

Laravel JWT guard and authentication tools — introducing



Milad Rahimi

Follow

Dec 6, 2017 · 5 min read



Acceder a medium.com con Google



David Moises Villegas Aguilar

david.villegas.aguilar@gmail.com

CONTINUAR COMO DAVID MOISES

Para crear tu cuenta, Google compartirá tu nombre, dirección de correo electrónico y foto de perfil con medium.com. Si continúas, aceptas las [condiciones del servicio](#) y la [política de privacidad](#) de medium.com.

If you have plan to use JWT for authenticating or authorizing your api clients, this article is for you.

This article might be out of date, you can find the latest version of package and its documentation here in Github.

Larajwt Package

LaraJwt is a Laravel package for generating JWT (JSON Web-based Token) from users and providing JWT guard for Laravel applications.

Installation

Run the following command in your Laravel root directory:

```
composer require miladrahimi/larajwt:2.*
```

Then run the following command to generate `jwt.php` (the package config) in your Laravel config directory:

```
php artisan vendor:publish --tag=larajwt-config
```

Notes on Installation

- The package service provider will be automatically discovered by Laravel package discovery.
- The `JwtAuth` alias for `MiladRahimi\La` registered.

Configuration

To configure the package open `jwt.php` file. It consists of following items:

- `key` : The secret key to sign the token, it uses your project key if you leave it empty.
- `ttl` : Time that token will be valid, token will be expired after this time (in seconds)
- `issuer` : Issuer claim
- `audience` : Audience claim
- `model_safe` : Set it true if you have different authentication for different models with LaraJwt, it ensures that token belongs to related model defined in `guard`.

Generate JWT from Users

Use the method below to generate JWT from users or any other authenticable entities (models):

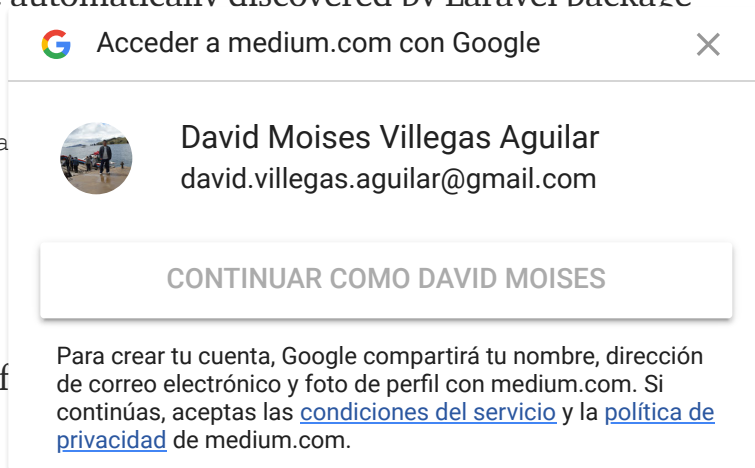
```
$jwt = JwtAuth::generateToken($user);
```

For example you may generate JWT from users in the sign-in process like this:

```
$credential = [
    'email' => $request->input('email'),
    'password' => $request->input('password'),
];

if(Auth::guard('api')->attempt($credential)) {
    $user = Auth::guard('api')->user();

    $jwt = JwtAuth::generateToken($user);
```



```
// Return successfull sign in
} else {
    // Return response for failed
}
```

If you want to store more information like email in the token method this way:

```
$customClaims = ['role' => 'admin', 'email' => $user->email];

$jwt = JwtAuth::generateToken($user, $customClaims);
```



Guards

Add as many as **guard** you need in your `config/auth.php` with `jwt` driver like this example:

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

    'api' => [
        'driver' => 'jwt',
        'provider' => 'users',
    ],
],
```

Authenticated Routes

After configuring **guards** in `config/auth.php` you can protect routes by the defined **guards**.

In our example we can protect route like this:

```
Route::group(['middleware' => 'auth:api'], function () {
    // Routes...
```

```
});
```

- Your clients must send header `Author`

Authenticated User

To retrieve current user and his info in yo can do it this way:

```
// To get current user
$user = Auth::guard('api')->user();
$user = Auth::guard('api')->getUser();

// To get current user id
$user = Auth::guard('api')->id();

// Is current user guest
$user = Auth::guard('api')->guest();

// To get current token
$jwt = Auth::guard('api')->getToken();

// To get current token claims
$claims = Auth::guard('api')->getClaims();

// To get a sepcific claim from current token
$role = Auth::guard('api')->getClaim('role');

// Logout current user (JWT will be cached in blacklist and NOT valid
in next requests).
Auth::guard('api')->logout();

// Logout current user (but it will be VALID next reuquests).
// It clears caches so user will be fetched and filters will be
executed again in next request.
Auth::guard('api')->logout(false);
```

Since Larajwt caches user fetching it can authenticate users without touching database.

Retrieve User Manually

You may need to retrieve user from generated JWTs manually, no worry! just do it this way:



```
$user = Jwtauth::retrieveUser($token);
```

It uses default user provider to fetch the user and pass it to the method as the second parameter.

```
$admin = Jwtauth::retrieveUser($token);
```



Retrieve JWT Claims Manually

You may even go further and need to retrieve JWT claims manually, it has considered too.

```
$claims = Jwtauth::retrieveClaims($jwt);
```

The mentioned method returns associative array of claims with following structure:

```
[
    'sub' => '666',
    'iss' => 'Your app name',
    'aud' => 'Your app audience',
    // ...
]
```

Cache

Larajwt caches JWTs in default, so after first authentication it remembers token as long as ttl which is set in config.

If you need to clear cache for a specific user you can use following method:

```
Jwtauth::clearCache($user);
```

You can pass user model (Authenticable) or its id to the `clearCache` method.

Filters

Filters are runnable closures which will be executed after authentication model.

For example if you have considered boolean value for active user, you probably want to check its value after authentication. If false or change Larajwt normal process to make it true.

You can register filters as many as you need in your authentication.

AuthServiceServiceProvider seems a good place to register filters.

```
class AuthServiceServiceProvider extends ServiceProvider
{
    // ...

    public function boot()
    {
        // ...

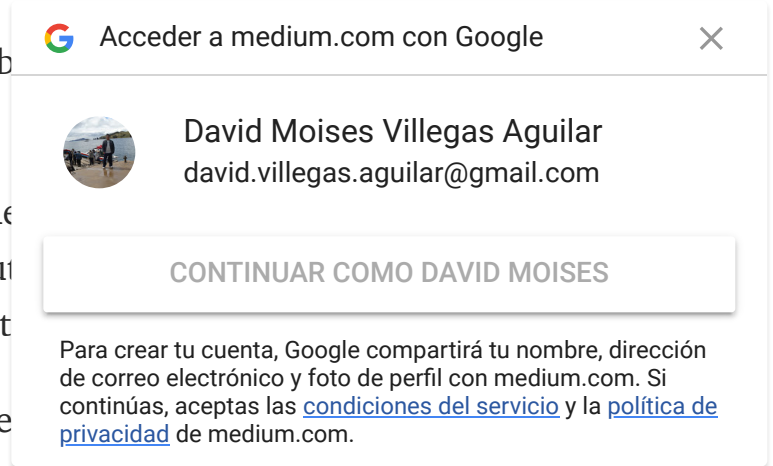
        $jwtAuth = $this->app->make(JwtAuthInterface::class);

        // Check if user is active or not
        $jwtAuth->registerFilter(function (User $user) {
            if ($user->is_active == true) {
                return $user;
            } else {
                return null;
            }
        });
    }
}
```

The `registerFilter` takes a closure with one argument to get authenticated user and it should return the user if there is no problem, it can return null if you want make the authentication failed.

Logout and JWT Invalidation

As mentioned with example above you can logout user with following method:

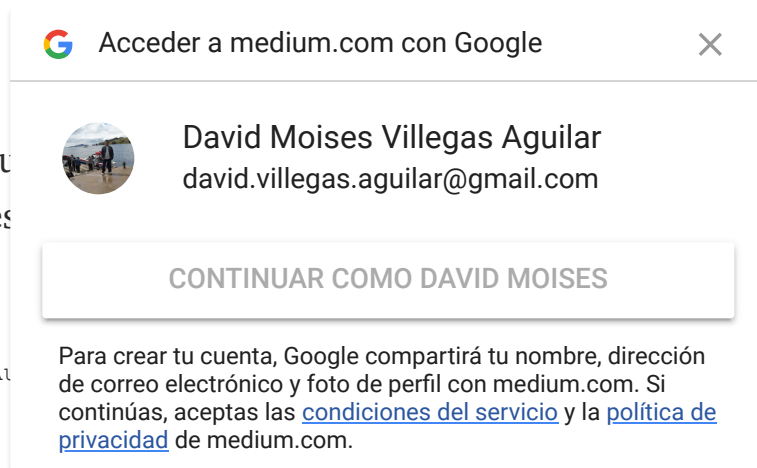


```
Auth::guard('api')->logout();
```

It takes one boolean parameter that is true so the token won't be valid in next request for current user and clear cache.

You can also invalidate tokens with `JwtAuth::invalidate($jti);`

```
JwtAuth::invalidate($jti);
```



Exceptions

```
Exception Class: LaraJwtConfiguringException
Exception Message: LaraJwt config not found.
```

This exception would be thrown if you had not published the package config (mentioned in Installation section).

JWT vs Stored Tokens

You may consider simple database-stored tokens as the alternative for JWT for authenticating, So we have provided some differences and comparison for you.

Cons

- More HTTP overhead, generated tokens are long.
- Force logout is more complex and tricky (LaraJwt handles it for you).

Pros

- No need to database column for storing generated tokens.
- No database touch if you only need user id.
- Less database touch if you cache user fetching (LaraJwt does it for you).


More information and Updates


If you are interested to use this package in your project, you can find it on Github via this link:

<https://github.com/miladrahimi/larajwt-guard-and-authentication-tools>

No rights reserved by the author. ©

[PHP](#)[Laravel](#)[Jwt](#)[Guard](#)[Authentication](#)

 Acceder a medium.com con Google ×



David Moises Villegas Aguilar
david.villegas.aguilar@gmail.com

CONTINUAR COMO DAVID MOISES

Para crear tu cuenta, Google compartirá tu nombre, dirección de correo electrónico y foto de perfil con medium.com. Si continúas, aceptas las [condiciones del servicio](#) y la [política de privacidad](#) de medium.com.

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