

LARAVEL + OAUTH2



CREANDO
NUESTRA API
EN LARAVEL
CON OAUTH2

Gonzalo Fernández
Full Stack Developer

ANTES DE EMPEZAR

Este tutorial fue creado para explicar todo lo necesario para el desarrollo de una **API** haciendo uso de **Laravel** y **Laravel Passport** para que esta cumpla los estándares del protocolo **OAuth2**, así como también poder probar esta **API** con la ayuda de ejemplos ya sea con **cURL** o **Python**, lo que **no** significa que sean las únicas formas de usar una **API** desarrollada con lo aprendido en este tutorial.

Además, este es el primer tutorial completo que hago, y, para ser honesto, cuando comencé a redactar este tutorial, nunca creí que tendría tantas páginas, por lo tanto, si sólo quieres aprender cómo crear una **API** en **Laravel**, puedes leer sólo la sección **Desarrollando LaraAuth**, desde la página **8**, pero, si quieres adentrarte en algunos conceptos importantes, y, probar tu **API**, lo ideal sería leerlo completo

Por último, como la finalidad de este tutorial es la creación de una **API** en **Laravel**, no ahondé mucho en temas de clientes, ejemplos, conceptos de **Laravel**, etc., por lo cual, se requiere tener **nociones básicas** de **Laravel**, aunque tampoco es excluyente, pero ayudará a comprender mejor el código de **LaraAuth**.

Cualquier duda o consulta, no dudes en contactarme.

INDICE

1. Introducción	4
2. OAuth2	5
3. Laravel Passport	7
4. Desarrollando LaraAuth	8
4.1. Passport	9
4.2. Migraciones	11
4.3. Modelos	12
4.4. Rutas	13
4.5. Controladores	14
5. Probando LaraAuth	15
5.1. cURL	17
5.2. Python	22
5.3. Próximamente	24
6. Bibliografía	25

1. Introducción

Cada vez son más las aplicaciones que ofrecen servicios mediante APIs, entre las más famosas están [Google Maps](#), [Amazon S3](#), [Facebook](#), entre otras. Tal es el impacto de estas APIs que muchos desarrolladores han llegado a crear proyectos enteros cuyas raíces recaen en estas, por ejemplo: **Panoramio**, un sitio web donde la gente podía subir sus fotos y georreferenciarlas gracias a la API de Google Maps, tal fue su éxito que en julio del 2007 Google compró el sitio a sus desarrolladores y lo integró a su servicio Google Maps.

No solo las aplicaciones web hacen uso de APIs, también las aplicaciones móviles y de otros tipos, básicamente, la mayoría de aplicaciones que no cuenten con una conexión directa a la Base de Datos del sistema y necesiten trabajar estos datos.

En mi caso particular, comencé a ahondar en esto mientras desarrollaba un proyecto para la empresa **Foodlf**, tenía toda la plataforma web desarrollada en **Laravel** y además, necesitaba mostrar en vivo la localización de los usuarios de una aplicación móvil (los “conductores”), en este punto, me di cuenta que no tenía como enviar los datos de la app móvil a mi plataforma web, así como obtener datos desde la plataforma a mi app, fue entonces cuando descubrí **Laravel Passport**, el cual provee de una implementación de **OAuth2** sencilla de usar, gracias a esto fue sumamente fácil la transmisión de datos entre mi aplicación móvil y mi servidor, y es precisamente sobre esto que trabajaremos en este tutorial, desarrollando nuestra aplicación [LaraAuth](#) (click para ir al repositorio en GitHub).

2. OAuth2

OAuth2 define un protocolo de autorización a seguir para permitir el acceso a información privada mediante **APIs**. Este protocolo de autorización consta de 3 **roles**:

1. **Cliente**: Es la aplicación que intenta acceder a la información almacenada en nuestro servidor.
2. **Servidor de Autorización y Recursos**: El servidor de recursos es el servidor que mantiene la **API** con la que se puede acceder a la información solicitada, en cambio, el servidor de Autorización es quien aprueba o deniega esta solicitud, en pequeñas implementaciones (por ejemplo, **LaraAuth**) estos son el mismo, pero en desarrollos a gran escala deberías considerar desarrollarlos como componentes separados.
3. **Dueño de Recursos**: Este es básicamente la entidad quien otorga acceso a sus datos mediante una **API**, no necesariamente debe ser una persona otorgando datos de su cuenta, puede ocurrir que necesitemos entregar datos “estándar” de nuestra aplicación como por ejemplo el rendimiento, estadísticas, u otras variables que quizá esta pueda otorgar, como el clima, esta autorización puede venir de parte de una cuenta “por defecto” creada sólo para ese propósito.

El primer paso en el desarrollo de una aplicación con **OAuth2** es “registrar” nuestra aplicación (ya lo veremos durante el desarrollo de **LaraAuth**), esto nos entregará un **Client-ID** y un **Client-Secret**, con los cuales nos podremos autenticar en nuestra **API**.

2. OAuth2

Una vez “registrada” nuestra aplicación (en este punto ya tendremos implementado **OAuth2**), el resto va por el lado de nuestro **Cliente** (en este tutorial no desarrollaremos uno pero sí realizaremos ejemplos para mostrar el funcionamiento de **LaraAuth**), el cual debe apegarse a alguno de los flujos de autenticación que ofrece **OAuth2**, para nuestra aplicación, sólo hablaremos del flujo con **Password Grant Type**, que consta de una autorización mediante las credenciales de acceso del usuario dueño de la información que necesitamos.

Para solicitar esta autorización, es necesario hacer una solicitud **POST** a nuestro servidor de Autorización / Recursos:

```
POST https://api.laraauth.com/token
grant_type=password&
username=DevTotal&
password=FacebookDeveloperCircles&
client_id=1&
client_secret=h08z123nLtqSZxpJ9vnP7CdHTFWqCUGxX9gcMKWZ
```

En donde:

- **grant_type=password**: Tipo de flujo OAuth2 solicitado.
- **username**: Usuario dueño de la información a solicitar.
- **password**: Contraseña del usuario dueño de la información.
- **client_id**: **Client-ID** obtenido al “registrar” nuestra app.
- **client_secret**: **Client-Secret** obtenido con el **Client-ID**.

Hecho esto, recibiremos un **access_token** con este, ya podemos realizar nuestras solicitudes autorizadas a nuestra aplicación. Este token nos autorizará por un tiempo determinado y tendrá que ser refrescado constantemente, pero esto lo veremos con **Laravel Passport**.

3. Laravel Passport

Laravel Passport es el paquete PHP desarrollado por Laravel para facilitar el desarrollo de APIs entregando la parte de la autenticación prácticamente en bandeja, este paquete provee de una implementación completa de un servidor OAuth2 para nuestra aplicación que podemos configurar en cosa de minutos.



Además, también trae una implementación **Front-End** (opcional) para manejar algunos aspectos básicos de nuestro servidor OAuth2 (aunque, para efectos del desarrollo de **LaraAuth**, no revisaremos en detalle):

My Application Matt Stauffer ▾

OAuth Clients [Create New Client](#)

You have not created any OAuth clients.

Personal Access Tokens [Create New Token](#)

You have not created any personal access tokens.

OAuth Clients [Create New Client](#)

Client ID	Name	Secret	
3	Consumer.dev	yx0JrP0L9gqbXxoxoF15I22IytF0peCnUXD3aE0d	Edit Delete

Cabe destacar que, no es necesario el uso de esta plataforma que nos provee Laravel Passport para agregar clientes y manejar el sistema, esto también puede realizarse de la mano de comandos.

4. Desarrollando LaraAuth

Ahora que tenemos un mejor manejo de los principales conceptos que usaremos en este proyecto, comencemos a desarrollar **LaraAuth**, como primer paso, debemos crear nuestro proyecto **Laravel** e instalar **Passport**:

```
laravel new LaraAuth
```

```
composer require laravel/Passport
```

```
D:\proyectos\Talleres\Taller Laravel + OAuth\LaraAuth>composer require laravel/passport
Using version ^3.0 for laravel/passport
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
- Installing phpseclib/phpseclib (2.0.6)
  Loading from cache
- Installing psr/http-message (1.0.1)
  Loading from cache
- Installing zendframework/zend-diactoros (1.4.0)
  Loading from cache
- Installing symfony/psr-http-message-bridge (v1.0.0)
  Loading from cache
- Installing defuse/php-encryption (v2.1.0)
  Downloading: 100%
  ...
- Installing laravel/passport (v3.0.0)
  Downloading: 100%
```

Una vez listo nuestro proyecto nuevo, registraremos Passport en el arreglo **providers** del archivo **config/app.php** para ser cargado en nuestro proyecto. Hecho esto, también implementaremos el sistema de autenticación por defecto de Laravel con el comando:

```
php artisan make:auth
```

```
'providers' => [
    ...
    /*
     * Application Service Providers...
     */
    ...
    Laravel\Passport\PassportServiceProvider::class,
],
```

```
D:\proyectos\Talleres\Taller Laravel + OAuth\LaraAuth>php artisan make:auth
Authentication scaffolding generated successfully.
```


4.1 Passport

Hemos integrado a nuestro proyecto Passport y el sistema de autenticación por defecto de Laravel, para esta aplicación **no modificaremos** las migraciones por defecto, por lo tanto, procederemos a migrar y ejecutar el comando de instalación de Passport (se asume que han configurado su archivo `.env` con los datos de su BD):

```
php artisan migrate
php artisan passport:install
```

```
D:\proyectos\Talleres\Taller Laravel + OAuth\LaraAuth>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table
Migrating: 2016_06_01_000001_create_oauth_auth_codes_table
Migrated: 2016_06_01_000001_create_oauth_auth_codes_table
Migrating: 2016_06_01_000002_create_oauth_access_tokens_table
Migrated: 2016_06_01_000002_create_oauth_access_tokens_table
Migrating: 2016_06_01_000003_create_oauth_refresh_tokens_table
Migrated: 2016_06_01_000003_create_oauth_refresh_tokens_table
Migrating: 2016_06_01_000004_create_oauth_clients_table
Migrated: 2016_06_01_000004_create_oauth_clients_table
Migrating: 2016_06_01_000005_create_oauth_personal_access_clients_table
Migrated: 2016_06_01_000005_create_oauth_personal_access_clients_table

D:\proyectos\Talleres\Taller Laravel + OAuth\LaraAuth>php artisan passport:install
Encryption keys generated successfully.
Personal access client created successfully.
Client ID: 1
Client Secret: yHbEyHbAv2u1dWyhBgk0aJ944Xgelic5Ditoq03
Password grant client created successfully.
Client ID: 2
Client Secret: h08z123nLtqSZxpJ9vnP7CdHTFWqCUGxX9gcMKWZ
```

Hecho esto, hemos creado 2 **clientes** con los que podemos autenticarnos en nuestra **API**, pero para efectos de esta aplicación, sólo haremos uso del 2do, con su respectivo **Client-ID** y **Client-Secret**, estos son los datos antes mencionados que obtenemos al “registrar” nuestra aplicación, con ellos,

4.1 Passport

Luego de guardar estos datos, debemos agregar el trait `Laravel\Passport\HasApiTokens` en nuestro modelo `App\User`, esto nos atribuirá algunos métodos que nos permitirán inspeccionar el token y alcance del usuario autenticado mediante nuestra API.

```
<?php

namespace App;

use Laravel\Passport\HasApiTokens;
use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use HasApiTokens, Notifiable;
```

Además, debemos llamar al método `Passport::routes` dentro del método `boot` de nuestro `app/Providers/AuthServiceProvider`, de modo de poder registrar las rutas necesarias para solicitar / revocar tokens, entre otras cosas. Finalmente, en nuestro archivo `config/auth.php`, debemos cambiar el `driver` del `guard API` a `passport`, de modo que nuestra aplicación autentifique las solicitudes API mediante el `Passport TokenGuard`. Hecho esto, finalmente habremos configurado `Passport`.

```
<?php

namespace App\Providers;

use Laravel\Passport\Passport;
use Illuminate\Support\Facades\Gate;
use
Illuminate\Foundation\Support\Providers\Auth
ServiceProvider as ServiceProvider;

class AuthServiceProvider extends
ServiceProvider
{
    ...
    public function boot()
    {
        $this->registerPolicies();

        Passport::routes();
    }
}
```

```
<?php

return [

    ...

    'guards' => [
        'web' => [
            'driver' => 'session',
            'provider' => 'users',
        ],

        'api' => [
            'driver' => 'passport',
            'provider' => 'users',
        ],
    ],

    ...

];
```

4.2 Migraciones

Vamos a integrar en nuestra aplicación un dato del cual un usuario pueda ser dueño, de este modo, este dato apuntará a él y podrá ser obtenido mediante nuestra **API**, digamos que este dato es acerca de los **pokémons** del usuario (porque, claro, toda aplicación debería al menos ser capaz de almacenar la información de los pokémons de sus clientes). Para esto, crearemos una tabla **pokemones** y un modelo **Pokémon** para establecer una relación **Muchos a Uno** con el modelo **User**, esto mediante los comandos:

```
php artisan make:migration create_pokemones_table --create=pokemones
```

```
php artisan make:model Pokemon
```

```
D:\proyectos\Talleres\Taller Laravel + OAuth\LaraAuth>php artisan make:migration
create_pokemones_table --create=pokemones
Created Migration: 2017_07_23_043832_create_pokemones_table

D:\proyectos\Talleres\Taller Laravel + OAuth\LaraAuth>php artisan make:model Pokemon
Model created successfully.
```

Creado el archivo de migraciones `create_pokemones_table`, procedemos a agregar en él los cambios de la tabla para así poder migrar con el comando: `php artisan migrate`

```
class CreatePokemonesTable extends Migration
{
    public function up()
    {
        Schema::create('pokemones', function (Blueprint $table) {
            $table->increments('id');           // id único del registro
            $table->integer('user_id');         // id del dueño, para crear una relación muchos a uno
            $table->integer('pokedex_id');     // id en la pokedex, sólo para mostrar
            $table->string('apodo');           // apodo asignado por el dueño al pokemon
            $table->string('nombre');          // nombre original del pokemon
            $table->integer('nivel');          // nivel actual del pokemon
            $table->integer('exp');            // experiencia actual del pokemon
            $table->timestamps();              // created_at y updated_at
        });
    }

    public function down()
    {
        Schema::dropIfExists('pokemones');
    }
}
```

4.3 Modelos

Ahora que tenemos nuestra tabla **pokemones** creada, debemos establecer la relación **Muchos a Uno**, para esto, crearemos la función **pokemones** en el modelo **User** que nos entregará una instancia **HasMany**, devolviendo así todos los pokémons del usuario.

```
<?php
...

class User extends Authenticatable
{
    ...

    public function pokemones()
    {
        return $this->hasMany('App\Pokemon');
    }
}
```

Luego, en el modelo **Pokemon**, crearemos la función **user** que nos entregará una instancia **BelongsTo**, devolviéndonos así un objeto de tipo **User**, con los datos del usuario dueño del Pokémon. Además, como nuestra tabla no cumple con el estándar (el nombre de tabla debe ser el nombre del modelo con **s**), debemos indicarlo en el modelo **Pokemon**. Por último, debemos indicar qué campos son **asignables** masivamente con el arreglo **\$fillable**.

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Pokemon extends Model
{
    protected $table = 'pokemones';
    protected $fillable = ['pokedex_id', 'apodo', 'nombre', 'nivel', 'exp'];

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

4.4 Rutas

Como en todas las APIs, es necesario tener urls específicas para manejar las solicitudes de un cliente, en **Laravel**, las rutas para apis se manejan en el archivo **routes/api.php** y a las urls registradas se les antepondrá **/api/** automáticamente. Para ordenar nuestras rutas, crearemos un grupo de rutas del **middleware auth:api**, las cuales, serán manejadas por el controlador **HomeController** que viene por defecto con el sistema de cuentas de Laravel (para esta aplicación, no hay motivos para crear otro):

```
<?php

use Illuminate\Http\Request;

/*
|-----
| API Routes
|-----
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

Route::middleware('auth:api')->group(function () {

    // Devuelve los datos del usuario autenticado
    // Parámetros: Ninguno
    Route::post('/user', 'HomeController@user');

    // Devuelve los pokemones del usuario autenticado
    // Parámetros: Ninguno
    Route::post('/user/pokemones', 'HomeController@user_pokemones');

    // Agrega un Pokémon a los pokemones del usuario autenticado
    // Parámetros: pokedex_id => ID Pokedex del pokémon
    //               apodo => Apodo del Pokémon
    //               nombre => Nombre original del Pokémon
    //               nivel => Nivel del Pokémon
    //               exp => Experiencia del Pokémon
    Route::post('/user/pokemones/agregar', 'HomeController@user_pokemones_agregar');

    // Devuelve los datos del ID buscado y un error al no encontrarlo
    // Parámetros: user_id => Id del usuario buscado
    Route::post('/user/buscar', 'HomeController@user_buscar');

    // Devuelve los datos pokemon del ID buscado y un error al no encontrarlo
    // Parámetros: pokemon_id => Id del pokemon buscado
    Route::post('/pokemon/buscar', 'HomeController@pokemon_buscar');

});
```

4.5 Controladores

Como ya indicamos en las **rut**as, manejaremos las solicitudes a nuestra **API** mediante el controlador **HomeController**, para esto, debemos crear las funciones que serán llamadas, no ahondaré demasiado en la descripción del código de cada función, pues, el principal objetivo del tutorial es poder realizar lo básico de una **API**, pero la mayoría es uso de las **Relaciones** de Laravel.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\User;
use App\Pokemon;

class HomeController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function index(Request $request)
    {
        return view('home');
    }

    public function user(Request $request)
    {
        return $request->user();
    }

    public function user_pokemones(Request $request)
    {
        return $request->user()->pokemones;
    }

    public function user_pokemones_agregar(Request $request)
    {
        return $request->user()->pokemones()->create($request->all());
    }

    public function user_buscar(Request $request)
    {
        return ($r = User::find($request->user_id)) == NULL ? "Error: ID no encontrado" : $r;
    }

    public function pokemon_buscar(Request $request)
    {
        return ($r = Pokemon::find($request->pokemon_id)) == NULL ? "Error: ID no encontrado" : $r;
    }
}
```

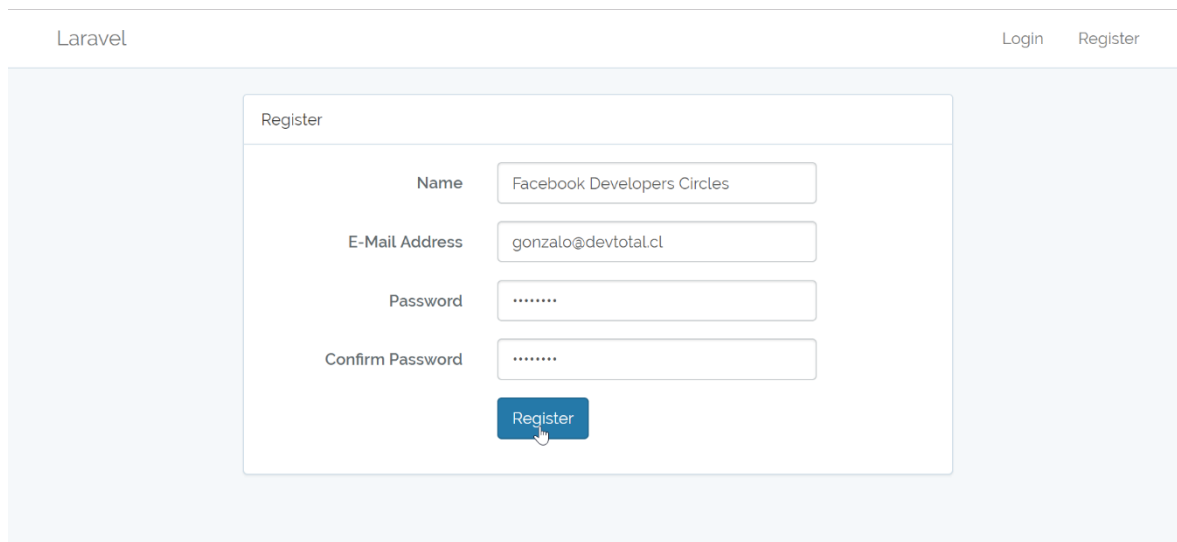
5. Probando LaraAuth

Finalmente, hemos terminado nuestra aplicación **LaraAuth**, ahora debemos probarla, para esto, debemos iniciar nuestra aplicación Laravel, con el comando:

```
php artisan serve --host="0.0.0.0" --port=80
```

```
D:\proyectos\Talleres\Taller Laravel + OAuth\LaraAuth>php artisan serve --host="0.0.0.0" --port=80
Laravel development server started: <http://0.0.0.0:80>
```

Y nos registraremos en <http://localhost/register>, para así tener una cuenta de pruebas



Ahora tenemos una cuenta de pruebas lista para trabajar desde nuestros clientes de ejemplo, en mi caso, los datos para autenticarnos en la **API** serán:

- Nombre: Facebook Developer Circles
- E-Mail: gonzalo@devtotal.cl
- Password: Tutorial
- Client ID: 2
- Secret: h08zl23nLtqSZxpj9vnP7CdHTFWqCUgxX9gcMKWZ

5. Probando LaraAuth

Vale la pena recordar que los datos **Client ID** y **Client Secret** los obtuvimos cuando “registramos” nuestra aplicación. Para los ejemplos que implementaremos, todos harán en resumen lo siguiente:

1. Solicitar un **Token** para **acceder** y para **refrescar** (access_token y refresh_token) mediante una solicitud **POST** con nuestros datos de autenticación.
2. Consultar los **datos** de nuestra cuenta mediante una solicitud **POST** sin parámetros extras y con nuestro **Token** en un header.
3. Crear un **Pokémon** en nuestra cuenta mediante una solicitud **POST** con los datos de este como parámetros y con nuestro **Token** en un header.
4. Consultar los **Pokemones** de nuestra cuenta mediante una solicitud **POST** sin parámetros extras y con nuestro **Token** en un header.
5. Buscar una cuenta existente y una inexistente mediante una solicitud **POST** con la **ID** buscada como parámetro y con nuestro **Token** en un header.
6. Buscar un **Pokémon** existente y uno inexistente mediante una solicitud **POST** con la **ID** buscada y con nuestro **Token** en un header.
7. Refrescar nuestro **Token** de acceso mediante una solicitud **POST** con el **Token de refresco**, el **Client-ID** y **Client-Secret** como parámetros.

El **header** de nuestro **Token** al momento de enviarlo en las acciones 2 – 6, será “Authorization: Bearer **Token**”, en los ejemplos quedará todo más claro.

5.1 cURL

3. Crear un Pokémon en nuestra Cuenta:

```
curl -H "Content-Type: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6IjVmMWU4MTJiMWIxYTA4NWMDNDU3MDU3ZjRjNmEzZWZmMjQ1MDMxZWZhYjZjMmQ5MmRhMTgxNjI2NTg0YTliNDZiZGE1NWFlYjRlYzk1MDYwIn0.eyJhdWQiOiIiYiIiwianRpIjoibWYxZTgxMmIxYjFhMDg1YzQ0NTcwNTdmNGM2YTlnZmYyNDUwMzFlZmFiNmMyZDkyZGEwODE2MjY1ODRhOWI0NmJkYTU1YWViNGVjOTUwNjAiLCJpYXQiOiJlMDEwODgyNzYsIm5iZiI6MTUwMTE4ODI3NiwiZXhwIjoxNTMyNzI0Mjc2LCJzdWIiOiIzIiwic2NvcGVzIjpbIioiXX0.KsHymWqXLvTw-94Y7rdWj1_I86eNrpKR1UHsIEmND-Yyf8cp1JiVRQg7gYSgfG2IU1Yfu834-WnI5shoMKne1vtWkhPt0jVZleQS7_HurmfkZbDHBQh1RSJcYmpPFHvZBbwITHSYU-lwMdU7YIhiS6DKA4GatrN0r4mf-PYxU63h74ri2MBzaeyXvs3XUeII0u8x8ILRjDo8lCUJ0toTCqU7ZKQLrLGNSRLBuyVoY10aRvPaYF27kd0_9sVtwt7ZZ1G0Q i7HVRy_QS0x_VJu0_m-Ziw4JQ1Ur-hLcKtxNQNpHFV-K3oCD-9Q9sGzjIhHsxfeW81mWtpyOHmE_MBwRYSgBAsybqbPJSOGoknpUzWDQ7BaMD1L21T-rpgS2i1cr6rRF9K7Nra1Rc88ebZxhH1bWx0jmbTaB0wNqDStWCuErH4LcyNzK1lHpvhqV0uqV1GUcVRT03Zjs8KT1LsreLQA P2jYjUt9MdQH7qqotNu7oZiZF07WwZJWaI_nAfe_8uUs78f0rYwSHS69nM2kqx2S9M0E79roc1HSiEgbCy-WXfhLrvZgt8TXeFKKF3xisio-1YXs3ejCPD0008hXpIJJTGmP6wL81m1rpxvcMun9MAFRrRBjThqzArac8pE-Bis0ojtPdJnEHq85CELzuTI2DKCsL_KH1_sm9p1JjQ" -X POST -d '{"pokedex_id":123412,"apodo":"Jacinto","nombre":"Obama","nivel":1234123,"exp":324234}' http://localhost/api/user/pokemones/agregar
```

```
{ "pokedex_id":123412,"apodo":"Jacinto","nombre":"Obama","nivel":1234123,"exp":324234,"user_id":3,"updated_at":"2017-07-27 21:46:47","created_at":"2017-07-27 21:46:47","id":4}
```

4. Consultar los Pokémon de nuestra Cuenta

```
curl -H "Content-Type: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6IjVmMWU4MTJiMWIxYTA4NWMDNDU3MDU3ZjRjNmEzZWZmMjQ1MDMxZWZhYjZjMmQ5MmRhMTgxNjI2NTg0YTliNDZiZGE1NWFlYjRlYzk1MDYwIn0.eyJhdWQiOiIiYiIiwianRpIjoibWYxZTgxMmIxYjFhMDg1YzQ0NTcwNTdmNGM2YTlnZmYyNDUwMzFlZmFiNmMyZDkyZGEwODE2MjY1ODRhOWI0NmJkYTU1YWViNGVjOTUwNjAiLCJpYXQiOiJlMDEwODgyNzYsIm5iZiI6MTUwMTE4ODI3NiwiZXhwIjoxNTMyNzI0Mjc2LCJzdWIiOiIzIiwic2NvcGVzIjpbIioiXX0.KsHymWqXLvTw-94Y7rdWj1_I86eNrpKR1UHsIEmND-Yyf8cp1JiVRQg7gYSgfG2IU1Yfu834-WnI5shoMKne1vtWkhPt0jVZleQS7_HurmfkZbDHBQh1RSJcYmpPFHvZBbwITHSYU-lwMdU7YIhiS6DKA4GatrN0r4mf-PYxU63h74ri2MBzaeyXvs3XUeII0u8x8ILRjDo8lCUJ0toTCqU7ZKQLrLGNSRLBuyVoY10aRvPaYF27kd0_9sVtwt7ZZ1G0Q i7HVRy_QS0x_VJu0_m-Ziw4JQ1Ur-hLcKtxNQNpHFV-K3oCD-9Q9sGzjIhHsxfeW81mWtpyOHmE_MBwRYSgBAsybqbPJSOGoknpUzWDQ7BaMD1L21T-rpgS2i1cr6rRF9K7Nra1Rc88ebZxhH1bWx0jmbTaB0wNqDStWCuErH4LcyNzK1lHpvhqV0uqV1GUcVRT03Zjs8KT1LsreLQA P2jYjUt9MdQH7qqotNu7oZiZF07WwZJWaI_nAfe_8uUs78f0rYwSHS69nM2kqx2S9M0E79roc1HSiEgbCy-WXfhLrvZgt8TXeFKKF3xisio-1YXs3ejCPD0008hXpIJJTGmP6wL81m1rpxvcMun9MAFRrRBjThqzArac8pE-Bis0ojtPdJnEHq85CELzuTI2DKCsL_KH1_sm9p1JjQ" -X POST http://localhost/api/user/pokemones
```

```
[{"id":4,"user_id":3,"pokedex_id":123412,"apodo":"Jacinto","nombre":"Obama","nivel":1234123,"exp":324234,"created_at":"2017-07-27 21:46:47","updated_at":"2017-07-27 21:46:47"}]
```

5.1 cURL

5. Buscar una Cuenta existente y una no existente:

```
curl -H "Content-Type: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6IjVmMWU4MTJiMWIxYTA4NWw0NDU3MDU3ZjRjNmEzZWZmMjQ1MDMxZWZhYjZjMmQ5MmRhMTgxNjI2NTg0YTliNDZiZGE1NWFlYjRlYzYkMDYwIn0.eyJhdWQiOiIyIiwianRpIjoibWYxZTgxMmIxYjFhMDg1YzQ0NTcwNTdmNGM2YTNIzWYyNDUwMzFlZmFiNmMyZDkyZGExODE2MjY1ODRhOWI0NmJkYTU1YWVlNGVjOTUwNjAiLCJpYXQ0eIMDExODgyNzYsIm5iZiI6MTUwMTE4ODI3NiwiZXhwIjoxNTMyNzI0Mjc2LCJzdWIiOiIzIiwic2NvcGVzIjpbIioiXX0.KsHymWqXLvTw-94Y7rdWj1_I86eNrpKR1UHsIEmND-Yyf8cp1JiVRQg7gYSgfG2IU1Yfu834-WnI5shoMKne1vtWkhPt0jVZleQS7_HurmfkZbDHBQh1RSJcYmpPFHvZBBwITHSYU-lwMdU7YIhIS6DKA4GatrN0r4mf-PYxU63h74ri2MBzaeyXvs3XUeII0u8x8ILRjDo8lCUJ0toTCqU7ZKQLrLGNSRLBuyVoY10aRvPaYF27kd0_9sVtwt7ZZ1G0Q i7HVRy_QS0x_VJu0_m-Ziw4JQ1Ur-hLcKtxnQnpHFV-K3oCD-9Q9sGzjIhHsxfew81mwtpyOHmE_MBwRYSgBAsybqbPJSOGoknpUzWDQ7BaMD1L21T-rpgS2i1cr6rRF9K7Nra1Rc88ebZxhH1bWx0jmbTaB0wNqDStWCuErH4LcyNzK11HpvhqV0uqV1GUcVRT03Zjs8KT1LsreLQA P2jYjUt9MdQH7qqotNu7oZiZF07WwZJWaI_nAfe_8uUs78f0rYwSHS69nM2kqx2S9M0E79roc1HSiEgbCy-WXfhLrvZgt8TXeFKF3xisio-1YXs3ejCPD0008hXpIJJTGmP6wL81m1rpxvcMun9MAFRrRBjThqzArac8pE-Bis0ojtPdJnEHq85CELzuTIZDKCsL_KH1_sm9plJjQ" -X POST -d '{"user_id":1}' http://localhost/api/user/buscar
```

```
curl -H "Content-Type: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6IjVmMWU4MTJiMWIxYTA4NWw0NDU3MDU3ZjRjNmEzZWZmMjQ1MDMxZWZhYjZjMmQ5MmRhMTgxNjI2NTg0YTliNDZiZGE1NWFlYjRlYzYkMDYwIn0.eyJhdWQiOiIyIiwianRpIjoibWYxZTgxMmIxYjFhMDg1YzQ0NTcwNTdmNGM2YTNIzWYyNDUwMzFlZmFiNmMyZDkyZGExODE2MjY1ODRhOWI0NmJkYTU1YWVlNGVjOTUwNjAiLCJpYXQ0eIMDExODgyNzYsIm5iZiI6MTUwMTE4ODI3NiwiZXhwIjoxNTMyNzI0Mjc2LCJzdWIiOiIzIiwic2NvcGVzIjpbIioiXX0.KsHymWqXLvTw-94Y7rdWj1_I86eNrpKR1UHsIEmND-Yyf8cp1JiVRQg7gYSgfG2IU1Yfu834-WnI5shoMKne1vtWkhPt0jVZleQS7_HurmfkZbDHBQh1RSJcYmpPFHvZBBwITHSYU-lwMdU7YIhIS6DKA4GatrN0r4mf-PYxU63h74ri2MBzaeyXvs3XUeII0u8x8ILRjDo8lCUJ0toTCqU7ZKQLrLGNSRLBuyVoY10aRvPaYF27kd0_9sVtwt7ZZ1G0Q i7HVRy_QS0x_VJu0_m-Ziw4JQ1Ur-hLcKtxnQnpHFV-K3oCD-9Q9sGzjIhHsxfew81mwtpyOHmE_MBwRYSgBAsybqbPJSOGoknpUzWDQ7BaMD1L21T-rpgS2i1cr6rRF9K7Nra1Rc88ebZxhH1bWx0jmbTaB0wNqDStWCuErH4LcyNzK11HpvhqV0uqV1GUcVRT03Zjs8KT1LsreLQA P2jYjUt9MdQH7qqotNu7oZiZF07WwZJWaI_nAfe_8uUs78f0rYwSHS69nM2kqx2S9M0E79roc1HSiEgbCy-WXfhLrvZgt8TXeFKF3xisio-1YXs3ejCPD0008hXpIJJTGmP6wL81m1rpxvcMun9MAFRrRBjThqzArac8pE-Bis0ojtPdJnEHq85CELzuTIZDKCsL_KH1_sm9plJjQ" -X POST -d '{"user_id":99999}' http://localhost/api/user/buscar
```

```
{"id":1,"name":"Gonzalo","email":"evd.totalblood@gmail.com","created_at":"2017-07-22 10:40:56","updated_at":"2017-07-22 10:40:56"}
```

```
Error: ID no encontrado
```

5.1 cURL

6. Buscar un Pokémon existente y uno inexistente:

```
curl -H "Content-Type: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6IjVmMWU4MTJiMWIxYTA4NWw0NDU3MDU3ZjRjNmEzZWZmMjQ1MDMxZWZhYjZjMmQ5MmRhMTgxNjI2NTg0YTliNDZiZGE1NWFlYjRlYzk1MDYwIn0.eyJhdWQiOiIyIiwianRpIjoibWYxZTgxMmIxYjFhMDg1YzQ0NTcwNTdmNGM2YTNIzWYyNDUwMzFlZmFiNmMyZDkyZGExODE2MjY1ODRhOWI0NmJkYTU1YWViNGVjOTUwNjAiLCJpYXQiOjE1MDExODgyNzYsIm5iZiI6MTUwMTE4ODI3NiwiZXhwIjoxNTMyNzI0Mjc2LCJzdWIiOiIzIiwic2NvcGVzIjpbIioiXX0.KsHymWqXLvTw-94Y7rdWj1_I86eNrpKR1UHsIEmND-Yyf8cp1JiVRQg7gYSgfG2IU1Yfu834-WnI5shoMKne1vtWkhPt0jVZleQS7_HurmfkZbDHBQh1RSJcYmpPFHvZBBwITHSYU-lwMdU7YIhIS6DKA4GatrN0r4mf-PYxU63h74ri2MBzaeyXvs3XUeII0u8x8ILRjDo8lCUJ0toTCqU7ZKQLrLGNSRLBuyVoY10aRvPaYF27kd0_9sVtwt7ZZ1G0Q i7HVRy_QS0x_VJu0_m-Ziw4JQ1Ur-hLcKtxnQnpHFV-K3oCD-9Q9sGzjIhHsxfew81mwtpyOHmE_MBwRYSgBAsybqbPJSOGoknpUzWDQ7BaMD1L21T-rpgS2i1cr6rRF9K7Nra1Rc88ebZxhH1bWx0jmbTaB0wNqDStWCuErH4LcyNzK11HpvhqV0uqV1GUcVRT03Zjs8KT1LsreLQA P2jYjUt9MdQH7qqotNu7oZiZF07WwZJWaI_nAfe_8uUs78f0rYwSHS69nM2kqx2S9M0E79roc1HSIEgbCy-WXfhLrvZgt8TXeFKF3xisio-1YXs3ejCPD0008hXpIJJTGmP6wL81m1rpxvcMun9MAFRrRBjThqzArac8pE-Bis0ojtPdJnEHq85CELzuTIZDKCsL_KH1_sm9plJjQ" -X POST -d '{"pokemon_id":1}' http://localhost/api/pokemon/buscar
```

```
curl -H "Content-Type: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6IjVmMWU4MTJiMWIxYTA4NWw0NDU3MDU3ZjRjNmEzZWZmMjQ1MDMxZWZhYjZjMmQ5MmRhMTgxNjI2NTg0YTliNDZiZGE1NWFlYjRlYzk1MDYwIn0.eyJhdWQiOiIyIiwianRpIjoibWYxZTgxMmIxYjFhMDg1YzQ0NTcwNTdmNGM2YTNIzWYyNDUwMzFlZmFiNmMyZDkyZGExODE2MjY1ODRhOWI0NmJkYTU1YWViNGVjOTUwNjAiLCJpYXQiOjE1MDExODgyNzYsIm5iZiI6MTUwMTE4ODI3NiwiZXhwIjoxNTMyNzI0Mjc2LCJzdWIiOiIzIiwic2NvcGVzIjpbIioiXX0.KsHymWqXLvTw-94Y7rdWj1_I86eNrpKR1UHsIEmND-Yyf8cp1JiVRQg7gYSgfG2IU1Yfu834-WnI5shoMKne1vtWkhPt0jVZleQS7_HurmfkZbDHBQh1RSJcYmpPFHvZBBwITHSYU-lwMdU7YIhIS6DKA4GatrN0r4mf-PYxU63h74ri2MBzaeyXvs3XUeII0u8x8ILRjDo8lCUJ0toTCqU7ZKQLrLGNSRLBuyVoY10aRvPaYF27kd0_9sVtwt7ZZ1G0Q i7HVRy_QS0x_VJu0_m-Ziw4JQ1Ur-hLcKtxnQnpHFV-K3oCD-9Q9sGzjIhHsxfew81mwtpyOHmE_MBwRYSgBAsybqbPJSOGoknpUzWDQ7BaMD1L21T-rpgS2i1cr6rRF9K7Nra1Rc88ebZxhH1bWx0jmbTaB0wNqDStWCuErH4LcyNzK11HpvhqV0uqV1GUcVRT03Zjs8KT1LsreLQA P2jYjUt9MdQH7qqotNu7oZiZF07WwZJWaI_nAfe_8uUs78f0rYwSHS69nM2kqx2S9M0E79roc1HSIEgbCy-WXfhLrvZgt8TXeFKF3xisio-1YXs3ejCPD0008hXpIJJTGmP6wL81m1rpxvcMun9MAFRrRBjThqzArac8pE-Bis0ojtPdJnEHq85CELzuTIZDKCsL_KH1_sm9plJjQ" -X POST -d '{"pokemon_id":99999}' http://localhost/api/pokemon/buscar
```

```
{"id":1,"user_id":1,"pokedex_id":12345,"apodo":"ElNene","nombre":"Square","nivel":123,"exp":1234567,"created_at":"2017-07-23 18:03:38","updated_at":"2017-07-23 18:03:38"}
```

```
Error: ID no encontrado
```


5.2 Python

```
import requests
import json

token = ""
refresh_token = ""
client_id = 2
client_secret = "h08z123nLtqSZxpJ9vnP7CdHTFWqCUgxX9gcMKWZ"

def ObtenerToken(user, password):
    global token
    global refresh_token

    url = 'http://localhost/oauth/token'
    payload = {
        "grant_type": "password", "client_id": client_id, "client_secret": client_secret, "username": user, "password": password, "scope": ""
    }
    r = requests.post(url, data=payload)
    d = json.loads(r.text)

    token = d['access_token']
    refresh_token = d['refresh_token']

def MiCuenta():
    if token == "":
        print("Consigue un Token de Acceso antes de usar esta opción")
        return

    url = 'http://localhost/api/user'
    r = requests.post(url, headers={"Authorization": "Bearer "+token})
    print(r.text)

def CrearPokemon(id_pokedex, apodo, nombre, nivel, exp):
    if token == "":
        print("Consigue un Token de Acceso antes de usar esta opción")
        return

    url = 'http://localhost/api/user/pokemones/agregar'
    payload = {
        "pokedex_id": id_pokedex, "apodo": apodo, "nombre": nombre, "nivel": nivel, "exp": exp
    }
    r = requests.post(url, data=payload, headers={"Authorization": "Bearer "+token})
    print(r.text)

def MisPokemones():
    if token == "":
        print("Consigue un Token de Acceso antes de usar esta opción")
        return

    url = 'http://localhost/api/user/pokemones'
    r = requests.post(url, headers={"Authorization": "Bearer "+token})
    print(r.text)

def BuscarUsuario(user_id):
    if token == "":
        print("Consigue un Token de Acceso antes de usar esta opción")
        return

    url = 'http://localhost/api/user/buscar'
    payload = {"user_id": user_id}
    r = requests.post(url, data=payload, headers={"Authorization": "Bearer "+token})
    print(r.text)
```

5.2 Python

```
def BuscarPokemon(pokemon_id):

    if token == "":
        print("Consigue un Token de Acceso antes de usar esta opción")
        return

    url = 'http://localhost/api/pokemon/buscar'
    payload = {"pokemon_id":pokemon_id}
    r = requests.post(url, data=payload, headers={"Authorization":"Bearer "+token})
    print(r.text)

def RefrescarToken():
    global token
    global refresh_token

    url = 'http://localhost/oauth/token'
    payload =
{"grant_type":"refresh_token","client_id":client_id,"client_secret":client_secret,"refresh_token
":refresh_token,"scope":"*"}
    r = requests.post(url, data=payload)
    d = json.loads(r.text)

    token = d['access_token']
    refresh_token = d['refresh_token']

def menu(opcion):

    if opcion == "1":
        ObtenerToken(input("Ingresar Usuario: "), input("Ingresar Contraseña: "))
    elif opcion == "2":
        MiCuenta()
    elif opcion == "3":
        CrearPokemon(input("ID Pokedex: "), input("Apodo: "), input("Nombre: "),
input("Nivel: "), input("Exp: "))
    elif opcion == "4":
        MisPokemones()
    elif opcion == "5":
        BuscarUsuario(input("Ingresar ID: "))
    elif opcion == "6":
        BuscarPokemon(input("Ingresar ID:"))
    elif opcion == "7":
        RefrescarToken()
    elif opcion == "8":
        print("Tutorial Creado por Gonzalo de DevTotal.Cl para Facebook Developer
Circles")
        return 0

    print("Escoga una Opción:")

    print("1. Obtener un Token")
    print("2. Datos de mi Cuenta")
    print("3. Crear un Pokémon")
    print("4. Consultar mis Pokémons")
    print("5. Buscar un Usuario por su ID")
    print("6. Buscar un Pokémon por su ID")
    print("7. Refrescar Token")
    print("8. Salir")

    return menu(input("\n"))

menu(0)
```

5.3 Próximamente

Cualquier **API** construida con este tutorial puede ser consumida de distintas maneras, no sólo con algún cliente en **Python** o con comandos **cURL**, más adelante integraré más ejemplos a este tutorial siempre y cuando tenga tiempo para hacerlo, por otro lado, sé que no ahondé mucho en los clientes ya que mi objetivo principal era explicar el cómo podemos crear herramientas para probar nuestra **API** desarrollada, por lo mismo, cualquier duda que puedas tener, sólo contáctame, espero pronto hacer un **MeetUp** sobre este tema.

Este tutorial ha sido desarrollado para Facebook Developer Circles

Autor: [Gonzalo Fernández](#) – Full Stack Developer.



<https://github.com/DevTotal>



<https://linkedin.com/in/DevTotal/>

6. Bibliografía

1. Joy Of Data - [OAuth 2.0 for Google \(Analytics\) API with Python Explained](#)
2. Laravel Documentación Oficial - [Laravel Passport](#)
3. Aaron Parecki - [OAuth2 Simplified](#)
4. OAuth2 Core - [Sitio Oficial](#)
5. Mattstauffer.co - [Introducing Laravel Passport](#)
6. Laravel News - [Password Grant Types](#)