API CREDENTIALS

oSMAN khoder – SENIOR DEVELOPER

2018

Contents

[ISSUING ACCESS TOKENS 2](#_Toc516666231)

[MANGE CLIENTS 2](#_Toc516666232)

[JSON API 2](#_Toc516666233)

[GET CLIENT 2](#_Toc516666234)

[CREATE CLIENT 2](#_Toc516666235)

[UPDATE CLIENT 2](#_Toc516666236)

[DELETE CLIENT 3](#_Toc516666237)

[REQUESTING TOKENS 3](#_Toc516666238)

[REDIRECTING FOR AUTHORIZATION 3](#_Toc516666239)

[APPROVING THE REQUEST 3](#_Toc516666240)

[CONVERTING AUTHORIZATION CODE TO ACCESS TOKENS 3](#_Toc516666241)

[VERIFY TOKENS 4](#_Toc516666242)

[REFRESHING TOKENS 5](#_Toc516666243)

[Password Grant Tokens 5](#_Toc516666244)

[Creating A Password Grant Client 5](#_Toc516666245)

[Requesting Tokens 5](#_Toc516666246)

[Client Credentials Grant Tokens 6](#_Toc516666247)

# [ISSUING](https://laravel.com/docs/5.6/passport#issuing-access-tokens) ACCESS TOKENS

Using OAuth2 with authorization codes is how most developers are familiar with OAuth2. When using authorization codes, a client application will redirect a user to your server where they will either approve or deny the request to issue an access token to the client.

## MANGE CLIENTS

**To create a Client you have to do the following:**

* Register a user on the api credentials web site.
* Login
* Press Create new Client

## JSON API

### GET CLIENT

#### GET /oauth/clients

This route returns all of the clients for the authenticated user. This is primarily useful for listing all of the user's clients so that they may edit or delete them

### CREATE CLIENT

#### POST /oauth/clients

This route is used to create new clients. It requires two pieces of data: the client's name and a redirect URL. The redirect URL is where the user will be redirected after approving or denying a request for authorization.

When a client is created, it will be issued a client ID and client secret. These values will be used when requesting access tokens from your application. The client creation route will return the new client instance.

### UPDATE CLIENT

#### PUT /oauth/clients/{client-id}

This route is used to update clients. It requires two pieces of data: the client's name and a redirect URL. The redirect URL is where the user will be redirected after approving or denying a request for authorization. The route will return the updated client instance.

### DELETE CLIENT

#### DELETE /oauth/clients/{client-id}

This route is used to delete clients

## REQUESTING TOKENS

### REDIRECTING FOR AUTHORIZATION

Once a client has been created, developers may use their client ID and secret to request an authorization code and access token from your application. First, the consuming application should make a redirect request to your application's /oauth/authorize route like so:

Laravel way:

Route::get('/redirect', function () {

$query = http\_build\_query([

'client\_id' => 'client-id',

'redirect\_uri' => 'http://example.com/callback',

'response\_type' => 'code',

'scope' => '',

]);

return redirect('http://localhost:8000.com/oauth/authorize?' . $query);

});

Or simply call this url:

<http://localhost:8000.com/oauth/authorize?client_id=id&response_type=code&redirect_uri=http://example.com/callback>

### APPROVING THE REQUEST

When receiving authorization requests, Passport will automatically display a template to the user allowing them to approve or deny the authorization request. If they approve the request, they will be redirected back to the redirect\_uri  that was specified by the consuming application. The redirect\_uri  must match the redirect URL that was specified when the client was created.

### CONVERTING AUTHORIZATION CODE TO ACCESS TOKENS

If the user approves the authorization request, they will be redirected back to the consuming application. The consumer should then issue a POST request to your application to request an access token. The request should include the authorization code that was issued by your application when the user approved the authorization request. In this example, we'll use the Guzzle HTTP library to make the POST request:

Route::get('/callback', function (Request $request) {

$http = new GuzzleHttp\Client;

$response = $http->post('http://your-app.com/oauth/token', [

'form\_params' => [

'grant\_type' => 'authorization\_code',

'client\_id' => 'client-id',

'client\_secret' => 'client-secret',

'redirect\_uri' => 'http://example.com/callback',

'code' => $request->code,

],

]);

return json\_decode((string) $response->getBody(), true);

});

This /oauth/token route will return a JSON response containing access\_token, refresh\_token, and expires\_in attributes. The expires\_in attribute contains the number of seconds until the access token expires.

After this request new block (Authorized Applications) shown in the dashboard. You can at any time revoke an app (client). When a client has revoked, all of her access\_token  will not work.

Note: *code* is the *code* that was generated when you approve a client authorization request.

### VERIFY TOKENS

To verify the access token get’s in the previews request you have to call /api/user  with the bearer token in the header like the follow ajax call.

|  |
| --- |
| $.ajax({ |
| url: "http://your-app:8080/api/user", |
| type: 'GET',  type: 'application/json', |
| success: function(user, status) { |
| return console.log("The returned user", user); |
| }, |
| beforeSend: function(xhr, settings) {  xhr.setRequestHeader('Authorization','Bearer ' + tokenString );  } //set tokenString before send |
| });  *This request return the user (id, name, username, created\_at, updated\_at)* |

## REFRESHING TOKENS

If your application issues short-lived access tokens, users will need to refresh their access tokens via the refresh token that was provided to them when the access token was issued. In this example, we'll use the Guzzle HTTP library to refresh the token:

$http = new GuzzleHttp\Client;

$response = $http->post('http://your-app.com/oauth/token', [

'form\_params' => [

'grant\_type' => 'refresh\_token',

'refresh\_token' => 'the-refresh-token',

'client\_id' => 'client-id',

'client\_secret' => 'client-secret',

'scope' => '',

],

]);

return json\_decode((string) $response->getBody(), true);

This /oauth/token route will return a JSON response containing access\_token, refresh\_token, and expires\_in attributes. The expires\_in attribute contains the number of seconds until the access token expires.

# [Password Grant Tokens](https://laravel.com/docs/5.6/passport#password-grant-tokens)

The OAuth2 password grant allows your other first-party clients, such as a mobile application, to obtain an access token using an e-mail address / username and password. This allows you to issue access tokens securely to your first-party clients without requiring your users to go through the entire OAuth2 authorization code redirect flow.

## Creating A Password Grant Client

Before your application can issue tokens via the password grant, you will need to create a password grant client. You may do this using the passport:client command with the --password option. If you have already run the passport:install command, you do not need to run this command:

$ php artisan passport:client --password

## Requesting Tokens

Once you have created a password grant client, you may request an access token by issuing a POST request to the /oauth/token route with the user's email address and password. Remember, this route is already registered by the Passport::routes method so there is no need to define it manually. If the request is successful, you will receive an access\_token and refresh\_token in the JSON response from the server:

$http = new GuzzleHttp\Client;

$response = $http->post('http://your-app.com/oauth/token', [

'form\_params' => [

'grant\_type' => 'password',

'client\_id' => 'client-id',

'client\_secret' => 'client-secret',

'username' => 'taylor@laravel.com',

'password' => 'my-password',

'scope' => '',

],

]);

return json\_decode((string) $response->getBody(), true);

# [Client Credentials Grant Tokens](https://laravel.com/docs/5.6/passport#client-credentials-grant-tokens)

The client credentials grant is suitable for machine-to-machine authentication. For example, you might use this grant in a scheduled job which is performing maintenance tasks over an API.

To retrieve a token, make a request to the oauth/token endpoint:

$guzzle = new GuzzleHttp\Client;

$response = $guzzle->post('http://your-app.com/oauth/token', [

'form\_params' => [

'grant\_type' => 'client\_credentials',

'client\_id' => 'client-id',

'client\_secret' => 'client-secret',

'scope' => 'your-scope',

],

]);

return json\_decode((string) $response->getBody(), true)['access\_token'];