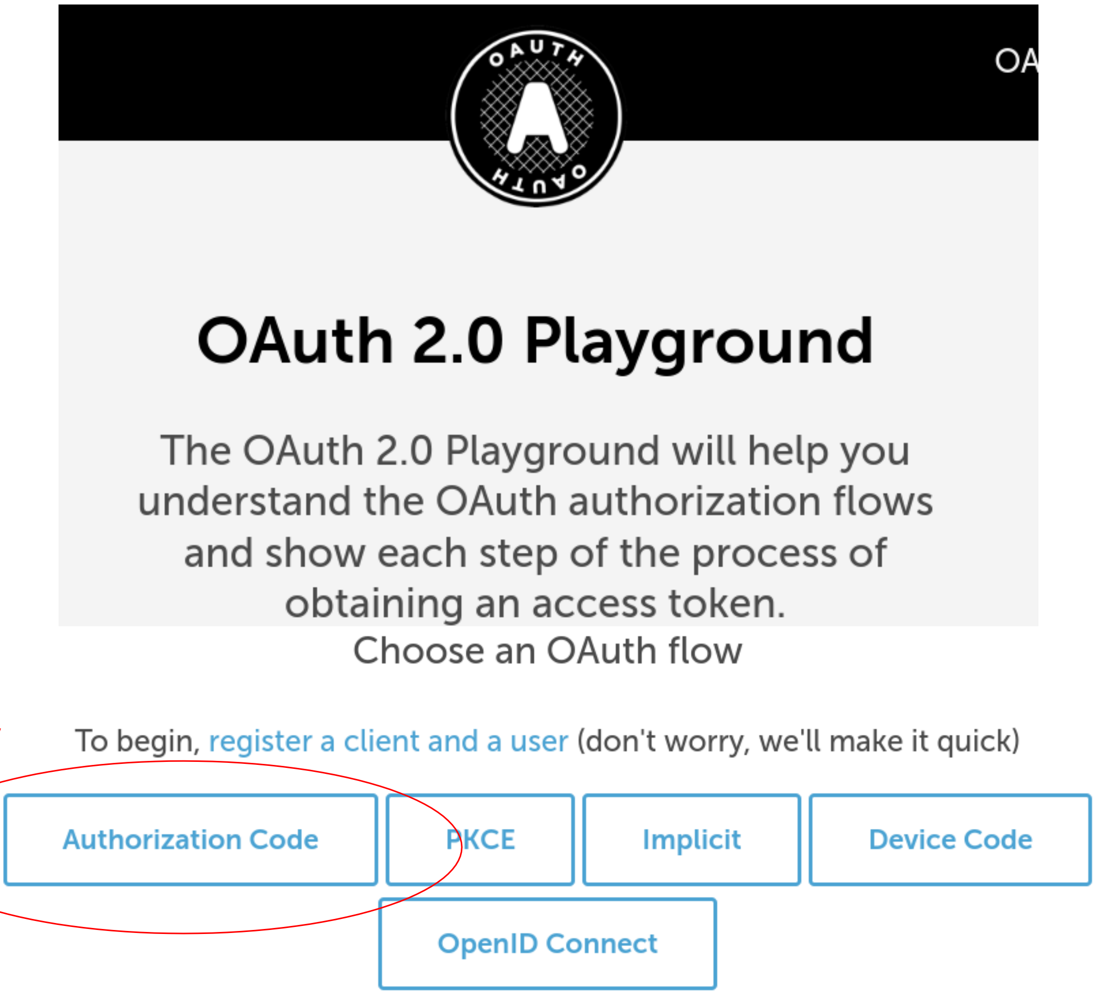
OAuth Practice

Using the OAuth Playground

Exercise: OAuth 2.0 Playground

https://www.oauth.com/playground/


OA

OAuth 2.0 Playground

The OAuth 2.0 Playground will help you understand the OAuth authorization flows and show each step of the process of obtaining an access token.

Choose an OAuth flow

To begin, register a client and a user (don't worry, we'll make it quick)

Choose "Authorization Code" Flow

Exercise: OAuth 2.0 Playground

Choose "**Authorization Code**" flow.

Work through the exercise.

Note:

you need to save the User login and password the site gives. Click "open in new window" to preserve it.

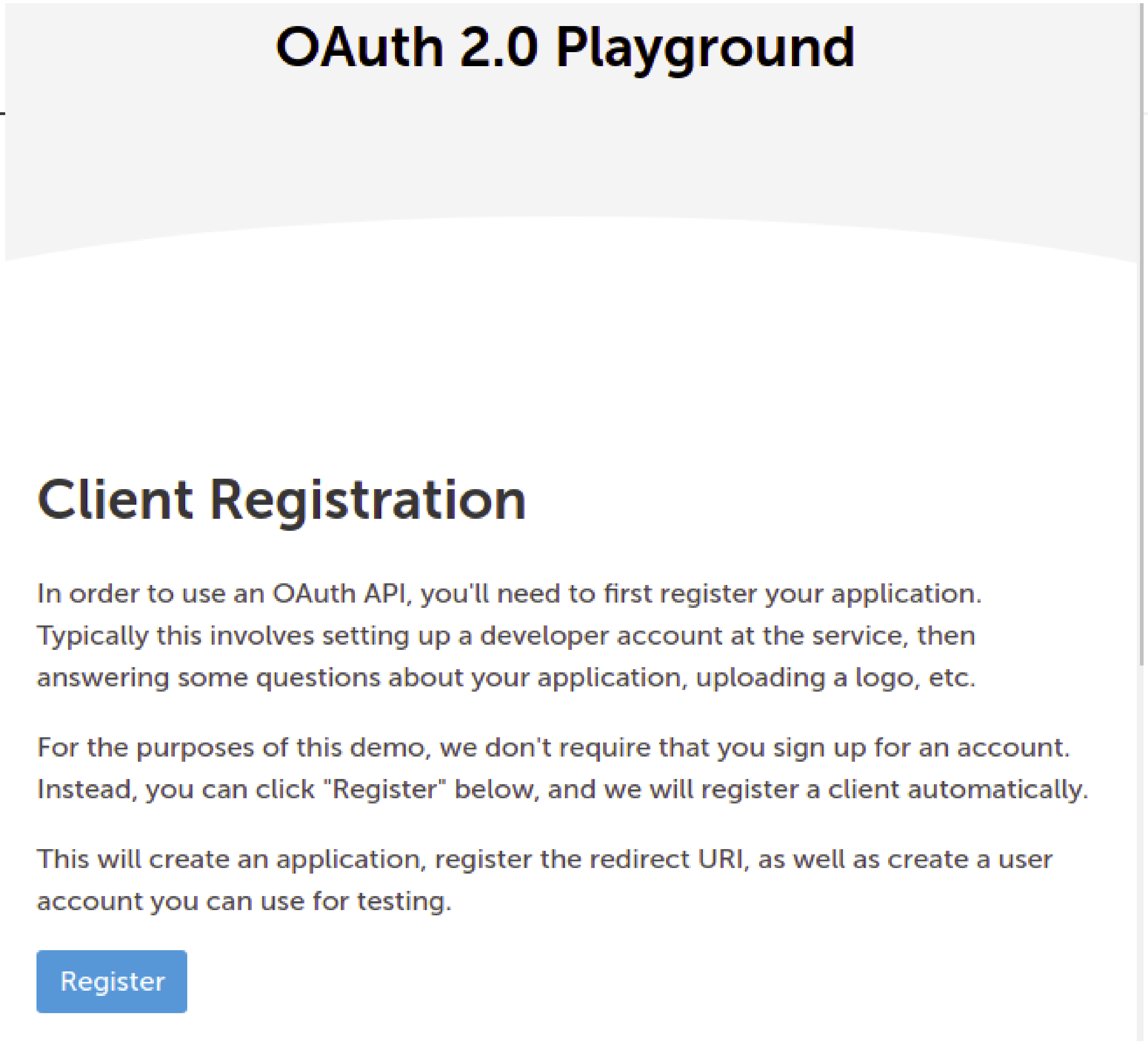
Next Slide for Important Terms

OAuth Terms

Client - application that wants access to a "user's" resource(s).

Authorization Server - a host that handle authentication on behalf of the Resource Server. This is where you register a Client application.

User - any end user of the Client application.



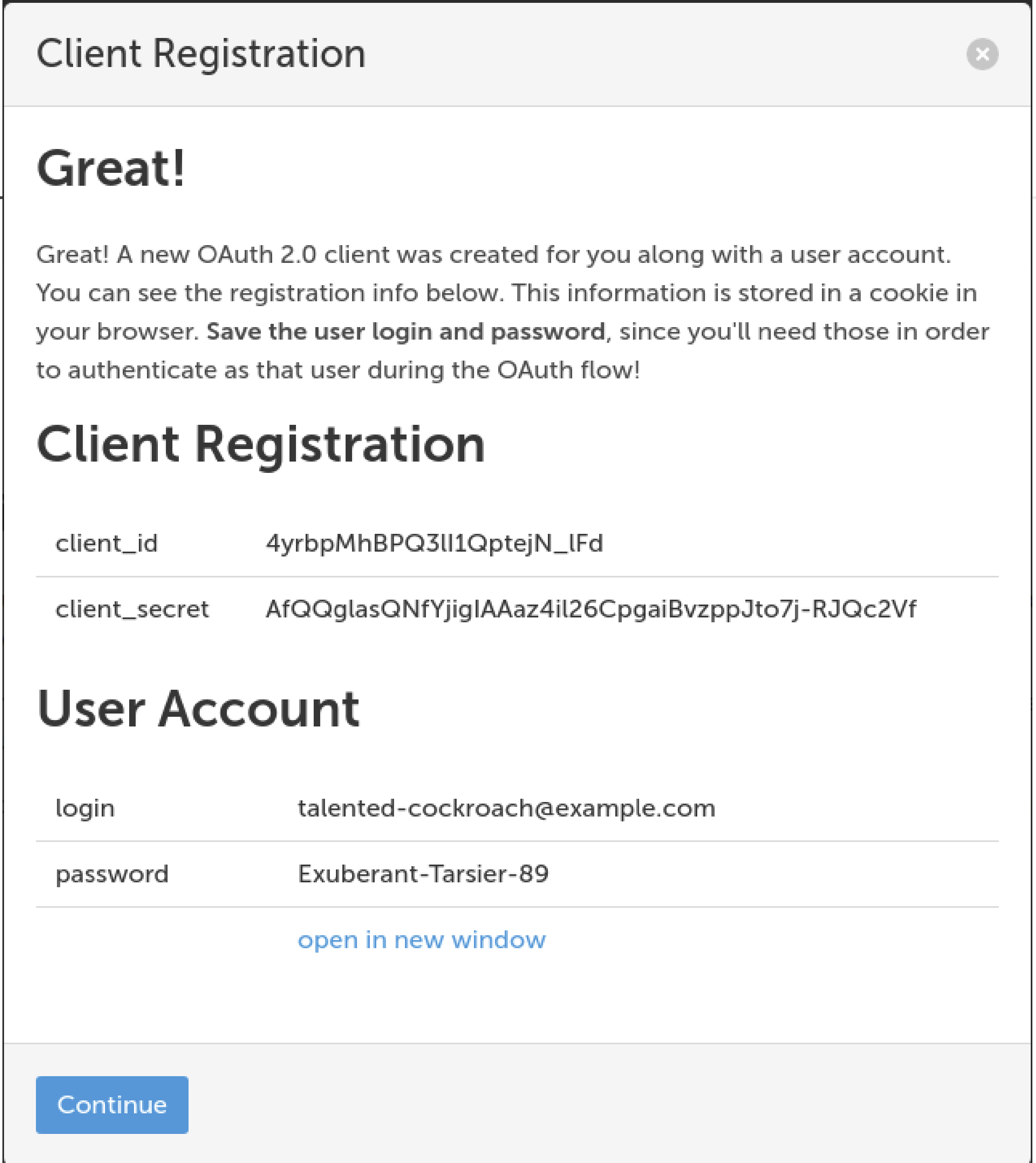
OAuth 2.0 Playground

Client Registration

In order to use an OAuth API, you'll need to first register your application. Typically this involves setting up a developer account at the service, then answering some questions about your application, uploading a logo, etc.

For the purposes of this demo, we don't require that you sign up for an account. Instead, you can click "Register" below, and we will register a client automatically.

This will create an application, register the redirect URI, as well as create a user account you can use for testing.



Client Registration

Great!

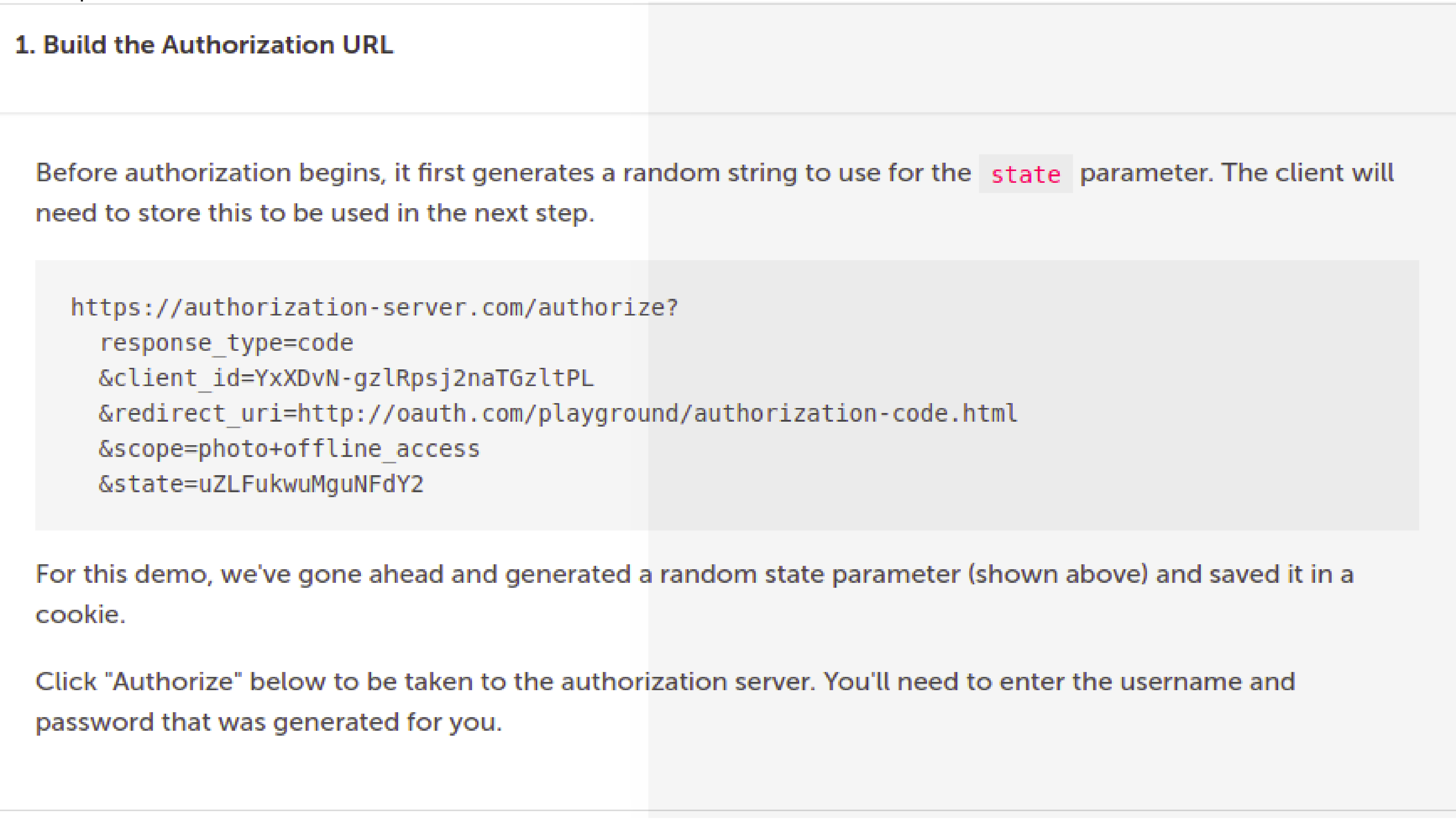
Great! A new OAuth 2.0 client was created for you along with a user account. You can see the registration info below. This information is stored in a cookie in your browser. **Save the user login and password**, since you'll need those in order to authenticate as that user during the OAuth flow!

Client Registration

client_id	4yrbpMhBPQ3lI1QptejN_lFd
client_secret	AfQQglasQNfYjigIAAaz4il26CpgaiBvzppJto7j-RJQc2Vf

User Account

login	talented-cockroach@example.com
password	Exuberant-Tarsier-89
	open in new window



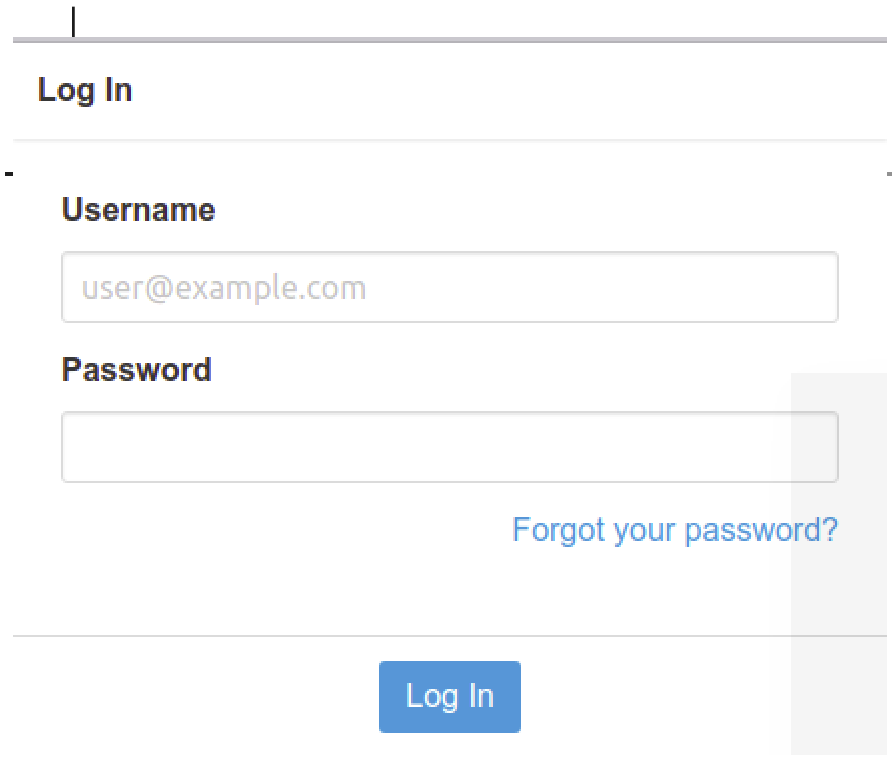
1. Build the Authorization URL

Before authorization begins, it first generates a random string to use for the `state` parameter. The client will need to store this to be used in the next step.

```
https://authorization-server.com/authorize?
  response_type=code
  &client_id=YxXDvN-gzlRpsj2naTGzltPL
  &redirect_uri=http://oauth.com/playground/authorization-code.html
  &scope=photo+offline_access
  &state=uZLFukwuMguNFdY2
```

For this demo, we've gone ahead and generated a random state parameter (shown above) and saved it in a cookie.

Click "Authorize" below to be taken to the authorization server. You'll need to enter the username and password that was generated for you.



Log In

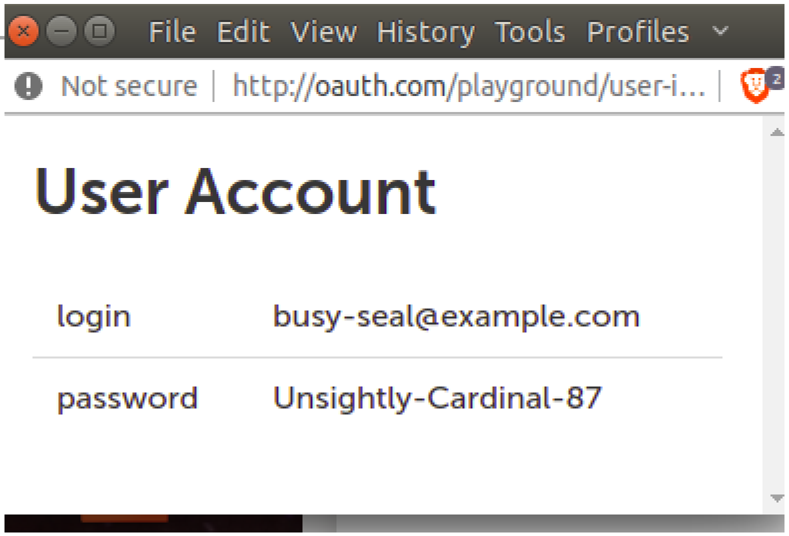
Username

user@example.com

Password

Forgot your password?

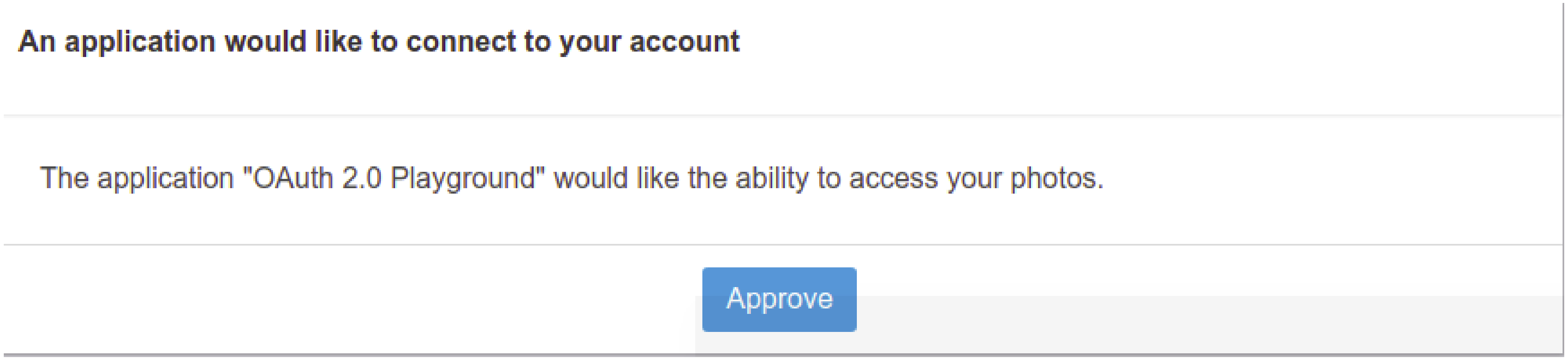
Log In



File Edit View History Tools Profiles
Not secure | http://oauth.com/playground/user-i...

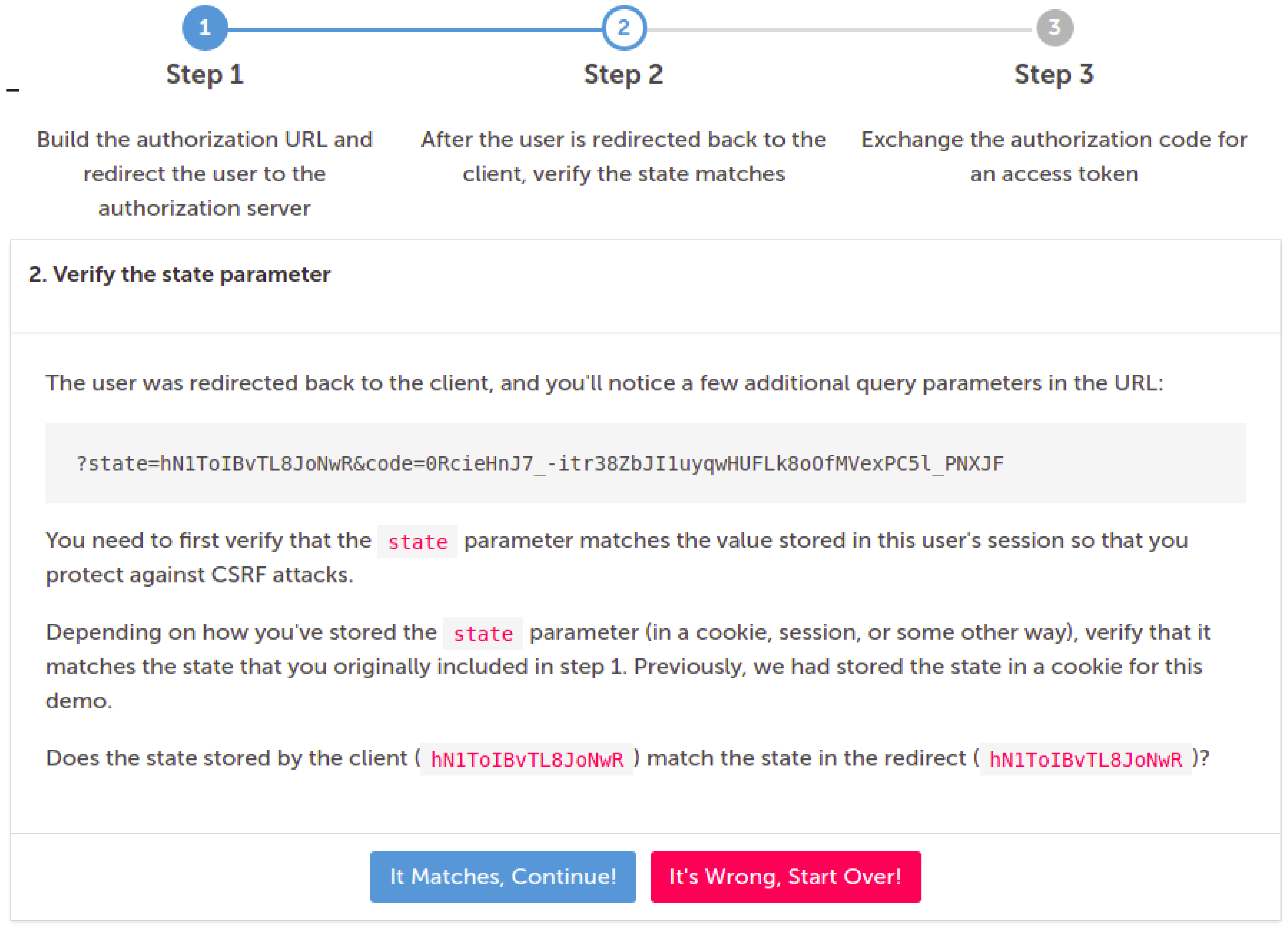
User Account

login	busy-seal@example.com
password	Unsightly-Cardinal-87



An application would like to connect to your account

The application "OAuth 2.0 Playground" would like the ability to access your photos.



1 Step 1

Build the authorization URL and redirect the user to the authorization server

2 Step 2

After the user is redirected back to the client, verify the state matches

3 Step 3

Exchange the authorization code for an access token

2. Verify the state parameter

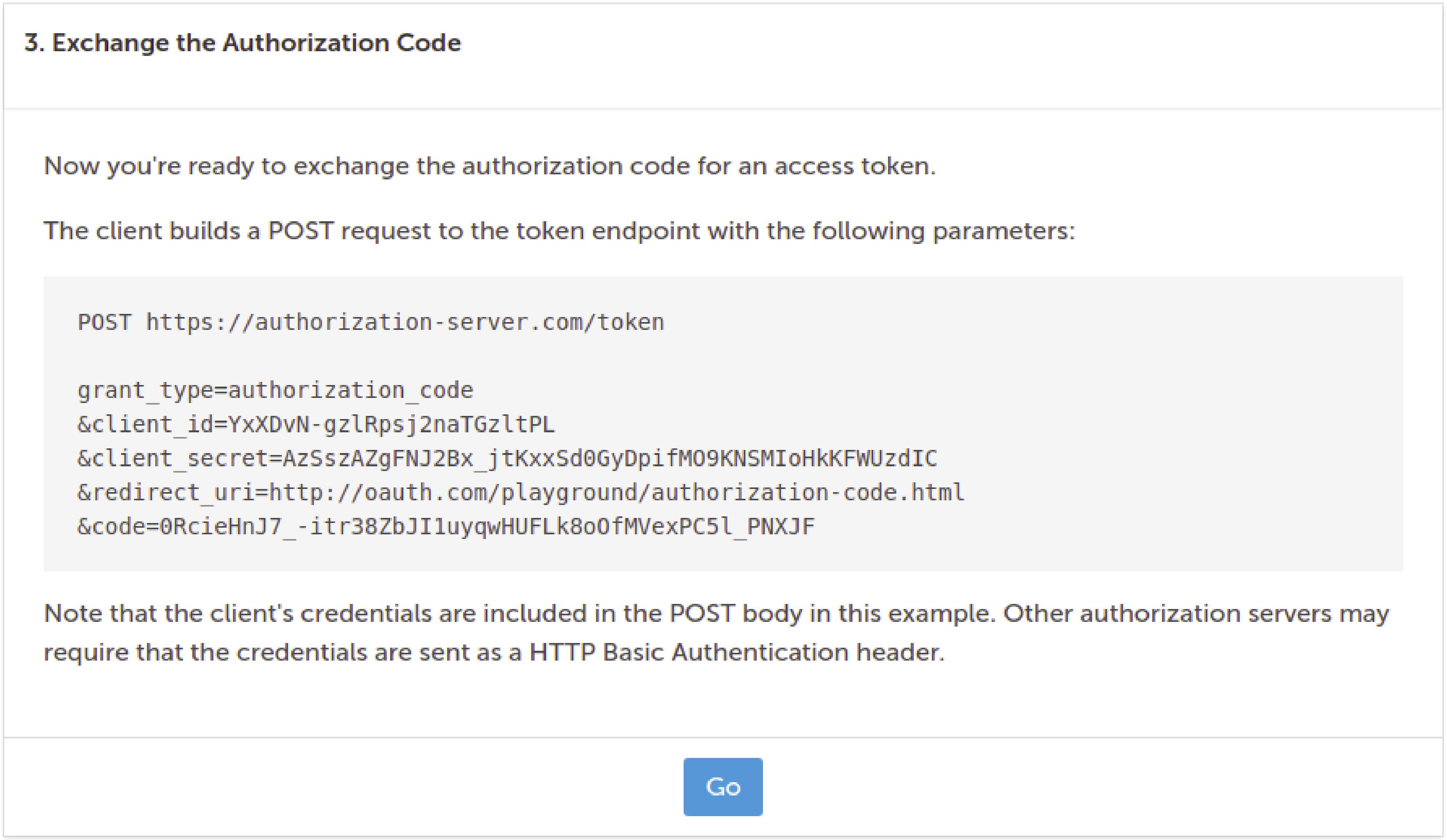
The user was redirected back to the client, and you'll notice a few additional query parameters in the URL:

```
?state=hN1ToIBvTL8JoNwR&code=0RcieHnJ7_-itr38ZbJI1uyqwHUFLk8oOfMVexPC5l_PNXJF
```

You need to first verify that the `state` parameter matches the value stored in this user's session so that you protect against CSRF attacks.

Depending on how you've stored the `state` parameter (in a cookie, session, or some other way), verify that it matches the state that you originally included in step 1. Previously, we had stored the state in a cookie for this demo.

Does the state stored by the client (`hN1ToIBvTL8JoNwR`) match the state in the redirect (`hN1ToIBvTL8JoNwR`)?



3. Exchange the Authorization Code

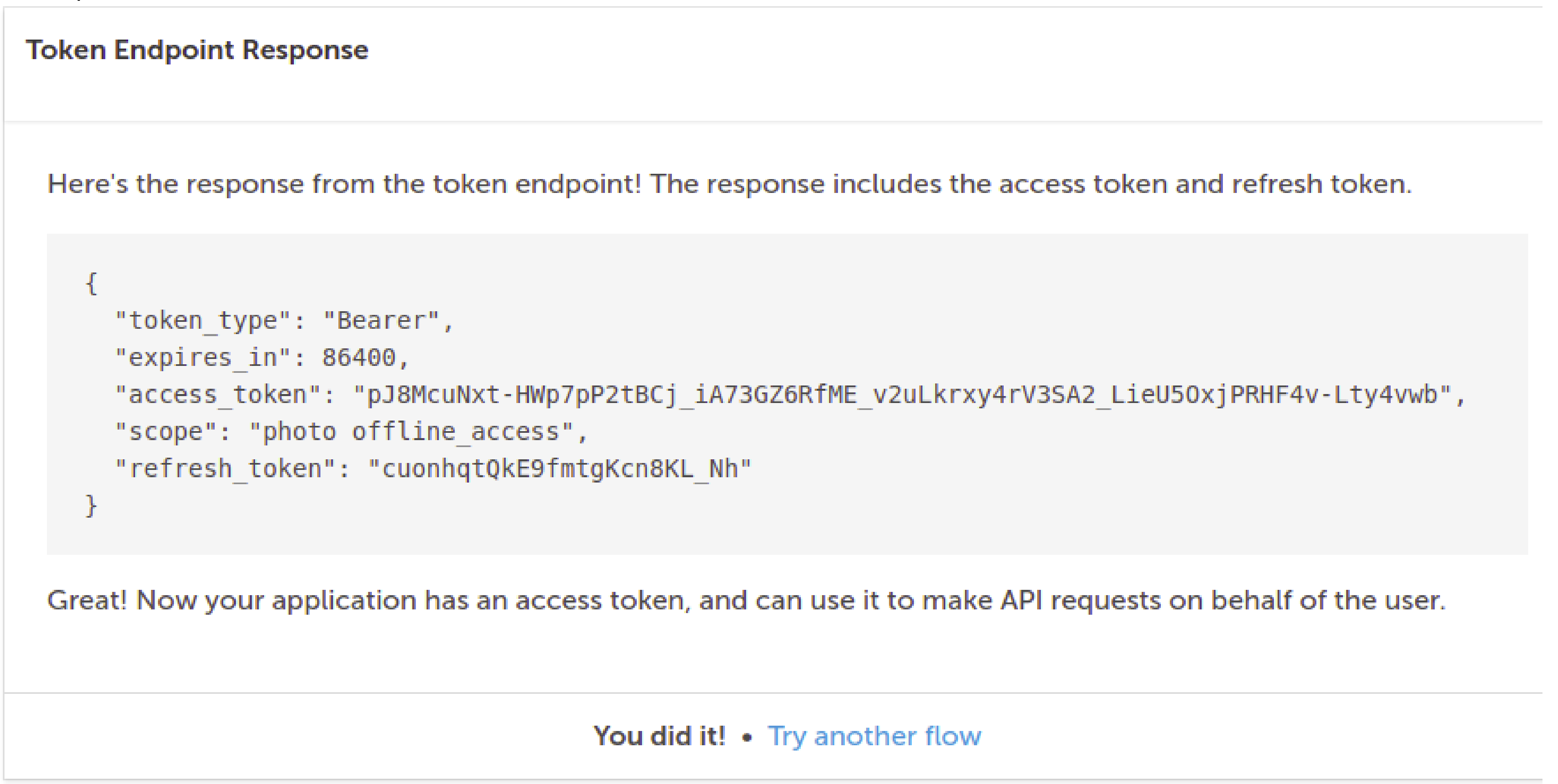
Now you're ready to exchange the authorization code for an access token.

The client builds a POST request to the token endpoint with the following parameters:

```
POST https://authorization-server.com/token

grant_type=authorization_code
&client_id=YxXDvN-gzlRpsj2naTGzltPL
&client_secret=AzSszAZgFNJ2Bx_jtKxxSd0GyDpifMO9KNSMIoHkKFWUzdIC
&redirect_uri=http://oauth.com/playground/authorization-code.html
&code=0RcieHnJ7_-itr38ZbJI1uyqwHUFLk8oOfMVexPC5l_PNXJF
```

Note that the client's credentials are included in the POST body in this example. Other authorization servers may require that the credentials are sent as a HTTP Basic Authentication header.



Token Endpoint Response

Here's the response from the token endpoint! The response includes the access token and refresh token.

```
{
  "token_type": "Bearer",
  "expires_in": 86400,
  "access_token": "pJ8McuNxt-HWp7pP2tBCj_iA73GZ6RfME_v2uLkrxy4rV3SA2_LieU5OxjPRHF4v-Lty4vwb",
  "scope": "photo offline_access",
  "refresh_token": "cuonhqtQkE9fmtgKcn8KL_Nh"
}
```

Great! Now your application has an access token, and can use it to make API requests on behalf of the user.

You did it! • Try another flow