OAuth Middleware for Slim


by Lorna Mitchell MVB · Oct. 10, 13 ·
Integration Zone · Not set

OAuth can be anything you want it to be, the standards are lax and give you plenty of room for getting the right implementation for your system. However you proceed, though, you'll need to check an access token on every request - and in a Slim application, a middeware can help enormously since it hooks in to every request by design. I've recently implemented this and thought I would share.

Acquiring an Access Token

This is an RPC-esque API, and there's an action that looks very much like a user login method. In OAuth terms, this is the "client credentials flow". Since the consumer of this API is created by the same organisation as the API itself, there's a high degree of trust between the two. To make things very simple, the consumer asks the user for their credentials, and passes them to the API (note: the consumer does not store them, and will never use them again. The trusted consumer application is the ONLY time you give your password to any 3rd party). The API checks them, and sends back an access token which the consumer then uses on all future requests.

This is a basic login type of flow, similar to what you would build on any website, and the access token is just a generated hash of some kind. You could do

pretty much anything here, generate something and then store it so you can tie it back to the correct user again later. For the record, my code looks like this:

```
$token = bin2hex(openssl_random_pseudo_bytes(16));
```

The consumer then captures this token and adds it in the Authorisation header of all future requests, so that the header looks like this:

```
Authorisation: OAuth [token]
```

Checking the Token

Now we need to check the token on every request where we expect the user to be authenticated. You could do this with a check at the top of each appropriate controller action, but Slim has a mechanism that makes this even easier: its middleware provides hooks that can be used on every request.

Middleware is a self-contained class, here's mine:

```
namespace MyLib\Middleware; class OAuth2Auth extends
\Slim\Middleware { protected $headers = array(); pro
tected $config; protected $mysql; // PDO public func
tion __construct($headers, $config, $mysql) { $this-
>headers = $headers; $this->config = $config; $this-
>mysql = $mysql; } public function call() { // no au
th header if(!isset($this->headers['Authorization'])
) { $this->app->getLog()->error("No authorization he
ader"); $view = $this->app->view(); $view->setData("
data", array("message" => "Access credentials not su
pplied")); $view->render('error.php', 400); } else
try { $authHeader = $this->headers['Authorization']
$auth = new \Service\Mysql\AuthService($this->mysql,
$this->config); $validated_user_id = $auth->verifyOA
uth($authHeader); $this->app->user_id = $validated_u
ser_id; } catch (\Exception $e) { $view = $this->app
->view(); $view->setData("data", array("message" =>
$e->getMessage())); $view->render('error.php', $e->g
etCode()); } } // this line is required for the appl
ication to proceed $this->next->call(); } }
```

Hopefully this code is easy enough to follow, the constructor of this class accepts some dependencies including the incoming headers for this request. The `call()` method is then called when it's this middleware's turn ... and at the end of that method, we call the next middleware in the list. Within this main

method, we check the incoming header against the database and save the user's information so that we can use it again if we want to use their identity when processing the detail of this request later. This application requires all requests to be authenticated, but if yours allows some unauthenticated access, you could just as easily not set the user details and continue - then check if those details are present as appropriate to each route.

Adding Middleware to Slim Framework

I found this a super-easy way to get the same code firing on every request without remembering anything :) It gets set up after I've instantiated the app (which is in `$app`) and connected to the database (the PDO object is stored in $mysql). Then in `index.php` I just add these lines:

```
// Add OAuth middleware $headers = apache_request_he
aders(); $app->add(new \MyLib\Middleware\OAuth2Auth(
$headers, $config, $mysql));
```

Hopefully if you need to implement something similar, this gives you an idea of the kind of pattern you can follow. Additional tips, questions or comments are all welcome, just use the comment form below :)

Further Reading

- PHP OAuth Provider: Access Tokens
- 7 Steps to Better APIs
- Pretty-Printing JSON with Python's JSON Tool
- Twitter Search API Using PHP and Guzzle
- API Design
- Changing Content Type with Slim Framework

free. Access your trial here.

Like This Article? Read More From DZone


Elasticsearch on Google Cloud Platform


When Scrum Is Not The Right Answer


Shadow Root DOM and Custom HTML Tags Automation Using Selenium


Free DZone Refcard
Foundations of RESTful Architecture

Topics:

Published at DZone with permission of Lorna Mitchell , DZone MVB. See the original article here.
Opinions expressed by DZone contributors are their own.