МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Інститут комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та мереж

Лабораторна робота №7
з дисципліни: «Спеціалізовані мови програмування»
на тему: «Робота з API та веб-сервісами»

Виконала:
студентка групи ІТ-32
Моляща Ю.А.
Прийняв:
Щербак С.С.

Львів - 2023

Лабораторна робота №7

«Робота з API та веб-сервісами»

Мета роботи: створення консольного об’єктно - орієнтованого додатка з використанням API

Завдання на лабораторну роботу.

Завдання 1: Вибір провайдера API

Виберіть надійний API, який надає через HTTP необхідні дані для віддаленого зберігання, вивантаження або реалізуйте свій. Для прикладу це може бути jsonplaceholder.org

Завдання 2: Інтеграція API

Виберіть бібліотеку для роботи з API та обробки HTTP запитів (для прикладу це може бути бібліотека Requests). Інтегруйте обраний API в ваш консольний додаток на Python. Ознайомтеся з документацією API та налаштуйте необхідний API-ключ чи облікові дані.

Завдання 3: Введення користувача

Розробіть користувальницький інтерфейс, який дозволяє користувачам візуалізувати всі доступні дані в табличному вигляді та у вигляді списку. Реалізуйте механізм для збору та перевірки введеного даних користувачем.

Завдання 4: Розбір введення користувача

Створіть розбірник для видобування та інтерпретації виразів користувача на основі регулярних виразів, наприклад, для візуалізації дат, телефонів, тощо. Переконайтеся, що розбірник обробляє різні формати введення та надає зворотний зв'язок про помилки.

Завдання 5: Відображення результатів

Реалізуйте логіку для візуалізації даних через API в консолі. Обробляйте відповіді API для отримання даних у вигляді таблиць, списків. Заголовки таблиць, списків мають виділяться кольором та шрифтом, які задається користувачем.

Завдання 6: Збереження даних

Реалізуйте можливості збереження даних у чіткому та читабельному форматі JSON, CSV та TXT.

Завдання 7: Обробка помилок

Розробіть надійний механізм обробки помилок для керування помилками API, некоректним введенням користувача та іншими можливими проблемами. Надавайте інформативні повідомлення про помилки.

Завдання 8: Ведення історії обчислень

Включіть функцію, яка реєструє запити користувача, включаючи введені запити та відповідні результати. Дозвольте користувачам переглядати та рецензувати історію своїх запитів.

Завдання 9: Юніт-тести

Напишіть юніт-тести для перевірки функціональності вашого додатку. Тестуйте різні операції, граничні випадки та сценарії помилок.

Хід роботи.

Код програми:

data_visualization.py

```
import os
import sys


current_dir = os.path.dirname(os.path.abspath(__file__))
parent_dir = os.path.abspath(os.path.join(current_dir, '..'))
sys.path.append(parent_dir)


from prettytable import PrettyTable
from spotify_api import get_artists_top_tracks, get_artists_albums, get_artists_related_artists, get_artist_followers, get_artist_genres
from spotify_api import get_track_artist, get_track_album, get_track_duration, get_track_popularity, get_track_release_date, get_track_genres, get_track_explicit
from utility.data_utils import save_data


FOLDER_PATH_OUTPUT = 'source/lab7/data/output/'


def display_data(data, field_names, entity_name):
    if not data:
        print(f"No {entity_name} data to display")
        return None

    table = PrettyTable()
    table.field_names = field_names
    for idx, item in enumerate(data, start=1):
        table.add_row([idx] + [item['name']])
```

```
    print(f"{entity_name.capitalize()} data:")
    print(table)

def display_artists_top_tracks(token, artist_id, artist_name):
    tracks = get_artists_top_tracks(token, artist_id)
    field_names = ["#", "Track name"]
    display_data(tracks, field_names, "track")
    save_data(tracks, FOLDER_PATH_OUTPUT, f'top_tracks_{artist_name}', 'json')

def display_artists_albums(token, artist_id, artist_name):
    albums = get_artists_albums(token, artist_id)
    field_names = ["#", "Album name"]
    display_data(albums, field_names, "album")
    save_data(albums, FOLDER_PATH_OUTPUT, f'albums_{artist_name}', 'json')

def display_artists_related_artists(token, artist_id, artist_name):
    related_artists = get_artists_related_artists(token, artist_id)
    field_names = ["#", "Artist name"]
    display_data(related_artists, field_names, "related artist")
                            save_data(related_artists,       FOLDER_PATH_OUTPUT,
f'related_artists_{artist_name}', 'json')

def display_artists_followers(token, artist_id, artist_name):
    followers = get_artist_followers(token, artist_id)
    print(f"Followers for {artist_name}: {followers}")
          save_data(followers,  FOLDER_PATH_OUTPUT,  f'followers_{artist_name}',
'json')

def display_artists_genres(token, artist_id, artist_name):
    genres = get_artist_genres(token, artist_id)
    print(f"Genres for {artist_name}: {', '.join(genres)}")
    save_data(genres, FOLDER_PATH_OUTPUT, f'genres_{artist_name}', 'json')

def display_track_artist(token, track_id, track_name):
    artist = get_track_artist(token, track_id)
```

```
    print(f"Artist for {track_name}: {artist['name']}")
    save_data(artist, FOLDER_PATH_OUTPUT, f'artist_{track_name}', 'json')

def display_track_album(token, track_id, track_name):
    album = get_track_album(token, track_id)
    print(f"Album for {track_name}: {album['name']}")
    save_data(album, FOLDER_PATH_OUTPUT, f'album_{track_name}', 'json')

def display_track_duration(token, track_id, track_name):
    duration = get_track_duration(token, track_id)
    print(f"Duration for {track_name}: {duration}")
    save_data(duration, FOLDER_PATH_OUTPUT, f'duration_{track_name}', 'json')

def display_track_popularity(token, track_id, track_name):
    popularity = get_track_popularity(token, track_id)
    print(f"Popularity for {track_name}: {popularity}")
        save_data(popularity,  FOLDER_PATH_OUTPUT,  f'popularity_{track_name}',
'json')

def display_track_release_date(token, track_id, track_name):
    release_date = get_track_release_date(token, track_id)
    print(f"Release date for {track_name}: {release_date}")
      save_data(release_date,  FOLDER_PATH_OUTPUT,  f'release_date_{track_name}',
'json')

def display_track_genres(token, track_id, track_name):
    genres = get_track_genres(token, track_id)
    print(f"Genres for {track_name}: {', '.join(genres)}")
    save_data(genres, FOLDER_PATH_OUTPUT, f'genres_{track_name}', 'json')

def display_track_explicit(token, track_id, track_name):
    explicit = get_track_explicit(token, track_id)
    print(f"Explicit for {track_name}: {explicit}")
    save_data(explicit, FOLDER_PATH_OUTPUT, f'explicit_{track_name}', 'json')
```

main.py

```
import sys
import os

# Include the parent directory in the system's import path
current_dir = os.path.dirname(os.path.abspath(__file__))
parent_dir = os.path.abspath(os.path.join(current_dir, '..'))
sys.path.append(parent_dir)

from lab7.spotify_api import get_token
from lab7.user_input_handling import search_for_artist_menu, search_for_track_menu
from utility.data_utils import view_user_history

FOLDER_PATH_USER_LOGS = 'source/lab7/data/user_logs.log'


def main():
    token = get_token()
    while True:
        print("\nMAIN MENU")
        print("Options:")
        print("1. Search for an artist")
        print("2. Search for a track")
        print("3. Display history")
        print("0. Exit")

        user_input = input("Enter option: ")

        if user_input == '1':
            search_for_artist_menu(token)
        elif user_input == '2':
            search_for_track_menu(token)
        elif user_input == '3':
```

```
            print("User history:")
            view_user_history(FOLDER_PATH_USER_LOGS)
        elif user_input == '0':
            print("Exiting...")
            break
        else:
            print("Invalid option")

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

menus.py

```
from data_visualization import display_artists_top_tracks, display_artists_albums, display_artists_related_artists, display_artists_followers, display_artists_genres
from data_visualization import display_track_artist, display_track_album, display_track_duration, display_track_popularity, display_track_release_date, display_track_genres, display_track_explicit


def search_for_artist_options_menu(token, artist_id, artist_name):
    while True:
        print("\nARTIST SEARCH MENU")
        print("Options:")
        print("1. Get Top Tracks")
        print("2. Get Albums")
        print("3. Get Related Artists")
        print("4. Get Followers")
        print("5. Get Genres")
        print("0. Back")

        user_input = input("Enter option: ")

        if user_input == '1':
            display_artists_top_tracks(token, artist_id, artist_name)
        elif user_input == '2':
            display_artists_albums(token, artist_id, artist_name)
```

```
        elif user_input == '3':
            display_artists_related_artists(token, artist_id, artist_name)
        elif user_input == '4':
            display_artists_followers(token, artist_id, artist_name)
        elif user_input == '5':
            display_artists_genres(token, artist_id, artist_name)
        elif user_input == '0':
            break
        else:
            print("Invalid option")

def search_for_track_options_menu(token, track_id, track_name):
    while True:
        print("\nTRACK SEARCH MENU")
        print("Options:")
        print("1. Get Artist")
        print("2. Get Album")
        print("3. Get Duration")
        print("4. Get Popularity")
        print("5. Get Release Date")
        print("6. Get Genres")
        print("7. Get Explicit")
        print("0. Back")

        user_input = input("Enter option: ")

        if user_input == '1':
            display_track_artist(token, track_id, track_name)
        elif user_input == '2':
            display_track_album(token, track_id, track_name)
        elif user_input == '3':
            display_track_duration(token, track_id, track_name)
        elif user_input == '4':
            display_track_popularity(token, track_id, track_name)
```

```
        elif user_input == '5':
            display_track_release_date(token, track_id, track_name)
        elif user_input == '6':
            display_track_genres(token, track_id, track_name)
        elif user_input == '7':
            display_track_explicit(token, track_id, track_name)
        elif user_input == '0':
            break
        else:
            print("Invalid option")
```

spotify_api.py

```
"""This module contains functions for making requests to the Spotify API.

How authentication works in Spotify API:

Two types of authentication:
    1. Client Credentials Flow:
        - Used for non-user related endpoints
    2. Authorization Code Flow
        - Used for user-related endpoints

We are using Client Credentials Flow. It works like this:
    1. Request access token from Spotify API (send client_id,
       client_secret, grant_type=client_credentials)
    2. Server returns access token.
    3. Use access token to make requests to Spotify Web API.
"""

import os
import json
import base64
from requests import get, post

# Get client id and secret from environment variables
```

```
client_id=os.environ.get("SPOTIFY_CLIENT_ID")
client_secret=os.environ.get("SPOTIFY_CLIENT_SECRET")

def get_token():
    """Request access token from Spotify API.
    Returns:
        token (str): Access token.
    Raises:
        Exception: raise an HTTPError if the HTTP request returned an
        unsuccessful status code.
    """
    auth_string = client_id + ':' + client_secret
    auth_bytes = auth_string.encode('utf-8')
    auth_base64 = str(base64.b64encode(auth_bytes), 'utf-8')

    url = 'https://accounts.spotify.com/api/token'
    headers = {
        'Authorization': 'Basic ' + auth_base64,
        'Content-Type': 'application/x-www-form-urlencoded'
        }
    data = {'grant_type': 'client_credentials'}

    try:
        result = post(url, headers=headers, data=data)
        result.raise_for_status()
        json_result = json.loads(result.content)
        token = json_result['access_token']
        return token
    except Exception as e:
        print('Error getting token: ' + str(e))
        return None

def get_auth_token(token):
    """Construct authorization header to be used in requests to Spotify API."""
```

```
    return {"Authorization": "Bearer " + token}

def search_for_artist(token, artist_name):
    """Search for artist by name.

    Args:
        token (str): Access token.
        artist_name (str): Artist name.

    Returns:
        json_result (dict): JSON object with artist data.

    Raises:
        Exception: raise an HTTPError if the HTTP request returned an
        unsuccessful status code.
    """
    url = 'https://api.spotify.com/v1/search'
    headers = get_auth_token(token)
    query = f'?q={artist_name}&type=artist&limit=1'

    query_url = url + query

    try:
        result = get(query_url, headers=headers)
        result.raise_for_status()  # Check for HTTP errors
        json_result = json.loads(result.content)['artists']['items']

        if not json_result:
            print(f'No results found for artist: {artist_name}')
            return None

        return json_result[0]

    except Exception as e:
```

```
        print(f"Error searching for artist: {e}")
        return None

def search_for_track(token, track_name):
    """Search for track by name.

    Args:
        token (str): Access token.
        track_name (str): Track name.

    Returns:
        json_result (dict): JSON object with track data.

    Raises:
        Exception: raise an HTTPError if the HTTP request returned an
        unsuccessful status code.
    """
    url = 'https://api.spotify.com/v1/search'
    headers = get_auth_token(token)
    query = f'?q={track_name}&type=track&limit=1'

    query_url = url + query

    try:
        result = get(query_url, headers=headers)
        result.raise_for_status()  # Check for HTTP errors
        json_result = json.loads(result.content)['tracks']['items']

        if not json_result:
            print(f'No results found for track: {track_name}')
            return None

        return json_result[0]
```

```
    except Exception as e:
        print(f"Error searching for track: {e}")
        return None

# Get functions for artist data

def get_artists_top_tracks(token, artist_id):
                                                                    url                =
f'https://api.spotify.com/v1/artists/{artist_id}/top-tracks?country=US'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        json_result = json.loads(result.content)['tracks']
        return json_result
    except Exception as e:
        print(f"Error getting top tracks: {e}")
        return None


def get_artists_albums(token, artist_id):
    url = f'https://api.spotify.com/v1/artists/{artist_id}/albums?country=US'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        json_result = json.loads(result.content)['items']
        return json_result
    except Exception as e:
        print(f"Error getting albums: {e}")
        return None
```

```
def get_artists_related_artists(token, artist_id):
    url = f'https://api.spotify.com/v1/artists/{artist_id}/related-artists'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        json_result = json.loads(result.content)['artists']
        return json_result
    except Exception as e:
        print(f"Error getting related artists: {e}")
        return None

def get_artist_followers(token, artist_id):
    url = f'https://api.spotify.com/v1/artists/{artist_id}'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        json_result = json.loads(result.content)['followers']['total']
        return json_result
    except Exception as e:
        print(f"Error getting followers: {e}")
        return None

def get_artist_genres(token, artist_id):
    url = f'https://api.spotify.com/v1/artists/{artist_id}'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        json_result = json.loads(result.content)['genres']
```

```
        return json_result
    except Exception as e:
        print(f"Error getting genres: {e}")
        return None

# Get functions for track data

def get_track_artist(token, track_id):
    url = f'https://api.spotify.com/v1/tracks/{track_id}'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        json_result = json.loads(result.content)['artists'][0]
        return json_result
    except Exception as e:
        print(f"Error getting artist: {e}")
        return None

def get_track_album(token, track_id):
    url = f'https://api.spotify.com/v1/tracks/{track_id}'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        json_result = json.loads(result.content)['album']
        return json_result
    except Exception as e:
        print(f"Error getting album: {e}")
        return None

def get_track_duration(token, track_id):
```

```
    url = f'https://api.spotify.com/v1/tracks/{track_id}'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        json_result = json.loads(result.content)['duration_ms']
        return json_result
    except Exception as e:
        print(f"Error getting duration: {e}")
        return None

def get_track_popularity(token, track_id):
    url = f'https://api.spotify.com/v1/tracks/{track_id}'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        json_result = json.loads(result.content)['popularity']
        return json_result
    except Exception as e:
        print(f"Error getting popularity: {e}")
        return None

def get_track_release_date(token, track_id):
    url = f'https://api.spotify.com/v1/tracks/{track_id}'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        json_result = json.loads(result.content)['album']['release_date']
        return json_result
```

```
    except Exception as e:
        print(f"Error getting release date: {e}")
        return None

def get_track_genres(token, track_id):
    url = f'https://api.spotify.com/v1/tracks/{track_id}'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        artist_id = json.loads(result.content)['artists'][0]['id']
        artist_genres = get_artist_genres(token, artist_id)
        return artist_genres
    except Exception as e:
        print(f"Error getting genres: {e}")
        return None

def get_track_explicit(token, track_id):
    url = f'https://api.spotify.com/v1/tracks/{track_id}'
    headers = get_auth_token(token)

    try:
        result = get(url, headers=headers)
        result.raise_for_status()
        json_result = json.loads(result.content)['explicit']
        return json_result
    except Exception as e:
        print(f"Error getting explicit: {e}")
        return None
```

test_.py

```
import unittest
from spotify_api import get_token, get_auth_token
```

```
class TestLab7(unittest.TestCase):
    def test_get_token(self):
        token = get_token()
        self.assertIsNotNone(token)
        self.assertIsInstance(token, str)
        self.assertGreater(len(token), 0)

    def test_get_auth_token(self):
        token = get_token()
        headers = get_auth_token(token)
        self.assertIsNotNone(headers)
        self.assertIsInstance(headers, dict)
        self.assertIn('Authorization', headers)
        self.assertIn('Bearer', headers['Authorization'])
        self.assertIn(token, headers['Authorization'])

if __name__ == '__main__':
    unittest.main()
```

user_input_handling.py

```
from spotify_api import search_for_artist, search_for_track
from menus import search_for_artist_options_menu, search_for_track_options_menu
from utility.data_utils import log_user_history

FOLDER_PATH_USER_LOGS = 'source/lab7/data/user_logs.log'

def get_artist_name_from_user():
    artist_name = input("Enter artist name: ")
    if not artist_name.strip():
        raise ValueError("Artist name cannot be empty")
    return artist_name

def get_track_name_from_user():
    track_name = input("Enter track name: ")
    if not track_name.strip():
```

```
        raise ValueError("Track name cannot be empty")
    return track_name

def search_for_artist_menu(token):
    try:
        artist_name = get_artist_name_from_user()
        result = search_for_artist(token, artist_name)
        if result:
            artist_id = result['id']
            log_message = f"Artist found: {result['name']}"
            print(log_message)
            search_for_artist_options_menu(token, artist_id, artist_name)
        else:
            log_message = f"No results found for artist: {artist_name}"
            print(log_message)
              log_user_history("Search  for  Artist",  artist_name,  log_message,
FOLDER_PATH_USER_LOGS)
    except ValueError as e:
        print(f"An error occurred: {e}")

def search_for_track_menu(token):
    try:
        track_name = get_track_name_from_user()
        result = search_for_track(token, track_name)
        if result:
            track_id = result['id']
            log_message = f"Track found: {result['name']}"
            print(log_message)
            search_for_track_options_menu(token, track_id, track_name)
        else:
            log_message = f"No results found for track: {track_name}"
            print(log_message)
               log_user_history("Search  for  Track",  track_name,  log_message,
FOLDER_PATH_USER_LOGS)
    except ValueError as e:
```

```
        print(f"An error occurred: {e}")
```

Висновок: під час виконання лабораторної роботи було створено консольний об’єктно - орієнтований додаток з використанням API