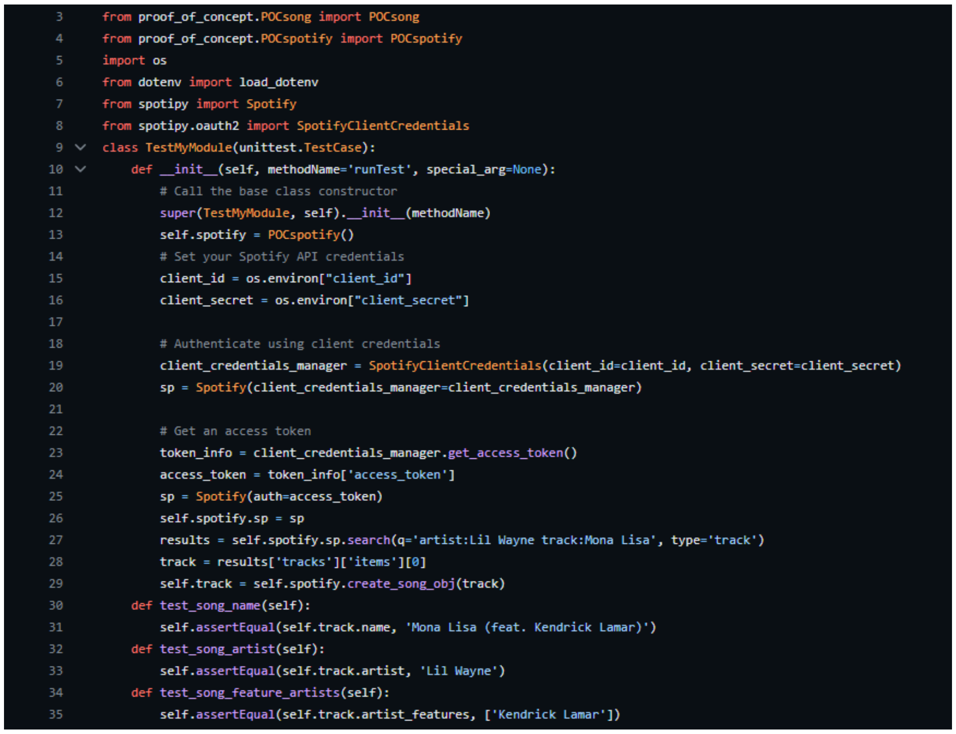
CI & Testing Deliverable

Authors: Garrett Phillips, Benjamin Boardley, Jack Gardner
Course: ECE49595O - Spring 2024
Date: 02/07/2024

1. The requirement / success criteria that we will encode into our pipeline will be:
   "Users can create content-based posts referencing songs, playlists, or music groupings."
   We will integrate an automatic test into our github actions pipeline to ensure that a user can successfully create a post referencing songs, playlist, etc. The test would simulate a user creating a post, referencing a song or playlists through an admin account, it would then verify that the post is created successfully and can be viewed / accessed by other users. This would ensure that this aspect of the application's functionality is continuously tested and validated with each code change, helping to maintain the overall quality and reliability of the application.

2. We are currently performing unit testing surrounding the functionality of our proof-of-concept. There are multiple classes (i.e. POCsong, POCplaylist, etc.) that hold the information received from the music streaming service (i.e. Spotify & Apple music), this allows for each api call to dump the information into one object or set of objects. This will ultimately add to the functionality of the program. The unit test shown below, which is run each time a pull request is created from a feature branch to our development branch, is performed on the POCsong class:

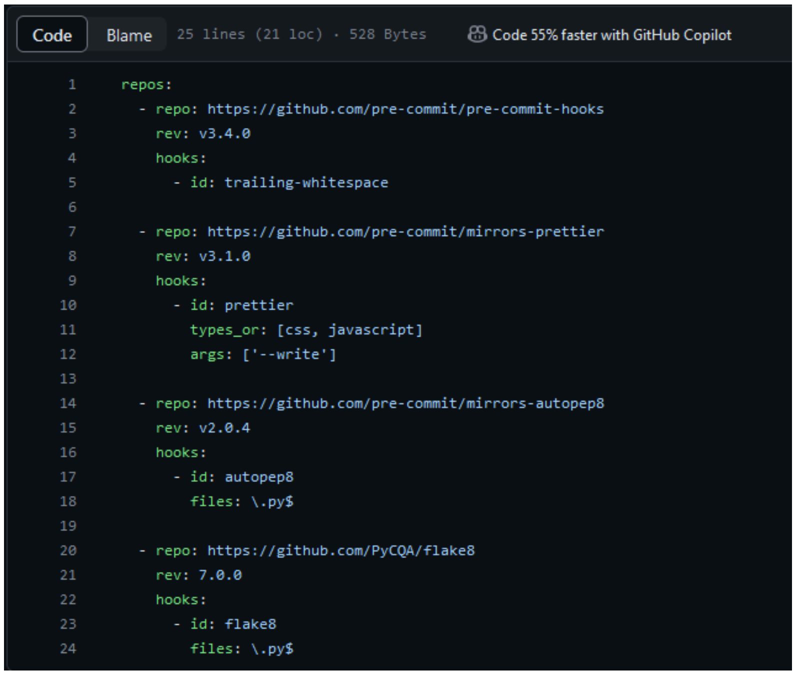


```python
from proof_of_concept.POCsong import POCsong
from proof_of_concept.POCspotify import POCspotify
import os
from dotenv import load_dotenv
from spotipy import Spotify
from spotipy.oauth2 import SpotifyClientCredentials
class TestMyModule(unittest.TestCase):
    def __init__(self, methodName='runTest', special_arg=None):
        # Call the base class constructor
        super(TestMyModule, self).__init__(methodName)
        self.spotify = POCspotify()
        # Set your Spotify API credentials
        client_id = os.environ["client_id"]
        client_secret = os.environ["client_secret"]

        # Authenticate using client credentials
        client_credentials_manager = SpotifyClientCredentials(client_id=client_id, client_secret=client_secret)
        sp = Spotify(client_credentials_manager=client_credentials_manager)

        # Get an access token
        token_info = client_credentials_manager.get_access_token()
        access_token = token_info['access_token']
        sp = Spotify(auth=access_token)
        self.spotify.sp = sp
        results = self.spotify.sp.search(q='artist:Lil Wayne track:Mona Lisa', type='track')
        track = results['tracks']['items'][0]
        self.track = self.spotify.create_song_obj(track)
    def test_song_name(self):
        self.assertEqual(self.track.name, 'Mona Lisa (feat. Kendrick Lamar)')
    def test_song_artist(self):
        self.assertEqual(self.track.artist, 'Lil Wayne')
    def test_song_feature_artists(self):
        self.assertEqual(self.track.artist_features, ['Kendrick Lamar'])
```

3. One way in which we incorporated the use of automated software quality assurance into our repository is through the use of pre-commit hooks. The pre-commit hooks we have currently implemented are as follows:


Code
Blame
25 lines (21 loc) · 528 Bytes
Code 55% faster with GitHub Copilot

```yaml
repos:
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v3.4.0
    hooks:
      - id: trailing-whitespace

  - repo: https://github.com/pre-commit/mirrors-prettier
    rev: v3.1.0
    hooks:
      - id: prettier
        types_or: [css, javascript]
        args: ['--write']

  - repo: https://github.com/pre-commit/mirrors-autopep8
    rev: v2.0.4
    hooks:
      - id: autopep8
        files: \.py$

  - repo: https://github.com/PyCQA/flake8
    rev: 7.0.0
    hooks:
      - id: flake8
        files: \.py$
```

This includes a whitespace linter, JS/CSS linter (Prettier), and a python linter (flake8 and autopep8). The python hook runs the Flake8 linter on Python files to detect and report style and syntax errors that Autopep8 hook was not able to fix. Our commit hooks can be set up using a setup.sh file we have present in our repository.