Notebook 01: Token acquisition notebook

Goal & Overview:

The goal of this notebook is to obtain valid tokens for several spotify users, whose accounts will be used in our data acquisition, and later analysis on spotify audio data. To do this, we drew from the Spotify web API, which proved fairly user friendly as it mapped closely with the terminology, and examples we used in class. Initially, we had considered using the Google Roads API to get travel data. However, after considering the small pay off of data for a sizable amount of work, we decided to use the Spotify API.

Set up, and creating an auth_url:

In [70]:
```
#General Set-up
import requests
from requests_oauthlib import OAuth2Session
import importlib
import key
import json
import pprint
import pprint
pp = pprint.PrettyPrinter(indent=2)
```

In [97]:
```
with open('keychain.txt') as json_file: #loads keychain
    keychain = json.load(json_file)

client_id = keychain['spotify']['client_id'] #creates some useful variab
les for the o-auth dance
scope = keychain['spotify']['scope']
redirect = keychain['spotify']['redirect_uris'][1]

session = OAuth2Session(client_id,scope=scope,redirect_uri=redirect)
auth_url, state = session.authorization_url(keychain['spotify']['auth_ur
i'])
print(auth_url) #valid auth_url
```

```
https://accounts.spotify.com/authorize?response_type=code&client_id=0eb
82343ee0c43e2ba1266eaa498701f&redirect_uri=https%3A%2F%2Flocalhost&scop
e=playlist-read-private+playlist-read-collaborative&state=k4U2VEhCMEN08
8nWhuMUHxGaG4SxEI
```

Additional Comments:

- While this process seems pretty straight forward, we were confused about where to grab the authorization code after clicking on the authorization url. However, we discovered that the code was embedded in the url, which was a small, but tricky difference from the google drive example.

Acquiring Users tokens, refreshing, and dumping JSON:

In [33]:
```
#Authorization codes or sampled users
code =
code2 =
code3 =
code4 =
code5 =
```

In [ ]:
```
#fetching tokens
token = session.fetch_token(keychain['spotify']['token_uri'],
                                    code=code,
                                    client_secret=keychain['spotify']['cl
ient_secret'])
token2 = session.fetch_token(keychain['spotify']['token_uri'],
                                    code=code2,
                                    client_secret=keychain['spotify']['cl
ient_secret'])
token3 = session.fetch_token(keychain['spotify']['token_uri'],
                                    code=code3,
                                    client_secret=keychain['spotify']['cl
ient_secret'])
token4 = session.fetch_token(keychain['spotify']['token_uri'],
                                    code=code4,
                                    client_secret=keychain['spotify']['cl
ient_secret'])
token5 = session.fetch_token(keychain['spotify']['token_uri'],
                                    code=code5,
                                    client_secret=keychain['spotify']['cl
ient_secret'])
print(token)
print(token2)
print(token3)
print(token4)
print(token5)
```

In [36]:
```
#assigns tokens to the keychain
keychain['spotify']['owners']['eric'] =  token
keychain['spotify']['owners']['miranda'] = token2
keychain['spotify']['owners']['gezim'] = token3
keychain['spotify']['owners']['luke'] = token4
keychain['spotify']['owners']['ben'] = token5
```

In [41]:
```
with open('keychain.txt', 'w') as outfile: #updates the keychain file wi
th new tokens
    json.dump(keychain, outfile)
```

In [ ]:
```
#refreshing my token
refresh_url = keychain['spotify']['token_uri']
token = session.refresh_token(refresh_url, #refreshes token
            client_id=keychain['spotify']['client_id'],
            client_secret=keychain['spotify']['client_secret'])
```

Additional Comments:

- Our biggest hang-up on this section was thinking about how to dump, and load to the token data to the keychain file. We could probably have done this part of the notebook more programmatically, but we acquired the tokens slowely throughout our project, and we can only run the fetch_token function once. NOTE: we do use functional abstraction in notebook 02 to refresh users' tokens, and access end-points with said tokens.

Prototyping for Notebook 02

In [94]:
```
token = keychain['spotify']['owners']['eric']
D = {}
D['access_token'] = token['access_token']
url = 'https://api.spotify.com/v1/me'
response = requests.get(url, params=D)
user = json.loads(response.text)
print(user)
print(user['id'])
user_id = user['id'] #getting user id, need to access user's playlists
```

```
{'country': 'US', 'display_name': 'Eric Buehler', 'email': 'buehlere77@
gmail.com', 'external_urls': {'spotify': 'https://open.spotify.com/use
r/1266353543'}, 'followers': {'href': None, 'total': 44}, 'href': 'http
s://api.spotify.com/v1/users/1266353543', 'id': '1266353543', 'images':
[{'height': None, 'url': 'https://scontent.xx.fbcdn.net/v/t1.0-1/p200x2
00/20994067_2015853548440610_4949905420302764024_n.jpg?oh=47ce6406b472d
97fb54053eb548f800d&oe=5A8939F6', 'width': None}], 'product': 'premiu
m', 'type': 'user', 'uri': 'spotify:user:1266353543'}
1266353543
```

In [95]:
```
D = {}
D['access_token'] = token['access_token']
url = 'https://api.spotify.com/v1/users/{}/playlists'
response = requests.get(url.format(user_id), params=D) #accessing user's
 playlists
playlist = json.loads(response.text)
#pp.pprint(playlist)
print(playlist['items'][0]['id']) #playlist id
playlist_id = playlist['items'][5]['id'] #saving a track id from playlis
t
```

```
37i9dQZF1E9UBfOE5yO2Rg
```

In [96]:
```
D = {}
D['access_token'] = token['access_token']
url = 'https://api.spotify.com/v1/users/{}/playlists/{}/tracks'
response = requests.get(url.format(user_id,playlist_id), params=D) #grab
bing track info
tracks = json.loads(response.text)
#pp.pprint(tracks)
print(tracks['items'][0]['track']['id']) #track id
```

```
4iVKvR6Y3LwBaOBSQKsqj0
```

Additional Comments:

- These get requests are the basis for the functional abstraction used in notebook02. user ID ---> playlist info ---> track info ---> audio info

In [ ]:
```

```