# SQL

Execute the following SQL file to create the tables and populate the demo information:

\_db\BK\_LOCAL\_2014-06-12\_OAUTH.sql

# Laravel 4 Package Installer

This package allows for quick and easy installation of supported Laravel 4 packages. This packages installs packages and adds any necessary ServiceProviders and Aliases.

## Installing Package Installer

This package installer is available through packagist and composer.

Add "rtablada/package-installer": "dev-master" to your composer.json or run composer require rtablada/package-installer.

*RCS: Typed “dev-master” for the version*

Then you have to add "Rtablada\PackageInstaller\PackageInstallerServiceProvider" to your list of providers in your app/config/app.php. Luckily, this will be one of the last package Aliases you will ever have to add manually!

"require": {

"laravel/framework": "4.2.\*"

,"rtablada/package-installer": "dev-master"

},

<https://github.com/lucadegasperi/oauth2-server-laravel>

php artisan package:install lucadegasperi/oauth2-server-laravel

# PHP OAuth 2.0 Server for Laravel

A wrapper package for the standards compliant OAuth 2.0 authorization server and resource server written in PHP by the [League of Extraordinary Packages](http://www.thephpleague.com).

The package assumes you have a good-enough knowledge of the principles behind the [OAuth 2.0 Specification](http://tools.ietf.org/html/rfc6749).

## Package Installation

### With Laravel Package Installer

The easiest way to install this package is via [Laravel Package Installer](https://github.com/rtablada/package-installer), this will set all the service providers and aliases for you. Run this artisan command to install the package:

php artisan package:install lucadegasperi/oauth2-server-laravel

### Manual Install

alternatively, you can manually install the package via composer. add the following line to your composer.json file:

"lucadegasperi/oauth2-server-laravel": "1.0.x"

Add this line of code to the providers array located in your app/config/app.php file:

'LucaDegasperi\OAuth2Server\OAuth2ServerServiceProvider',

And this lines to the aliases array:

'AuthorizationServer' => 'LucaDegasperi\OAuth2Server\Facades\AuthorizationServerFacade',

'ResourceServer' => 'LucaDegasperi\OAuth2Server\Facades\ResourceServerFacade',

### Configuration

In order to use the OAuth2 server publish its configuration first

php artisan config:publish lucadegasperi/oauth2-server-laravel

Afterwards edit the file app/config/packages/lucadegasperi/oauth2-server-laravel/oauth2.php to suit your needs.

### Migrations

This package comes with all the migrations you need to run a full featured oauth2 server. Run:

php artisan migrate --package="lucadegasperi/oauth2-server-laravel"

## Issuing an access token

You can use different grant types to issue an access token depending on which suits your use cases. A detailed description of the different grant types can be found [here](https://github.com/php-loep/oauth2-server/wiki/Which-OAuth-2.0-grant-should-I-use%3F)

A client should make a POST request with the appropriate parameters (depending on the grant type used) to an access token endpoint like the one defined here. This package will take care of all the fuss for you.

Route::post('oauth/access\_token', function()

{

return AuthorizationServer::performAccessTokenFlow();

});

### Authorization Code flow

The most common grant type is the authorization\_code. It's also the longest to setup, but don't worry, it will be easy.

First the client must obtain the authorization (not an access token) from the resource owner to access the resources on its behalf. The client application should redirect the user to the authorization page with the correct query string parameters, for example:

https://www.example.com/oauth/authorize?

client\_id=the\_client\_id&

redirect\_uri=client\_redirect\_uri&

response\_type=code&

scope=scope1,scope2&

state=1234567890

Route::get('/oauth/authorize', array('before' => 'check-authorization-params|auth', function()

{

// get the data from the check-authorization-params filter

$params = Session::get('authorize-params');

// get the user id

$params['user\_id'] = Auth::user()->id;

// display the authorization form

return View::make('authorization-form', array('params' => $params));

}));

Route::post('/oauth/authorize', array('before' => 'check-authorization-params|auth|csrf', function()

{

// get the data from the check-authorization-params filter

$params = Session::get('authorize-params');

// get the user id

$params['user\_id'] = Auth::user()->id;

// check if the user approved or denied the authorization request

if (Input::get('approve') !== null) {

$code = AuthorizationServer::newAuthorizeRequest('user', $params['user\_id'], $params);

Session::forget('authorize-params');

return Redirect::to(AuthorizationServer::makeRedirectWithCode($code, $params));

}

if (Input::get('deny') !== null) {

Session::forget('authorize-params');

return Redirect::to(AuthorizationServer::makeRedirectWithError($params));

}

}));

If the authorization process is successful the client will be redirected to its redirect\_uri parameter with an authorization code in the query string like in the example below

https://www.yourclient.com/redirect?code=XYZ123

The client can now use this code to make an access token request in the background.

POST https://www.example.com/oauth/access\_token?

grant\_type=authorization\_code&

client\_id=the\_client\_id&

client\_secret=the\_client\_secret&

redirect\_uri=client\_redirect\_uri&

code=the\_authorization\_code&

scope=scope1,scope2&

state=123456789

### Password flow

This grant type is the easiest to use and is ideal for highly trusted clients. To enable this grant type add the code below to the grant\_types array located at app/config/packages/lucadegasperi/oauth2-server-laravel/oauth2.php

'password' => array(

'class' => 'League\OAuth2\Server\Grant\Password',

'access\_token\_ttl' => 604800,

'callback' => function($username, $password){

$credentials = array(

'email' => $username,

'password' => $password,

);

$valid = Auth::validate($credentials);

if (!$valid) {

return false;

}

return Auth::getProvider()->retrieveByCredentials($credentials)->id;

}

),

An example request for an access token using this grant type might look like this.

POST https://www.example.com/oauth/access\_token?

grant\_type=password&

client\_id=the\_client\_id&

client\_secret=the\_client\_secret&

username=the\_username&

password=the\_password&

scope=scope1,scope2&

state=123456789

### Client Credentials flow

Sometimes the client and the resource owner are the same thing. This grant types allows for client to access your API on their own behalf. To enable this grant type add the code below to the grant\_types array located at app/config/packages/lucadegasperi/oauth2-server-laravel/oauth2.php

'client\_credentials' => array(

'class' => 'League\OAuth2\Server\Grant\ClientCredentials',

'access\_token\_ttl' => 3600,

),

An example request for an access token using this grant type might look like this.

POST https://www.example.com/oauth/access\_token?

grant\_type=client\_credentials&

client\_id=the\_client\_id&

client\_secret=the\_client\_secret&

scope=scope1,scope2&

state=123456789

### Refresh token flow

Access tokens do expire but by using the refresh token flow you can exchange a refresh token for an access token. When this grant type is enabled, every access token request will also issue a refresh token you can use to get a new access token when the current one expires. Configure this grant type in the grant\_types array located at app/config/packages/lucadegasperi/oauth2-server-laravel/oauth2.php like this:

'refresh\_token' => array(

'class' => 'League\OAuth2\Server\Grant\RefreshToken',

'access\_token\_ttl' => 3600,

'refresh\_token\_ttl' => 604800,

'rotate\_refresh\_tokens' => false,

),

An example request for an access token using this grant type might look like this.

POST https://www.example.com/oauth/access\_token?

grant\_type=refresh\_token&

refresh\_token=the\_refresh\_token&

client\_id=the\_client\_id&

client\_secret=the\_client\_secret&

state=123456789

## Securing the API endpoints

You can protect your laravel routes with oauth by applying the oauth before filter to them like in the example shown below

Route::get('secure-route', array('before' => 'oauth', function(){

return "oauth secured route";

}));

Additionaly you can provide the allowed scopes to the oauth before filter by passing them in the filter name.

Route::get('secure-route', array('before' => 'oauth:scope1,scope2', function(){

return "oauth secured route";

}));

An interesting addition is the possibility to limit an endpoint to a specific owner type when using the client credentials grant type. It can be achieved by adding the oauth-owner before filter to your route.

Route::get('secure-route', array('before' => 'oauth:scope1,scope2|oauth-owner:client', function(){

return "oauth secured route for clients only";

}));

## Accessing the API

To access any api route, after issuing a token you must pass it as a parameter to the api call:

http://www.example.com/secure-route?access\_token= 'valid\_token'

## Getting the token owner ID and type

When accessing your API with an access token, you might want to know who's the owner of that token. The server makes it trivial.

$ownerId = ResourceServer::getOwnerId();

When using the client\_credentials grant type you might have users mixed with clients, to distinguish them use the following method

$ownerType = ResourceServer::getOwnerType();

The aim of this package is to make working with oauth2 server stuff in Laravel a breeze. You can still access all the undelying power of the league/oauth2-server package via the ResourceServer and AuthorizationServer facades.

## Support

Bugs and feature request are tracked on [GitHub](https://github.com/lucadegasperi/oauth2-server-laravel/issues)

## License

This package is released under the MIT License.

## Credits

The code on which this package is [based](https://github.com/php-loep/oauth2-server/), is principally developed and maintained by [Alex Bilbie](https://twitter.com/alexbilbie).

# Guzzle

## Installing via Composer

The recommended way to install Guzzle is through [Composer](http://getcomposer.org).

# Install Composer

curl -sS https://getcomposer.org/installer | php

Next, update your project's composer.json file to include Guzzle:

{

"require": {

"guzzlehttp/guzzle": "~4.0"

}

}

After installing, you need to require Composer's autoloader:

require 'vendor/autoload.php';