












## **LARAVEL FRAMEWORK**

### **BÀI 8: RESTFUL API TRONG LARAVEL PHẦN 1**








- Tổng quan về Restful API
- Triển khai Restful trong Laravel



## Phần I: Validation

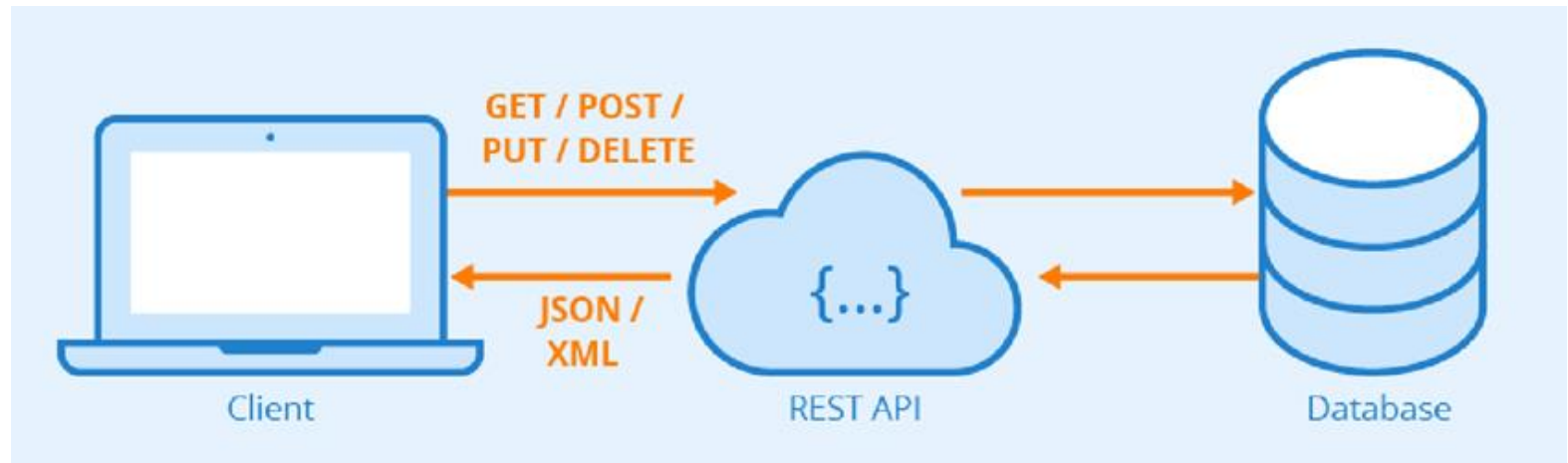
-  Giới thiệu restful api
-  Các method trong http
-  Uri trong restful api
-  Giới thiệu về json,
-  Các status code http cần biết
-  Tạo table trong database
-  Tạo model và cấu hình
-  Tạo eloquent resource
-  Tạo controller resource

## Phần II:

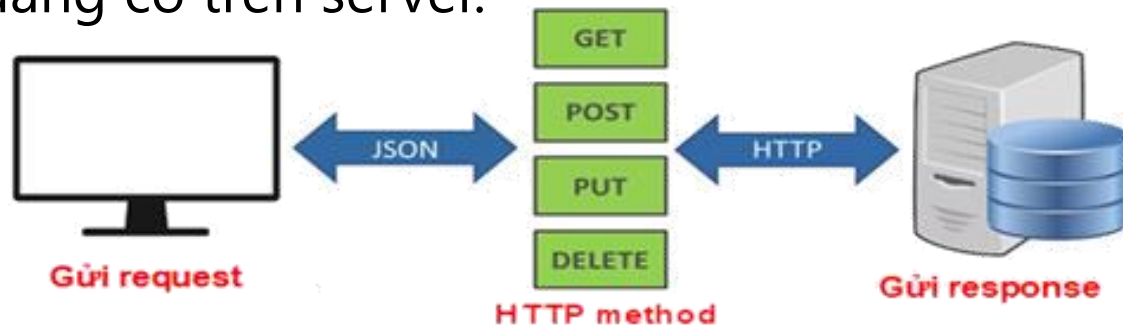
-  Tạo route api trong laravel
-  Code tạo resource mới
-  Code cập nhật một resource
-  Code trả về danh sách resource
-  Code xóa một resource
-  Code thông tin chi tiết một resource
-  Authentication API



- ❑ RESTful API là một chuẩn phổ biến ngày nay dùng để thiết kế giao tiếp cho ứng dụng web nhằm trao đổi và quản lý các resource từ xa.
- ❑ RESTful hoạt động dựa trên http với hoạt động request từ xa đến server để thực hiện lấy, thêm,xóa , sửa dữ liệu
- ❑ Trong Restful, dữ liệu trao đổi giữa client/server là JSON hoặc XML, những JSON thường dùng hơn
- ❑ Tool hay giúp test restful api là Postman. Bạn vào địa chỉ sau để download và cài <https://postman.com/downloads/>



- ❑ Giao thức http hỗ trợ tạo request đến server với nhiều method khác nhau: GET, POST, PUT, PATCH, DELETE...
- ❑ Restful dựa vào các method này để diễn tả các hành động xem, thêm, sửa, xóa dữ liệu (tài nguyên) trên server
- ❑ Method **GET**: dùng để diễn tả hành động muốn xem chi tiết 1 tài nguyên hoặc lấy danh sách tài nguyên trên server.
- ❑ Method **POST**: dùng để diễn tả hành động muốn thêm 1 tài nguyên mới trên server.
- ❑ Method **PUT**: dùng để diễn tả hành động muốn cập nhật 1 tài nguyên đang có trên server.
- ❑ Method **DELETE**: dùng để diễn tả hành động muốn xóa 1 tài nguyên đang có trên server.



Restful API quản lý tài nguyên thông qua các uri. Ví dụ:

❑ Lấy resource từ server

GET **<http://abc.com/api/product>** → trả về danh sách sản phẩm

GET **<http://abc.com/api/product/1>** → trả về sản phẩm có id là 1

❑ Tạo mới một resource trên server

POST **<http://abc.com/api/product>** → tạo sản phẩm mới

❑ Cập nhật 1 resource trên server

PUT **<http://abc.com/api/product/1>** → cập nhập sản phẩm có id là 1

❑ Xoá 1 resource trên server

DELETE **<http://abc.com/api/product/1>** → Xóa sản phẩm có id là 1

- ❑ JSON là định dạng dữ liệu chính được dùng trong restful api
- ❑ JSON rất ngắn gọn, giúp diễn tả dữ liệu có cấu trúc.
- ❑ Một đối tượng dữ liệu json được đặt trong dấu { }, trong đó dữ liệu được diễn tả thành từng cặp key:value cách nhau bởi dấu phẩy.
- ❑ Mảng các đối tượng dữ liệu json được diễn tả bằng dấu [ ]
- ❑ Ví dụ dữ liệu json mô tả 1 sản phẩm

```
{ "id": 1, "name": "Nói với tuổi 20", "price": 250000 }
```

- ❑ Ví dụ dữ liệu json mô tả mảng sản phẩm

```
[  
  { "id": 1, "name": "Nói với tuổi 20", "price": 250000 } ,  
  { "id": 2, "name": "Hiếu về trái tim", "price": 120000 } ,  
  { "id": 3, "name": "Con đường hạnh phúc", "price": 600000 } ,  
]
```

Khi request một API, server thường có trả về status code, giá trị của status cho biết kết quả của request

- ❑ 200 OK –thành công, cho các phương thức GET, PUT, PATCH, DELETE.
- ❑ 201 Created – Trả về khi resource được tạo thành công.
- ❑ 204 No Content – Trả về khi resource xóa thành công.
- ❑ 304 Not Modified – Client có thể sử dụng dữ liệu cache.
- ❑ 400 Bad Request – Request không hợp lệ
- ❑ 401 Unauthorized – Request cần phải có auth.
- ❑ 403 Forbidden – bị từ chối không cho phép.
- ❑ 404 Not Found – Không tìm thấy resource từ URI
- ❑ 405 Method Not Allowed – Phương thức bị cấm với user hiện tại.
- ❑ 429 Too Many Requests – Request bị từ chối do bị giới hạn



Để tạo restful api quản trị dữ liệu từ xa , các việc sau cần thực hiện:

1. Tạo table để chứa dữ liệu nếu chưa tạo (**Product**)
2. Tạo model (**Product**) và khai báo các field trong model
3. Tạo Eloquent resource để transform data
4. Tạo Controller resource chứa các action xem, thêm sửa, xóa resource
5. Tạo các route api dẫn vào các action trong controller
6. Viết code cho chức năng thêm resource và test
7. Viết code cho chức năng sửa resource và test
8. Viết code cho hiện danh sách resource và test
9. Viết code hiện chi tiết 1 resource và test
10. Viết code xóa 1 resource và test

## 1. Tạo table (**product**) để lưu dữ liệu


❖ Tạo migration:

**php artisan make:migration create\_products\_table**

❖ Khai báo các field trong file migration mới tạo

```
public function up() {
    Schema::create('products', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->integer('price');
        $table->timestamps();
    });
}
```

❖ Chạy migration để tạo table: **php artisan migrate**

Server: 127.0.0.1 » Database: jet » Table: products							
#	Name	Type	Collation	Attributes	Null	Default	Extra
1	id 	bigint(20)		UNSIGNED	No	None	AUTO_INCREMENT
2	name	varchar(255)	utf8mb4_unicode_ci		No	None	
3	price	int(11)			No	None	
4	created_at	timestamp			Yes	NULL	
5	updated_at	timestamp			Yes	NULL	

## 2. Tạo model Eloquent (product) để tương tác với table

❖ Tạo migration:

**php artisan make:model Product**

❖ Khai báo các field trong file model mới tạo

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
class Product extends Model {
    use HasFactory;
    protected $fillable = ['name', 'price'];
}
```

❖ Chèn vài dòng dữ liệu vào table (dùng seeder)

3. Eloquent resource giúp chuyển (transform) dữ liệu cho thân thiện với người dùng. Ví dụ như định dạng thời gian, đổi tên các field cho đẹp.... trước khi response.
- ❖ Để tạo 1 class eloquent resource (product), chạy lệnh sau:  
**php artisan make:resource Product**
  - ❖ File resource được lưu trong **app/Http/Resources** , trong đó có hàm toArray() – là nơi khai báo các định dạng mong muốn

```
<?php
namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\JsonResource;
class Product extends JsonResource {
    public function toArray($request) {
        return parent::toArray($request);
    }
}
```

- ❖ Ví dụ: định nghĩa tên thân thiện cho field và định dạng lại dữ liệu các field

```
<?php
namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\JsonResource;
class Product extends JsonResource {
    public function toArray($request) {
        return [
            'id' => $this->id,
            'tensp' => $this->name,
            'gia' => $this->price,
            'created_at' => $this->created_at->format('d/m/Y'),
            'updated_at' => $this->updated_at->format('d/m/Y'),
        ];
    }
}
```

## 4. Tạo resource controller để có sẵn các hàm tương tác db

```
php artisan make:controller ProductController --resource
```

- ❑ Mở file controller resource, sẽ thấy các hàm có sẵn
- ❑ Thêm các lệnh nhúng validator để kiểm tra dữ liệu, nhúng model và eloquent resource để dùng

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Validator;
use App\Models\Product;
use App\Http\Resources\Product as ProductResource;

class ProductController extends Controller {
    public function index(){}
    public function create() {}
    public function store(Request $request) { }
    public function show($id) { }
    public function edit($id) { }
    public function update(Request $request,$id){}
    public function destroy($id) { }
}
```

- ❑ **index()**: nơi code để trả về list các tài nguyên
- ❑ **store()**: Nơi code để tạo (lưu) tài nguyên mới
- ❑ **update()**: nơi code để cập nhật 1 tài nguyên.
- ❑ **destroy()**: nơi code để xóa 1 tài nguyên
- ❑ **create() , edit()** có thể xóa



**DEMO**

Demo xác thực dữ liệu form login





## **LARAVEL FRAMEWORK**

### **BÀI 8: RESTFUL API WITH LARAVEL PHẦN 2**



5. Tạo route để đưa user đến các action trong controller
- ❖ Các route api được định nghĩa trong file routes/api.php.
  - ❖ Khi thực hiện request, các route sẽ có prefix là chữ **api**,  
(có thể đổi trong *app\Providers\RouteServiceProvider.php*)
  - ❖ Định nghĩa route trong api.php có 2 cách:
    - Cách 1: Khai báo route với 1 method cụ thể (get, post), route (sp), action (index, store). Ví dụ:

```
use App\Http\Controllers\ProductController;  
Route::get('sp', [ProductController::class, 'index']);  
Route::post('sp', [ProductController::class, 'store']);
```

- Cách 2: Khai báo route với lệnh **Route::resource** để dùng chung route cho các method get, post, put, delete dẫn vào các action của controller

```
Route::resource('products', ProductController::class);
```

Lệnh trên tạo ra các route dẫn vào các action như sau

Method	URI	Action
POST	api/products	App\Http\Controllers\ProductController@store
GET HEAD	api/products	App\Http\Controllers\ProductController@index
GET HEAD	api/products/{product}	App\Http\Controllers\ProductController@show
PUT PATCH	api/products/{product}	App\Http\Controllers\ProductController@update
DELETE	api/products/{product}	App\Http\Controllers\ProductController@destroy

- ❑ Trong controller, hàm **store()** là nơi code tạo resource mới, sau đó trả về:
- ❖ **status** (true/false) cho biết hoạt động thành công/thất bại.
  - ❖ **message**: text mô tả kết quả hoạt động
  - ❖ **data** : tài nguyên mới tạo

```
public function store(Request $request) {
    $input = $request->all();
    $validator = Validator::make($input, [
        'name' => 'required', 'price' => 'required'
    ]);
    if($validator->fails()){
        $arr = [
            'success' => false,
            'message' => 'Lỗi kiểm tra dữ liệu',
            'data' => $validator->errors()
        ];
        return response()->json($arr, 200);
    }
    $product = Product::create($input);
    $arr=['status' => true,
        'message'=>"Sản phẩm đã lưu thành công",
        'data'=> new ProductResource($product)
    ];
    return response()->json($arr, 201);
}
```

POST http://localhost:8000/api/products Send

Params Authorization Headers (9) Body Pre-req

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

	KEY	VALUE
<input checked="" type="checkbox"/>	name	Iphone 13
<input checked="" type="checkbox"/>	price	32000000

```
{
  "status": true,
  "message": "Sản phẩm đã lưu thành công",
  "data": {
    "id": 3,
    "tensp": "Iphone 13",
    "gia": "32000000",
    "created_at": "27/12/2021",
    "updated_at": "27/12/2021"
  }
}
```

❑ Trong controller, hàm **update()** là nơi code để cập nhật tài nguyên, sau đó trả về:

- ❖ **status** (true/false) cho biết hoạt động thành công/thất bại.
- ❖ **message**: text mô tả kết quả hoạt động
- ❖ **data** : tài nguyên mới cập nhật

```
public function update(Request $request, Product $product){
    $input = $request->all();
    $validator = Validator::make($input, [
        'name' => 'required', 'price' => 'required'
    ]);
    if($validator->fails()){
        $arr = [
            'success' => false,
            'message' => 'Lỗi kiểm tra dữ liệu',
            'data' => $validator->errors()
        ];
        return response()->json($arr, 200);
    }
    $product->name = $input['name'];
    $product->price = $input['price'];
    $product->save();
    $arr = [
        'status' => true,
        'message' => 'Sản phẩm cập nhật thành công',
        'data' => new ProductResource($product)
    ];
    return response()->json($arr, 200);
}
```

PUT ▼ http://localhost:8000/api/products/3

Params ● Authorization Headers (9) ● Body ● Pre

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

	KEY	VALUE
<input checked="" type="checkbox"/>	name	Iphone 12
<input checked="" type="checkbox"/>	price	15000000

```
{
  "status": true,
  "message": "Sản phẩm đã cập nhật thành công",
  "data": {
    "id": 3,
    "tensp": "Iphone 12",
    "gia": "15000000",
    "created_at": "27/12/2021",
    "updated_at": "27/12/2021"
  }
}
```

- ❑ Trong controller, hàm **index()** là nơi code để trả về danh sách resource. Sau đó trả về:
- ❖ **status** (true/false) cho biết hoạt động thành công/thất bại.
  - ❖ **message**: text mô tả kết quả hoạt động
  - ❖ **data** : là mảng json các tài nguyên

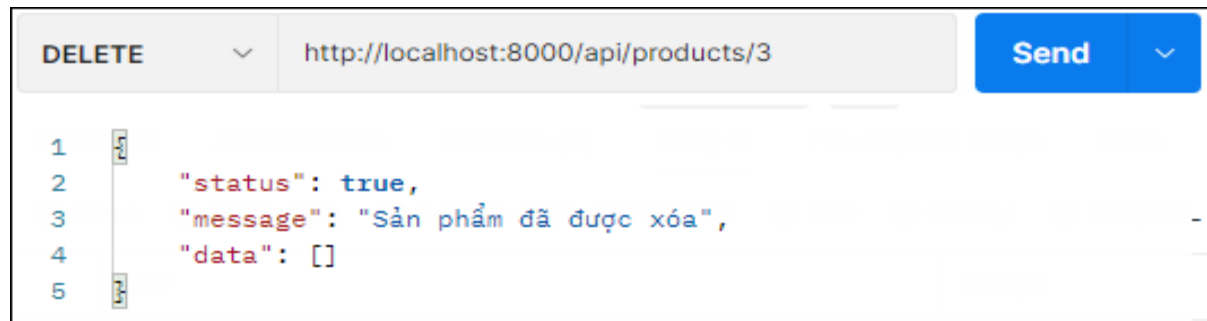
```
public function index() {  
    $products = Product::all();  
    $arr = [  
        'status' => true,  
        'message' => "Danh sách sản phẩm",  
  
        'data'=>ProductResource::collection($products)  
    ];  
    return response()->json($arr, 200);  
}
```

GET <http://localhost:8000/api/products/>

```
{  
  "status": true,  
  "message": "Danh sách sản phẩm",  
  "data": [  
    {  
      "id": 1,  
      "tensp": "Samsung Galary S8",  
      "gia": 68000000,  
      "created_at": "09/12/2021",  
      "updated_at": "16/12/2021"  
    },  
    {  
      "id": 2,  
      "tensp": "HTC M10",  
      "gia": 65000000,  
      "created_at": "13/12/2021",  
      "updated_at": "12/12/2021"  
    }  
  ]  
}
```

- ❑ Trong controller, hàm `destroy()` là nơi code để xóa 1 tài nguyên, sau đó trả về
- ❖ **status** (true/false) cho biết thành công/thất bại.
  - ❖ **message**: text mô tả kết quả hoạt động
  - ❖ **data** : không có

```
public function destroy(Product $product){  
    $product->delete();  
    $arr = [  
        'status' => true,  
        'message' => 'Sản phẩm đã được xóa',  
        'data' => [],  
    ];  
    return response()->json($arr, 200);  
}
```



❑ Trong controller, hàm **show()** là nơi code để lấy 1 tài nguyên, sau đó trả về

- ❖ **status** (true/false) cho biết thành công/thất bại.
- ❖ **message**: text mô tả kết quả hoạt động
- ❖ **data** : chi tiết tài nguyên

```
public function show($id) {  
    $product = Product::find($id);  
    if (is_null($product)) {  
        $arr = [  
            'success' => false,  
            'message' => 'Không có sản phẩm này',  
            'data' => []  
        ];  
        return response()->json($arr, 200);  
    }  
    $arr = [  
        'status' => true,  
        'message' => "Chi tiết sản phẩm ",  
        'data' => new ProductResource($product)  
    ];  
    return response()->json($arr, 201);  
}
```

GET http://localhost:8000/api/products/1

```
1 {  
2     "status": true,  
3     "message": "Chi tiết sản phẩm ",  
4     "data": {  
5         "id": 1,  
6         "tensp": "Samsung Galary S8",  
7         "gia": 680000000,  
8         "created_at": "09/12/2021",  
9         "updated_at": "16/12/2021"  
10    }  
11 }
```

Sau đây là các bước cơ bản hướng dẫn để triển khai authentication khi dùng API:

1. Cài gói sanctum:

```
composer require laravel/sanctum
```

2. Publish cấu hình mà migration của sanctum:

```
php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"
```

3. Chạy migration để tạo các table cần dùng

```
php artisan migrate
```

4. Tạo AuthController

```
php artisan make:controller API/AuthController
```



5. Mở Controllers/API/AuthController.php mới tạo viết các hàm thực hiện login, logout, register...

```
<?php
namespace App\Http\Controllers\API;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Auth;
use Validator;
use App\Models\User;
class AuthController extends Controller {
    public function register(Request $request) {
        $validator = Validator::make($request->all(), [
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:8'
        ]);
        if($validator->fails())return response()->json($validator->errors());
        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password)
        ]);
        $token = $user->createToken('auth_token')->plainTextToken;
        return response()->json(
            ['data' => $user, 'access_token' => $token, 'token_type' => 'Bearer', ]);
    }
    ///login
    ///logout
} //class
```

```
public function login(Request $request) {  
    if (!Auth::attempt($request->only('email', 'password'))) {  
        return response()->json(['message' => 'Unauthorized'], 401);  
    }  
    $user = User::where('email', $request['email'])->firstOrFail();  
    $token = $user->createToken('auth_token')->plainTextToken;  
    return response()->json(['message' => 'Chào ' . $user->name . '! Chúc an lành',  
        'access_token' => $token, 'token_type' => 'Bearer',]);  
}  
public function logout() {  
    auth()->user()->tokens()->delete();  
    return ['message' => 'Bạn đã thoát ứng dụng và token đã xóa'];  
}
```

## 6. Định nghĩa các route cần được bảo vệ trong routes/api.php

```
use App\Http\Controllers\API\AuthController;
//API route để đăng ký
Route::post('/login', [AuthController::class, 'register']);
//API route để đăng nhập
Route::post('/login', [AuthController::class, 'login']);
Route::group(['middleware' => ['auth:sanctum']], function () {
    Route::get('/profile', function(Request $request) {
        return auth()->user();
    });
    // API route thoát
    Route::post('/logout', [AuthController::class, 'logout']);
});
```

## 7. Bảo vệ các route

```
use App\Http\Controllers\API\AuthController;
//API route để đăng ký
Route::post('/dangky', [AuthController::class, 'register']);
//API route để đăng nhập
Route::post('/dangnhap', [AuthController::class, 'login']);
Route::group(['middleware' => ['auth:sanctum']], function () {
    Route::get('/profile', function(Request $request) {
        return auth()->user();
    });
    // API route thoát
    Route::post('/logout', [AuthController::class, 'logout']);
});
```

❑ Dùng Postman request uri đăng ký

POST http://localhost:8000/api/dangky Send

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

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

<input checked="" type="checkbox"/>	name	Text	Nguyễn Văn Tèo
<input checked="" type="checkbox"/>	email		teonv@gmail.com
<input checked="" type="checkbox"/>	password		12345678

```

1
2
3
4
5
6
7
8
9
10
11
    "data": {
      "name": "Nguyễn Văn Tèo",
      "email": "teonv@gmail.com",
      "updated_at": "2021-12-28T09:34:17.000000Z",
      "created_at": "2021-12-28T09:34:17.000000Z",
      "id": 1
    },
    "access_token": "1|mRjJJDTtFPSuYfUIOAYybkr0Ai9L1Jx1NSxKXd9w",
    "token_type": "Bearer"
  
```

❑ Thông tin User sẽ được chèn vào database, có cả token

Table: users				
id	name	email	email_verified_at	password
1	Nguyễn Văn Tèo	teonv@gmail.com	NULL	\$2y\$10\$q4PFS5UvO2D/qzHjcTMGruhiuisHCCK8bzSk

Table: personal_access_tokens				
id	tokenable_type	tokenable_id	name	token
1	App\Models\User	1	auth_token	03a0675544fe6be9283326b3f5e802b01149b28953e3

## ❑ Đăng nhập

**POST** ▼ http://localhost:8000/api/dangnhap

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

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

<input type="checkbox"/>	name	Nguyễn Văn Tèo
<input checked="" type="checkbox"/>	email	teonv@gmail.com
<input checked="" type="checkbox"/>	password	12345678

```
1 {  
2   "message": "Chào Nguyễn Văn Tèo! Chúc an lành",  
3   "access_token": "2|x4S7Jn8gGbfo6AMHn020eHyJ10mAFArntiVm1jvb",  
4   "token_type": "Bearer"  
5 }
```












# DEMO








-Demo xác thực số điện thoại  
VietNam bắt đầu “+84”



## Phần I: Validation

-  Giới thiệu restful api
-  Các method trong http
-  Uri trong restful api
-  Giới thiệu về json,
-  Các status code http cần biết
-  Tạo table trong database
-  Tạo model và cấu hình
-  Tạo eloquent resource
-  Tạo controller resource

## Phần II:

-  Tạo route api trong laravel
-  Code tạo resource mới
-  Code cập nhật một resource
-  Code trả về danh sách resource
-  Code xóa một resource
-  Code thông tin chi tiết một resource
-  Authentication API







**Cảm ơn**