JSON WEB TOKEN

Using tymon/jwt-auth of PHP/Laravel

Source Github: <https://github.com/tymondesigns/jwt-auth/>

Document: <https://github.com/tymondesigns/jwt-auth/wiki/>

Require

* Laravel 4 or 5 (see compatibility table)
* PHP 5.4+

Install

* Edit file composer.json to require the package

"require": {

"tymon/jwt-auth": "0.5.\*"

}

//sau đó dùng lệnh

Composer update

Or dùng lệnh để tự động require và add vào composer.json: composer require tymon/jwt-auth

* Add file config/app.php
* Chỗ provider

Tymon\JWTAuth\Providers\JWTAuthServiceProvider::class

* Aliases

'JWTAuth' => Tymon\JWTAuthFacades\JWTAuth::class,

'JWTFactory' => Tymon\JWTAuthFacades\JWTFactory::class

* We also need to publish the asset for this package. From the command line:

php artisan vendor:publish --provider="Tymon\JWTAuth\Providers\JWTAuthServiceProvider"

After you run this command you will see a new file jwt.php in the config folder.

* We need to generate a secret key:

php artisan jwt:generate

* Nếu sử dụng cho 2 server thì key này phải trùng với server bên kia mới có thể giải mã đúng token được

Configuration

* Secret key – secret
* K

Creating tokens

* Creating a token based on the users credentials: example AuthenticateController

use JWTAuth;

use Tymon\JWTAuth\Exceptions\JWTException;

class AuthenticateController extends Controller

{

public function authenticate(Request $request)

{

// grab credentials from the request

$credentials = $request->only('email', 'password');

try {

// attempt to verify the credentials and create a token for the user

if (! $token = JWTAuth::attempt($credentials)) {

return response()->json(['error' => 'invalid\_credentials'], 401);

}

} catch (JWTException $e) {

// something went wrong whilst attempting to encode the token

return response()->json(['error' => 'could\_not\_create\_token'], 500);

}

// all good so return the token

return response()->json(compact('token'));

}

}

* Creating a Token based on a user object
* You can also skip user authentication and just pass in a User object. e.g.

// grab some user

$user = User::first();

$token = JWTAuth::fromUser($user);

* The above two methods also have a second parameter where you can pass an array of custom claims. e.g.

$customClaims = ['foo' => 'bar', 'baz' => 'bob'];

JWTAuth::attempt($credentials, $customClaims);

// or

JWTAuth::fromUser($user, $customClaims);

* And these custom claims will be available alongside the other claims when decoding the token.
* Note: Be wary about adding lots of custom claims as this will increase the size of your token.
* Creating a Token based on anything you like
* I have provided access to the underlying classes and methods to offer advanced/custom functionality.
* Example using the built in Tymon\JWTAuth\PayloadFactory instance (or using the included JWTFactory facade):

$customClaims = ['foo' => 'bar', 'baz' => 'bob'];

$payload = JWTFactory::make($customClaims);

$token = JWTAuth::encode($payload);

* You can also chain claims directly onto the **Tymon\JWTAuth\PayloadFactory** instance, (or using the included **JWTFactory** facade)

// add a custom claim with a key of `foo` and a value of ['bar' => 'baz']

$payload = JWTFactory::sub(123)->aud('foo')->foo(['bar' => 'baz'])->make();

$token = JWTAuth::encode($payload);

Authentication

* To get the token from the request you can do:

// this will set the token on the object

JWTAuth::parseToken();

// and you can continue to chain methods

$user = JWTAuth::parseToken()->authenticate();

* To get the token value, you can call:

$token = JWTAuth::getToken();

* Set token

JWTAuth::setToken('foo.bar.baz');

* **Retreiving the Authenticated user from a token**

// somewhere in your controller

public function getAuthenticatedUser()

{

try {

if (! $user = JWTAuth::parseToken()->authenticate()) {

return response()->json(['user\_not\_found'], 404);

}

} catch (\Tymon\JWTAuth\Exceptions\TokenExpiredException $e) {

return response()->json(['token\_expired'], $e->getStatusCode());

} catch (\Tymon\JWTAuth\Exceptions\TokenInvalidException $e) {

return response()->json(['token\_invalid'], $e->getStatusCode());

} catch (\Tymon\JWTAuth\Exceptions\JWTException $e) {

return response()->json(['token\_absent'], $e->getStatusCode());

}

// the token is valid and we have found the user via the sub claim

return response()->json(compact('user'));

}

* Chú ý
* If using Laravel 5 (0.5.\*) then you have access to 2 included Middlewares:
* GetUserFromToken
* This will check the header and query string (as explained above) for the presence of a token, and attempts to decode it. The same events are fired, as above.
* RefreshToken

This middleware will again try to parse the token from the request, and in turn will refresh the token (thus invalidating the old one) and return it as part of the next response. This essentially yields a single use token flow, which reduces the window of attack if a token is compromised, since it is only valid for the single request.

* To use the middlewares you will have to register them in **app/Http/Kernel.php** under the **$routeMiddleware property**:

protected $routeMiddleware = [

...

'jwt.auth' => 'Tymon\JWTAuth\Middleware\GetUserFromToken',

'jwt.refresh' => 'Tymon\JWTAuth\Middleware\RefreshToken',

];

* Sau đó sử dụng middleware đó
* Trong route thì thêm param [‘middleware’ => [‘jwt.auth’,’jwt.refresh’]]
* Trong controller thì:

public function \_\_construct()

{

// Apply the jwt.auth middleware to all methods in this controller

// except for the authenticate method. We don't want to prevent

// the user from retrieving their token if they don't already have it

// $this->middleware('jwt.auth', ['except' => ['authenticate']]);

// $this->middleware('jwt.auth');

}