Deezer Album Tracker

This script uses the deezer public api to provide a list of albums released the past half year of artists in the configuration file. Configuration file will be created if it doesn't exist. Adding/removing artists can be done using command line options. The output can be emailed for easy use from cron with customisable subject line.

Prerequisites for fuzzy search:

```
pip install fuzzywuzzy
```

Usage:

```
usage: dat.py [-h] [--list] [--days DAYS] [--add ARTIST_NAME] [--delete
SEARCH_TERM] [--email]

Deezer Album Tracker

options:
  -h, --help            show this help message and exit
  --list                List all monitored artists
  --days DAYS           Amount of days to list
  --add ARTIST_NAME     Add a new artist
  --delete SEARCH_TERM  Delete an artist by fuzzy search
  --email               Email the output
```

Example output:

```
$ ./dat.py
Albums released in the past 6 months:
Release Date: 2024-01-12
Artist: Papa Roach
Album Name: Scars (feat. Chris Daughtry) (Live)

Release Date: 2023-11-03
Artist: Limp Bizkit
Album Name: Counterfeit Countdown

Release Date: 2023-10-31
Artist: Papa Roach
Album Name: Leave a Light On (Talk Away The Dark)
```

Example config file:

config.json

```
{
    "global": {
        "days": 180
```

```
    },
    "email": {
        "smtp_server": "emailserver",
        "smtp_port": 587,
        "sender_email": "emailaddress",
        "sender_password": "password",
        "email_recipients": [
            "email1@googlemail.com",
            "email2@googlemail.com"
        ],
        "email_subject": "Deezer Album Tracker"
    },
    "artist_ids": {
        "89": "Papa Roach",
        "566": "Foo Fighters",
        "93": "Limp Bizkit",
        "1070": "Puddle of Mudd",
        "373": "Staind"
    }
}
```

dat.py

```
#!/usr/bin/python
import ssl
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
import requests
import json
from datetime import datetime, timedelta
import time
import argparse
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
import os

# Constants for file paths
CONFIG_FILE = "config.json"

def load_config():
    if not os.path.exists(CONFIG_FILE):
        # Create default config file if it doesn't exist
        default_config = {
            "global": {
                "days": "180"
            },
            "email": {
                "smtp_server": "smtp.example.com",
```

```
                "smtp_port": 587,
                "sender_email": "sender@example.com",
                "sender_password": "password",
                "email_recipients": ["recipient1@example.com",
"recipient2@example.com"],
                "email_subject": "Deezer Album Tracker"
            },
            "artist_ids": []
        }
        with open(CONFIG_FILE, "w") as config_file:
            json.dump(default_config, config_file, indent=4)

    with open(CONFIG_FILE, "r") as config_file:
        return json.load(config_file)

def save_config(config):
    with open(CONFIG_FILE, "w") as config_file:
        json.dump(config, config_file, indent=4)

def send_email(subject, body, recipients):
    config = load_config()
    email_config = config.get('email', {})
    if not email_config:
        print("Email configuration not found in config file.")
        return

    smtp_server = email_config.get('smtp_server')
    smtp_port = email_config.get('smtp_port')
    sender_email = email_config.get('sender_email')
    sender_password = email_config.get('sender_password')

    msg = MIMEMultipart()
    msg['From'] = sender_email
    msg['To'] = ', '.join(email_config.get('email_recipients'))
    msg['Subject'] = f"{subject} - {datetime.now().strftime('%Y-%m-
%d')}"

    body = MIMEText(body)
    msg.attach(body)

    # Use TLS
    context = ssl.create_default_context()

    with smtplib.SMTP(smtp_server, smtp_port) as server:
        server.ehlo()  # Can be omitted
        server.starttls(context=context)
        server.ehlo()  # Can be omitted
        server.login(sender_email, sender_password)
        server.send_message(msg)

def get_artist_name(artist_id):
```

```
    url = f"https://api.deezer.com/artist/{artist_id}"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        return data.get('name', '')
    return ''

def get_artist_id(artist_name):
    url = f"https://api.deezer.com/search/artist?q={artist_name}"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        for artist in data.get('data', []):
            if fuzz.token_sort_ratio(artist_name, artist['name']) >=
90:
                return artist['id']
    return None

def get_albums(artist_ids,lookupdays):
    config = load_config()
    base_url = "https://api.deezer.com/artist/{}/albums"
    earliest_release = (datetime.now() -
timedelta(days=lookupdays)).strftime('%Y-%m-%d')
    albums = []
    request_count = 0
    start_time = time.time()

    for artist_id in artist_ids:
        url = base_url.format(artist_id)
        response = requests.get(url)
        request_count += 1
        if response.status_code == 200:
            data = response.json()
            artist_name = get_artist_name(artist_id)
            for album in data['data']:
                release_date = datetime.strptime(album['release_date'],
'%Y-%m-%d')
                if release_date >= datetime.strptime(earliest_release,
'%Y-%m-%d'):
                    albums.append({
                        'artist': artist_name,
                        'album_name': album['title'],
                        'release_date': album['release_date']
                    })

        # Deezer rate limit is 50 requests / 5 seconds. Limiting to
40/5 here:
        # Check if 40 requests have been made in less than 5 seconds
        if request_count == 40:
            elapsed_time = time.time() - start_time
```

```
            if elapsed_time < 5:
                time.sleep(5 - elapsed_time)
            # Reset request count and start time
            request_count = 0
            start_time = time.time()

    return sorted(albums, key=lambda x: x['release_date'],
reverse=True)

def list_artists():
    config = load_config()
    subscribed_artists = config.get('artist_ids', {})
    sorted_artists = dict(sorted(subscribed_artists.items(), key=lambda
item: item[1].casefold()))
    for artist_id, artist_name in sorted_artists.items():
        print(f"{artist_name}  ({artist_id})")

def add_artist(artist_name):
    config = load_config()
    artist_id = get_artist_id(artist_name)
    if artist_id:
        artist_name_from_api = get_artist_name(artist_id)  # Fetch
artist name from Deezer API
        config['artist_ids'][artist_id] = artist_name_from_api  # Add
artist name to config
        save_config(config)
        print(f"Artist '{artist_name_from_api}' added successfully.")
    else:
        print("Artist not found.")

def delete_artist(search_term):
    config = load_config()
    subscribed_artists = config.get('artist_ids', {})

    choices = process.extract(search_term, subscribed_artists.values(),
limit=5)
    print("Fuzzy search results:")
    for index, (artist_name, score) in enumerate(choices):
        print(f"{index + 1}. {artist_name} ({score})")
    choice_index = int(input("Enter the number of the artist to delete:
")) - 1
    if 0 <= choice_index < len(choices):
        artist_name = choices[choice_index][0]
        artist_id = [key for key, value in subscribed_artists.items()
if value == artist_name][0]
        del config[artist_id]
        save_config(config)
        print(f"Artist '{artist_name}' deleted successfully.")
    else:
        print("Invalid choice.")
```

```
def main():
    parser = argparse.ArgumentParser(description="Deezer Album
Tracker")
    parser.add_argument("--list", action="store_true", help="List all
monitored artists")
    parser.add_argument("--days", metavar="DAYS", help="Amount of days
to list")
    parser.add_argument("--add", metavar="ARTIST_NAME", help="Add a new
artist")
    parser.add_argument("--delete", metavar="SEARCH_TERM", help="Delete
an artist by fuzzy search")
    parser.add_argument("--email", action="store_true", help="Email the
output")

    args = parser.parse_args()

    if args.list:
        list_artists()
    elif args.add:
        add_artist(args.add)
    elif args.delete:
        delete_artist(args.delete)
    else:
        config = load_config()
        artist_ids = config.get('artist_ids', [])
        if args.days:
            lookupdays=int(args.days)
        else:
            lookupdays=config.get('global', {})['days']
        albums = get_albums(artist_ids,lookupdays)

        print(f"Albums released in the past {lookupdays} days:")
        for album in albums:
            print("Release Date:", album['release_date'])
            print("Artist:", album['artist'])
            print("Album Name:", album['album_name'])
            print()

        if args.email:
            email_subject = config.get('email_subject', 'Deezer Album
Tracker')
            email_recipients = config.get('email_recipients', [])
            email_body = "\n".join([f"Release Date:
{album['release_date']}\nArtist: {album['artist']}\nAlbum Name:
{album['album_name']}\n" for album in albums])
            send_email(email_subject, email_body, email_recipients)

if __name__ == "__main__":
    main()
```