app.py

change #1

```python
import urllib.parse
from flask import Flask, redirect, url_for, render_template, request, jsonify, session
from dotenv import load_dotenv
import os
import spotipy
import urllib
import requests
from datetime import datetime
```

imported necessary libraries

change #2

```python
#initialize web app
app = Flask(__name__)
app.config['SECRET_KEY'] = os.urandom(64)
```

changed how the app generates its secret key, rather than pulling a fixed key from the .env file, it generates a new one each time the app is run now

change #3

```python
#store access token in flask session
cache_handler = FlaskSessionCacheHandler(session)
```

moved the scope variable outside of the login route

change #4

```python
#store access token in flask session
cache_handler = FlaskSessionCacheHandler(session)
```

created cache handler, with this the app can automatically check if the user has a valid access token

change #5

```python
#create authentication manager
sp_oauth = SpotifyOAuth(
    client_id=client_id,
    client_secret=client_secret,
    redirect_uri=REDIRECT_URI,
    scope=scope,
    cache_handler=cache_handler,
    show_dialog=True
)
```

created oauth manager, as the name entails it manages authorization, obtains tokens, manages scopes, etc.

change #6

```python
REDIRECT_URI = "http://localhost:5000/access_token"
```

```python
#get access token
@app.route('/access_token')
def access_token():
    #check if error occured while logging in
    if 'error' in request.args:
        return jsonify({"error": request.args['error']})

    #if user successfully logs in
    if 'code' in request.args:
        #create request_body to exchange it for access token
        request_body = {
            'code': request.args['code'],
            'grant_type': 'authorization_code',
            'redirect_uri': REDIRECT_URI,
            'client_id': client_id,
            'client_secret': client_secret
        }

        #send request body to token url to get access token
        response = requests.post(TOKEN_URL, data=request_body)
        token_info = response.json()

        #store acces token and refresh token within session
        session['access_token'] = token_info['access_token']
        session['refresh_token'] = token_info['refresh_token']

        #store token expiration date within token
        session['expires_at'] = datetime.now().timestamp() + token_info['expires_in']

        #redirect user to page with their display name and email
        return redirect('/display_name')

```

created route to exchange authorization for an access token necessary for making api calls

change #7

```python
@app.route('/refresh_token')
def refresh():
    #check if refresh token is in session
    if 'refresh_token' not in session:
        #if refresh token is not in session, prompt user to log in again
        return redirect('/login')

    #check if access token has expired
    if datetime.now().timestamp() > session['expires_at']:
        #build request body
        request_body = {
            'grant-type': 'refresh_token',
            'refresh_token': session['refresh_token'],
            'client_id': client_id,
            'client_secret': client_secret
        }

        #request fresh access token using request body
        response = requests.post(TOKEN_URL, data=request_body)
        #store fresh token
        new_token_info = response.json()

        #update session with new token and new expiration time
        session['access_token'] = new_token_info['access_token']
        session['expires_at'] = datetime.now().timestamp() + new_token_info['expires_in']

        #redirect user to page with their display name and email
        return redirect('/display_name')

```

created route to check if access token has expired, if it has, then the user is provided with a new one, given they have authenticated properly

change #8

```python
@app.route('/display_name')
def display_name():
    #if user does not have access token, prompt them to log in
    if 'access_token' not in session:
        return redirect('/login')

    #if token has expired, call for refresh token
    if datetime.now().timestamp() > session['expires_at']:
        return redirect('/refresh_token')

    sp = spotipy.Spotify(auth_manager=sp_oauth, auth=session['access_token'])

    #get user information
    username = sp.current_user()['display_name']
    user_email = sp.current_user()['email']

    #print username and email in terminal, used for debugging
    print(username)
    print(user_email)

    #render html page, pass username and user_email variables so they can be displayed
    return render_template('display_name.html', username=username, user_email=user_email)

```

created route to display the user's username and email once they log in successfully

instance of spotipy is created on line 139, this instance is used to get username and user email

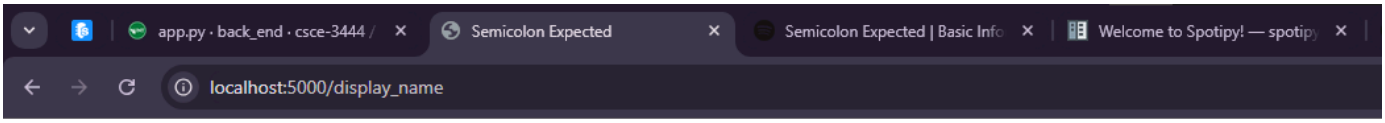
display_name.html

```html
{% extends "base.html" %}

{% block head %}
<title>Semicolon Expected</title>
{% endblock %}

{% block body %}
<h1>Username: {{ username }}</h1>
<h1>Email: {{ user_email }}</h1>
{% endblock %}
```

html for the page displaying the information gathered from display_name route

SUMMARY

changed how app secret key is generated and handled, created authorization manager and cache handler, user is now provided an spotify api access token after logging in, created route to refresh access token if it expires, created route which makes api calls to get the user's display name as well as email which is then displayed on screen



Username: angelsolis

Email: angelsolis2004@gmail.com