#### **OAuth and OpenID Connect**

OktaDev of Nate Barbettini

Industry best practices – Oauth and OpenID Connect

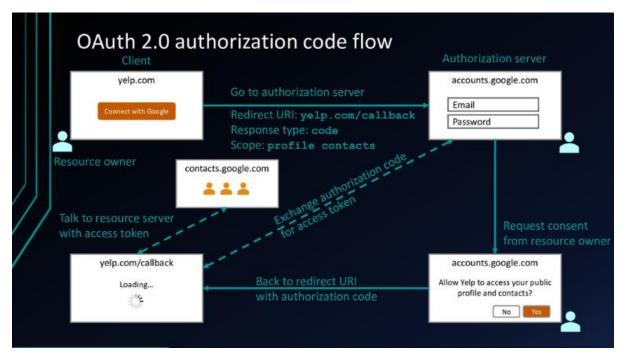
#### **Identity use cases (Pre-2010)**

- Simple login (forms and cookies)
- Single sign-on across sites (SAML employee accounts)
- Mobile app login (???)
- Delegated authorization (???) Where OAuth protocol began
  - How can a website access data w/o giving password?
    - Eg. Yelp example Wanted to obtain new users but requested user details (incl. password) to access your contacts
      - Yelp > Connect to Google Account > Prompted to login at Google domain (email and password) > Pop up to grant access to Yelp (yes/no) Oauth flow > redirected to URI/Callback (e.g. contacts for Google) > back to redirect URI > Exchange authorization code for access token > talk to resource server with access token

#### **OAuth 2.0 Terminology**

- Resource Owner End User
- Client Application used to access DB (i.e. Yelp)
- Authorization Server System used to grant access (i.e Google)
- Resource Server API holds data client wants to obtain (i.e. Google Contacts API)
- Authorization Grant Proves user gave consent
- Redirect URI aka Callback/Endpoint
- Access Token Client requirement. Key used to access data for specific requirement
- Scope Permissions (e.g. contact.read, contact.write, contact.delete)
- Consent List of scopes displayed to user whether app given access is allowed/not
- Back channel highly secure channel (e.g API/HTTP request between servers passes through SSL encryption, HTTPS etc.)
- Front channel less secure channel (e.g browser consists of loop holes

### **OAuth 2.0 Authorization Code Flow (General)**



### **Authorization Code vs. Access Token**

Back channel (highly secure channel) - Backend server to another API

Token exchange happens within backend

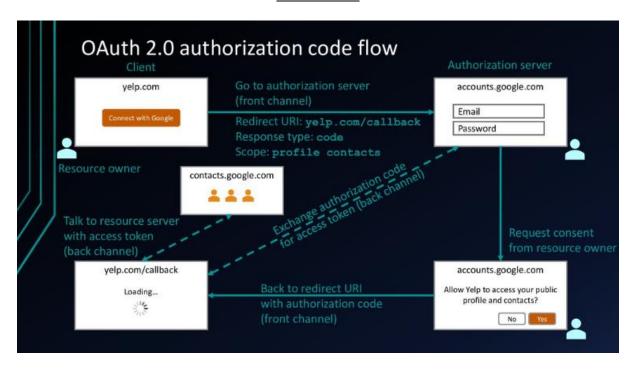
Client secret – don't want for it to be within the browser. Could be stolen

Talk to resource with access token

Front channel (less secure channel) – Browser with loop holes.

- Redirect URI, response type, scope, auth code (solid lines) – in the query parameters of address bar

Takes both into consideration so that it's highly secure



### Starting the Flow

```
Starting the flow

https://accounts.google.com/o/oauth2/v2/auth?
client_id=abc123&
    redirect_uri=https://yelp.com/callback&
    scope=profile&
    response_type=code&
    state=foobar
```

#### Create a client

Obtain ClientID and Client Secret

Receive authorization code

State – text value

```
Calling back

https://yelp.com/callback?
error=access_denied&
error_description=The user did not consent.

https://yelp.com/callback?
code=oMsCeLvIaQm6bTrgtp7&
state=foobar
```

Error received if parameters not set up accordingly

Authorization code returned if all goes accordingly

```
Exchange code for an access token

POST www.googleapis.com/oauth2/v4/token
Content-Type: application/x-www-form-urlencoded

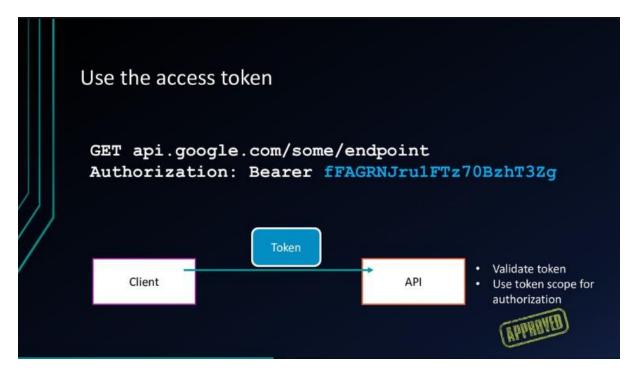
code=oMsCelvIaQm6bTrgtp7&
client_id=abc123&
client_secret=secret123&
grant_type=authorization_code
```

• HTTP Post to Auth server on different route, token endpoint

```
Authorization server returns an access token

{
    "access_token": "fFAGRNJru1FTz70BzhT3Zg",
    "expires_in": 3920,
    "token_type": "Bearer",
}
```

Auth code – displays number, duration of validity

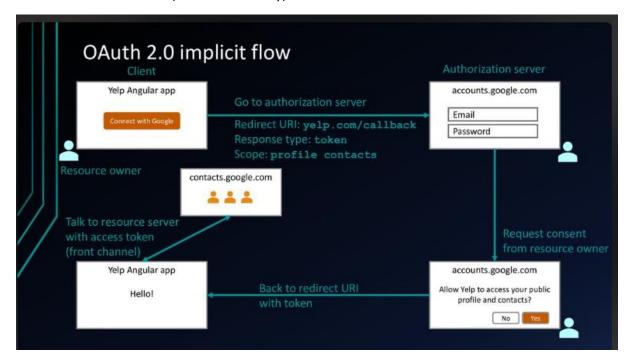


Access token used to make a request

Attach HTTP auth header to request with bearer

#### **OAuth 2.0 Flows**

- Auth code (best of both front and back channel)
- Implicit (front channel only) single page app in Java
  - Access token received without authorization code exchange step occurred in the background
- Resource owner password credentials (back channel only)
- Client credentials (back channel only)



Access token issued immediately. Review response type

### Identity use cases (pre -2014)

Don't use OAuth for authentication due to:

- No standard way to get user's information.
- Every implementation is a little different
- No common set of scopes

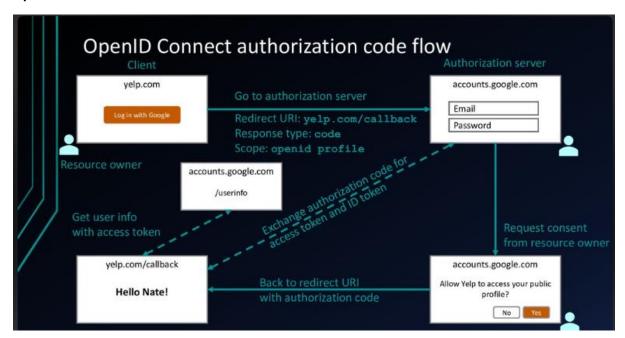
OpenID is for authentication

OAuth is for authorization

#### OpenID add to OAuth:

- ID Token
- UserInfo endpoint for getting more user info
- Standard set of scopes
- Standardized implementation

### **OpenID Connect authorization code flow**



- Get back ID token immediately consumed and decoded
- If more info required, use same access token to obtain the info

### Starting the flow

```
https://accounts.google.com/o/oauth2/v2/auth?
client_id=abc123&
  redirect_uri=https://yelp.com/callback&
  scope=openid profile&
  response_type=code&
  state=foobar
```

### Exchange code for access token and ID token

```
POST www.googleapis.com/oauth2/v4/token
Content-Type: application/x-www-form-urlencoded

code=oMsCeLvIaQm6bTrgtp7&
client_id=abc123&
client_secret=secret123&
grant_type=authorization_code
```

```
Authorization server returns access and ID tokens

{
   "access_token": "fFAGRNJru1FTz70BzhT3Zg",
   "id_token": "eyJraB03ds3F..."
   "expires_in": 3920,
   "token_type": "Bearer",
}
```



JWT - Json web token

Go into a decoder, information is returned with different properties

Signature determines if ID token hasn't been modified etc. Done independently of auth server

```
The ID token (JWT)

(Header)

{
    "iss": "https://accounts.google.com",
    "sub": "you@gmail.com",
    "name": "Nate Barbettini"
    "aud": "s6BhdRkqt3",
    "exp": 1311281970,
    "iat": 1311280970,
    "auth_time": 1311280969,
}

(Signature)
```

```
Calling the userinfo endpoint

GET www.googleapis.com/oauth2/v4/userinfo
Authorization: Bearer fFAGRNJru1FTz70BzhT3Zg

200 OK
Content-Type: application/json

{
    "sub": "you@gmail.com",
    "name": "Nate Barbettini"
    "profile_picture": "http://plus.g.co/123"
}
```

Make use of access token to call userinfo end point

```
Identity Cases (Today)

Simple Login (OpenID Connect – Authentication)

Single sign on across sites (OpenID Connect – Authentication)
```

Mobile app login (OpenID Connect – Authentication)

Delegated authorization (OAuth 2.0 – Authorization)

### Grant Flow to be used:

- Web application with backend server: authorization code flow
- Native mobile app (iOS and Android): authorization code flow with PKCE
- JS App with API backend: Implicit flow
- Microservices and APIs: client credentials flow

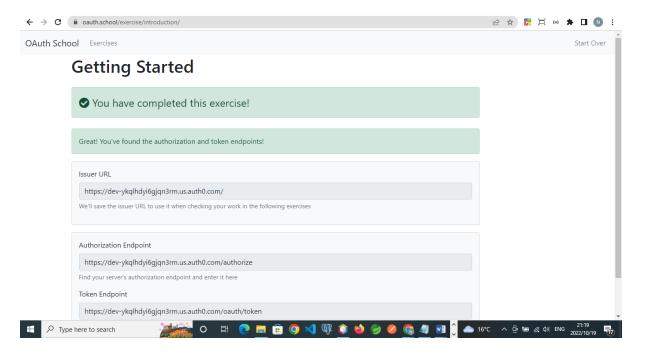
#### 1. Getting Started

Lesson Outcome: To sign up for a developer account, enabling one to follow allow with exercises

Create account

Create an API Resource

API resource is protected with AuthO access tokens



#### Set default audience

End goal is to use the access tokens with your own API, either set a default audience on AuthO tenant or specify audience in each authorization request.

If you wanted to segment your account into multiple API resources, have each OAuth client specify audience for each request

Find Issuer URI – Identifier of your API Endpoints – OpenID configuration URL

Find Auth and Token End Point

### 2. OAuth for Web Applications

Lesson Outcome: Obtain an access token using the authorization code flow and PKCE as a confidential client

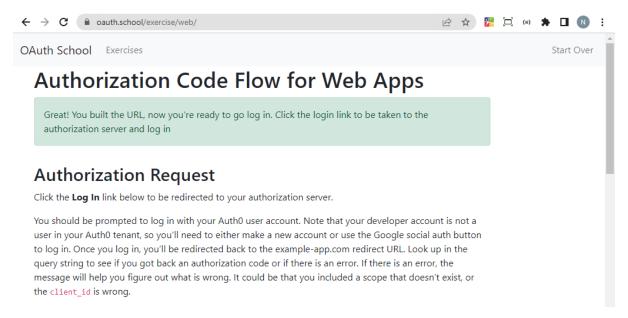
Authorization Code Flow + PKCE = OpenId Connect Flow designed to authenticate native or mobile application users

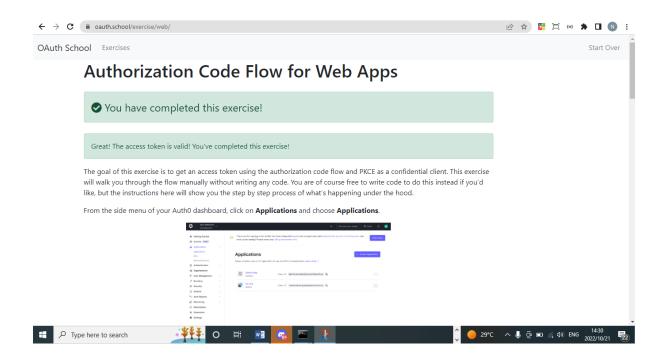
PKCE, pronounced "pixy" = Proof Key for Code Exchange

- Users aren't required to provide a client\_secret
- Reduces security risk for native apps
- Client app creates a unique string value, code verifier, hashes and encodes as a code challlenge.
- Client app initiates Authorization Code flow, hashed code\_challenge sent
- PKCE used by mobile apps

#### **Code Challenge creation**

- 1. Unique string created = code verifier
- 2. Base64-URL encoded SHA256 hash of random string generated. Generated from random string





### 3. Authorization Code Flow for Single-Page Apps

OAuth School Exercises Start Over

# Authorization Code Flow for Single-Page Apps

Great! You built the URL, now you're ready to go log in. Click the login link to be taken to the authorization server and log in

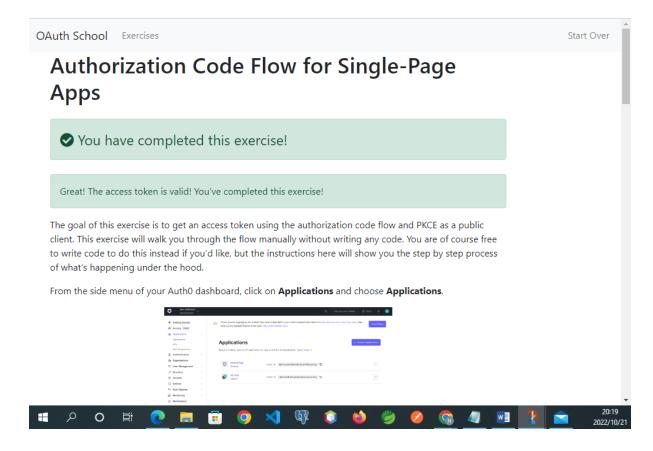
### **Authorization Request**

Click the **Log In** link below to be redirected to your authorization server.

You should be prompted to log in with your Auth0 user account. Note that your developer account is not a user in your Auth0 tenant, so you'll need to either make a new account or use the Google social auth button to log in. Once you log in, you'll be redirected back to the example-app.com redirect URL. Look up in the query string to see if you got back an authorization code or if there is an error. If there is an error, the message will help you figure out what is wrong. It could be that you included a scope that doesn't exist, or the client\_id is wrong.

Access token for Single Page apps generated

Get an access token using authorization code flow and PKCE as a public client



### 4. OAuth for Machine-to-Machine Applications

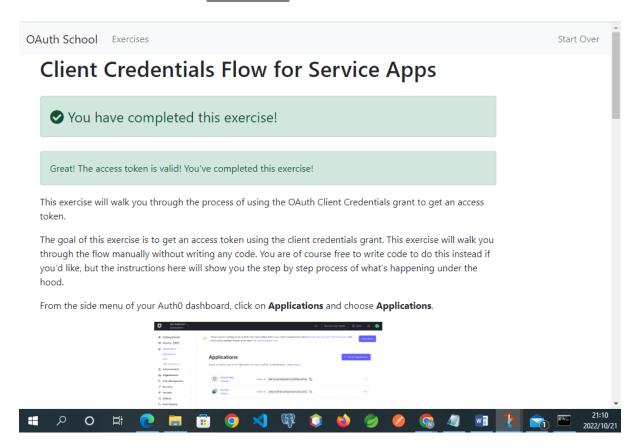
Client credentials flow for service apps

In creating the application, authorize link to API

A Machine to Machine Application represents a program that interacts with an API where there is no user involved. An example would be a server script that would be granted access to consume a Zip Codes API. It's a machine to machine interaction.

Since this is a machine-to-machine flow, there is no user involved in the flow so there is no browser involved either

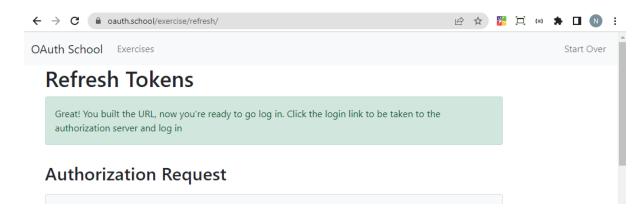
Application can make a direct request to the authorization server's token endpoint to get an access token.



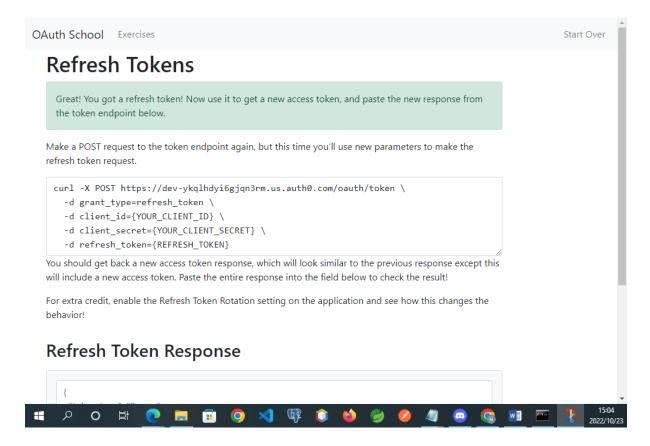
### 5. Refresh Tokens

In this exercise you'll learn how to obtain a refresh token and use it to get new access tokens

- Enable the Allow Offline Access Toggle
- Able to request refresh tokens for API
- Scope "offline\_access" added to request

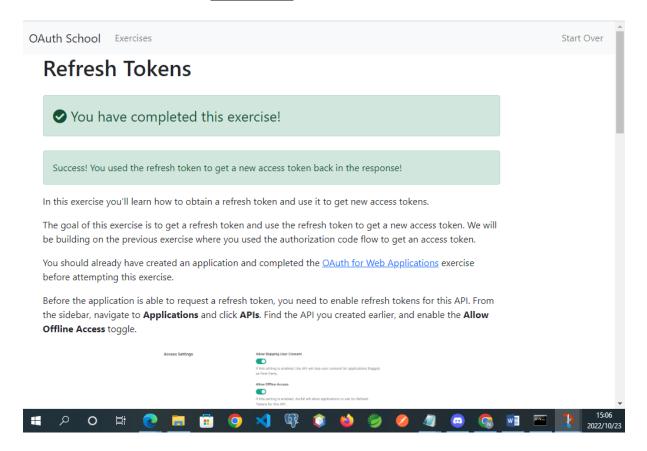


Received an error when creating authorization code. Went into advanced settings of application > Grant Types > Grants (Implicit, Auth Code, Refresh Token, Client Credentials)



Refresh Token received

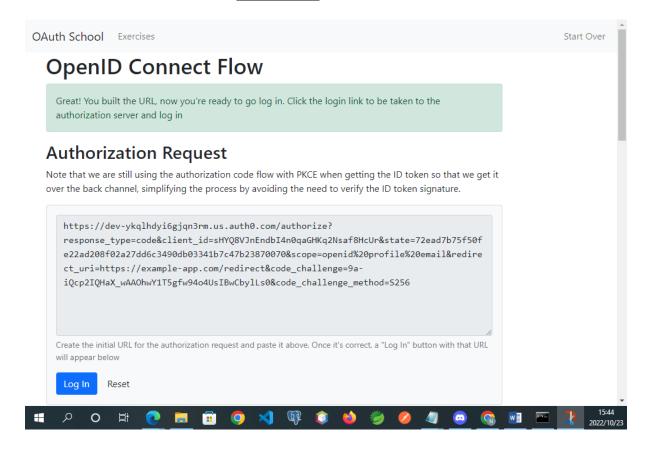
Posted another request using cURL, with the refresh token received

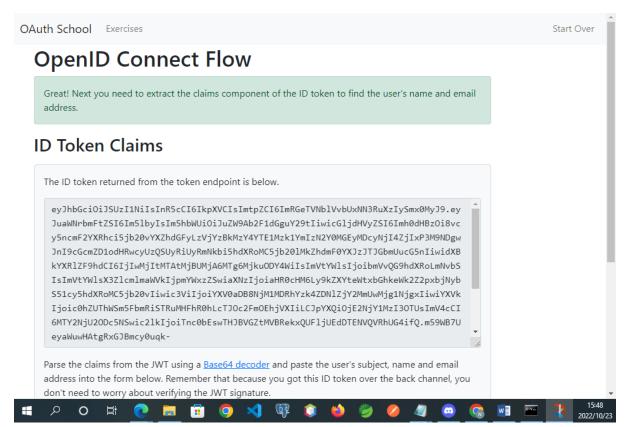


#### 6. OpenID Connect

- Request an OpenID Connect ID token and extract the user's information from it using a JWT decoder.
- We will be building on the previous exercise where you used the authorization code flow to get an access token.
- To get an ID token, you need to add the Openid scope to the authorization request. You can also add the profile and email scopes to get more information about the user.

When adding in scopes into authorization request, include "%20" between different scopes. All space delimited





Next Steps: Extract claims component of ID token to find user's name and email address

Decode ID Token

#### Base64 Decode

```
CMTePix60-jyFiNc21 XOA-caBr--
vMJvWt8xyJLczVX531EXb_qpvHjkUEEPOOpcG42JZB9wAm
3Z6fyFDyYbarNg0-3q0vZHJeOpBaGCObLOp61-
RHkI9pTog2iGap2_BK8PE5xuaivOsLRD8xj129-
VibAQwum4217HSU4Ep7sevVS44Orjfg7B6UODH2qcCf26E
TSP2Araux-ns_TaoN6A

Decode | Encode t

{
    "alg": "RS256",
    "tyP": "JWT",
    "kid": "dFy5MnUomLM7tn_22Jlt3"
}

{
    "nickname": "neo",
    "name": "heo@oauth.com",
    "picture":
"https://s.gravatar.com/avatar/5cc0d368a15395b
b37f40a2072628f21?
s=4804r=pg4d=https%3A%2F%2Fcdn.auth0.com%2Fava
tars%2Fne.png",
    "updated at": "2022-10-20T20:18:29.868Z",
    "email": "neo@oauth.com",
    "email verified": false,
    "isg": "https://dev-
```

You can learn more about OAuth 2.0 by reading OAuth 2.0 Simplified



OAuth School Exercises Start Over

### **OpenID Connect Flow**

✓ You have completed this exercise!

email: neo@oauth.com name: neo@oauth.com

Great work! You found the user's name and email address from the ID token!

### **ID Token Claims**

### The Nuts and Bolts of OAuth 2.0

This website is a companion to the course <u>The Nuts and Bolts of OAuth 2.0</u> and <u>Hands-on Introduction to OAuth 2.0</u> by Aaron Parecki. You should enroll in the course if you'd like to use this website!

#### What You'll Learn

OAuth 2.0, OpenID, PKCE, deprecated flows, JWTs, API Gateways, and scopes. No programming knowledge needed.

- ✓ The basics of OAuth 2.0 and OpenID Connect
- ✓ Best practices for developing OAuth applications (server-side, native, and SPAs)
- $\checkmark$  How to implement an OAuth client from scratch
- √ How to protect an API with JWT access tokens

### **Exercises**

- Getting Started
- OAuth for Web Applications
- OAuth for Single-Page Applications
- OAuth for Machine-to-Machine Applications
- Refresh Tokens
- OpenID Connect

