

Protect Your API with OAuth 2

Rob Allen ~ @akrabad

Authentication

Know who is logging into your API

- Rate limiting
- Revoke application access if its a problem
- Allow users to revoke 3rd party applications

How?

Authorization header:

```
GET /books/1 HTTP/1.1
Host: api.example.com
Accept: application/json
Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW11
```

```
base64_encode("Aladdin:OpenSesame")
=> QWxhZGRpbjpPcGVuU2VzYW11
```

Problems

- All clients have to know user's credentials
- Credentials are passed in every request

OAuth2



The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service

oauth.net

Roles

- The user (*Resource Owner*)
- The (third-party) application (*Client*)
- The API (*Resource Server*)
- The Authorisation server

Grant types

Grant type	Use case
Authorization code	3rd party web or native
Password	1st party
Client credentials	application (no user)
Implicit	3rd party JS app

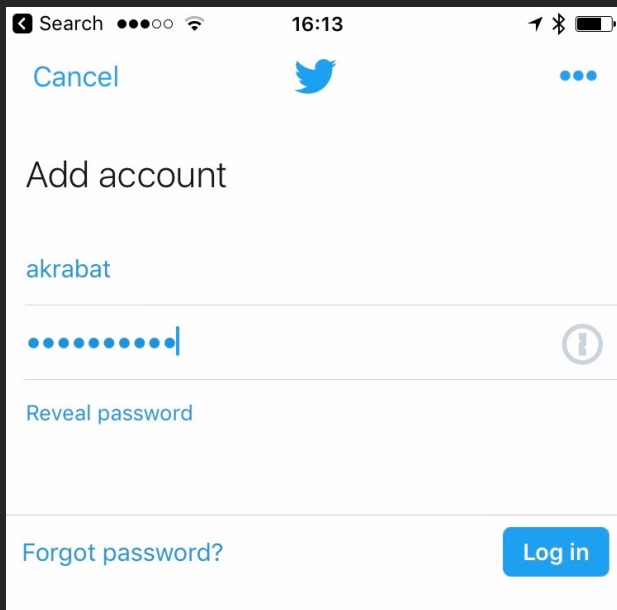
Tokens

OAuth uses a bearer token

```
GET /books/1 HTTP/1.1  
Host: api.example.com  
Accept: application/json  
Authorization: Bearer {some-string-here}
```


Password grant (for 1st party apps)

Password grant



The image shows a mobile app interface for logging in to Twitter. At the top, there's a status bar with a back arrow, the word "Search", signal strength, Wi-Fi, the time "16:13", and location, Bluetooth, and battery icons. Below the status bar, the interface has a white background. At the top left is a "Cancel" link. In the center is the Twitter bird logo. At the top right are three dots. Below this is the text "Add account". Underneath is the username "akrabat" in a light blue font. Below the username is a password field with ten dots and a cursor at the end. To the right of the password field is an information icon (a circle with an 'i'). Below the password field is a "Reveal password" link. At the bottom left is a "Forgot password?" link. At the bottom right is a blue "Log in" button.

Search 16:13

Cancel

Twitter logo

Add account

akrabat

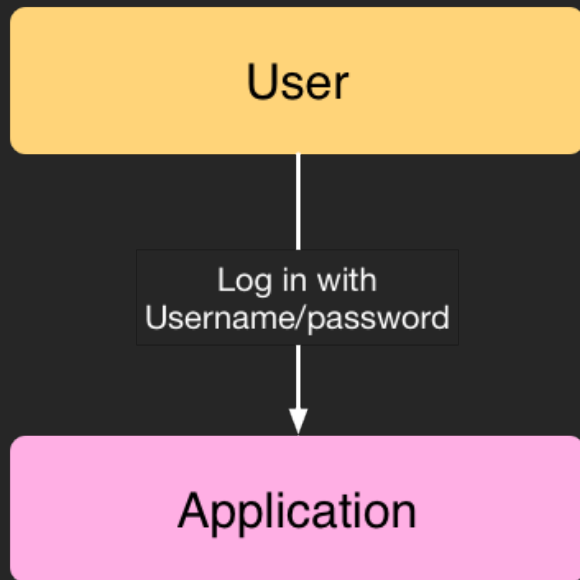
.....|

Reveal password

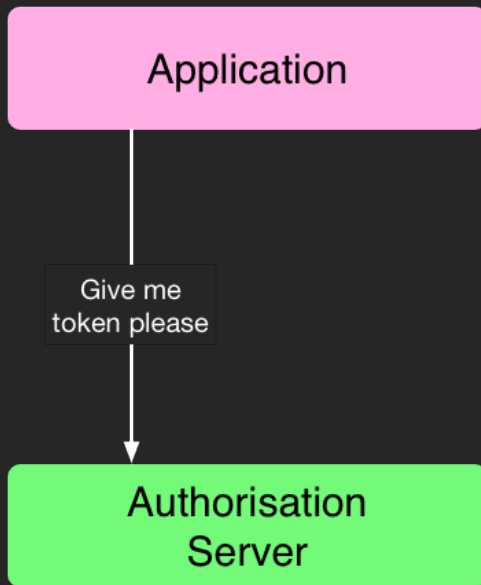
Forgot password?

Log in

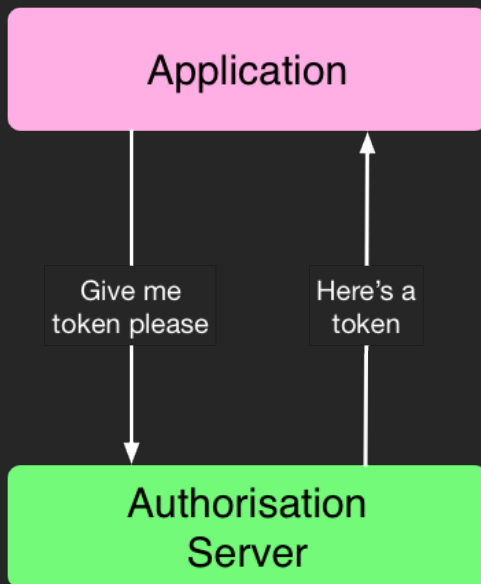
Password flow



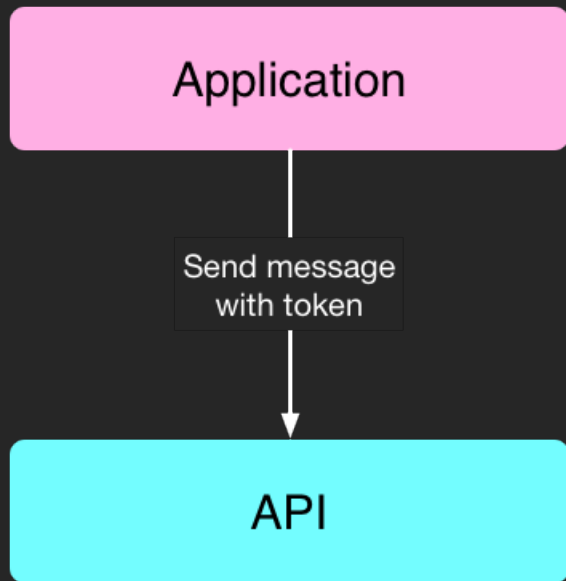
Password flow



Password flow



Password flow



Implementing in PHP

OAuth 2.0 Server PHP

by Brent Shaffer

```
$ composer require bshaffer/oauth2-server-php
```

- Implements Authorise and Token endpoints
- Multiple storage backends: PDO, Redis, Mongo, Cassandra, DynamoDB, etc

Steps to implement

For the client and user credentials grants:

1. Set up the database tables
2. Register the OAuth2 Server
3. Implement the Authorise endpoint

Database tables

- `CREATE TABLE oauth_clients ...`
- `CREATE TABLE oauth_access_tokens ...`
- `CREATE TABLE oauth_authorization_codes ...`
- `CREATE TABLE oauth_refresh_tokens ...`
- `CREATE TABLE oauth_users ...`
- `CREATE TABLE oauth_scopes ...`
- `CREATE TABLE oauth_jwt ...`

(SQL is in the Cookbook in the docs)

Create a Server

```
1 use MyAuth\PdoStorage;
2 use OAuth2\GrantType\UserCredentials;
3
4 $container['OAuth2Server'] = function ($c) {
5     $pdo = $c->get('db');
6     $storage = new PdoStorage($pdo);
7
8     $server = new \OAuth2\Server($storage);
```

Add the grant

```
1 use MyAuth\PdoStorage;
2 use OAuth2\GrantType\UserCredentials;
3
4 $container['OAuth2Server'] = function ($c) {
5     $pdo = $c->get('db');
6     $storage = new PdoStorage($pdo);
7
8     $server = new \OAuth2\Server($storage);
9
10    /* Add the password grant type */
11    $userCreds = new UserCredentials($storage);
12    $server->addGrantType($userCreds);
13
14    return $server;
15 };
```

Aside: use Bcrypt

```
namespace MyAuth;
```

```
class PDOStorage extends \OAuth2\Storage\Pdo
{
    protected function checkPassword($user, $pwd)
    {
        return password_verify($pwd, $user['password']);
    }
}
```

Credentials

We need a client:

```
1 INSERT INTO oauth_clients
2   (client_id, client_secret, redirect_uri)
3 VALUES
4   ("mywebsite", "$2y$10$mzP0fR...BHu", null);
```

Credentials

We need a client:

```
1 INSERT INTO oauth_clients
2   (client_id, client_secret, redirect_uri)
3 VALUES
4   ("mywebsite", "$2y$10$mzP0fR...BHu", null);
```

& a user:

```
1 INSERT INTO oauth_users
2   (username, password, first_name, last_name)
3 VALUES
4   ("rob", "$2y$10$Qq1CsK...LV6", "Rob", "Allen");
```

Token endpoint

```
1 $app->post(  
2     '/token',  
3     function ($request, $response) {  
4         $server = $this->get('OAuth2Server');  
5         $req = \OAuth2\Request::createFromGlobals();  
6  
7         $server->handleTokenRequest($req)->send();  
8         exit;  
9     }  
10 );
```


How does this work?

```
1 $ curl -i -X POST http://localhost:8888/token \  
2   -H "Accept: application/json" \  
3   -H "Content-Type: application/json" \  
4   -d '${  
5       "grant_type": "password"  
6       "client_id": "mywebsite",  
7       "client_secret": "abcdef",  
8       "username": "rob",  
9       "password": "123456"  
10  }'
```

Response

```
1 HTTP/1.1 200 OK
2 Host: localhost:8888
3 Content-Type: application/json
4
5 {
6   "access_token": "65077f90e3baae8aa863",
7   "expires_in": 3600,
8   "token_type": "Bearer",
9   "scope": null,
10  "refresh_token": "be071d2c6193d32a353d"
11 }
```

Protecting your API endpoints

Is the token valid?

```
1 /* test for valid Auth header */
2 $req = \OAuth2\Request::createFromGlobals();
3 if (!$server->verifyResourceRequest($req)) {
4     /* not valid */
5 }
6
7 /* get information */
8 $token = $server->getAccessTokenData($req);
9 $username = $token['user_id'];
```

Unauthorised API call

```
1 $ curl -i -H "Accept: application/json" \  
2   http://localhost:8888/authors  
3  
4 HTTP/1.1 401 Unauthorized  
5 Host: localhost:8888  
6 Connection: close  
7 X-Powered-By: PHP/7.0.15  
8 WWW-Authenticate: Bearer realm="Service"  
9 Content-Type: application/json
```

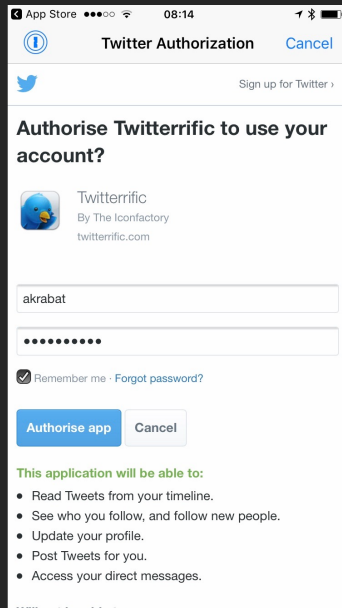
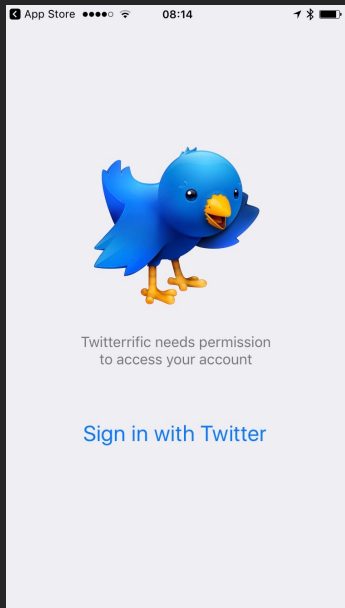
Authorised API call

```
1 $ curl -i -H "Accept: application/json" \  
2   -H "Authorization: Bearer 65077f90e3baae8aa863" \  
3   http://localhost:8888/authors  
4  
5 HTTP/1.1 200 OK  
6 Host: localhost:8888  
7 Connection: close  
8 X-Powered-By: PHP/7.0.15  
9 Content-type: application/hal+json  
10  
11 {  
12     "count": 6,  
13     "_links": {  
14     ...
```

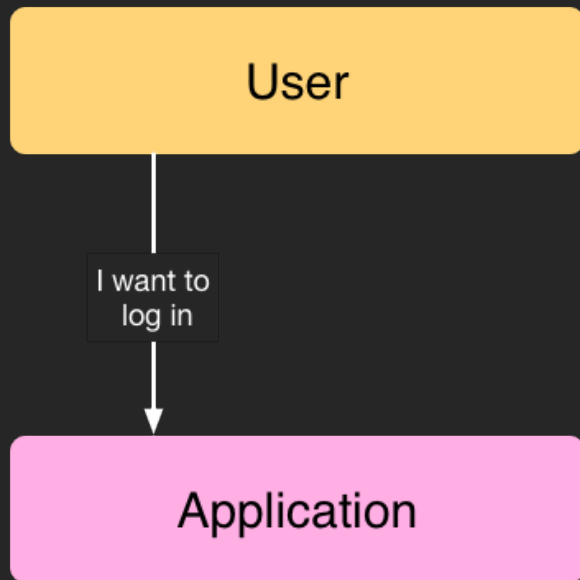
Authorisation Code

(for 3rd party apps)

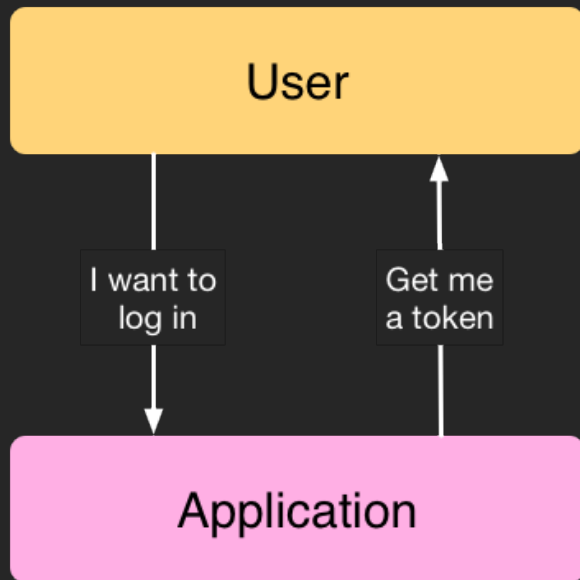
Authorisation code



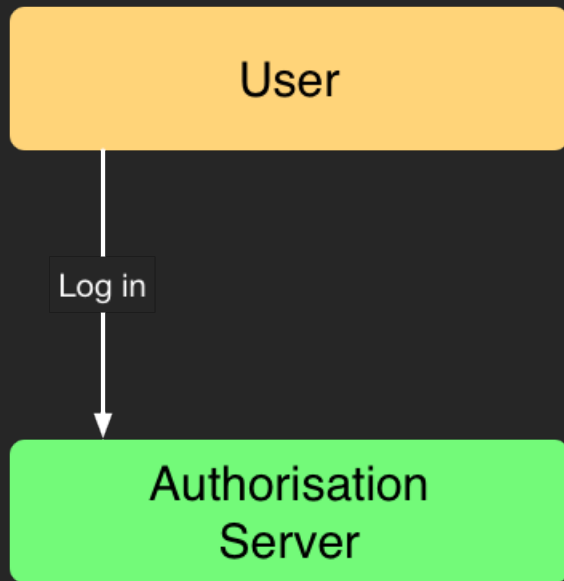
Authorisation code flow



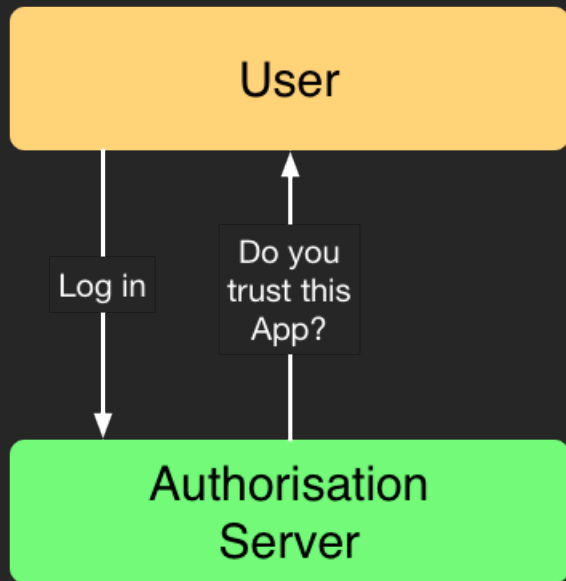
Authorisation code flow



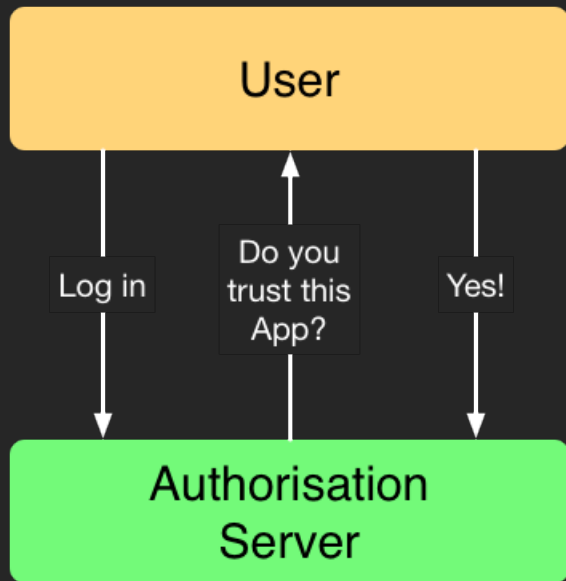
Authorisation code flow



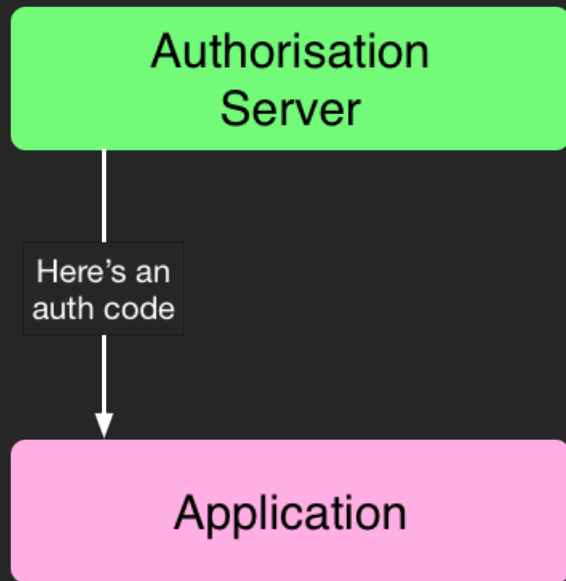
Authorisation code flow



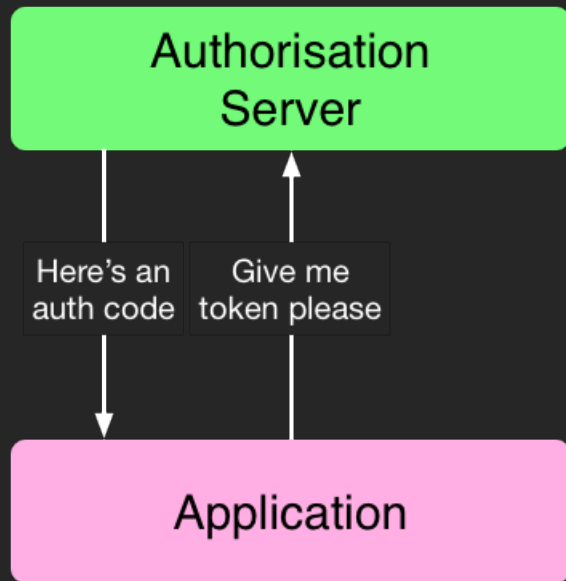
Authorisation code flow



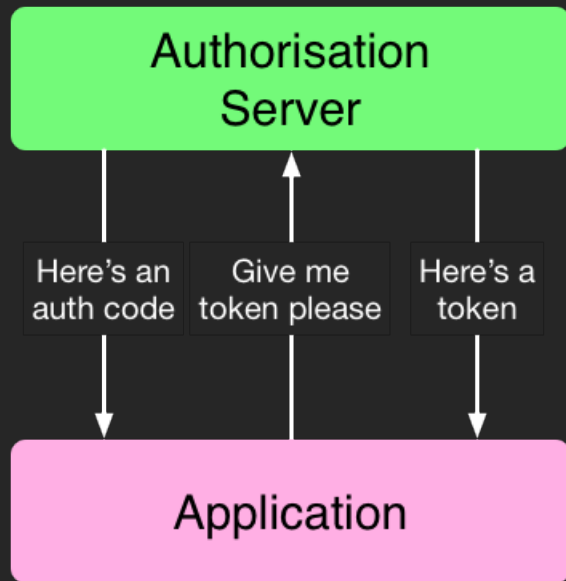
Authorisation code flow



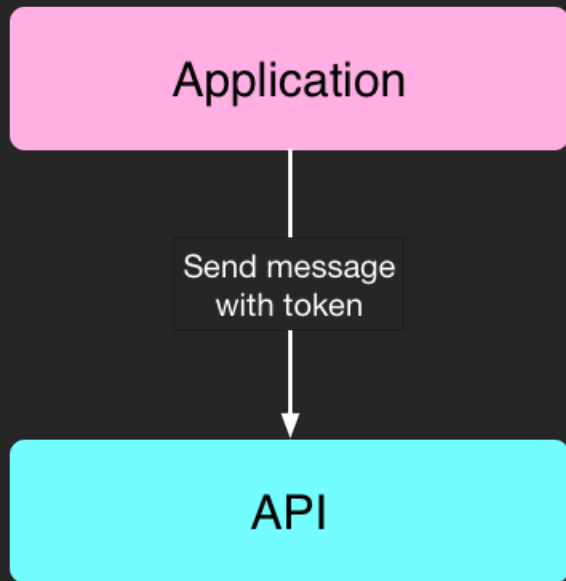
Authorisation code flow



Authorisation code flow



Authorisation code flow



Implementing in PHP

Required pieces

1. A website that talks to the Authorisation server
2. A new endpoint in the Authorisation server to provide auth codes

Process

1. 3rd party app sends user to our website:
2. User logs in to our website and authorises app
3. Our website gets code from our API
4. Our website redirects user back to app (or displays a code)

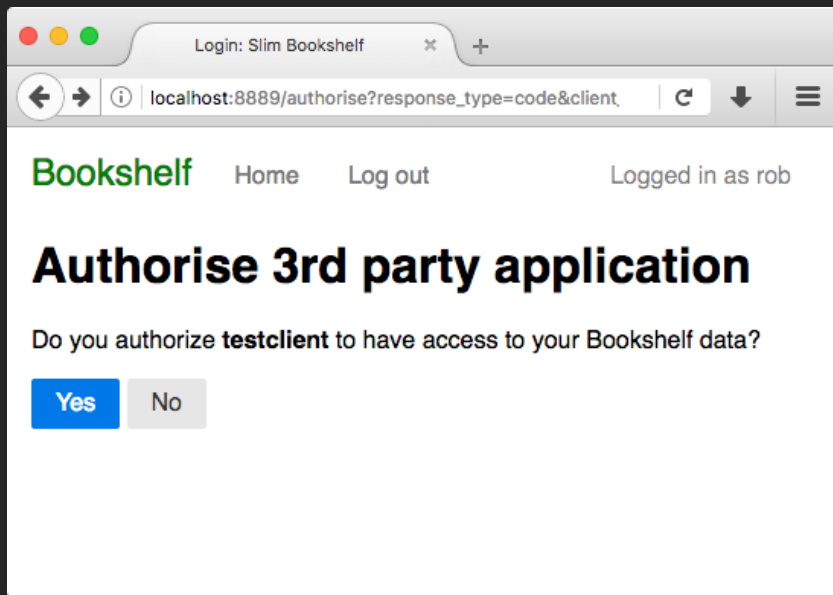
Add the grant

```
1 $container['OAuth2Server'] = function ($c) {  
2     // ...  
3     $server = new \OAuth2\Server($storage);  
4  
5     /* Add the password grant type */  
6     $userCreds = new UserCredentials($storage);  
7     $server->addGrantType($userCreds);  
8  
9     return $server;  
10 };
```

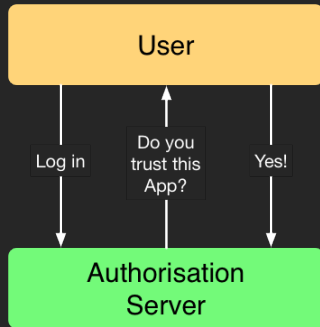
Add the grant

```
1 $container['OAuth2Server'] = function ($c) {  
2     // ...  
3     $server = new \OAuth2\Server($storage);  
4  
5     /* Add the password grant type */  
6     $userCreds = new UserCredentials($storage);  
7     $server->addGrantType($userCreds);  
8  
9     /* Add authorisation code grant type */  
10    $authCode = new AuthorizationCode($storage);  
11    $server->addGrantType($authCode);  
12  
13    return $server;  
14 };
```

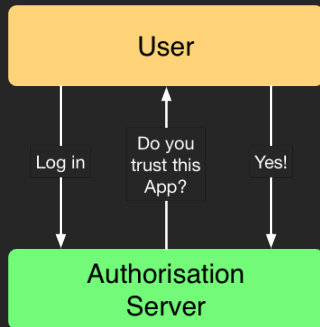
Authorise



Remember this?



It's more like this...



Website sends to API

Pressing Yes does this:

```
1 $data['code'] = 'token';
2 $data['client_id'] = $_GET['client_id'];
3 $data['redirect_uri'] = $_GET['redirect_uri'];
4 $data['state'] = $_GET['state'];
5
6 $apiResponse = $guzzle->post('/authorise', [
7     'json' => $data,
8     'headers' => [
9         'Authorization' => 'Bearer ' . $webAccessToken,
10     ]
11 ]);
```

API handles authorisation

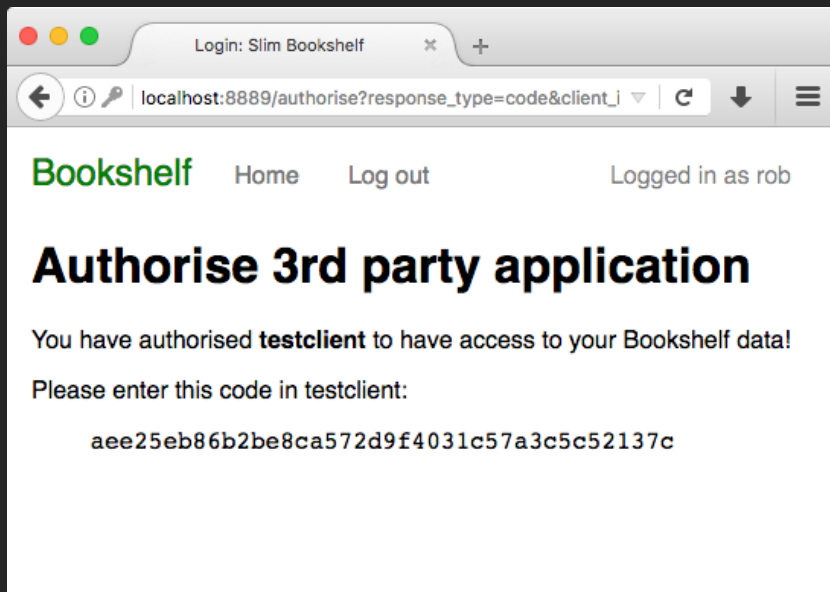
API's /authorise endpoint:

```
1 if (!$server->validateAuthorizeRequest($req, $res)) {  
2     $srvResponse->send(); exit;  
3 }  
4  
5 $server->handleAuthorizeRequest($req, $res, true);  
6 $srvResponse->send(); exit;
```

Website handles response

```
1 if ($apiResponse-&gtgetStatusCode() != 302) {
2     throw new Exception("Failed to get code");
3 }
4 $loc = $apiResponse->getHeaderLine('Location');
5
6 if ($this->isValidUrl($loc)) {
7     /* location is valid - redirect */
8     return $response->withRedirect($loc);
9 }
10
11 /* invalid url - display the code to user */
12 parse_str($parts['query'], $queryParams);
13 return $renderer->renderPage($queryParams['code']);
```

Authorised



Get token from code

```
1 $ curl -X "POST" http://localhost:8888/token \  
2   -H "Accept: application/json" \  
3   -H "Content-Type: application/json" \  
4   -d '${  
5     "grant_type": "authorization_code",  
6     "client_id": "testclient",  
7     "client_secret": "abcdef",  
8     "code": "aee25eb86b2be8ca572d9f4031c57a3c5c52137c",  
9   }'
```

Response

```
1 HTTP/1.1 200 OK
2 Host: localhost:8888
3 Connection: close
4 Content-Type: application/json
5
6 {
7   "access_token": "df7fcb455efb9a2c9544",
8   "expires_in": 3600,
9   "token_type": "Bearer",
10  "scope": null,
11  "refresh_token": "bb87ffbef191bdda55b1"
12 }
```

JWT bearer tokens

JWT

- Cryptographically signed block of data
- Potentially faster
- A JWT consists of
 - Header
 - Payload
 - Signature

Also: JWT is pronounced *"jot"*

Payload

```
1 {  
2   "id": "394a71988caa6cc30601e43f5b6569d52cd7f",  
3   "jti": "394a71988caa6cc30601e43f5b6569d52cd7f",  
4   "iss": "{issuer_id}",  
5   "aud": "{client_id}",  
6   "sub": "{user_id}",  
7   "exp": 1483711650,  
8   "iat": 1483708050,  
9   "token_type": "bearer",  
10  "scope": "read write delete"  
11 }
```

Implementation

1. Update token creation to create JWT tokens
2. Update validation to check for JWT tokens

Previously

```
1 $container['OAuth2Server'] = function ($c) {  
2     $pdo = $c->get('db');  
3     $storage = new PDOStorage($pdo);  
4  
5     $server = new \OAuth2\Server($storage);  
6  
7     // ... add grants ...
```

Enable JWT

```
1 $container['OAuth2Server'] = function ($c) {  
2     $pdo = $c->get('db');  
3     $storage = new PDOStorage($pdo);  
4  
5     $server = new \OAuth2\Server($storage, [  
6         'use_jwt_access_tokens' => true,  
7     ]);  
8  
9     // ... add grants ...
```

Get a token

```
1 $ curl -i -X POST http://localhost:8888/token \  
2   -H "Accept: application/json" \  
3   -H "Content-Type: application/json" \  
4   -d '${  
5       "grant_type": "password"  
6       "client_id": "mywebsite",  
7       "client_secret": "abcdef",  
8       "username": "rob",  
9       "password": "123456"  
10  }'
```

Response

```
1 HTTP/1.1 200 OK
2 Host: localhost:8888
3 Connection: close
4 Content-Type: application/json
5
6 {
7   "access_token": "eyJ0eXAiOi...BLUWlojjm24HmNbOMg",
8   "expires_in": 3600,
9   "token_type": "Bearer",
10  "scope": null,
11  "refresh_token": "be071d2c6193d32a353d"
12 }
```

Validation

Use an in-memory OAuth2 Server:

```
1 $storage = new OAuth2\Storage\Memory([
2     'keys' => [
3         'public_key' => $publicKey,
4     ]
5 ]);
6
7 $server = new OAuth2\Server($storage, [
8     'use_jwt_access_tokens' => true,
9 ]);
```


Validation

The validation code doesn't change

```
1 /* test for valid Auth header */
2 $req = \OAuth2\Request::createFromGlobals();
3 if (!$server->verifyResourceRequest($req)) {
4     /* not valid */
5 }
6
7 /* get information */
8 $token = $server->getAccessTokenData($req);
9 $username = $token['user_id'];
```

Refresh tokens

Refresh tokens

- Access tokens expire quickly
- Use the refresh token to get a new access token
- Guard refresh tokens!

```
1 $ curl -i -X POST http://localhost:8888/token \  
2   -H "Accept: application/json" \  
3   -H "Content-Type: application/json" \  
4   -d '${  
5       "grant_type": "refresh_token"  
6       "client_id": "testclient",  
7       "client_secret": "abcdef",  
8       "refresh_token": "be071d2c6193d32a353d"  
9   }'
```

Response

```
1 HTTP/1.1 200 OK
2 Host: localhost:8888
3 Connection: close
4 Content-Type: application/json
5
6 {
7   "access_token": "eyJ0eXAiOi...tjD8whWBt8h4oRlu0MA",
8   "expires_in": 3600,
9   "token_type": "Bearer",
10  "scope": null
11 }
```

Summary

Summary

- Authorization header contains token
- Two actors
 - Client (id & secret)
 - User (username & password)
- Grants:
 - Password: 1st party apps
 - Authorisation code: 3rd party apps
- JWT for speed and scale

Resources

This talk:

- <https://github.com/akrabat/slim-bookshelf-api>
- <https://akrabat.com/talks/protect-your-api-with-oauth2-codetalks/>

Around the web:

- <https://oauth.net/2/>
- <http://bshaffer.github.io/oauth2-server-php-docs>
- <https://aaronparecki.com/oauth-2-simplified/>

Questions?

Rob Allen ~ @akrabat

Thank you!

Rob Allen ~ @akrabat