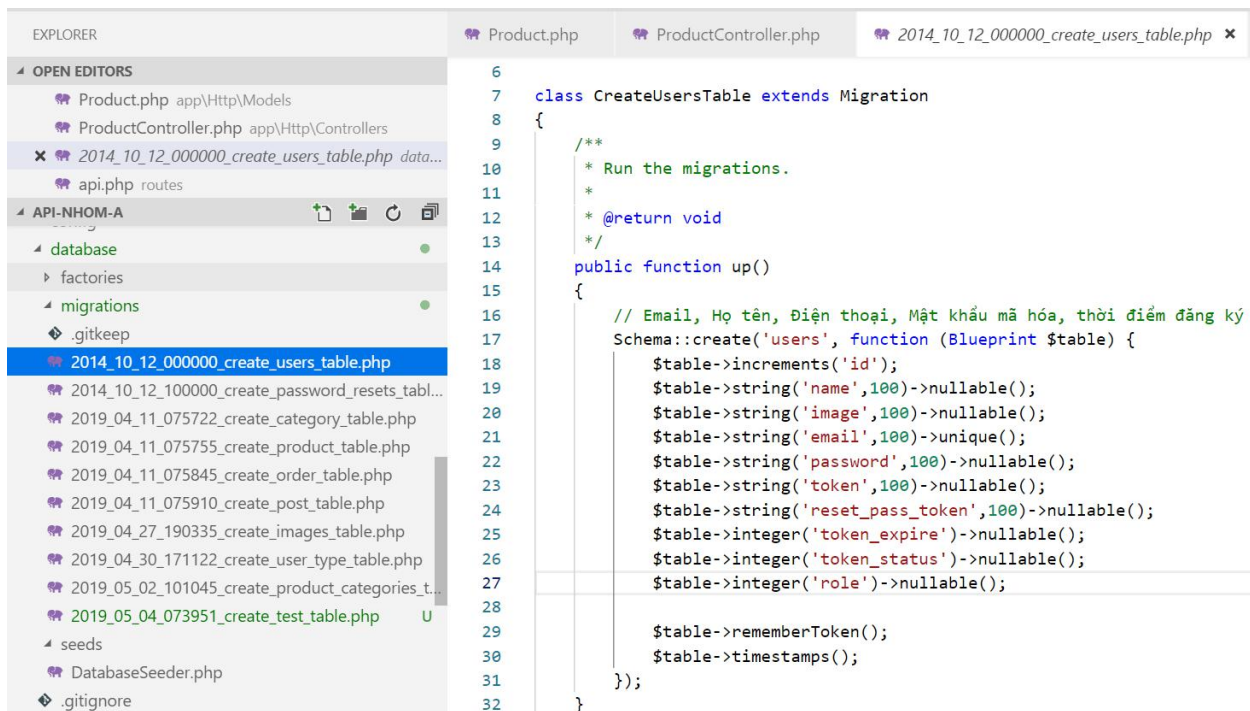
Hình 2. File cấu hình `.env`

Tiếp theo, chúng ta có thể tạo dữ liệu mẫu trực tiếp trên `localhost/phpmyadmin` hoặc chúng ta có thể dùng lệnh để tạo dữ liệu. Ở đây, nhóm sẽ hướng dẫn các bạn sử dụng câu lệnh để tạo dữ liệu mẫu.

Mở Terminal và chạy dòng lệnh sau:

```
php artisan make:migration create_users_table --create=users
```

Sau khi câu lệnh được thực hiện, hệ thống sẽ tạo ra 1 file như hình bên dưới trong `database/migrations`, tại đây các bạn có thể thêm các thuộc tính cho bảng users.



```

6
7 class CreateUsersTable extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         // Email, Họ tên, Điện thoại, Mật khẩu mã hóa, thời điểm đăng ký
17         Schema::create('users', function (Blueprint $table) {
18             $table->increments('id');
19             $table->string('name',100)->nullable();
20             $table->string('image',100)->nullable();
21             $table->string('email',100)->unique();
22             $table->string('password',100)->nullable();
23             $table->string('token',100)->nullable();
24             $table->string('reset_pass_token',100)->nullable();
25             $table->integer('token_expire')->nullable();
26             $table->integer('token_status')->nullable();
27             $table->integer('role')->nullable();
28
29             $table->rememberToken();
30             $table->timestamps();
31         });
32     }

```

Hình 3. Thêm các cột cho bảng user

Sau khi thêm các cột cho bảng, chúng ta mở file `DatabaseSeeder.php` và thêm vào các dòng sau để khi chạy câu lệnh dữ liệu sẽ tự động thêm vào database. Mặc định file này đã có sẵn class `DatabaseSeeder` có function `run()` gọi tới các seed khác, cho phép bạn có thể điều khiển được thứ tự thêm dữ liệu theo cách của bạn.

Có rất nhiều cách thêm dữ liệu bằng seeder nhưng ở đây chúng ta sẽ thêm dữ liệu trực tiếp trong function run().

```
<?php

use Faker\Factory;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        DB::table('users')->insert([
            'name' => 'Dieu My',
            'email' => 'truongdieumy97@gmail.com',
            'password' => bcrypt('123456'),
            'role' => '-1',
        ]);
    }
}
```

Hình 4. Thêm dữ liệu mẫu

Sau khi thực hiện các bước trên, chúng ta chạy câu lệnh sau để thực thi chèn dữ liệu vào bảng users. Trước khi thực hiện điều này, để chắc chắn sẽ không phát sinh lỗi, chúng ta nên thêm các thuộc tính sau để hệ thống biết được dữ liệu sẽ được thêm cụ thể vào bảng nào.



Tại Model mà chúng ta cần thêm dữ liệu, hãy tạo 2 thuộc tính `$table` và `$primaryKey` như sau:

```
class User extends Authenticatable
{
    use Notifiable;

    protected $table = "users";
    protected $primaryKey = "id";

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

Hình 5. Thêm tên bảng và khóa chính

Sau đó, chúng ta chạy lệnh bên dưới:

`php artisan migrate:refresh --seed` hoặc `php artisan db:seed`

Như vậy, dữ liệu đã có và kết nối đến cơ sở dữ liệu hoàn tất.

## | ĐẶC TẢ YÊU CẦU

### Các API đã viết

Phương thức	Url	Mô tả
GET	api/users	Lấy toàn bộ danh sách user
POST	api/users/page	Phân trang danh sách user
GET	api/users/search	Tìm kiếm user theo name hoặc email, phân trang kết quả tìm kiếm
GET	api/users/{id}	Lấy thông tin user theo id
POST	api/users/login	Đăng nhập với email và password
GET	api/users/logout	Đăng xuất
POST	api/users/create	Thêm mới 1 user
PUT	api/users/update/{id}	Cập nhật thông tin của user theo id
PUT	api/users/change-password/{id}	Thay đổi mật khẩu theo id của user
DELETE	api/users/delete/{id}	Xoá user theo id
POST	api/users/update-image	Cập nhật hình ảnh cho user
GET	api/products	Lấy danh sách toàn bộ sản phẩm
POST	api/products/search	Tìm kiếm sản phẩm
GET	api/products/{id}	Lấy thông tin sản phẩm theo id
POST	api/products/create	Thêm mới 1 sản phẩm
PUT	api/products/update	Thay đổi thông tin sản phẩm
DELETE	api/products/delete	Xoá sản phẩm khỏi danh sách
Các chức năng trên khi thực hiện request cần phải có token xác thực người dùng.		

## Danh sách các web service mà chúng ta cần

STT	Title	Url	Description	Method	Header	Parameter	Request response
1	Update user avatar image	/api/users/update-with-image	update avatar image for user	POST	header authorization	+ token + email + image	<p>If success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: a success message</li> </ul> <p>If token is not valid</p> <ul style="list-style-type: none"> <li>Header response status: 401</li> <li>Body: a message “the token is not valid”</li> </ul> <p>If image_value is not valid</p> <ul style="list-style-type: none"> <li>Header response status: 201</li> <li>Body: a message “The image must be a file of type: jpeg, jpg, png, gif, svg.”</li> </ul> <p>If fail</p> <ul style="list-style-type: none"> <li>Header: response status: 201</li> <li>Body: a error message</li> </ul>
2	Update user information	api/users/update/{id}	Update user informations such as name, email	POST	header authorization	+ email + token + name	<p>If success</p> <ul style="list-style-type: none"> <li>Response status: 200</li> <li>Body: a successful message</li> </ul> <p>If token if not valid</p> <ul style="list-style-type: none"> <li>Header: response status: 401</li> <li>Body: a message “the token is not valid”</li> </ul> <p>If fail</p> <ul style="list-style-type: none"> <li>Response status: 201</li> <li>Body: a error message</li> </ul>
3	Search user by email and name, paginate result	api/users/search	Search information user by name and email	GET	header authorization	+ Key + Page + Token	<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: Show list result, paginate 2 result for page</li> </ul>

							<p>Token null</p> <ul style="list-style-type: none"> <li>Response status: 204</li> <li>Body: Message “Unauthorized user”</li> </ul>
4	Login user with create Token	api/users/login	Use to login user by email, password and create token for check user	POST	header authorization	+ email + password	<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: Show information user login and token user</li> </ul> <p>Fails</p> <ul style="list-style-type: none"> <li>Response status: 401</li> <li>Body: message login fail</li> </ul>
5	Delete user	api/users/delete/{id}	Delete user	DELETE	header authorization	+ token	<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: Message “Delete success”</li> </ul> <p>Token null</p> <ul style="list-style-type: none"> <li>Header: response status: 406</li> <li>Body: Message “Unauthorized user”</li> </ul> <p>Role user</p> <ul style="list-style-type: none"> <li>Header: response status: 404</li> <li>Body: Message “You must be Admin to delete user”</li> </ul>
6	Create User, only access admin this API	api/users/create	For client to create a new user object	POST	header authorization	+ name + email + password + token + role	<p>If success:</p> <ul style="list-style-type: none"> <li>Response status 201</li> <li>Body: Message: Create success</li> </ul> <p>If failed:</p> <p>+ The email has already been taken</p> <ul style="list-style-type: none"> <li>Response status 401</li> <li>Body: Message: The email has already been taken.</li> </ul> <p>+ If name, password absurd</p> <ul style="list-style-type: none"> <li>Response status 401</li> <li>Body: Message: The name must be at</li> </ul>

							<p>least 3 characters, the password must be at least 6 characters.</p> <p>+ If token invalid</p> <ul style="list-style-type: none"> <li>• Response status 401</li> <li>• Body: Message: Unauthorized user.</li> </ul> <p>+ If can not admin</p> <ul style="list-style-type: none"> <li>• Response status 401</li> <li>• Body: Message: Can't create an object because you don't have permission.</li> </ul>
7	Change the password	api/users/change-password/{id}	Update password of user with token when login	PUT	header authorization	+ password + token	<p>If success:</p> <ul style="list-style-type: none"> <li>• Response status: 200</li> <li>• Body: a message change the password success.</li> </ul> <p>If failed:</p> <p>+ If token invalid</p> <ul style="list-style-type: none"> <li>• Response status: 401</li> <li>• Body: a message show notification unauthorized user</li> </ul>
8	Get all user	api/users	Get all information user	GET	header authorization	+ token	<p>Success</p> <ul style="list-style-type: none"> <li>• Header: response status: 200</li> <li>• Body: Status: 200, List user: Show list user</li> </ul> <p>Fail</p> <ul style="list-style-type: none"> <li>• Response status: 401</li> <li>• Body: Status: 401 , Show message: "Account has not been verified"</li> </ul>
9	Get user by id	api/users/{id}	Get information user by id	GET	header authorization	+ token	<p>Success</p> <ul style="list-style-type: none"> <li>• Response status: 200</li> <li>• Body: Status: 200, User: Show user by id</li> </ul> <p>Fail</p> <ul style="list-style-type: none"> <li>• Header: response status: 201</li> </ul>

							<ul style="list-style-type: none"> <li>Body: Status: 201, Show message: "Find not user"</li> </ul>
10	Page user	api/users/page	Page list user	POST	header authorization	+ page + token	<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 206</li> <li>Body: Status: 206, List user: show list about 5 user</li> </ul> <p>Fail</p> <ul style="list-style-type: none"> <li>Response status: 401</li> <li>Body: Status: 401, Error: Account has not been verified</li> </ul>
11	Logout	api/users/logout	logout user	GET	header authorization	+ token	<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: Status: 200, Message: "Logout success"</li> </ul> <p>Fail</p> <ul style="list-style-type: none"> <li>Response status: 401</li> <li>Body: Status: 401, Message: "Unauthorized user"</li> </ul>
12	Get all product	api/products	Show list product	GET	header authorization		<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: Status: 200, List product: Show list product</li> </ul>
13	Search product with keyword	api/products/search	Search product with keyword	POST	header authorization	+keyword	<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: Status: 200, Product: show product</li> </ul> <p>Fail</p> <ul style="list-style-type: none"> <li>Response status: 200</li> </ul> <p>Body: Status: 200, Product: []</p>
14	Get product by id	api/products/{id}	Show product by id	GET	header authorization	+ id	<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: Status: 200, Product: show</li> </ul>

							<p>product</p> <p>Fail</p> <ul style="list-style-type: none"> <li>Response status: 401</li> </ul> <p>Body: Staus:401, Message: Find not product</p>
15	Create product	api/products/create	Create new product	POST	header authorization	+name +price +description +cate_id	<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: Status: 200, Success: "Create success"</li> </ul> <p>Fail</p> <ul style="list-style-type: none"> <li>Response status: 400</li> <li>Body: Staus: 400, Message: "Create fail"</li> </ul>
16	Update image product	api/products/update-image	Update product image	POST	header authorization	+id +image	<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: Status: 200, Success: "Update success"</li> </ul> <p>Fail</p> <ul style="list-style-type: none"> <li>Response status: 400</li> </ul> <p>Body: Staus: 400, Message: "Update fail"</p>
17	Update product by id	api/products/update	Update product	PUT	header authorization	+ id + name + price +description	<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: Status: 200, Success: "Update success"</li> </ul> <p>Fail</p> <ul style="list-style-type: none"> <li>Response status: 400</li> </ul> <p>Body: Staus: 400, Message: "Update fail"</p>
18	Delete product by id	api/products/delete	Delete product by id	DELETE	header authorization	+ id	<p>Success</p> <ul style="list-style-type: none"> <li>Header: response status: 200</li> <li>Body: Status: 200, Success: "Delete success"</li> </ul> <p>Fail</p>

							<ul style="list-style-type: none"> <li>Response status: 400</li> </ul> <p>Body: Staus: 400, Message: “Find not product”</p>
--	--	--	--	--	--	--	---

## | TEST API

Để kiểm tra xem các API mà chúng ta đã liệt kê ra có đúng hay không bạn cần phải sử dụng phần mềm Postman để test. Nếu chưa có, hãy cài đặt và bắt đầu test.

Trước hết, để chạy test được các API trên Postman bạn cần phải chạy server bằng lệnh:

`php artisan serve`

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

php artisan serve
Laravel development server started: <http://127.0.0.1:8000>
PS D:\wamp64\www\api-nhom-a> php artisan serve
Laravel development server started: <http://127.0.0.1:8000>

```

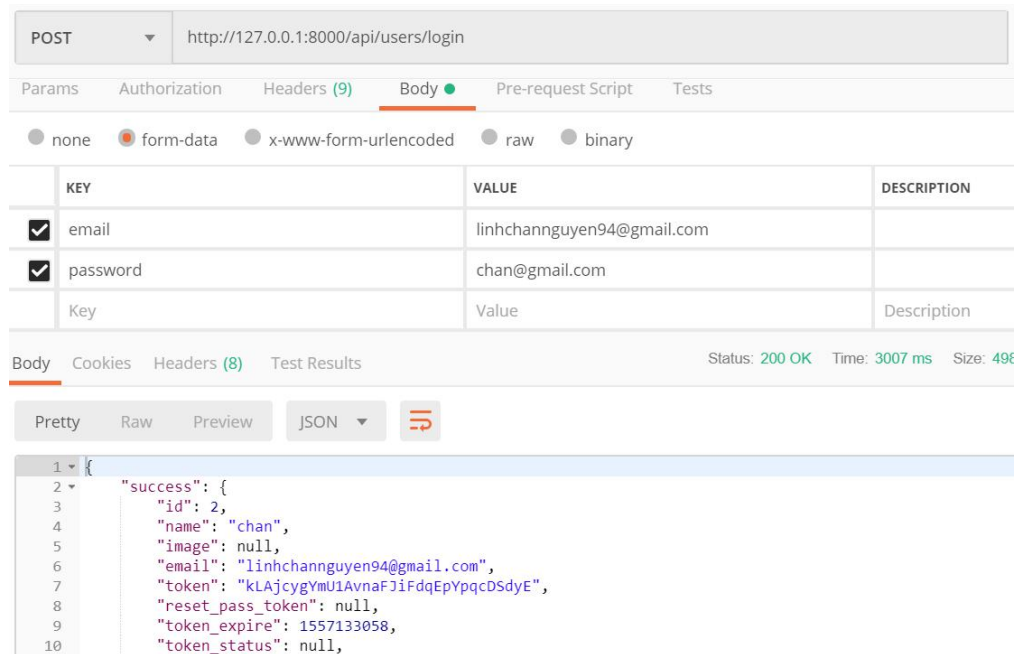
*Hình 6. Start server*

Sau đó copy url <http://127.0.0.1:8000> bỏ vào url bên dưới.



## Login

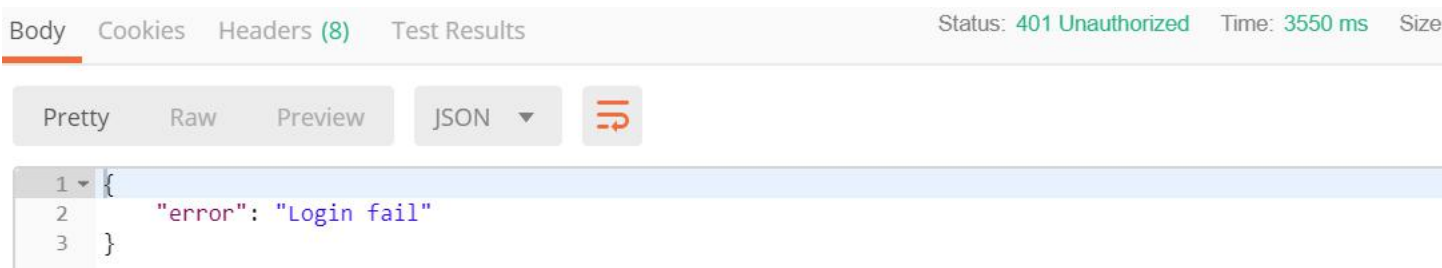
Click vào tab Body -> chọn form-data sau đó thêm vào 2 trường email và password. Chọn phương thức POST ở option kế url. Nếu dữ liệu đúng sẽ có kết quả như sau:



Hình 7. Login success

Khi user login success nghĩa là bạn đã đăng nhập thành công thì user đó sẽ được tạo ra 1 token để xác thực được user đó là ai và có quyền gì.

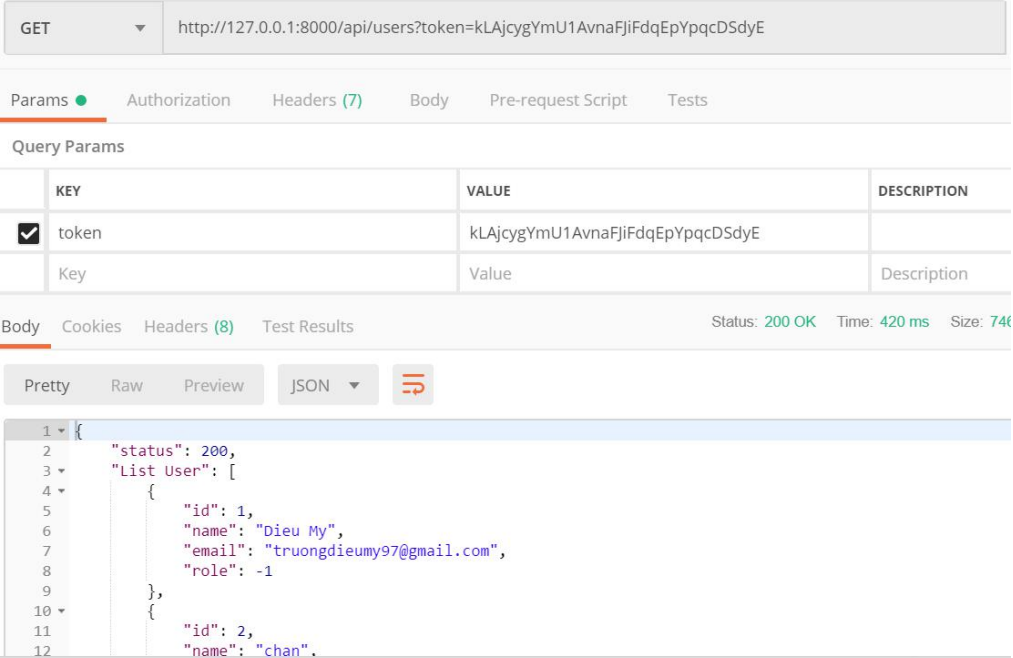
Nếu dữ liệu không đúng:



Hình 8. Login fail

## Lấy danh sách user dựa vào token

Click tab Params, thêm vào trường token và token phải có giá trị đúng với user token được tạo ra mà chúng ta vừa đăng nhập. Nếu đúng hệ thống sẽ trả về như sau:



GET http://127.0.0.1:8000/api/users?token=kLAjcygYmU1AvnaFjiFdqEpYpqcDSdyE

Params Authorization Headers (7) Body Pre-request Script Tests

Query Params

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> token	kLAjcygYmU1AvnaFjiFdqEpYpqcDSdyE	
Key	Value	Description

Body Cookies Headers (8) Test Results Status: 200 OK Time: 420 ms Size: 744

Pretty Raw Preview JSON

```
1 {
2   "status": 200,
3   "List User": [
4     {
5       "id": 1,
6       "name": "Dieu My",
7       "email": "truongdieumy97@gmail.com",
8       "role": -1
9     },
10    {
11      "id": 2,
12      "name": "chan".
```

Hình 9. Lấy danh sách user

Nếu token không đúng:



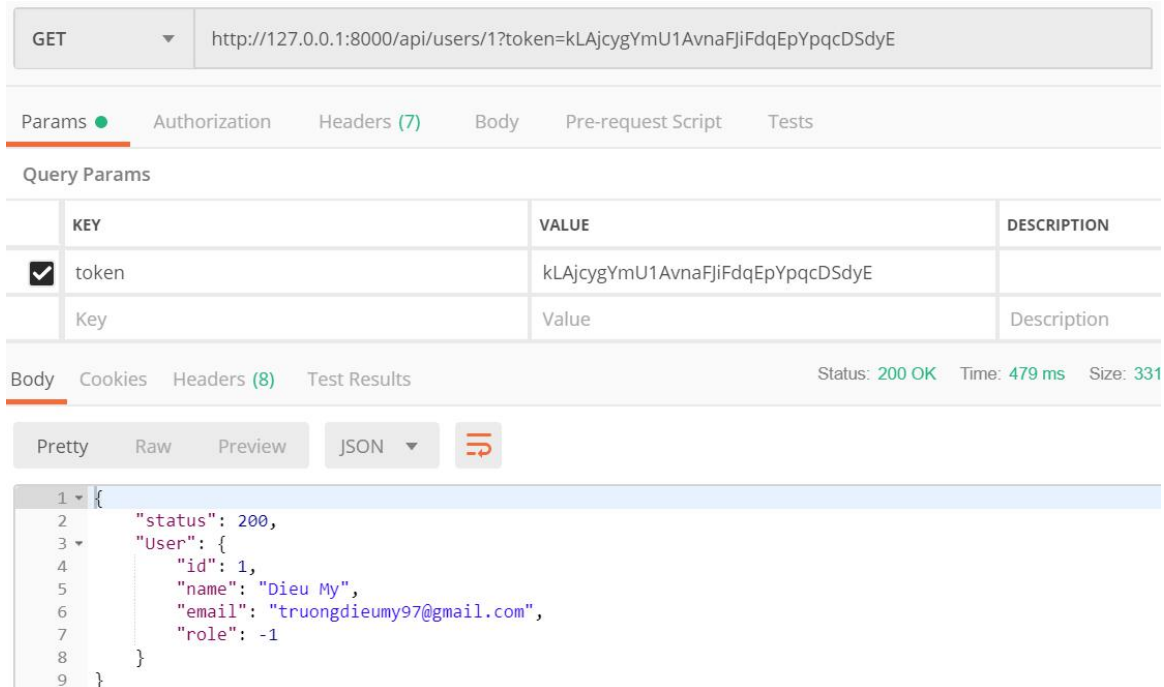
Body Cookies Headers (8) Test Results Status: 401 Unauthorized Time: 429 ms

Pretty Raw Preview JSON

```
1 {
2   "status": 401,
3   "error": "Account has not been verified"
4 }
```

Hình 10. Lấy danh sách khi token không đúng

## Lấy thông tin user theo id

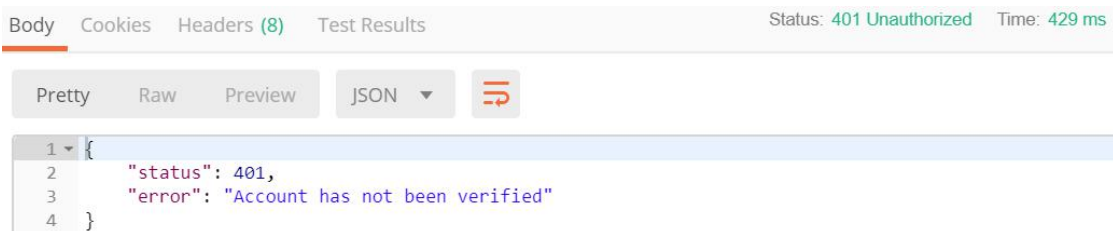


The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:8000/api/users/1?token=kLAjcygYmU1AvnaFjiFdqEpYpqcDSdyE`. The 'Params' tab is active, showing a query parameter 'token' with the value 'kLAjcygYmU1AvnaFjiFdqEpYpqcDSdyE'. The 'Body' tab is also active, displaying the JSON response in 'Pretty' format. The response status is '200 OK' with a time of '479 ms' and a size of '331'.

KEY	VALUE	DESCRIPTION
token	kLAjcygYmU1AvnaFjiFdqEpYpqcDSdyE	
Key	Value	Description

```
{
  "status": 200,
  "User": {
    "id": 1,
    "name": "Dieu My",
    "email": "truongdieumy97@gmail.com",
    "role": -1
  }
}
```

Hình 11. Lấy thông tin user với token đúng

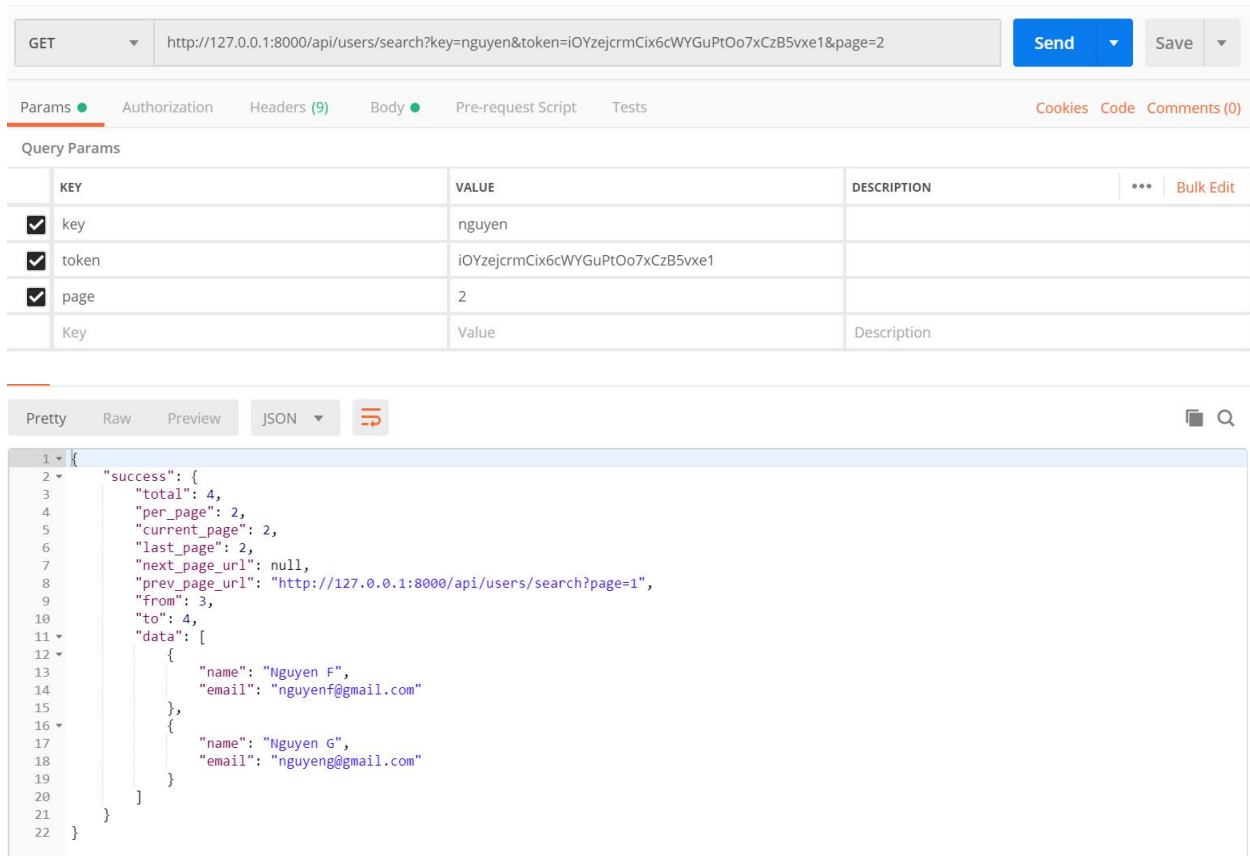


The screenshot shows a REST client interface with a GET request. The 'Body' tab is active, displaying the JSON response in 'Pretty' format. The response status is '401 Unauthorized' with a time of '429 ms'. The JSON response indicates an error: 'Account has not been verified'.

```
{
  "status": 401,
  "error": "Account has not been verified"
}
```

Hình 12. Lấy thông tin user với token sai

## Tìm kiếm user theo name hoặc email



The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:8000/api/users/search?key=nguyen&token=iOYzejcrmCix6cWYGuPtOo7xCzB5vxe1&page=2`. The request parameters are listed in a table below.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> key	nguyen	
<input checked="" type="checkbox"/> token	iOYzejcrmCix6cWYGuPtOo7xCzB5vxe1	
<input checked="" type="checkbox"/> page	2	
Key	Value	Description

The response is shown in JSON format:

```
1 {
2   "success": {
3     "total": 4,
4     "per_page": 2,
5     "current_page": 2,
6     "last_page": 2,
7     "next_page_url": null,
8     "prev_page_url": "http://127.0.0.1:8000/api/users/search?page=1",
9     "from": 3,
10    "to": 4,
11    "data": [
12      {
13        "name": "Nguyen F",
14        "email": "nguyenf@gmail.com"
15      },
16      {
17        "name": "Nguyen G",
18        "email": "nguyeng@gmail.com"
19      }
20    ]
21  }
22 }
```

Hình 13. Tìm kiếm thông tin user theo name hoặc email

## Phân trang user

POST ▼ http://127.0.0.1:8000/api/users/page?token=kLAjcygYmU1AvnaFjiFdqEpYpqcDSdyE&page=1

<input checked="" type="checkbox"/>	token	kLAjcygYmU1AvnaFjiFdqEpYpqcDSdyE	
<input checked="" type="checkbox"/>	page	1	
	Key	Value	D

Body Cookies Headers (8) Test Results Status: 206 Partial Content Time: 4

Pretty Raw Preview JSON ⌵

```

2  "status": 206,
3  "List User": {
4    "current_page": 1,
5    "data": [
6      {
7        "id": 1,
8        "name": "Dieu My",
9        "email": "truongdieumy97@gmail.com",
10       "role": -1
11      },
12      {
13        "id": 2,
14        "name": "chan",
15        "email": "linhchannguyen94@gmail.com",
16        "role": 1
17      },
18      {

```

Hình 14. Phân trang user (trang 1)

POST ▼ http://127.0.0.1:8000/api/users/page?token=kLAjcygYmU1AvnaFjiFdqEpYpqcDSdyE&page=2

<input checked="" type="checkbox"/>	token	kLAjcygYmU1AvnaFjiFdqEpYpqcDSdyE	
<input checked="" type="checkbox"/>	page	2	
	Key	Value	T

Body Cookies Headers (8) Test Results Status: 206 Partial Content Time: 4

Pretty Raw Preview JSON ⌵

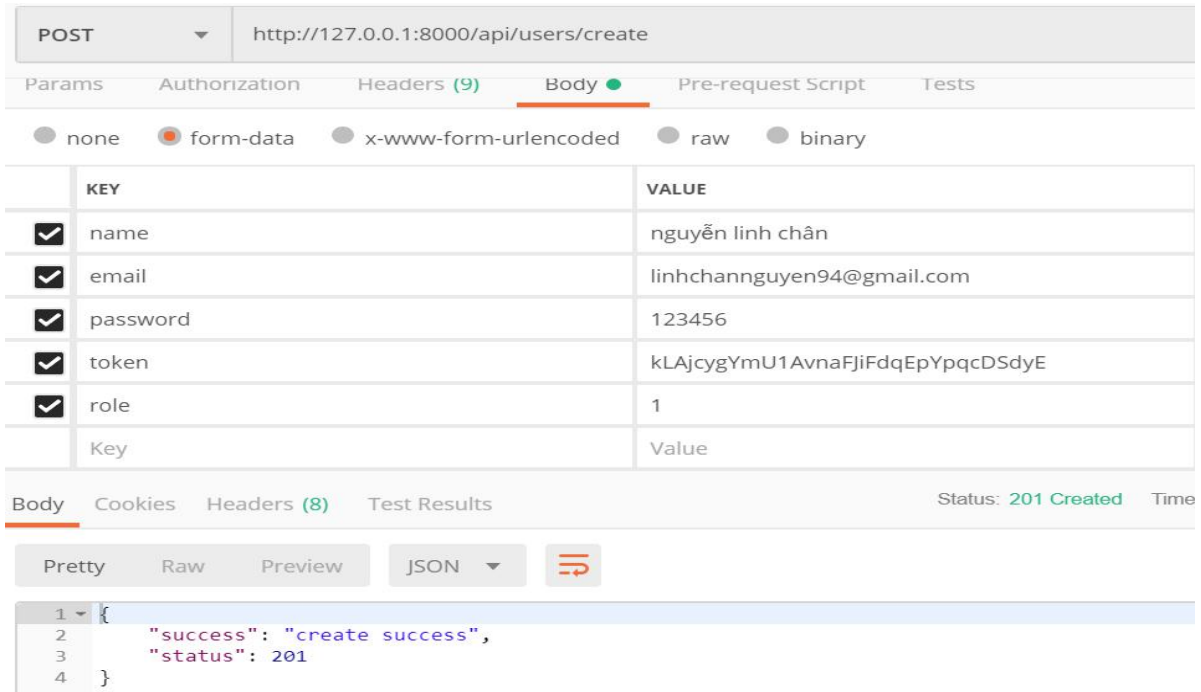
```

2  "status": 206,
3  "List User": {
4    "current_page": 2,
5    "data": [
6      {
7        "id": 6,
8        "name": "Nguyen F",
9        "email": "nguyenf@gmail.com",
10       "role": null
11      },
12      {
13        "id": 7,
14        "name": "Nguyen G",
15        "email": "nguyeng@gmail.com",
16        "role": null
17      },
18      {

```

Hình 15. Phân trang user (trang 2)

## Thêm mới 1 user



POST http://127.0.0.1:8000/api/users/create

Params Authorization Headers (9) **Body** Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary

	KEY	VALUE
<input checked="" type="checkbox"/>	name	nguyễn linh chân
<input checked="" type="checkbox"/>	email	linhchannguyen94@gmail.com
<input checked="" type="checkbox"/>	password	123456
<input checked="" type="checkbox"/>	token	kLAjcygYmU1AvnaFjiFdqEpYpqcDSdyE
<input checked="" type="checkbox"/>	role	1
	Key	Value

Body Cookies Headers (8) Test Results Status: 201 Created Time

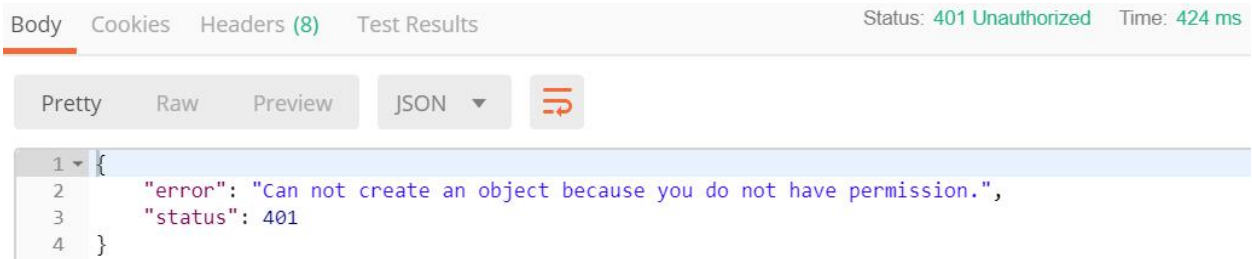
Pretty Raw Preview JSON

```

1 {
2   "success": "create success",
3   "status": 201
4 }
```

Hình 16. Thêm mới 1 user

Nếu bạn không phải là admin, hệ thống sẽ thông báo lỗi và không cho bạn tạo tài khoản:



Body Cookies Headers (8) Test Results Status: 401 Unauthorized Time: 424 ms

Pretty Raw Preview JSON

```

1 {
2   "error": "Can not create an object because you do not have permission.",
3   "status": 401
4 }
```

Hình 17. Lỗi tạo tài khoản do không có quyền

## Cập nhật thông tin user

PUT ▼ http://127.0.0.1:8000/api/users/update/1

Params Authorization Headers (9) **Body** ● Pre-request Script Tests

● none ● form-data ● x-www-form-urlencoded ● raw ● binary

	KEY	VALUE
<input checked="" type="checkbox"/>	name	Truong Dieu My
<input checked="" type="checkbox"/>	token	Hly8y6qmew0jbfGsBjZnKNX5aaM9K4p2
<input checked="" type="checkbox"/>	role	1
	Key	Value

Body Cookies Headers (8) Test Results Status: 200 OK Time

Pretty Raw Preview JSON ≡

```

1 {
2   "success": "Cap nhat user thanh cong"
3 }
```

Hình 18. Cập nhật thông tin user

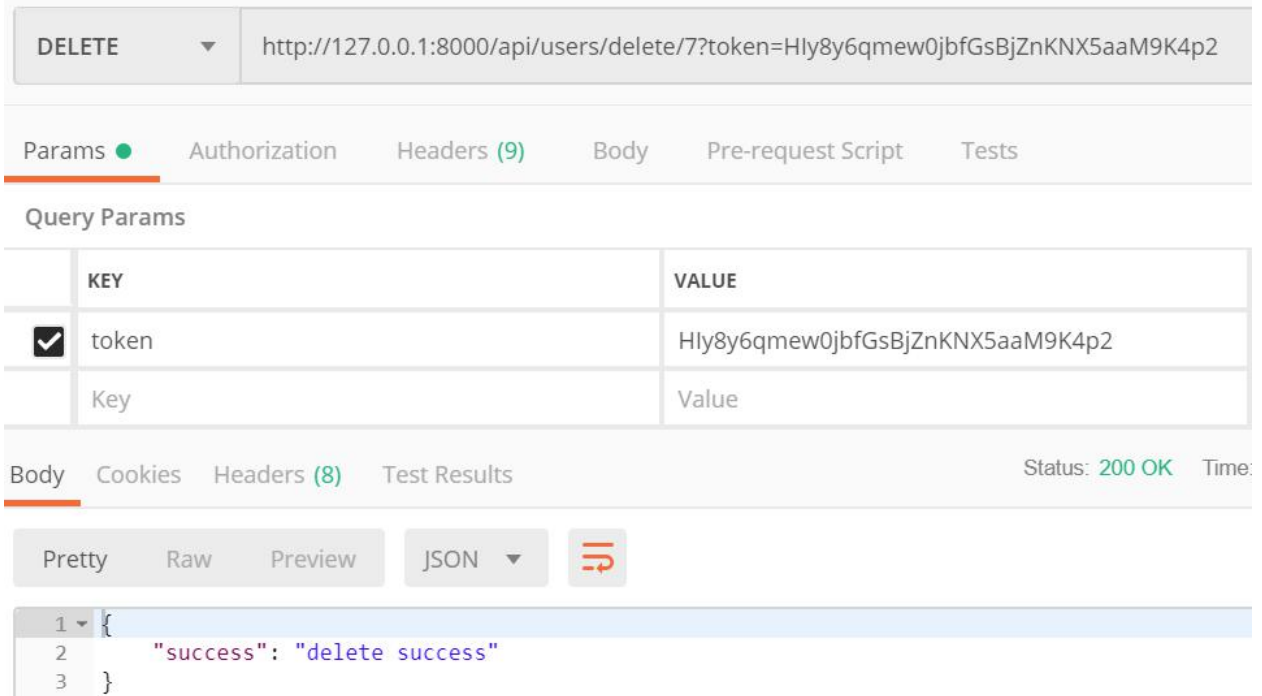
Pretty Raw Preview JSON ▼ ≡

```

1 {
2   "error": "Loi xac thuc nguoi dung"
3 }
```

Hình 19. Lỗi cập nhật do không phải admin

## Xóa user theo id



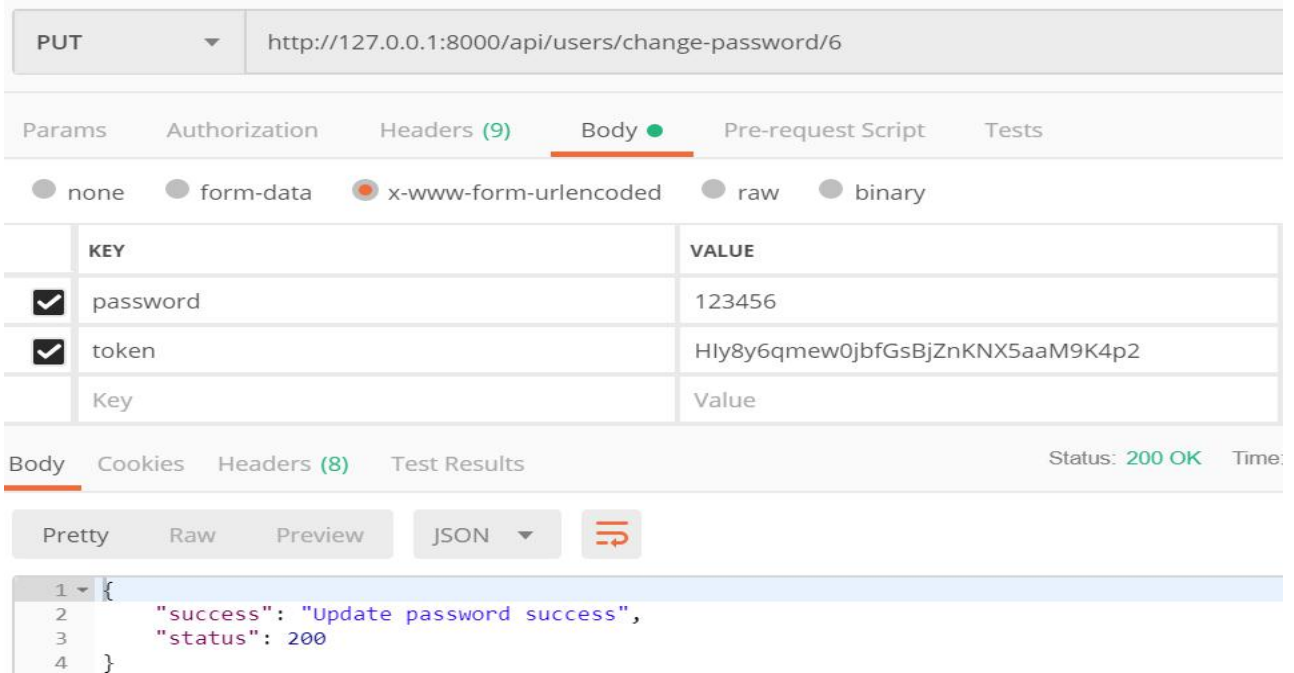
The screenshot shows a REST client interface with a DELETE request to `http://127.0.0.1:8000/api/users/delete/7?token=Hly8y6qmew0jbfGsBjZnKNX5aaM9K4p2`. The 'Params' tab is active, showing a query parameter 'token' with the value 'Hly8y6qmew0jbfGsBjZnKNX5aaM9K4p2'. The 'Body' tab is also visible, showing a JSON response: `{ "success": "delete success" }`. The status is 200 OK.

KEY	VALUE
token	Hly8y6qmew0jbfGsBjZnKNX5aaM9K4p2
Key	Value

```
{
  "success": "delete success"
}
```

Hình 20. Xóa user dựa theo id

## Thay đổi mật khẩu



The screenshot shows a REST client interface with a PUT request to `http://127.0.0.1:8000/api/users/change-password/6`. The 'Body' tab is active, showing a JSON response: `{ "success": "Update password success", "status": 200 }`. The status is 200 OK.

KEY	VALUE
password	123456
token	Hly8y6qmew0jbfGsBjZnKNX5aaM9K4p2
Key	Value

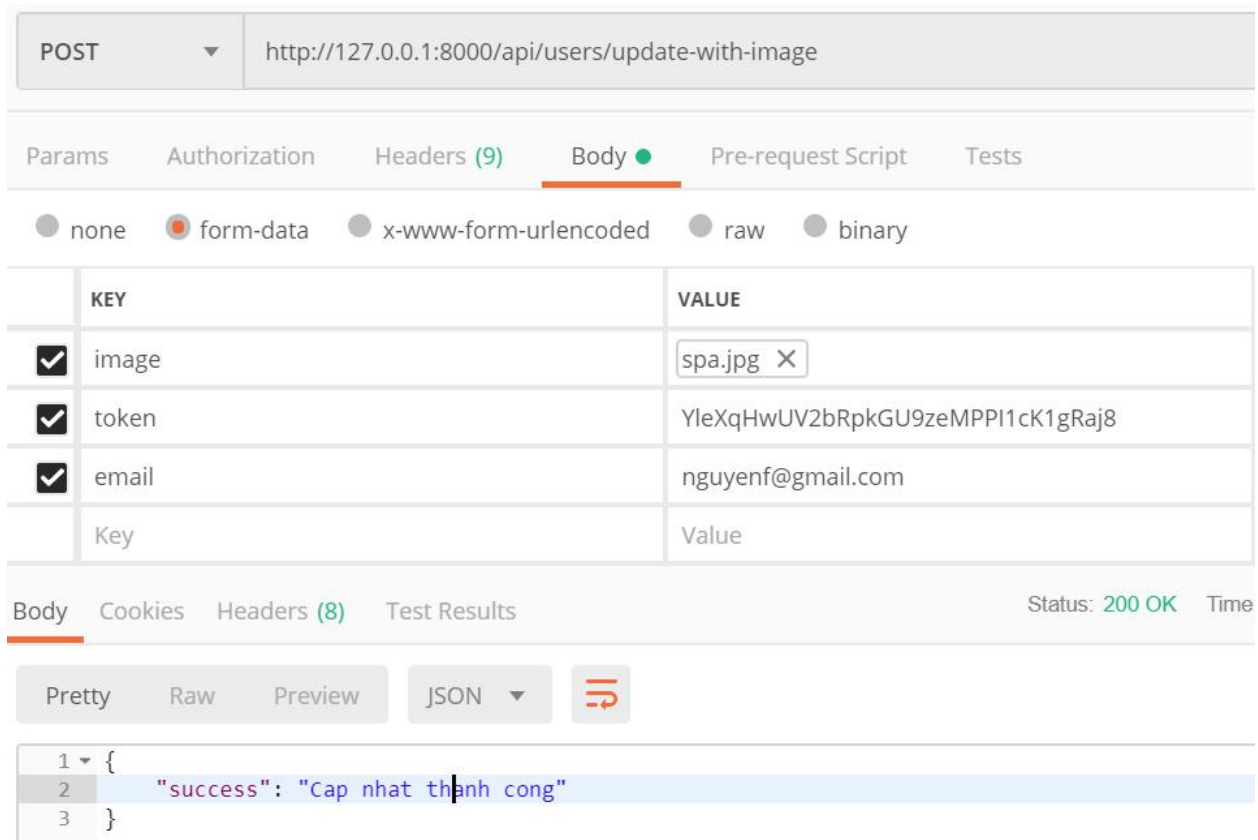
```
{
  "success": "Update password success",
  "status": 200
}
```

Hình 21. Thay đổi mật khẩu



## Cập nhật hình ảnh user

Khi user đăng nhập, dựa vào token của user và email tức là chỉ có user đó mới được quyền thay đổi hình ảnh cá nhân.



The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://127.0.0.1:8000/api/users/update-with-image
- Body Type:** form-data
- Body Data:**

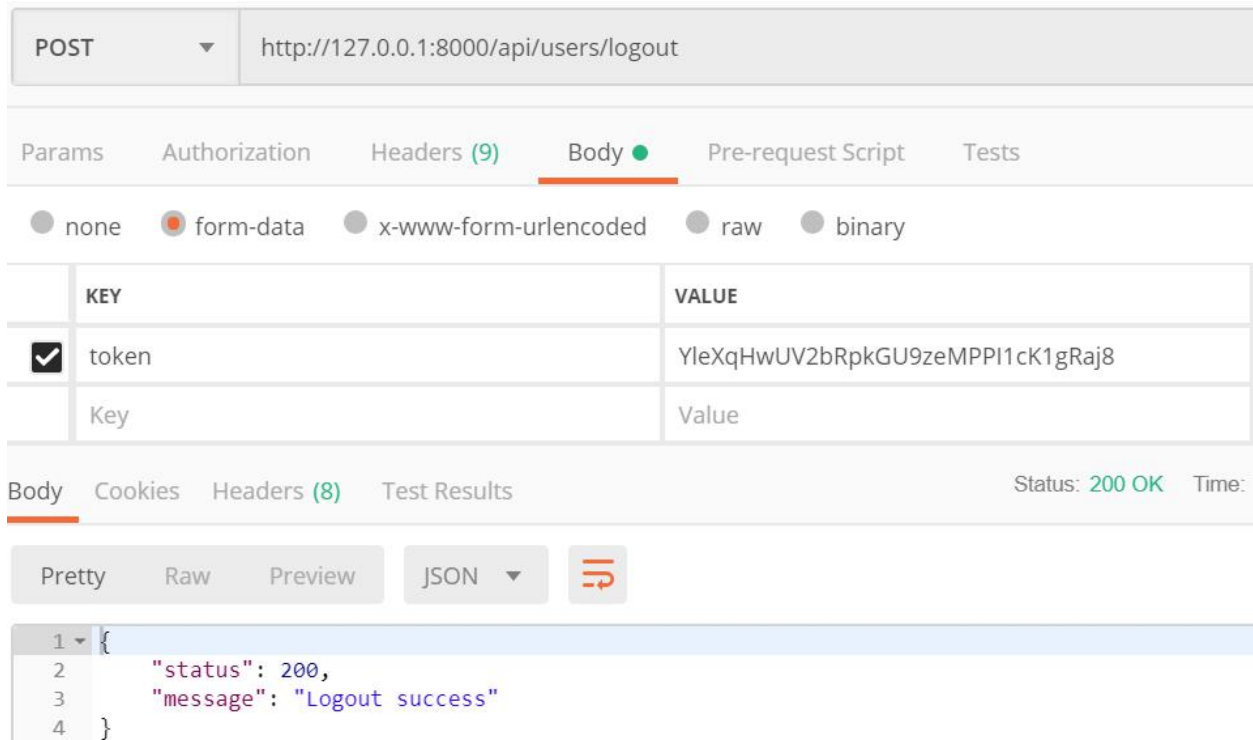
KEY	VALUE
image	spa.jpg
token	YleXqHwUV2bRpkGU9zeMPPI1cK1gRaj8
email	nguyenf@gmail.com
Key	Value
- Status:** 200 OK
- Response Body (JSON):**

```

1 {
2   "success": "Cap nhat thanh cong"
3 }
```

Hình 22. Cập nhật hình ảnh user

## Logout



The screenshot shows a REST client interface with the following details:

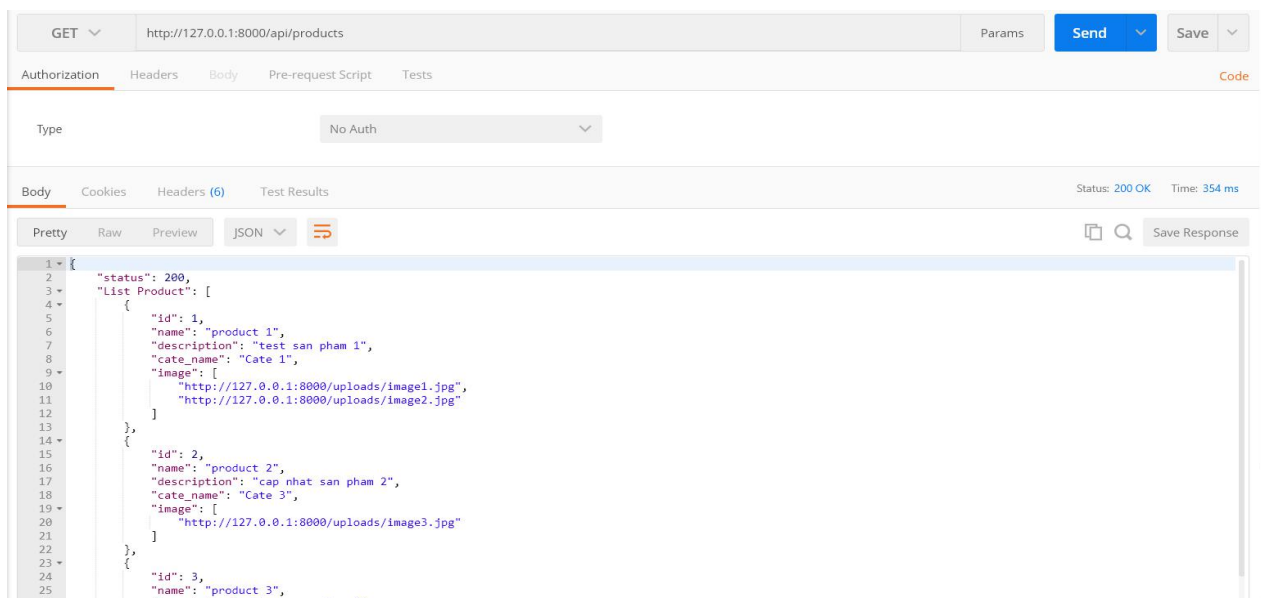
- Method:** POST
- URL:** http://127.0.0.1:8000/api/users/logout
- Body Type:** form-data
- Body Content:**

KEY	VALUE
token	YleXqHwUV2bRpkGU9zeMPPI1cK1gRaj8
Key	Value
- Status:** 200 OK
- Response Body (JSON):**

```
{
  "status": 200,
  "message": "Logout success"
}
```

Hình 23. Logout

## Lấy danh sách tất cả các sản phẩm



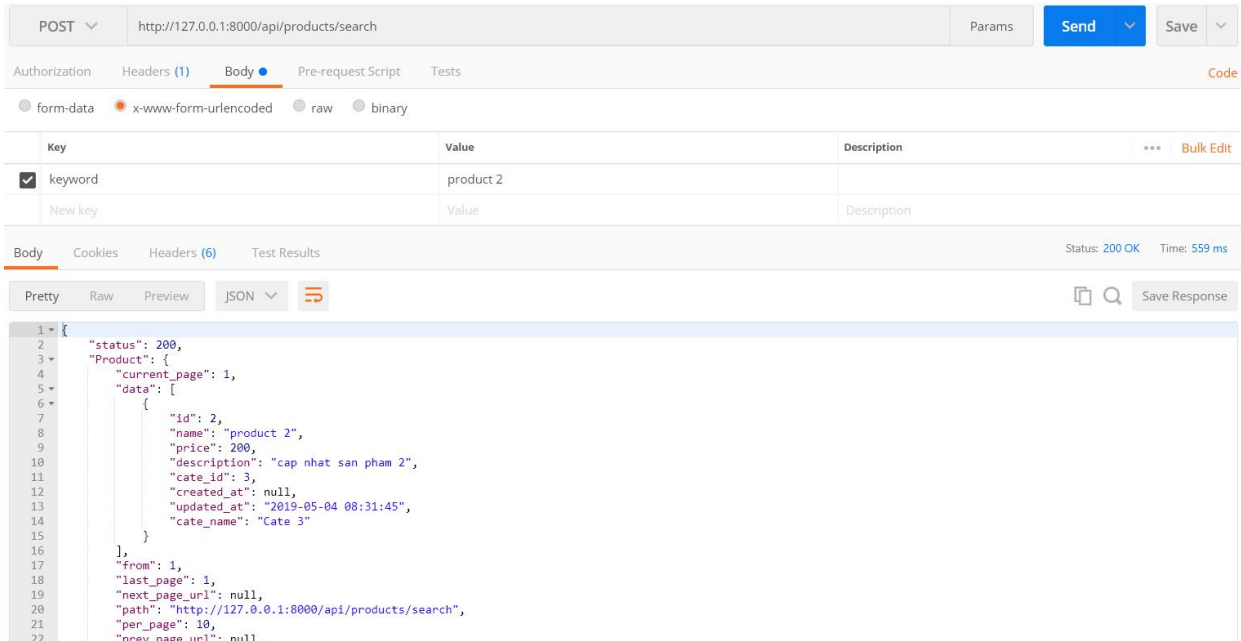
The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://127.0.0.1:8000/api/products
- Body Type:** No Auth
- Status:** 200 OK
- Time:** 354 ms
- Response Body (JSON):**

```
{
  "status": 200,
  "List Product": [
    {
      "id": 1,
      "name": "product 1",
      "description": "test san pham 1",
      "cate_name": "Cate 1",
      "image": [
        "http://127.0.0.1:8000/uploads/image1.jpg",
        "http://127.0.0.1:8000/uploads/image2.jpg"
      ]
    },
    {
      "id": 2,
      "name": "product 2",
      "description": "cap nhat san pham 2",
      "cate_name": "Cate 3",
      "image": [
        "http://127.0.0.1:8000/uploads/image3.jpg"
      ]
    },
    {
      "id": 3,
      "name": "product 3",
      "description": "test san pham 3"
    }
  ]
}
```

Hình 24. Thêm mới 1 product

## Tìm kiếm sản phẩm




POST ▼ http://127.0.0.1:8000/api/products/search Params Send ▼ Save ▼

Authorization Headers (1) Body ● Pre-request Script Tests Code

☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary

Key	Value	Description
<input checked="" type="checkbox"/> keyword	product 2	
New key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 559 ms

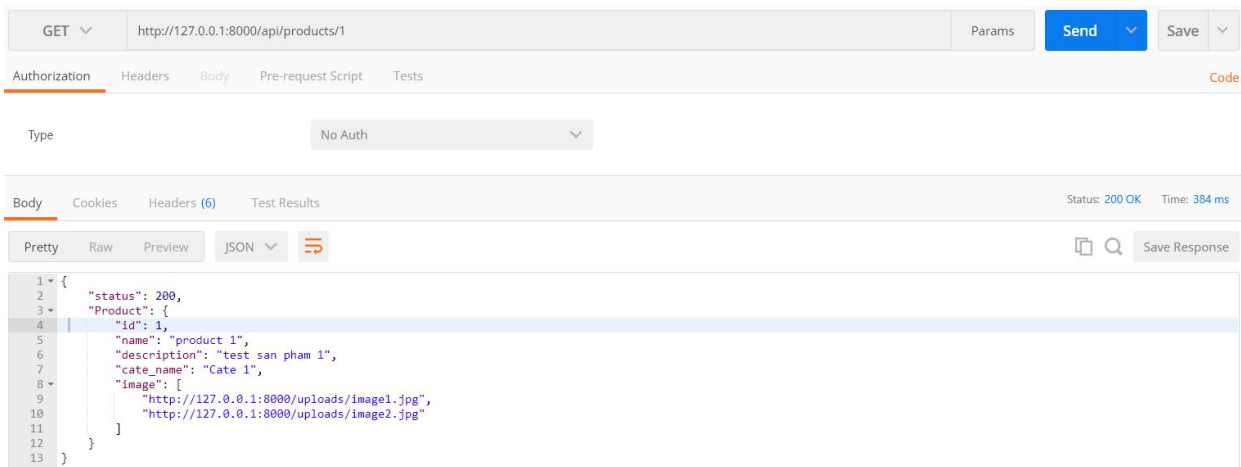
Pretty Raw Preview JSON ▼  Save Response

```

1 {
2   "status": 200,
3   "Product": {
4     "current_page": 1,
5     "data": [
6       {
7         "id": 2,
8         "name": "product 2",
9         "price": 200,
10        "description": "cap nhat san pham 2",
11        "cate_id": 3,
12        "created_at": null,
13        "updated_at": "2019-05-04 08:31:45",
14        "cate_name": "Cate 3"
15      }
16    ],
17    "from": 1,
18    "last_page": 1,
19    "next_page_url": null,
20    "path": "http://127.0.0.1:8000/api/products/search",
21    "per_page": 10,
22    "prev_page_url": null
23  }
24 }
```

Hình 25. Tìm kiếm 1 product

## Lấy sản phẩm theo id




GET ▼ http://127.0.0.1:8000/api/products/1 Params Send ▼ Save ▼

Authorization Headers Body Pre-request Script Tests Code

Type No Auth ▼

Body Cookies Headers (6) Test Results Status: 200 OK Time: 384 ms

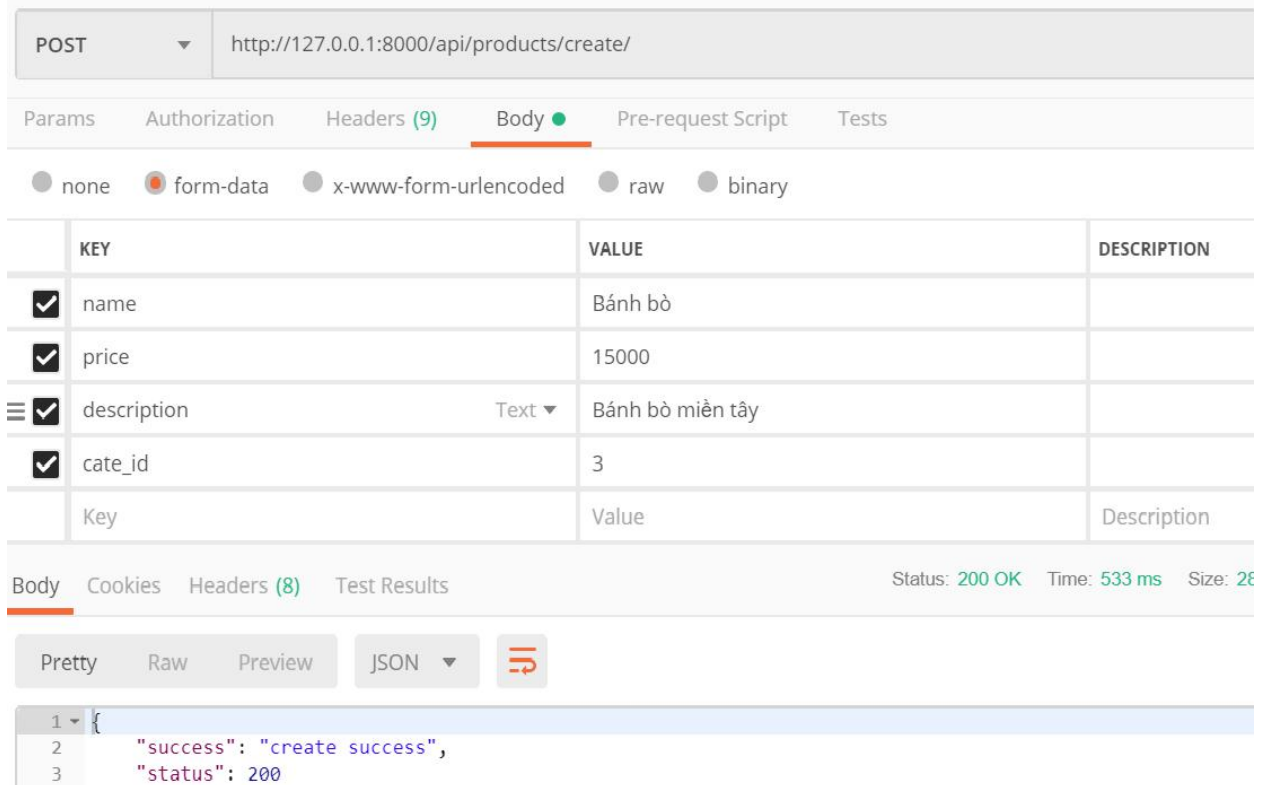
Pretty Raw Preview JSON ▼  Save Response

```

1 {
2   "status": 200,
3   "Product": {
4     "id": 1,
5     "name": "product 1",
6     "description": "test san pham 1",
7     "cate_name": "Cate 1",
8     "image": [
9       "http://127.0.0.1:8000/uploads/image1.jpg",
10      "http://127.0.0.1:8000/uploads/image2.jpg"
11    ]
12   }
13 }
```

Hình 26. Lấy 1 product theo id

## Thêm sản phẩm vào danh sách



POST http://127.0.0.1:8000/api/products/create/

Params Authorization Headers (9) **Body** Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	name	Bánh bò	
<input checked="" type="checkbox"/>	price	15000	
<input checked="" type="checkbox"/>	description	Text ▼ Bánh bò miền tây	
<input checked="" type="checkbox"/>	cate_id	3	
	Key	Value	Description

Body Cookies Headers (8) Test Results Status: 200 OK Time: 533 ms Size: 28

Pretty Raw Preview JSON ▼

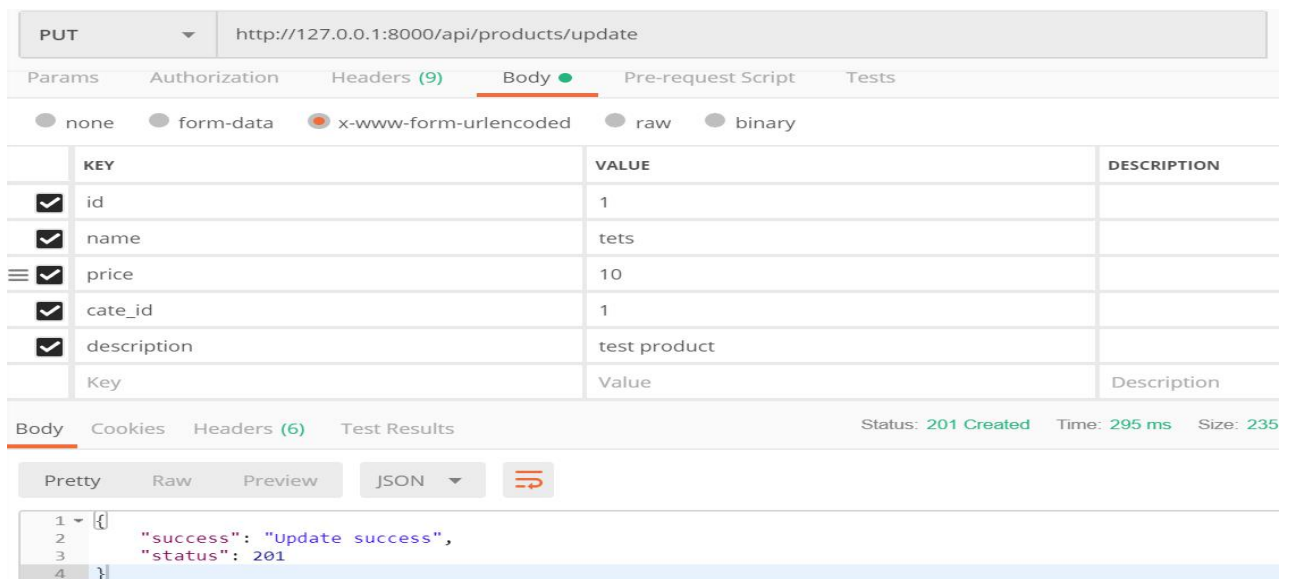
```

1 {
2   "success": "create success",
3   "status": 200

```

Hình 27. Thêm 1 product

## Sửa thông tin sản phẩm



PUT http://127.0.0.1:8000/api/products/update

Params Authorization Headers (9) **Body** Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	id	1	
<input checked="" type="checkbox"/>	name	tets	
<input checked="" type="checkbox"/>	price	10	
<input checked="" type="checkbox"/>	cate_id	1	
<input checked="" type="checkbox"/>	description	test product	
	Key	Value	Description

Body Cookies Headers (6) Test Results Status: 201 Created Time: 295 ms Size: 235

Pretty Raw Preview JSON ▼

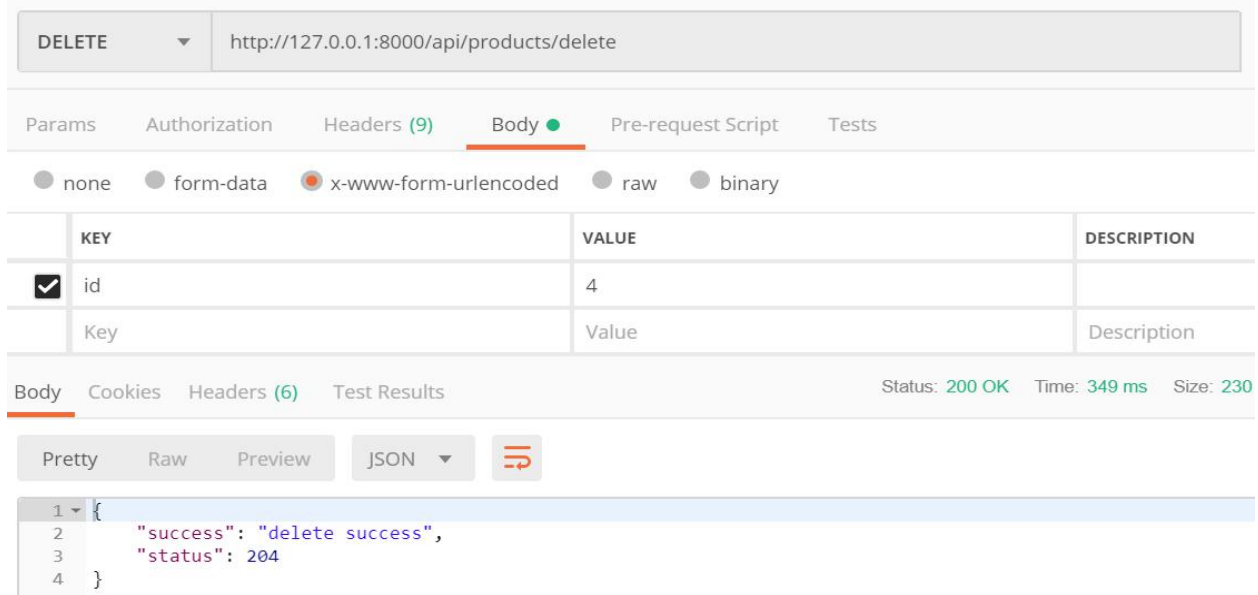
```

1 {
2   "success": "Update success",
3   "status": 201
4 }

```

Hình 28. Cập nhật 1 product theo id

## Xóa sản phẩm khỏi danh sách



The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** http://127.0.0.1:8000/api/products/delete
- Params:** None
- Authorization:** None
- Headers:** 9 headers (not shown)
- Body:** Selected, with format **x-www-form-urlencoded**. The body contains a table with the following data:

KEY	VALUE	DESCRIPTION
id	4	
Key	Value	Description
- Pre-request Script:** None
- Tests:** None
- Status:** 200 OK
- Time:** 349 ms
- Size:** 230
- Body:** Pretty, Raw, Preview tabs. The JSON body is: 

```
{  "success": "delete success",  "status": 204}
```

Hình 29. Xóa 1 product theo id

## | HƯỚNG DẪN TẠO 1 PACKAGE TRONG LARAVEL

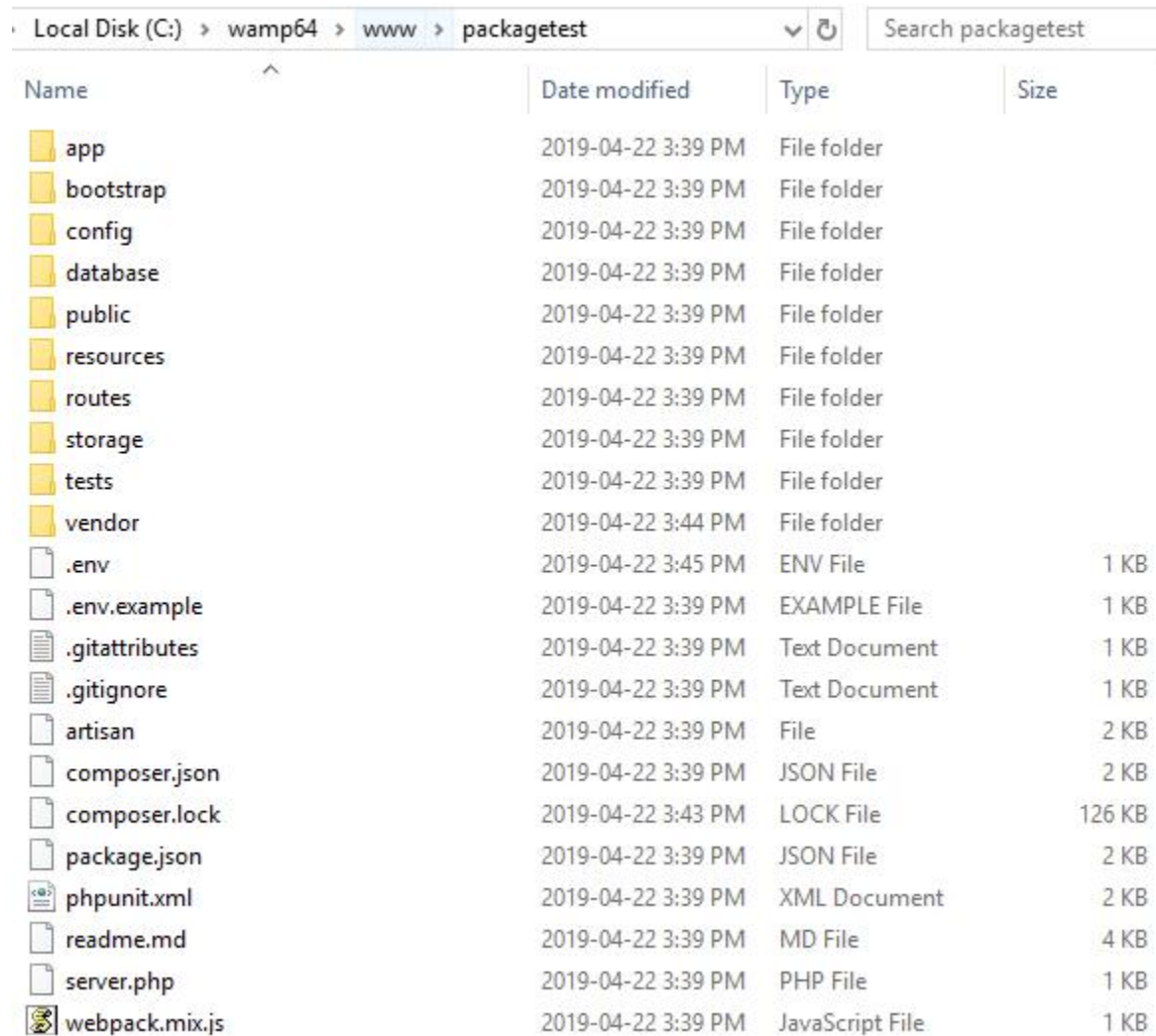
### CÁCH 1:

Ở cách này, chúng ta sẽ tạo 1 package chạy trên nền Laravel để giúp chúng ta hiểu rõ về cách sử dụng những thư viện hỗ trợ sẵn trong Laravel, đồng thời giúp cho các bạn có thể tiếp cận với Laravel dễ dàng cài đặt và sử dụng hơn sau khi tạo package bằng cách 2 (nêu sau).

Để bắt đầu, chúng ta cần 1 source Laravel, ở đây nhóm sử dụng Laravel phiên bản 5.4. Chúng ta khởi tạo 1 source Laravel bằng cách sử dụng **Composer** (phải cài đặt composer mới có thể sử dụng được lệnh command composer). Có nhiều cú pháp khởi tạo 1 source mới, có thể tham khảo tại <https://laravel.com/docs/5.6/installation>.

Sau khi cài đặt thành công **Composer**, chúng ta sẽ vào thư mục www trong Wamp/Xamp/Ampmp mở lệnh command và gõ dòng lệnh:

```
composer create-project --prefer-dist laravel/laravel packagetest
```



Name	Date modified	Type	Size
app	2019-04-22 3:39 PM	File folder	
bootstrap	2019-04-22 3:39 PM	File folder	
config	2019-04-22 3:39 PM	File folder	
database	2019-04-22 3:39 PM	File folder	
public	2019-04-22 3:39 PM	File folder	
resources	2019-04-22 3:39 PM	File folder	
routes	2019-04-22 3:39 PM	File folder	
storage	2019-04-22 3:39 PM	File folder	
tests	2019-04-22 3:39 PM	File folder	
vendor	2019-04-22 3:44 PM	File folder	
.env	2019-04-22 3:45 PM	ENV File	1 KB
.env.example	2019-04-22 3:39 PM	EXAMPLE File	1 KB
.gitattributes	2019-04-22 3:39 PM	Text Document	1 KB
.gitignore	2019-04-22 3:39 PM	Text Document	1 KB
artisan	2019-04-22 3:39 PM	File	2 KB
composer.json	2019-04-22 3:39 PM	JSON File	2 KB
composer.lock	2019-04-22 3:43 PM	LOCK File	126 KB
package.json	2019-04-22 3:39 PM	JSON File	2 KB
phpunit.xml	2019-04-22 3:39 PM	XML Document	2 KB
readme.md	2019-04-22 3:39 PM	MD File	4 KB
server.php	2019-04-22 3:39 PM	PHP File	1 KB
webpack.mix.js	2019-04-22 3:39 PM	JavaScript File	1 KB

Hình 30. Cài đặt thành công

Khi hoàn tất, chúng ta cần cấu hình file .env, nếu sau khi hoàn tất không tìm thấy file .env, gõ lệnh:

```
cp .env.example .env
```

Cấu hình file .env như sau:



```
.env
1 APP_ENV=local
2 APP_KEY=base64:BnwGEJeg4D0pKac1cJcmioXX5WUD9ukM9MkM3jtXf3w=
3 APP_DEBUG=true
4 APP_LOG_LEVEL=debug
5 APP_URL=http://localhost
6
7 DB_CONNECTION=mysql
8 DB_HOST=127.0.0.1
9 DB_PORT=3306
10 DB_DATABASE=spa
11 DB_USERNAME=root
12 DB_PASSWORD=''
13
14 BROADCAST_DRIVER=log
15 CACHE_DRIVER=file
16 SESSION_DRIVER=file
17 QUEUE_DRIVER=sync
18
19 REDIS_HOST=127.0.0.1
20 REDIS_PASSWORD=null
21 REDIS_PORT=6379
```

Hình 31. Cấu hình file .env

Cần tạo 1 database trên MySQL với tên packagetest để kết nối. Sau đó chạy câu lệnh sau để kiểm tra có kết nối không:

```
php artisan serve
```

Truy cập vào url để kiểm tra:

```
PS C:\wamp64\www\spa> php artisan serve
Laravel development server started: <http://127.0.0.1:8000>
■
```

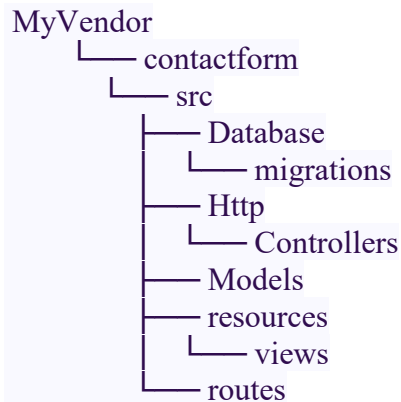
Hình 32. Start server



Tiếp theo, chúng ta sẽ tạo thư mục sau:

`packages/MyVendor/contactform/`

Sau đó, tạo các file và folder như sau sẽ tạo nên 1 package:



Tại thư mục packages, click chuột phải chọn Open in Terminal hoặc mở command và gõ lệnh:

`composer init`

Sau đó, nó sẽ hỏi một số câu hỏi để điền vào file `composer.json`, nhấn enter cho đến khi khởi tạo xong file `composer.json` và sau đó muốn thay đổi thì cứ vào file `composer.json` thay đổi. Khi hoàn tất, file sẽ như sau:

```
{
  "name": "MyVendor/Contactform",
  "description": "A contact form package for laravel",
  "authors": [{
    "name": "samuel ogundipe",
    "email": "email@email.com"
  }],
  "require": {}
}
```

Hình 33. Cấu hình file `composer.json`

Thêm đoạn code sau để chúng tự động tải các tập tin:

```
{
    "name": "MyVendor/Contactform",
    "description": "A contact form package for laravel",
    "authors": [{
        "name": "samuel ogundipe",
        "email": "email@email.com"
    }],
    "require": {},
    "autoload": {
        "psr-4": {
            "MyVendor\\Contactform\\": "src/"
        }
    }
}
```

Hình 34. Cấu hình file composer.json

Thêm các file vào package. Đầu tiên, cần xác định một nhà cung cấp dịch vụ cho package. Nhà cung cấp dịch vụ là những gì mà Laravel sử dụng để xác định các file sẽ được tải và truy cập bởi package của bạn.

Trong thư mục `src/` tạo file `ContactFormServiceProvider.php`, sau đó thêm đoạn code như trong hình để package của bạn được gọi ở mọi nơi:

```
1  <?php
2  namespace MyVendor\Contactform;
3  use Illuminate\Support\ServiceProvider;
4
5  class ContactFormServiceProvider extends ServiceProvider {
6      public function boot() {
7      }
8      public function register(){}
9  }
```

Hình 35. Cấu hình service provider

Để Laravel biết cách tải package và sử dụng các chức năng, chúng ta cần phải thêm đoạn code ở trong source gốc tại file composer.json

```
"autoload": {
    "classmap": [
        "database/seeds",
        "database/factories"
    ],
    "psr-4": {
        "MyVendor\\Contactform\\": "packages/MyVendor/contactform/src",
        "App\\": "app/"
    }
},
"autoload-dev": {
    "psr-4": {
        "MyVendor\\Contactform\\": "packages/MyVendor/contactform/src",
        "Tests\\": "tests/"
    }
},
```

Hình 36. Cấu hình file composer.json

Bây giờ hãy kiểm tra và xem package có được tải đúng không. Bên trong phương thức boot() của ContactFormServiceProvider.php, hãy thêm một đường dẫn và tải nó:

```
// MyVendor\\contactform\\src\\ContactFormServiceProvider.php
$this->loadRoutesFrom(__DIR__.'/routes/web.php');
```

Hình 37. Tự động load file web.php

`__DIR__` truy cập trực tiếp đến thư mục hiện tại nơi chứa file.

`routes/web.php` truy cập trực tiếp đến thư mục routes của package mà bạn tạo, nó sẽ sử dụng routes nằm trong thư mục src của package mà bạn tạo, không phải các routes Laravel mặc định.

Tiếp theo, tạo file `web.php` nằm trong thư mục routes của package và thêm đoạn code sau:

```
<?php
// MyVendor\contactform\src\routes\web.php
Route::get('contact', function(){
    return 'Packagist';
});
?>
```

Hình 38. Trả về view với nội dung Packagist

Tiếp theo, chúng ta cần thêm nhà cung cấp dịch vụ mới trong file `config/app.php` nằm trong source gốc, nó nằm bên trong array `'providers'`:

```
// config/app.php
'providers' => [
    ...,
    App\Providers\RouteServiceProvider::class,
    // Our new package class
    MyVendor\Contactform\ContactFormServiceProvider::class,
],
```

Hình 39. Gọi provider để chạy package

Sau đó, start server bằng lệnh sau và truy cập url: `http://127.0.0.1:8000//contact` để kiểm tra:

`php artisan serve`

Kết quả như  
sau:



Hình 40. Package đã được chạy trên nền Laravel

Đến đây, bạn đã biết được package của bạn đã tạo đúng cách. Bây giờ hãy tạo view cho trang contact và để Laravel biết cách tải trang đó như thế nào. Trong phương thức boot(), hãy thêm:

```
// MyVendor\contactform\src\ContactFormServiceProvider.php
$this->loadViewsFrom(__DIR__.'resources/views', 'contactform');
```

`resources/views` để truy cập trực tiếp đến thư mục resources mà bạn đã tạo cho package chứ không phải thư mục resources Laravel mặc định.

Để phân biệt giữa views mặc định của Laravel và views của package bạn tạo, chúng ta phải thêm một tham số phụ vào hàm `loadviewsfrom()` và tham số phụ đó phải là tên của package của bạn đó là `contactform`. Vì vậy, bất cứ khi nào Laravel muốn tải view trong package của bạn, nó phải sử dụng cú pháp `packagename::view`.

Tiếp theo chúng ta sẽ thêm đoạn code HTML cho trang contact. Tại thư mục `resources/views` của package, tạo file `contact.blade.php` và thêm đoạn code sau:

Bây giờ, chúng ta cần phải thay đổi đường dẫn để dẫn đến trực tiếp trang mà bạn thiết kế. Sửa lại đoạn code trong file `web.php` như sau:

```
// MyVendor\contactform\src\routes\web.php
Route::get('contact', function(){
    return view('contactform::contact');
});

Route::post('contact', function(){
    // logic goes here
})->name('contact');
```

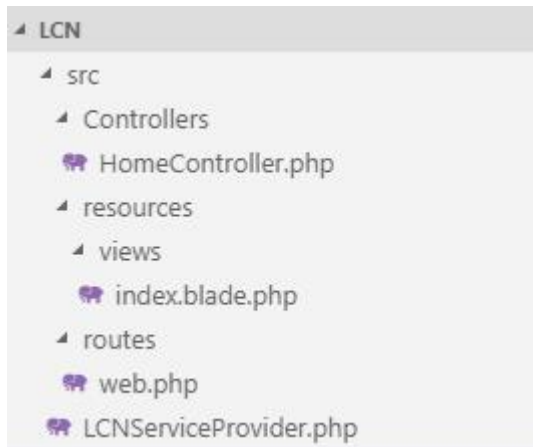
Hình 41. Đường dẫn route hiển thị view

Sau đó, refresh lại trang và xem kết quả. Để quản lý package của bạn, bạn cần phải lưu giữ nó bằng git hoặc ứng dụng nào đó để tiện cho việc cập nhật và phát triển thêm các tính năng cho package đó.

## CÁCH 2:

Ở cách 2 này, chúng ta sẽ tạo package không nằm trong source Laravel.

Đầu tiên, các bạn tạo 1 folder với tên package tùy ý và chứa các thành phần như sau:

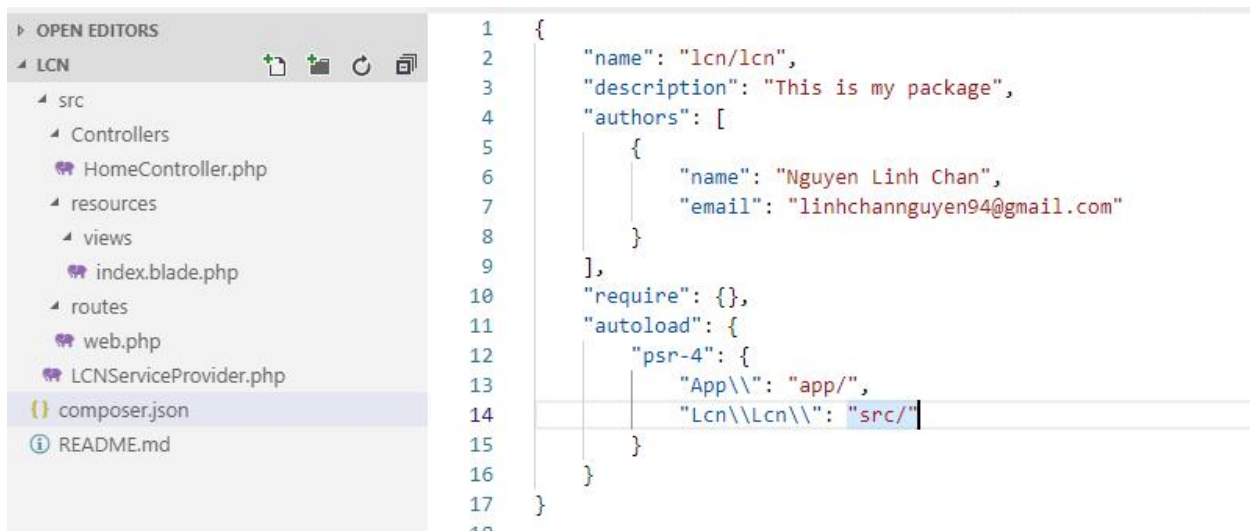


Hình 42. Cấu trúc package

Sau đó, mở Terminal lên và khởi tạo file composer.json như cách 1:

`composer init`

Sau đó nhấn enter để cài mặc định file composer.json. Khi hoàn tất, hãy cấu hình file composer.json như sau:



Hình 43 Cấu hình file composer.json

Các bạn chú ý vào chỗ “psr-4”... “Lcn\Lcn\:: “src/”

Để package của bạn có thể chạy trên nền Laravel thì ServiceProvider là thứ chúng ta cần quan tâm. Vì vậy, chúng ta hãy bắt đầu từ file `LCNServiceProvider.php`. Hãy thêm đoạn code sau:

```
<?php

namespace Lcn\Lcn;

use Illuminate\Support\ServiceProvider;

class LCNServiceProvider extends ServiceProvider {
    /**
     * Bootstrap the application services.
     *
     * @return void
     */
    public function boot()
    {
        $this->loadRoutesFrom(__DIR__.'/routes/web.php');
        $this->loadViewsFrom(__DIR__.'/resources/views', 'lcn');
    }

    /**
     * Register the application services.
     *
     * @return void
     */
    public function register()
    {
    }
}
```

Hình 44. Cấu hình service provider

Các bạn nên chú ý đến tham số phụ ‘lcn’ vì nó sẽ là điểm dễ xảy ra lỗi khi chúng ta gọi đến view (phần này mình sẽ nói sau)

Chúng ta bắt đầu tập trung vào file `web.php` nằm trong thư mục routes. Đây cũng giống như trạm thu phí trên đường cao tốc, nếu bạn không trả tiền thì sẽ không được phép đi vào đường cao tốc. Chúng ta cần biết được hướng đi là đâu và bắt đầu từ đâu. Hãy thêm đoạn code sau và mình sẽ cùng tìm hiểu:



```
<?php
Route::group(['namespace'=>'Lcn\Lcn\Controllers'],function(){
    Route::get('/', 'HomeController@index');
});
```

Hình 45. Thiết lập đường dẫn route

‘/’ tham số đầu tiên truyền vào vd: /hello, /ahihi... là tham số mình sẽ truyền trên url, tùy vào tham số đó có công dụng gì, thường thì nó là động từ để thể hiện được url đó muốn gì.

‘HomeController@index’ tham số thứ 2 là để biết được nó sẽ vào Controller nào thực hiện yêu cầu của hành động (/hello, /ahihi). @index là function mà url đó muốn gọi đến để thực hiện nhiệm vụ. Mọi hành động đều nằm trong function này.

‘namespace’ => ‘Lcn\Lcn\Controllers’ namespace trong Laravel giống như PHP được chỉ định với một nhóm Controller, ở đây namespace chỉ định HomeController nằm trong thư mục Controllers.

Tiếp theo, hãy xem bên trong nó có gì nhé:

```
<?php

namespace Lcn\Lcn\Controllers;
use App\Http\Controllers\Controller;

class HomeController extends Controller
{
    public function index()
    {
        return view('lcn::index');
    }
}
```

Hình 46. Controller

Các bạn hãy nhớ đến lưu ý mình nói ở trên, tham số phụ ‘lcn’ nếu các bạn không thêm nó trước view mà bạn muốn thấy thì nó sẽ bị lỗi. Vì vậy chúng phải được thêm trước view mà bạn muốn chuyển đến.

Cuối cùng, là xem nó muốn hiển thị ra nhưng gì cho chúng ta. Đó là tùy vào các bạn, ở đây mình chỉ hiển thị ra đoạn text để thông báo rằng nó đã được gọi và trả về 1 trang với nội dung text sau:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  Welcom to my Package!!!
</body>
</html>
```

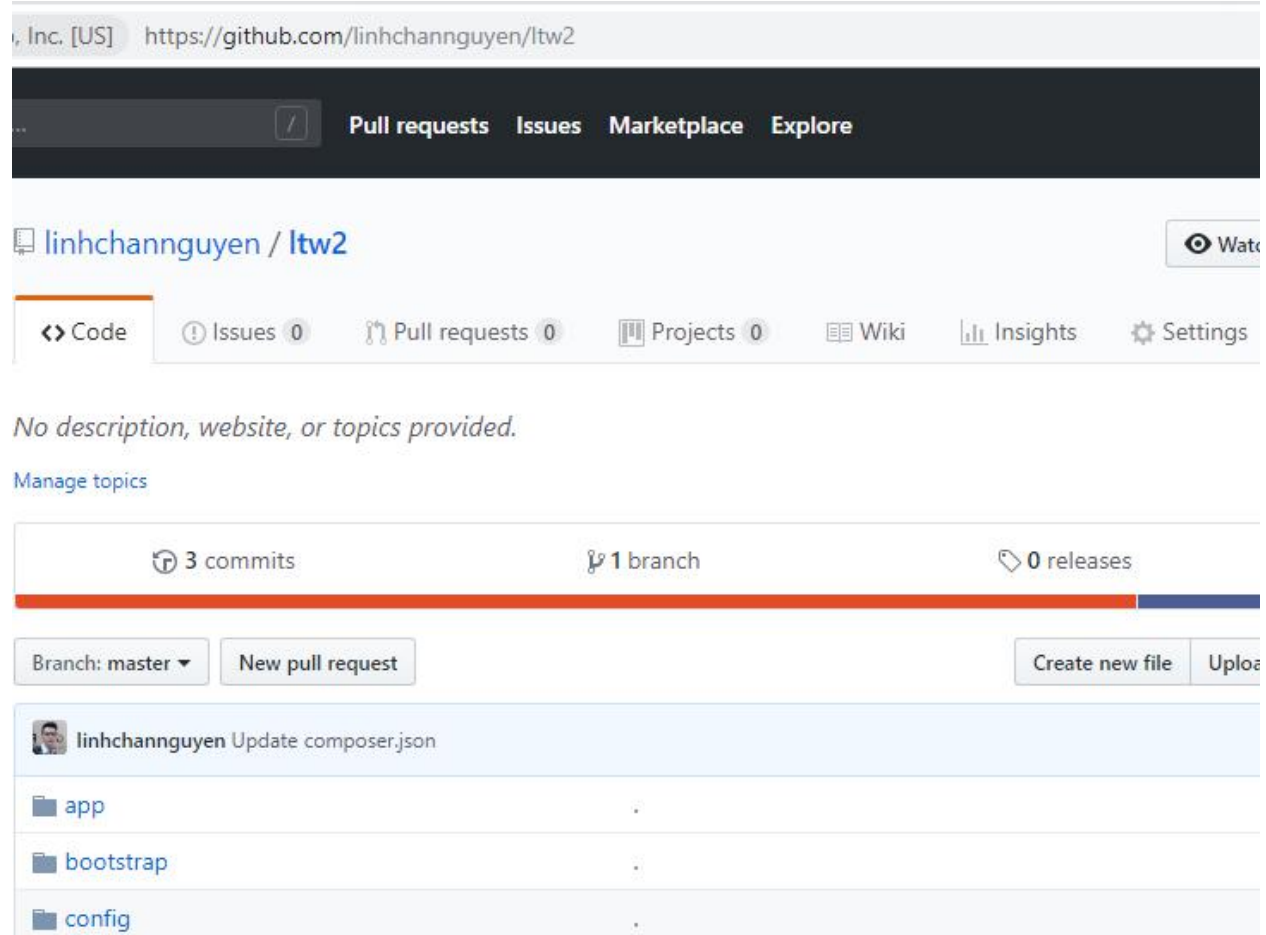
*Hình 47. View*

Như vậy chúng ta đã tạo thành công 1 package và tiếp theo chúng ta sẽ đưa package lên packagist và sau khi thành công, thử cài đặt package này và chạy thử nghiệm. Để tiến hành, chúng ta sẽ tiếp tục phần sau.

## | HƯỚNG DẪN PUBLISH 1 PACKAGE LÊN PACKAGIST

Sau khi package hoàn tất, chúng ta cần đưa source code lên git để quản lý.

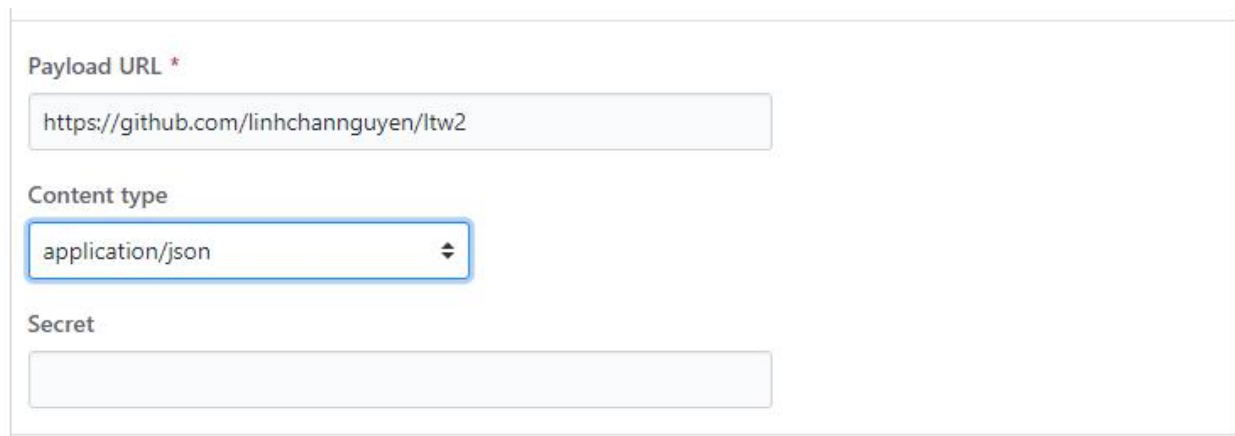
Để publish 1 package lên packagist, hãy vào repositories của package đó:



Hình 48. Github

Lưu ý url: copy lại

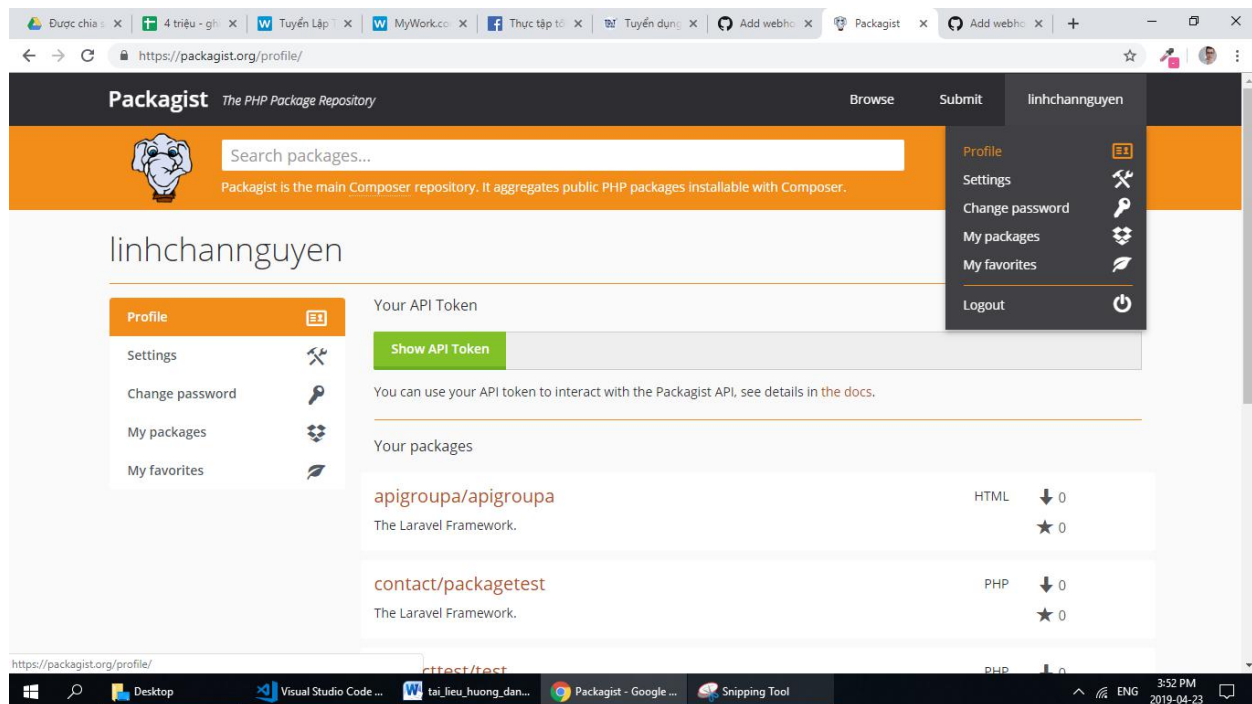
Chọn Settings => Webhooks, sau đó chọn Add webhook



Hình 49. Cấu hình Webhooks

Dán url đã copy vào Payload URL

Tại field Secret, lúc này các bạn hãy vào packagist => chọn Profile



Hình 50. Tìm token

Click vào Show API Token nó sẽ hiển thị ra token:



Hình 51. Token

Copy token và dán vào field Secret, sau đó Click

Add webhook

Tiếp theo, mở packagist và chọn Submit:

Repository URL (Git/Svn/Hg)

e.g.: <https://github.com/composer/composer>

Check

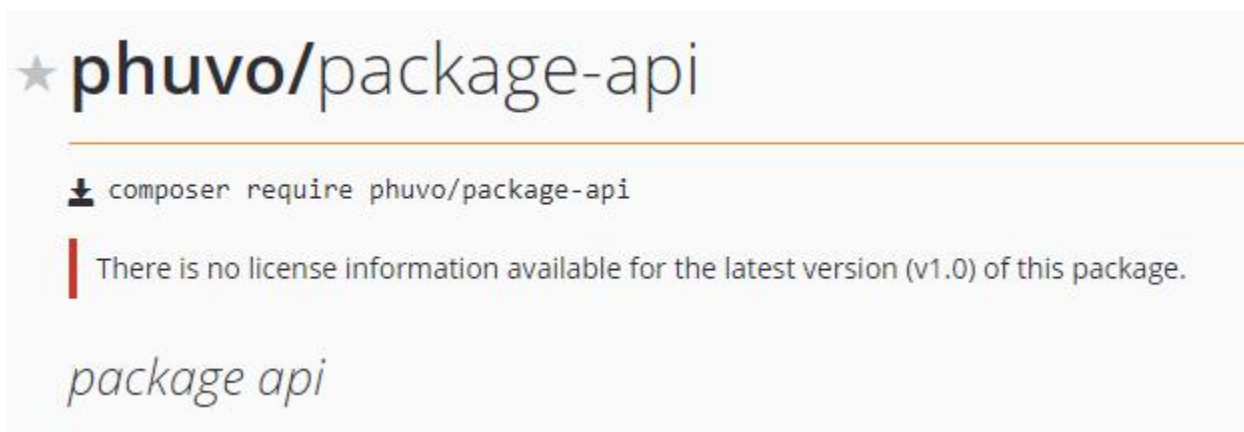
Hình 52. Kiểm tra package có tồn tại

Dán đường dẫn của repositories vào và Check, nếu như sau khi kiểm tra và thông báo tên package đã tồn tại, các bạn hãy vào file composer.json để thay đổi name để không bị trùng:

```
composer.json ×
1  {
2      "name": "laravel/laravel",
3      "description": "The Laravel Framework.",
4      "keywords": ["framework", "laravel"],
5      "license": "MIT",
6      "type": "project",
7      "require": {
```

Hình 53. Thay đổi tên package

Nếu thành công, sẽ có kết quả như sau:

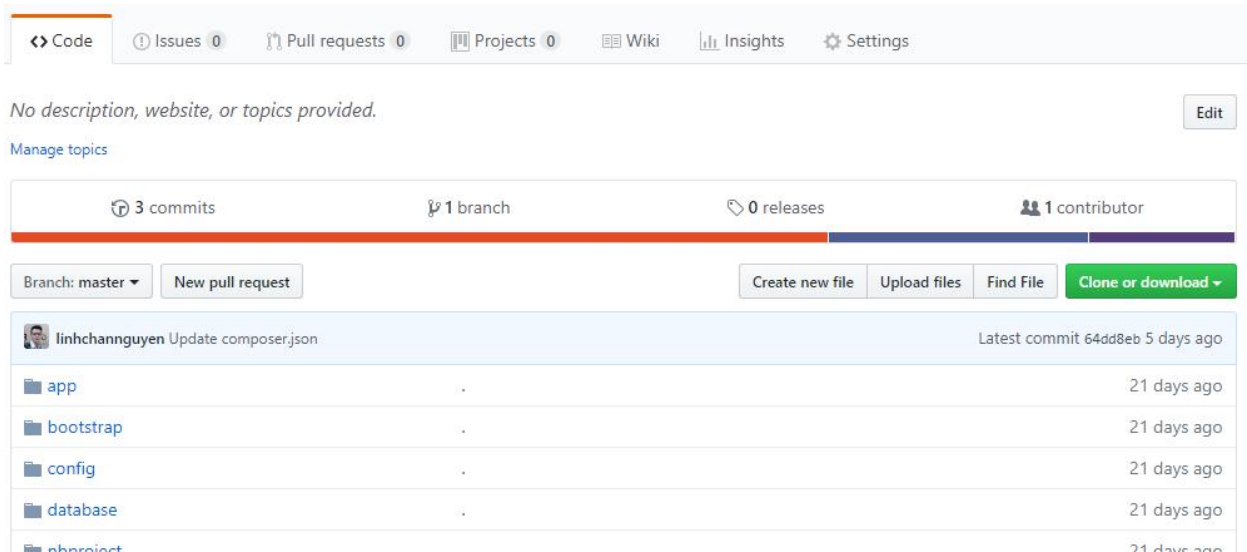


Hình 54. Tạo package thành công

## | HƯỚNG DẪN SỬ DỤNG PACKAGE

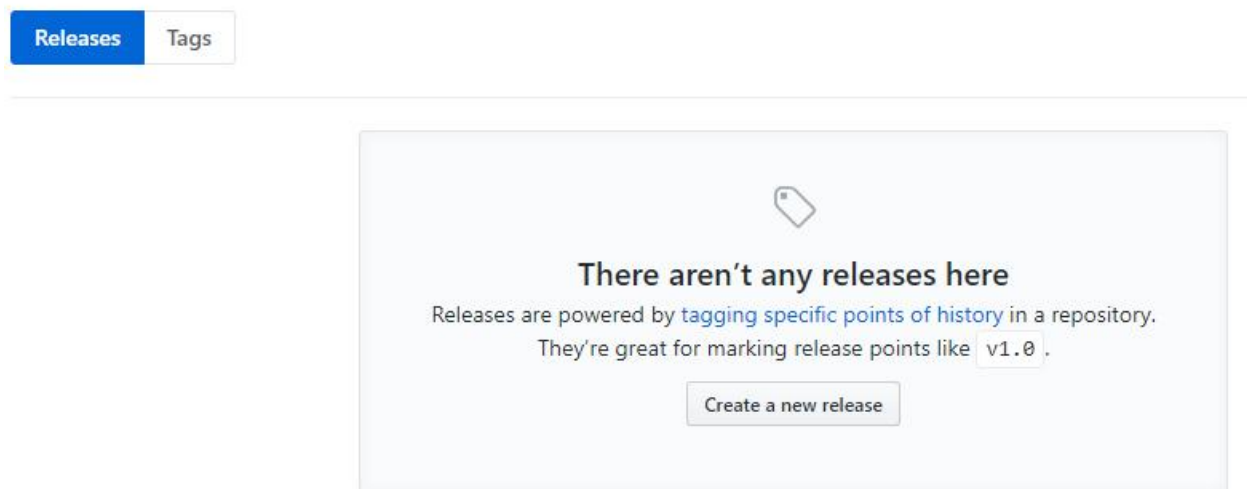
Để package của bạn được cài đặt và sử dụng, cần phải tạo cho nó 1 version.

Đầu tiên, vào repositories của package:



Hình 55. Tạo phiên bản cho package

Click releases



Hình 56. Tạo phiên bản cho package

Click Create a new releases và đặt tên phiên bản:

Releases

Tags

v1.0

@

Target: master

Excellent! This tag will be created from the target when you publish this release.

Release title

Write

Preview

Describe this release

Attach files by dragging & dropping, selecting or pasting them.

Hình 57. Tạo phiên bản cho package

Click Publish release để tạo version.

Sau khi đã tạo version cho package thành công, kết quả như sau:

 composer require phuvo/package-api

There is no license information available for the latest version (v1.0) of this package.

package api

Maintainers



Details

[github.com/gitvophu/package-api](https://github.com/gitvophu/package-api)

Source

Issues

Installs: 4

Dependents: 0

Suggesters: 0

Stars: 0

Watchers: 0

Forks: 0

Open Issues: 0

v1.0

2019-04-22 15:26 UTC

requires

requires (dev)

suggests

dev-master

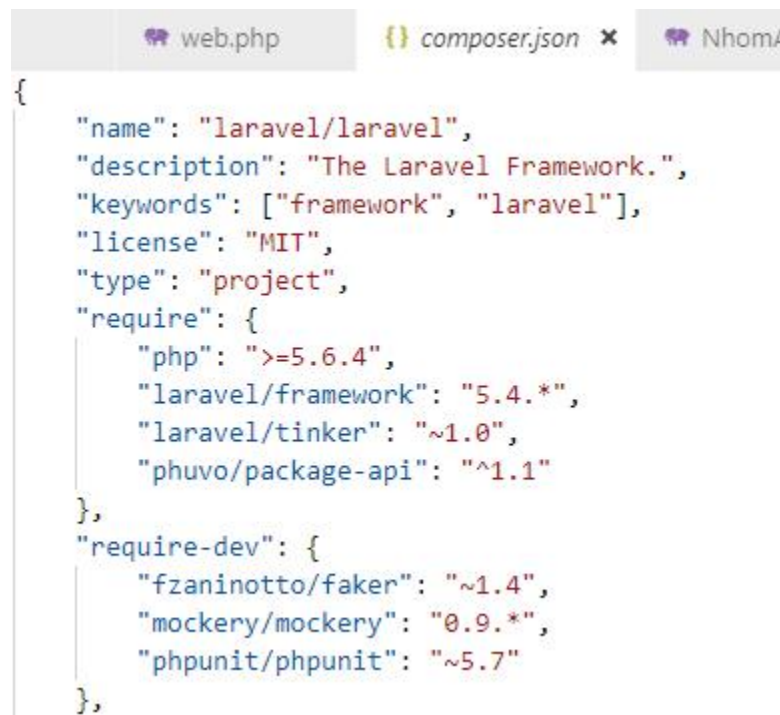
v1.0

Hình 58. Tạo phiên bản thành công



Tiếp theo, tiến hành cài đặt và sử dụng package. Copy `composer require phuvo/package-api` và mở Terminal trong sources Laravel chạy lệnh trên và chờ đợi đến khi nó cài đặt hoàn tất.


Sau khi package được cài, hãy kiểm tra thư mục vendor xem nó có xuất hiện không và kiểm tra file `composer.json` của Laravel:



```
{
  "name": "laravel/laravel",
  "description": "The Laravel Framework.",
  "keywords": ["framework", "laravel"],
  "license": "MIT",
  "type": "project",
  "require": {
    "php": ">=5.6.4",
    "laravel/framework": "5.4.*",
    "laravel/tinker": "~1.0",
    "phuvo/package-api": "^1.1"
  },
  "require-dev": {
    "fzaninotto/faker": "~1.4",
    "mockery/mockery": "0.9.*",
    "phpunit/phpunit": "~5.7"
  },
}
```

Hình 59. Kiểm tra đã cài đặt thành công package

Nếu trong “`require`” có tồn tại tên package của bạn, nghĩa là đã thành công, tiếp theo hãy thêm đoạn code sau vào “`provider`” trong `config/app.php`:



```
App\Providers\AppServiceProvider::class,
App\Providers\AuthServiceProvider::class,
// App\Providers\BroadcastServiceProvider::class,
App\Providers\EventServiceProvider::class,
App\Providers\RouteServiceProvider::class,
Phuvo\Apinhoma\NhomAServiceProvider::class,
```

Hình 60. Thêm provider để chạy package

Cuối cùng, start server và trải nghiệm.



## | TÀI LIỆU THAM KHẢO

<https://viblo.asia/p/tao-mot-package-don-gian-voi-laravel-5-7rVRqw94G4bP>

<https://pusher.com/tutorials/publish-laravel-packagist>

<https://viblo.asia/p/lam-nhu-the-nao-de-tao-mot-restful-api-bang-laravel-gGJ59X3DIX2>

<https://github.com/intrip/laravel-authentication-acl/blob/1.4/docs/index.md>