Web and Mobile Application Development

OAuth

Material created by:
David Augenblick, Bill Mongan, Dan Ziegler, Samantha Bewley, and Matt Burlick

Acknowledgements

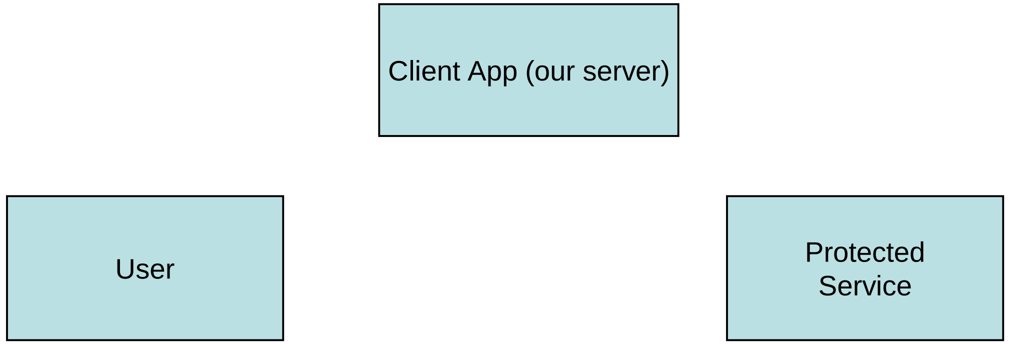
- Excerpts from the following link were utilized for this discussion along with a flow diagram by Vishy Ranganath from the same article.

https://hueniverse.com/oauth/guide

Oauth - Introduction

- Sometimes we want to get private/protected data from a web service
- We probably don’t want to make the users enter their username/password on our site and forward it to someone elses
    - Not very secure.
- Oauth (open authorization) provides a standard protocol for authorizing a user to get his/her data from a web service.

The standard (3-legged) user / client / server model



The standard (3-legged) user / client / server model

- User
  - The resource owner – has private data stored on a protected resource (service)
  - User has id and password necessary to interact with service
- Client Application/Site
  - Acts on user’s behalf to access user’s private data on service
  - However, user does not want to divulge id / password to client
    - But, client will need permission to access from user
- Service
  - Stores protected data owned by resource owner (user)
  - Requires owner’s permission for client to access owner’s data

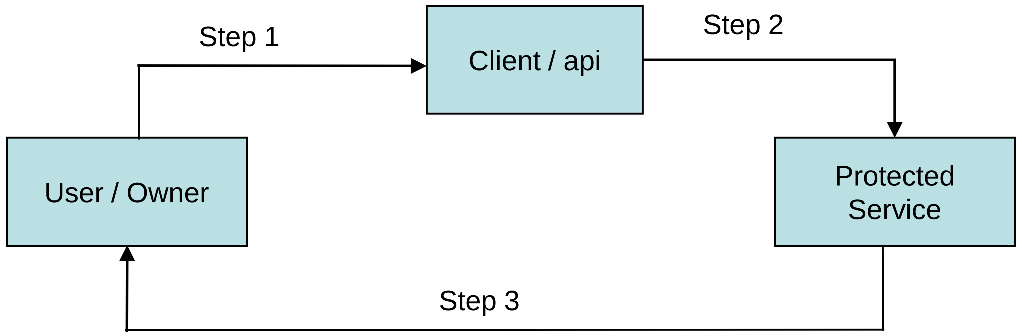
Example 1 (from Hueniverse)

- Example: Client api wants to access photos from a photo storage service
  - User / owner stores photos on a photo storage service
  - Client / api accesses photos from service to print and (snail) mail
  - Service – Photo storage site
  - The user / owner would like the client / api to retrieve photos from the service and send them to a friend

Example 2

- Dropbox Mover
  - User / owner has files stored on his / her private Dropbox account
  - Service = the Dropbox server
  - Client / api - you are to develop an application to:
    - Obtain user’s permission to access his / her Dropbox files
    - Move requested file(s) to user’s local directory

The Oauth Process
Diagram 1 – front end of process

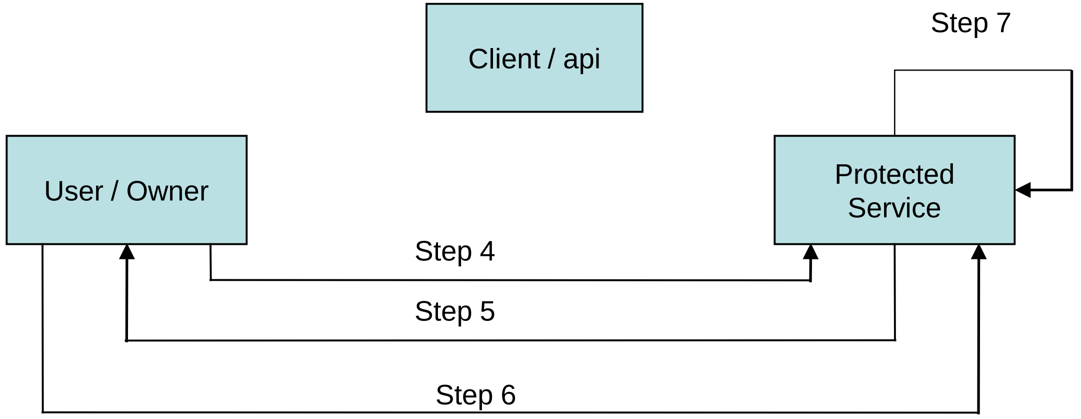


Step 1 – Resource owner requests client to obtain his/her files from service

Step 2 – Client, using it's credentials, redirects user to service

Step 3 – Service displays user authorization page to owner

The Oauth Process
Diagram 2 – steps 4 - 7



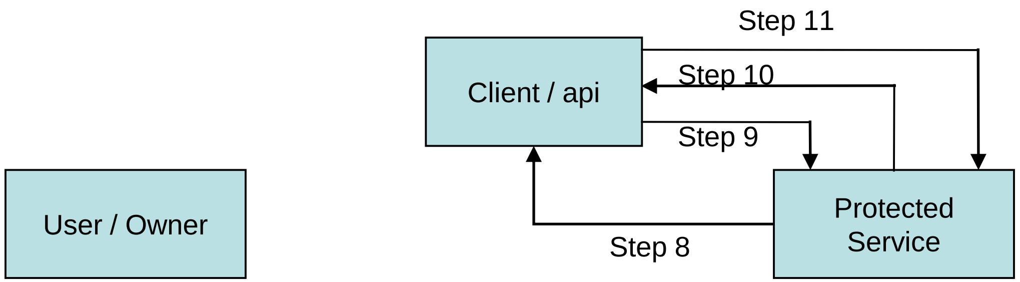
Step 4 – Owner signs into service using his/her key / password

Step 5 – Service asks if it is OK to grant limited access to client

Step 6 – Resource owner (user) allows (or denies)

Step 7 – Service creates an authorized request token for client

The Oauth Process
Diagram 3 – steps 8 - 11



Step 8 – Service re-directs owner back to client with the authorized request token

Step 9 – Client uses the authorized request token to obtain new an access token from service

Step 10 – Service sends back access token to client

Step 11 – Client supplies access token to service and accesses the protected resource

Example: Foursquare

- To illustrate how to use Oauth, let’s use of Oauth based authorization to enable a client application to retrieve and process a private owner's “recent check-ins” data from the Foursquare social network application.

Foursquare: Obtaining Client Key

- In order to initialize “the dance” the client must be authorized to interface with the API.
- This usually involves obtaining (via registration) a developer’s key.
- With Foursquare we must first have an account, then request a developer key.

Foursquare: Obtaining Client Key

- Once you have an account you can register a developer key here:

  https://developer.foursquare.com/overview/auth

- Unlike some applications (which only allow a single program per user app), it is possible to write several programs that are associated with the same user account.
- In addition to a developer key, you will create application keys for each application you create.
- These will be linked to end-user keys later (this happens when you click "Allow" as a user).

Foursquare: Obtaining Client Key

- After you register, create an app with any app name you like.
- Provide the location that Foursquare should redirect your token to.
    - If we're developing locally on port 8080 and using a Nodejs endpoint `FoursquareOauth` to handle the interactions we could do http://localhost:8080/FoursquareOauth
    - Or of you have your site hosted somewhere you'd use that URL.
- You will find your "Client ID" and "Client Secret" on the Foursquare developer site.
    - You will need these as inputs to your program

Foursquare: Obtaining Client Key

- To proceed we need to store following information
  1. Client ID. Effectively, this is your application key
  2. Callback URL (this must match what you provided)
  3. Version number (use a date beyond the present date – eg. 12/12/2017)
     - Foursquare wants it in the format YYYYMMDD
- Where should we keep this information?
  - None of this is really private information so no need to worry about security unless you really want to.

Foursquare: API Interface

- Of course we need to know which service URLs to send stuff to.
- This is typically defined in the service's documentation
- The Foursquare API Documentation is available at http://developer.foursquare.com and, in particular, https://developer.foursquare.com/overview/
- We can do many things, but for now we will just list recent check-ins made by our friends.

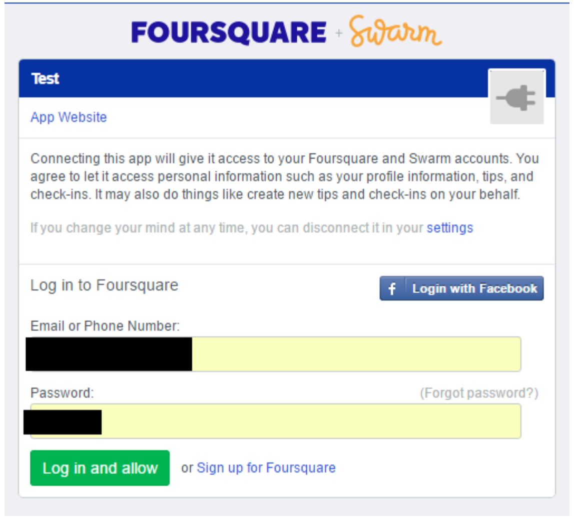
Authorizing the App to a User’s Account

- Now we should have everything we need
- Step 1-3 – Resource owner requests client to obtain his/her files from service
  - We’ll have a page with a “Authorize Foursquare” button that when clicked will redirect the user to the Foursquare Oauth authentication page:

```
<html>
<head>
<script>
function redirect(){
      var client_id = “XXXXXXXXXXXXXX“;  //this is our App ID.  No need to keep it private
      var redirect_url = "http://localhost:8080/FoursquareOauth";
      window.location="https://foursquare.com/oauth2/authenticate?client_id=" + client_id
      + "&response_type=code&redirect_uri="+redirect_url;
}
</script>
</head>
<body>
      <input type=button onclick="redirect()" value="Authorize Foursquare"/>
</body>
</html>
```

Authorizing the App to a User's Account

- Steps 3-6 – Service displays user authorization page to owner
  - Here the user will have to type their user/pass for the service if not already logged in.
  - Then they can select Allow


FOURSQUARE + Swarm

Test

App Website

Connecting this app will give it access to your Foursquare and Swarm accounts. You agree to let it access personal information such as your profile information, tips, and check-ins. It may also do things like create new tips and check-ins on your behalf.

If you change your mind at any time, you can disconnect it in your settings

Log in to Foursquare

Login with Facebook

Email or Phone Number:

[redacted]

Password: (Forgot password?)

[redacted]

Log in and allow or Sign up for Foursquare

Authorizing the App to a User's Account

- Steps 7/8 – Service creates an authorization token for client and redirects owner back to the client with the token
  - This token's purpose is basically just to say verify the callback and encode the user's ID.
- You will be redirected to your callback with a `code` field populated in the GET information:

  http://callback/?code=ABC123

- Let's assume we have a NodeJS endpoint `/FoursquareOauth`. We can then extract the `code` from the GET information easily.

```
app.get('/FoursquareOauth',function(req,res){
      res.write(req.query.code);
      res.end();
});
```

Authorizing the App to a User’s Account

- Steps 9/10 – Client now sends the authorized token back to the service to receive an access token from service
- To do this let’s create a Foursquare object (constructed from a module) that we pass the temporary code to and let it do the back-and-forth with Foursquare
- According to the API we get this by going to

  https://foursquare.com/oauth2/access_token?client_id=" + key + "&client_secret=" + consumersecret

  +"&grant_type=authorization_code&redirect_uri=" + callback + "&code=" + code

  - Where
    - `key` – Same App ID we used in steps 1-3
    - `client_secret` – Your secrete code. Although your Client ID is publically visible, your Client Secret isn’t and therefore you should read this in from a file not publically accessible
    - `redirect_uri` – Same redirect URI you provided during your registration
    - `code` – The authorized token obtained from Step 8

Authorizing the App to a User’s Account

- Step 9: We’ll create an `authenticate` method of our class that
  - Makes the `HTTPs` request to Foursquare for the access token
- Step 10: Once it gets the response back (async!), we extract the access token (it’ll be a JSON object!) and emit a signal indicating we’re authenticated
  - Once we see that event we can store the access token

```
{
    "access_token": “ABC123DEF456GHI789“
}
```

Authorizing the App to a User’s Account

- Step 9-10 – Client uses the authorized token to obtain access token from service

```
var key = “XXXXXXXXXXXXXXXXXXXX";
var consumersecret = fs.readFileSync('./FoursquareCredentials.txt','utf8');
var request = require(‘request’);


class FourSquare extends EventEmitter{
        constructor(){
                super();
                this.OauthKey='';
        }
        authenticate(code){
                if(this.OauthKey==‘’){ //avoid multiple authentications
                        var URL = ‘https://foursequre.com/oauth2/access_token?’;
                        URL+=‘client_id=" + key;
                        URL+=‘&client_secret=‘ + consumersecret;
                        URL+=‘&grant_type=authorization_code&redirect_uri=‘ + callback + ‘&code=‘ + code;
                        var self = this; //get a reference to this object for later use…

                        request.get(URL, function(error, response, body){
                                var json = JSON.parse(body);
                                self.OauthKey = json.access_token; //note the need to use self here
                                self.emit('authenticated');
                        });
                }
                else
                this.emit(‘authenticated’);
        }
}
```

```
app.get('/FoursquareOauth',function(req,res){
        fs.once('authenticated', function(msg){
                res.write(‘Authenticated!’);
                res.end();
        });
        fs.authenticate(req.query.code)
});
```

Example: Foursquare

- Step 11 – Client supplies access token to service and accesses the protected resource.
  - Using the access token we finally obtained, we can now get to the protected resources.
  - We just need to know which URLs to go to in order to get what we want
  - Again we'll need to look at the API documentation

Foursquare API

- Access your own checkins:
  - https://api.foursquare.com/v2/users/self/checkins
- Access a list of your friends' recent checkins:
  - https://api.foursquare.com/v2/checkins/recent
- Add to the end of these the following and you're good to go!

  "?limit=100&oauth_token=" + oauth_token + "&v=" + version

  - Where version is a date in the future in the format YYYYMMDD
- Of course read more of the documentation to see additional parameters you can add to these URL strings for more stuff

Example: Foursquare

- Ok so let’s retrieve a list of friends’ recent check-ins!
- Go to the URL

  https://api.foursquare.com/v2/checkins/recent?limit=100&oauth_token=" + oauth_token + "&v=" + version;

Example: Foursquare

```
{
 "response": {
  "recent": [
   {
    "id": "abc123",
    "createdAt": 1372597625,
    "type": "checkin",
    "timeZoneOffset": -240,
    "user": {
     "id": "12345",
     "firstName": "Joe",
     "lastName": "Smith",
    },
    "venue": {
     "id": "def456",
     "name": "Drexel University",
     "location": {
      "address": "3141 Chestnut Street",
      "lat": 40,
      "lng": -75,
      "postalCode": "19104",
      "city": "Philadelphia",
      "state": "PA",
      "country": "United States",
     }     }    }   ] }}
```

- And here’s the response!
- Tada! JSON

Getting the Check-ins

```
app.get('/FoursquareOauth',function(req,res){
        console.log(fs.OauthKey);
        fs.once('authenticated', function(msg){
                fs.getTable();
        });
        fs.once('gottable',function(msg){
                res.write(msg);
                res.end();
        });

        fs.authenticate(req.query.code)
});
```

```
//within Foursquare class…
        getTable (){
                var URL = https://api.foursquare.com/v2/checkins/recent?limit=100;
                URL+="&oauth_token=" + this.OauthKey + "&v=" + version,

                var self = this;

                request.get(URL, function(error,response,body){
                        var html = "<table><th>Name</th><th>Photo</th><th>Venue</th>";
                        var json = JSON.parse(body);
                        var keys = Object.keys(json.response.recent);
                        for(var i = 0, length=keys.length; i<length; i++){
                                var user = json.response.recent[keys[i]].user
                                html+="<tr><td>"+user.firstName+" " + user.lastName +"</td><td><img src='"+user.photo.prefix +
                                "40x40"+user.photo.suffix+"'/></td><td>"+json.response.recent[keys[i]].venue.name+"</td></tr>";
                        }
                        html += "</table>";
                        self.emit('gottable',html);
                });
        }
} //end of Foursquare class
```