

# Sistema de Autenticación API Rest con Laravel 5.6

## — Parte 1



Claudio Vallejo

Jul 9, 2018 · 6 min read

Este tiempo hemos estado desarrollando aplicaciones y servicios que permitan generar conectividad entre éstas y Laravel. En una de las muchísimas lecturas y recopilación de posibilidades, me encontré con una serie de 3 publicaciones (excelentemente bien escritas por [Alfredo Barron](#) a quién recomiendo leer y seguir) que permiten, de una forma simple y directa, comenzar con el mundo API Rest con Laravel.



. . .

### 1. Partir con una instalación fresca y nueva de laravel

Para ello, entendiendo que ya manejas algo del framework y su documentación (más info: <https://laravel.com/docs/5.6>) debes escribir en tu ventana de la terminal (asegúrate de estar en la carpeta donde quieres que se genere este nuevo proyecto):

```
laravel new apiAuth
```

Puedes escoger el nombre que quieras... en este caso utilizaremos “apiAuth” por ser muy descriptivo ;)

Recuerda que ahora debes ingresar a tu aplicación y configurar las variables de entorno (.env).

## 2. Instala el paquete de autenticación API — Passport

Este paquete es fundamental ya que, como su nombre lo indica, Laravel posee un sistema tradicional de autenticación pero, para el caso del desarrollo de una API, Laravel ofrece algo específico. Laravel Passport.

Las APIs utilizan típicamente tokens para autenticar usuarios pero no para mantener las sesiones entre los requests. Laravel ayuda a que la autenticación a través de la API sea muy simple con Laravel Passport, sistema que provee una implementación total de OAuth2 para tu aplicación de Laravel.

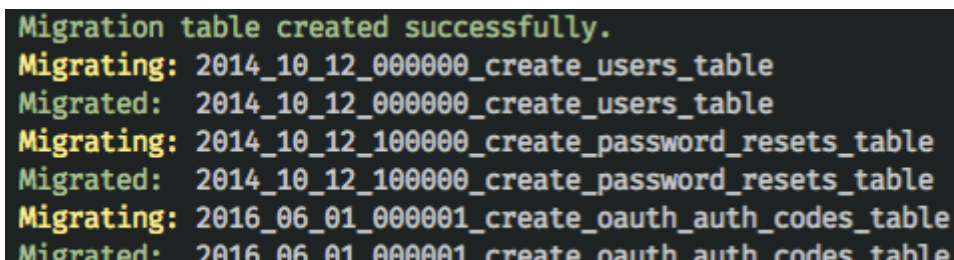
Seguiremos los pasos indicados en la documentación oficial (<https://laravel.com/docs/5.6/passport>)

### a. Comienza la instalación a través del manejador de paquetes, composer, a través del comando:

```
composer require laravel/passport
```

### b. Realizar la migración

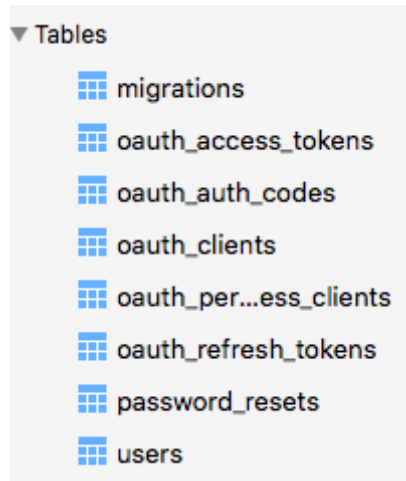
```
php artisan migrate
```



```
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_000001_create_oauth_auth_codes_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_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
```

output de la terminal



*La migración generará las tablas que necesitará tu aplicación para almacenar los clientes y sus token de acceso. Al costado izquierdo se pueden ver todas las tablas que genera el comando.*

### c. Instalación y generación de las llaves

Luego, debes ejecutar el comando `passport:install`. Este comando creará las llaves de encriptación necesarias para generar los tokens de acceso. Adicionalmente el comando creará el “personal access” y “password grant” de los clientes que se usarán para generar los tokens de acceso:

```
php artisan passport:install
```

### d. Configurar Passport

Luego de ejecutar este comando, hay que agregar el trait `Laravel\Passport\HasApiTokens` al modelo `App\User`. Este Trait provee algunos métodos de ayuda a tu modelo que te permitirán inspeccionar al token y scope de los usuarios autenticados:

```
<?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;

    protected $fillable = [
        'name', 'email', 'password',
    ];

    protected $hidden = [
        'password', 'remember_token',
    ];
}
```

Luego tu deberás llamar al método **Passport::routes** dentro del método **boot** en tu **app/Providers/AuthServiceProvider** . Este método registrará las rutas necesarias para emitir tokens de acceso y revocar tokens de acceso, clientes y tokens de acceso personal:

```
<?php

namespace App\Providers;

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

class AuthServiceProvider extends ServiceProvider
{
    protected $policies = [
        'App\Model' => 'App\Policies\ModelPolicy',
    ];

    public function boot()
    {
        $this->registerPolicies();

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

Para terminar en tu archivo de configuración `config/auth.php`, tu deberás ajustar la opción del `driver` de la autenticación de la `api` en el `'guards'` a `passport`. Esto le indicará a tu aplicación que use el `TokenGuard` de `Passport` al autenticar las solicitudes API entrantes:

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

### 3. Creación de las rutas de la API

A continuación lo que se requiere es la creación de las rutas necesarias para tu `api`. Para ello debes ingresar al servicio de rutas que Laravel provee en forma exclusiva para una `api` `routes/api.php`

```
<?php  
  
use Illuminate\Http\Request;  
  
Route::group(['prefix' => 'auth'], function () {  
    Route::post('login', 'AuthController@login');  
    Route::post('signup', 'AuthController@signup');  
  
    Route::group(['middleware' => 'auth:api'], function() {  
        Route::get('logout', 'AuthController@logout');  
        Route::get('user', 'AuthController@user');  
    });  
});
```

### 4. Creación del controlador para la autenticación

Al visualizar las rutas que hemos generado más arriba podrás notar que se especifica un controlador que aún no hemos creado. Para ello deberemos crear dicho controlador a través del comando:

```
php artisan make:controller AuthController
```

Luego, deberemos crear cada uno de los métodos que estamos llamando:

*signup / login / logout / user*

Para ello escribiremos dentro del controlador lo siguiente:

```
<?php

namespace App\Http\Controllers;

use App\User;
use Carbon\Carbon;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class AuthController extends Controller
{

    public function signup(Request $request)
    {
        $request->validate([
            'name'      => 'required|string',
            'email'     => 'required|string|email|unique:users',
            'password'  => 'required|string|confirmed',
        ]);

        $user = new User([
            'name'      => $request->name,
            'email'     => $request->email,
            'password'  => bcrypt($request->password),
        ]);

        $user->save();

        return response()->json([
            'message' => 'Successfully created user!'], 201);
    }

    public function login(Request $request)
    {
        $request->validate([
            'email'      => 'required|string|email',
            'password'   => 'required|string',
            'remember_me' => 'boolean',
        ]);

        $credentials = request(['email', 'password']);
        if (!Auth::attempt($credentials)) {
```

```

        return response()->json([
            'message' => 'Unauthorized'], 401);
    }

    $user = $request->user();
    $tokenResult = $user->createToken('Personal Access Token');
    $token = $tokenResult->token;

    if ($request->remember_me) {
        $token->expires_at = Carbon::now()->addWeeks(1);
    }

    $token->save();

    return response()->json([
        'access_token' => $tokenResult->accessToken,
        'token_type'    => 'Bearer',
        'expires_at'    => Carbon::parse(
            $tokenResult->token->expires_at)
            ->toDateTimeString(),
    ]);
}

public function logout(Request $request)
{
    $request->user()->token()->revoke();

    return response()->json(['message' =>
        'Successfully logged out']);
}

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

```

. . .

Desde este punto en adelante, tu aplicación está plenamente funcional y operativa. Solo necesitas correr tu sitio en el ambiente local que tengas (homestead, valet, single server o cualquier otro entorno). En el caso de un servidor simple o directo, basta con escribir el comando:

```
php artisan serve
```

• • •

Para las pruebas, utilicé Postman (tiene opción para extensión en chrome o como app).  
Más info: <https://www.getpostman.com/>

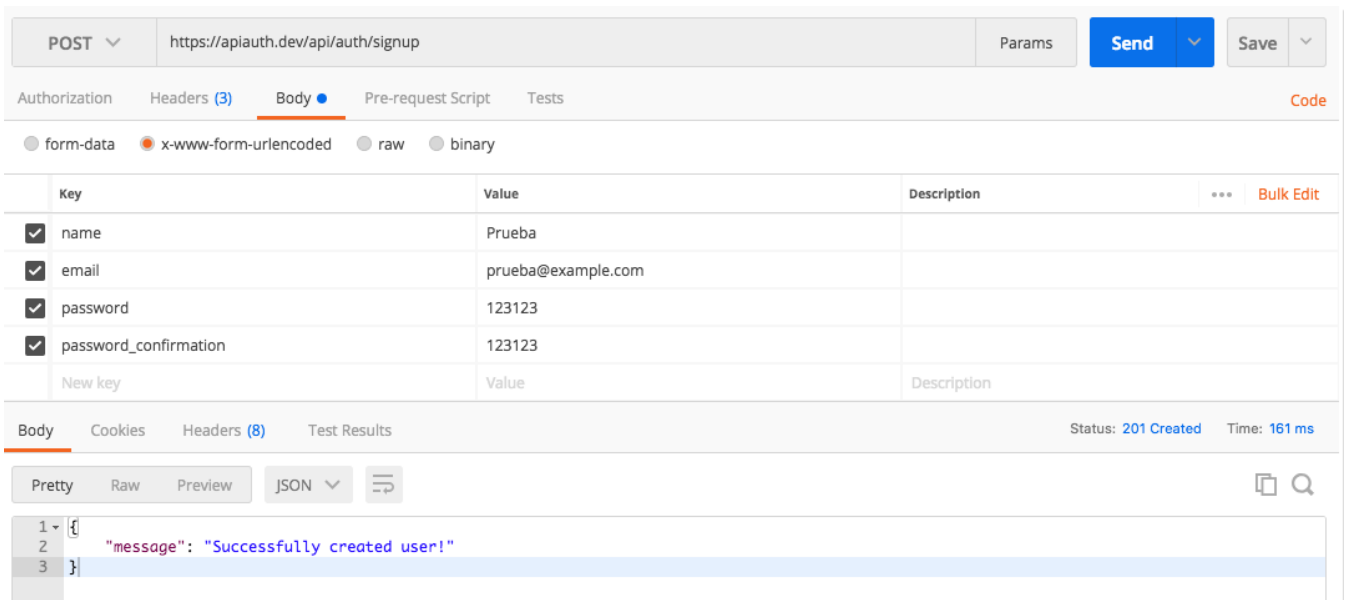
Para la correcta utilización, hay que configurar las siguientes dos cabeceras:

Content-Type: application/json  
X-Requested-With: XMLHttpRequest



postman con las cabeceras adicionales

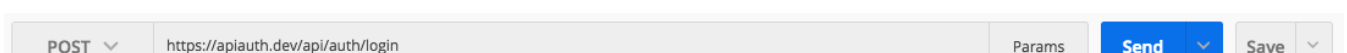
## Creación de usuario / signup



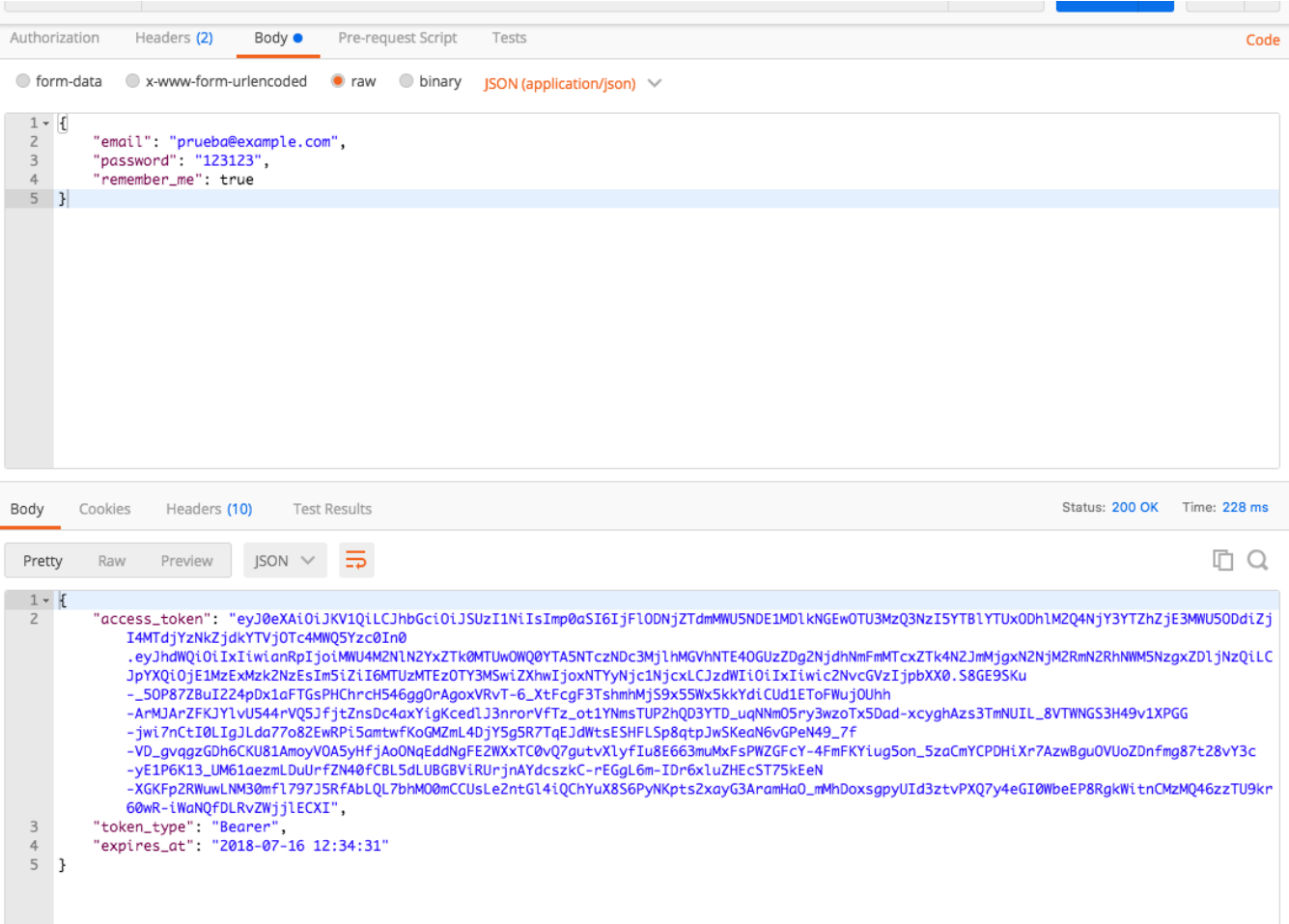
signup

## Ingreso de usuario / login

El “token\_type” entregado es Bearer (más info).

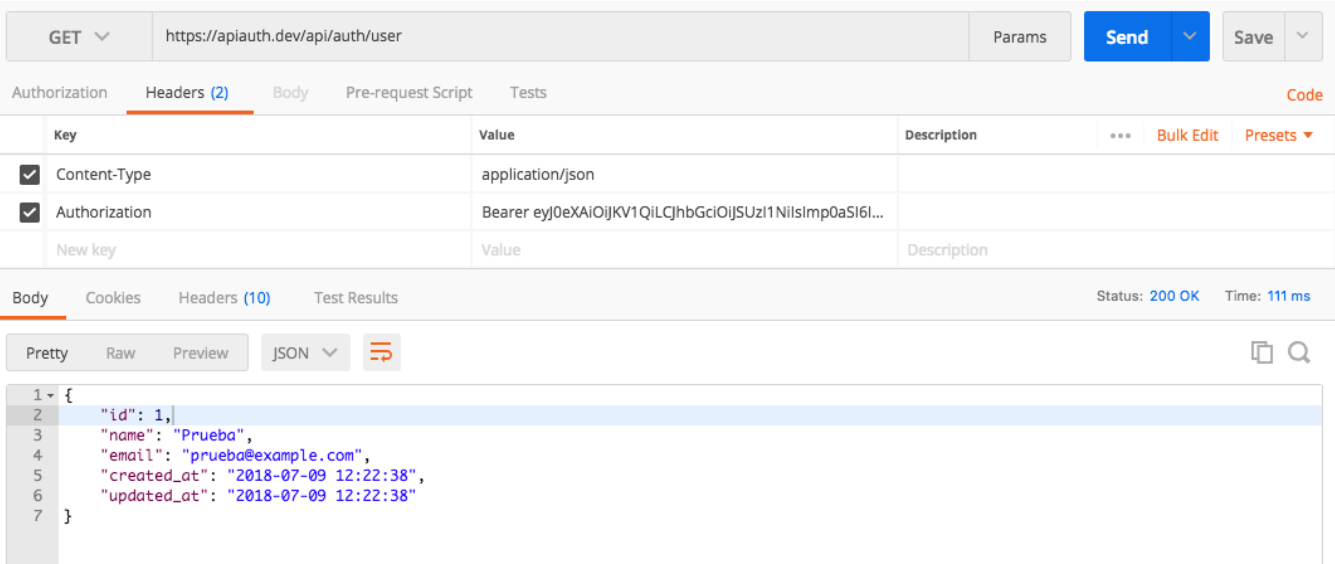






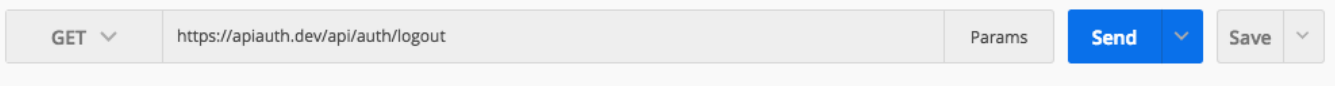
login

## Visualización de usuario / user



user

## Cerrar sesión usuario / logout



The screenshot shows a REST client interface with tabs for Authorization, Headers (1), Body, Pre-request Script, and Tests. The Headers tab is active, displaying a table with one header: 'Authorization' with a checked checkbox, a value 'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0...'. Below the table are fields for 'New key' and 'Value'. The bottom section shows the 'Body' tab with a JSON response: 

```
{  "message": "Successfully logged out"}
```

 The status bar indicates 'Status: 200 OK' and 'Time: 125 ms'.

logout

*Con esto hemos terminado la primera parte. En las próximas publicaciones iré adicionando funcionalidades.*

. . .

## Serie: Sistema de Autenticación API Rest con Laravel 5.6

- Parte 1 : Instalación y configuración
- Parte 2 : Confirmación de cuenta y notificaciones
- Parte 3 : Generar un avatar
- Parte 4 : Restablecer contraseña
- Parte 5 : Enviar notificaciones con espera en Redis

. . .

Gracias por tu lectura.

Si te ha gustado, podrías darme un aplauso y seguirme :)





Puedes compartir esta publicación en tus redes sociales

**Tweets with replies by Claudio Vallejo (@cvallejo) | Twitter**

The latest Tweets and replies from Claudio Vallejo (@cvallejo). Wine, programmer, diver & photographer lover...

twitter.com

[Laravel](#) [API](#) [Español](#) [Rest Api](#) [Oauth2](#)

[About](#) [Help](#) [Legal](#)

Get the Medium app

