МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж



Лабораторна робота №7

з дисципліни

Спеціалізовані мови програмування

на тему

Робота з API та веб-сервісами

Виконав:

ст. гр. ІТ-21сп

Олександр КОЗАК

Прийняв

доцент каф. ІСМ:

Сергій ЩЕРБАК

|  |  |
| --- | --- |
| **Балів** | **Дата** |
|  |  |

Львів-2023

**Мета**: Створення консольного об’єктно - орієнтованого додатка з використанням API

**Хід роботи:**

**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.

**album.py**

"""

Spotify Album Module

This module provides a class for representing albums and methods to interact with the Spotify API

to retrieve album information. It includes functionality to initialize an Album object, search for

an album by name, and obtain details such as the album's release date, artist information, and

Spotify link.

"""

from shared.settings import get\_lab\_settings

from classes.lab7.api\_classes.api\_error\_handling.api\_error\_handling import APIError, APIRequest

from classes.lab7.auth.auth import get\_auth\_header, get\_token

settings = get\_lab\_settings("lab7")

BASE\_URL = settings["urls"]["base\_url"]

class Album:

"""

Represents an album.

Attributes:

id (str): The ID of the album.

album\_name (str): The name of the album.

artist (dict): The artist of the album, containing the ID, name, and Spotify link.

release\_date (str): The release date of the album.

spotify\_link (str): The Spotify link of the album.

"""

def get\_album\_json\_from\_api(self, album\_name):

"""

Retrieves the JSON data of an album from the API based on the album name.

Args:

album\_name (str): The name of the album to search for.

Returns:

dict: The JSON data of the album, or None if no album with the given name exists.

Raises:

APIError: If there is an error while making the API request.

"""

api\_request = APIRequest(BASE\_URL)

try:

token = get\_token()

headers = get\_auth\_header(token)

album\_data = api\_request.make\_request("search", params={"q": album\_name, "type": "album", "limit": 1}, headers=headers)

if not album\_data:

print("No album with this name exists...")

return None

return album\_data["albums"]["items"][0]

except APIError as api\_error:

print(f"API Error: {api\_error.message}")

def init\_album(self, name):

"""

Initializes the album object with the given name.

Args:

name (str): The name of the album.

"""

album\_json = self.get\_album\_json\_from\_api(name)

if album\_json is None:

print("The object was not created")

return

self.set\_values(album\_json)

def set\_values(self, album\_json):

"""

Sets the values of the album object based on the album JSON data.

Args:

album\_json (dict): The album JSON data.

"""

self.id = album\_json["id"]

self.album\_name = album\_json["name"]

self.release\_date = album\_json["release\_date"]

artist\_id = album\_json["artists"][0]["id"]

artist\_name = album\_json["artists"][0]["name"]

artist\_link = album\_json["artists"][0]["external\_urls"]["spotify"]

self.artist = {"id": artist\_id, "name": artist\_name, "spotify\_link": artist\_link}

self.spotify\_link = album\_json["external\_urls"]["spotify"]

def get\_album\_formatted\_json(self):

"""

Returns a formatted dictionary representation of the album.

Returns:

dict: The formatted dictionary representation of the album.

"""

return {

'id': self.id,

'album\_name': self.album\_name,

'album\_artist': self.artist,

'release\_date': self.release\_date,

'spotify\_link': self.spotify\_link

}

**artist.py**

"""

Spotify API Module

This module contains classes and functions for interacting with the Spotify API.

It includes classes for working with artists, albums, tracks, etc.

"""

from shared.settings import get\_lab\_settings

from classes.lab7.api\_classes.api\_error\_handling.api\_error\_handling import APIError, APIRequest

from classes.lab7.auth.auth import get\_auth\_header, get\_token

settings = get\_lab\_settings("lab7")

BASE\_URL = settings["urls"]["base\_url"]

class Artist:

"""

Represents an artist and provides methods for retrieving artist information from an API.

"""

def \_\_init\_\_(self):

"""

Initializes an instance of the Artist class.

"""

self.id = None

self.artist\_name = None

self.spotify\_link = None

def \_\_str\_\_(self):

"""

Returns a string representation of the Artist object.

"""

return str(self.get\_artist\_formatted\_json())

def get\_artist\_json\_from\_api(self, artist\_name):

"""

Retrieves the JSON data for an artist from the API.

Parameters:

- artist\_name (str): The name of the artist.

Returns:

- dict: The JSON data for the artist, or None if the artist does not exist.

"""

api\_request = APIRequest(BASE\_URL)

try:

token = get\_token()

headers = get\_auth\_header(token)

artist\_data = api\_request.make\_request("search", params={"q": artist\_name, "type": "artist", "limit": 1}, headers=headers)

return artist\_data["artists"]["items"][0]

except APIError as api\_error:

print(f"API Error: {api\_error.message}")

return None

def init\_artist(self, name):

"""

Initializes the Artist object with the data for the specified artist.

Parameters:

- name (str): The name of the artist.

"""

artist\_json = self.get\_artist\_json\_from\_api(name)

if artist\_json is None:

print("The object was not created")

return

self.set\_values(artist\_json)

def set\_values(self, artist\_json):

"""

Sets the values of the Artist object using the provided artist JSON data.

Parameters:

- artist\_json (dict): The JSON data for the artist.

"""

self.id = artist\_json["id"]

self.artist\_name = artist\_json["name"]

self.spotify\_link = artist\_json["external\_urls"]["spotify"]

def get\_artist\_formatted\_json(self):

"""

Returns a formatted dictionary representation of the Artist object.

Returns:

- dict: The formatted dictionary representation of the Artist object.

"""

return {

'id': self.id,

'artist\_name': self.artist\_name,

'spotify\_link': self.spotify\_link

}

**data\_by\_artist.py**

"""

Spotify Data by Artist Module

This module provides a class for retrieving data related to a specific artist from the Spotify API.

It includes functionality to initialize an instance of the DataByArtist class, search for an artist,

retrieve albums and top tracks by the artist, and format the obtained data.

"""

import json

from requests import get, exceptions

from classes.lab7.auth.auth import get\_auth\_header, get\_token

from classes.lab7.api\_classes.artist import Artist

from classes.lab7.api\_classes.album import Album

from classes.lab7.api\_classes.track import Track

from shared.settings import get\_lab\_settings

settings = get\_lab\_settings("lab7")

BASE\_URL = settings["urls"]["base\_url"]

class DataByArtist(Artist):

"""

A class that represents data retrieval for a specific artist.

Attributes:

data (list): A list to store the retrieved data.

"""

def \_\_init\_\_(self):

"""

Initializes an instance of the DataByArtist class.

"""

self.data = []

super().\_\_init\_\_()

def init\_artist(self, name):

"""

Initializes the artist by name.

Args:

name (str): The name of the artist.

Returns:

bool: True if the artist is successfully initialized, False otherwise.

"""

return super().init\_artist(name)

def get\_albums\_by\_artist\_json\_from\_api(self):

"""

Retrieves the albums by the artist from the API.

Returns:

list: A list of albums in JSON format.

"""

try:

token = get\_token()

headers = get\_auth\_header(token)

url = BASE\_URL + f"artists/{self.id}/albums"

result = get(url, headers=headers)

json\_result = json.loads(result.content)["items"]

if not json\_result:

print("No top tracks found for this artist.")

return None

return json\_result

except exceptions.RequestException as exeption:

print(f"Error making API request: {exeption}")

return None

except json.JSONDecodeError as exeption:

print(f"Error decoding JSON response: {exeption}")

return None

except KeyError as exeption:

print(f"Unexpected response format: {exeption}")

return None

except Exception as exeption:

print(f"An unexpected error occurred: {exeption}")

return None

def get\_albums\_formatted\_json(self):

"""

Retrieves the albums by the artist in a formatted JSON format.

Returns:

list: A list of albums in a formatted JSON format.

"""

json\_albums = self.get\_albums\_by\_artist\_json\_from\_api()

data = []

for json\_album in json\_albums:

album = Album()

album.set\_values(json\_album)

data.append(album.get\_album\_formatted\_json())

return data

def get\_top\_tracks\_by\_artist\_json\_from\_api(self):

"""

Retrieves the top tracks by the artist from the API.

Returns:

list: A list of top tracks in JSON format.

"""

try:

token = get\_token()

headers = get\_auth\_header(token)

url = BASE\_URL + f"artists/{self.id}/top-tracks?country=UA"

result = get(url, headers=headers)

json\_result = json.loads(result.content)["tracks"]

if not json\_result:

print("No top tracks found for this artist.")

return None

return json\_result

except exceptions.RequestException as exeption:

print(f"Error making API request: {exeption}")

return None

except json.JSONDecodeError as exeption:

print(f"Error decoding JSON response: {exeption}")

return None

except KeyError as exeption:

print(f"Unexpected response format: {exeption}")

return None

except Exception as exeption:

print(f"An unexpected error occurred: {exeption}")

return None

def get\_tracks\_formatted\_json(self):

"""

Retrieves the top tracks by the artist in a formatted JSON format.

Returns:

list: A list of top tracks in a formatted JSON format.

"""

json\_tracks = self.get\_top\_tracks\_by\_artist\_json\_from\_api()

data = []

for json\_track in json\_tracks:

track = Track()

track.set\_values(json\_track)

data.append(track.get\_track\_formatted\_json())

return data

**recommendation.py**

"""

Spotify Recommendation Module

This module provides a class for retrieving track recommendations from the Spotify API.

It includes functionality to initialize a Recommendation object, retrieve and format track recommendations,

and formulate API requests based on seed artists, genres, and tracks.

"""

import json

from requests import get, exceptions

from classes.lab7.auth.auth import get\_auth\_header, get\_token

from classes.lab7.api\_classes.artist import Artist

from classes.lab7.api\_classes.track import Track

class Recommendation():

"""

Represents a recommendation object that retrieves track recommendations from the Spotify API.

Attributes:

limit (int): The maximum number of track recommendations to retrieve.

seed\_artists (list): A list of seed artist names.

seed\_genres (list): A list of seed genre names.

seed\_tracks (list): A list of seed track names.

"""

def \_\_init\_\_(self, limit=5, seed\_artists=None, seed\_genres=None, seed\_tracks=None):

"""

Initialize a Recommendation object.

Args:

limit (int): The maximum number of track recommendations to retrieve. Default is 5.

seed\_artists (list): A list of seed artist names. Default is None.

seed\_genres (list): A list of seed genre names. Default is None.

seed\_tracks (list): A list of seed track names. Default is None.

"""

self.limit = limit

self.seed\_artists = seed\_artists

self.seed\_genres = seed\_genres

self.seed\_tracks = seed\_tracks

def get\_track\_recommendation\_json\_from\_api(self):

"""

Retrieve track recommendation JSON from the Spotify API.

Returns:

list: A list of track recommendation JSON objects.

"""

try:

url = self.form\_url()

token = get\_token()

headers = get\_auth\_header(token)

result = get(url, headers=headers)

json\_result = json.loads(result.content)["tracks"]

if not json\_result:

print("No track recommendation...")

return None

return json\_result

except exceptions.RequestException as exception:

print(f"Error making API request: {exception}")

return None

except json.JSONDecodeError as exception:

print(f"Error decoding JSON response: {exception}")

return None

except KeyError as exception:

print(f"Unexpected response format: {exception}")

return None

except Exception as exception:

print(f"An unexpected error occurred: {exception}")

return None

def get\_track\_recommendation\_formatted\_json(self):

"""

Retrieve formatted track recommendation JSON from the Spotify API.

Returns:

list: A list of formatted track recommendation JSON objects.

"""

json\_track\_recommendation = self.get\_track\_recommendation\_json\_from\_api()

data = []

for json\_track in json\_track\_recommendation:

track = Track()

track.set\_values(json\_track)

data.append(track.get\_track\_formatted\_json())

return data

def get\_seed\_artists\_id(self):

"""

Get the IDs of the seed artists.

Returns:

list: A list of seed artist IDs.

"""

artists\_ids = []

for artist in self.seed\_artists:

obj = Artist()

obj.init\_artist(artist)

if obj.id is not None:

artists\_ids.append(obj.id)

return artists\_ids

def get\_seed\_tracks\_id(self):

"""

Get the IDs of the seed tracks.

Returns:

list: A list of seed track IDs.

"""

tracks\_ids = []

for track in self.seed\_tracks:

obj = Track()

obj.init\_track(track)

if obj.id is not None:

tracks\_ids.append(obj.id)

return tracks\_ids

def form\_url(self):

"""

Formulate the URL for the Spotify API request.

Returns:

str: The URL for the API request.

"""

url = f"https://api.spotify.com/v1/recommendations?limit={self.limit}"

if self.seed\_artists:

url += "&seed\_artists="

artists\_ids = self.get\_seed\_artists\_id()

url += self.add\_item\_to\_url(artists\_ids)

if self.seed\_genres:

url += "&seed\_genres="

url += self.add\_item\_to\_url(self.seed\_genres)

if self.seed\_tracks:

url += "&seed\_tracks="

tracks\_id = self.get\_seed\_tracks\_id()

url += self.add\_item\_to\_url(tracks\_id)

return url

def add\_item\_to\_url(self, items):

"""

Add items to the URL.

Args:

items (list): A list of items to add to the URL.

Returns:

str: The URL with the added items.

"""

url\_part = ""

if items is None:

return url\_part

for idx in range(len(items)):

if idx == len(items) - 1:

url\_part += str(items[idx])

else:

url\_part += str(items[idx]) + "%2C"

return url\_part

**lab\_menu.py**

"""

Spotify API Menu Module (Lab 7)

This module provides a menu function 'lab\_menu' for Lab 7, focusing on the Spotify API.

The menu allows users to perform various actions related to Spotify, such as searching for songs,

accessing artist information, viewing listening history, running tests, and exiting the program.

Classes:

- APIMenu: A class that implements the Spotify API menu functionality.

Usage:

Import this module and call the 'lab\_menu' function to start the Spotify API menu.

The program will display a menu with options for interacting with the Spotify API.

"""

import classes.lab7.tests.main as tests

from UI.menu import Menu

from UI.menu\_item import Item

from .api\_menu import APIMenu

def lab\_menu():

"""

This function displays a menu of Lab 7 for the Spotify API.

It allows the user to perform various actions such as searching for songs, accessing artist information,

viewing listening history, running tests, and exiting the program.

"""

api = APIMenu()

menu = Menu("\nSpotify API Menu (Lab 7)")

menu.set\_color("green")

menu.add\_item(Item("1", "Search Menu", api.search\_menu))

menu.add\_item(Item("2", "Artist Menu", api.player\_menu))

menu.add\_item(Item("3", "History Menu", api.history\_menu))

menu.add\_item(Item("0", "Exit", ))

menu.run()

**api\_menu.py**

"""

Spotify API Menu Module

This module provides a command-line interface for interacting with the Spotify API.

It includes a menu system with options to search for artists, tracks, and albums,

retrieve artist-related information, get recommendations, manage user history,

and save or print the obtained data in different formats.

Classes:

- APIMenu: A class representing the API Menu for Spotify API.

Attributes:

- HISTORY\_FILE\_PATH: The file path to store user history data.

Usage:

1. Create an instance of APIMenu.

2. Use the menu options to interact with the Spotify API and manage data.

"""

from UI.menu import Menu

from UI.menu\_item import Item

from shared.history import History

from shared.settings import get\_lab\_settings

from classes.lab7.api\_classes.artist import Artist

from classes.lab7.api\_classes.album import Album

from classes.lab7.api\_classes.track import Track

from classes.lab7.api\_classes.data\_by\_artist import DataByArtist

from classes.lab7.data\_manupulation.data\_saver import DataSaver

from classes.lab7.data\_manupulation.data\_visualization import DataVisualization

from classes.lab7.data\_manupulation.data\_from\_console import get\_name, get\_user\_input\_recommendations

from classes.lab7.api\_classes.recommendation import Recommendation

settings = get\_lab\_settings("lab7")

HISTORY\_FILE\_PATH = settings["history\_file\_path"]

class APIMenu:

"""

A class representing the API Menu for Spotify API.

Attributes:

- data: The data obtained from API calls.

- history: An instance of the History class to manage user history.

- data\_visualization: An instance of the DataVisualization class to visualize data.

Methods:

- history\_menu: Displays the history menu.

- search\_menu: Displays the search menu.

- search\_artist: Searches for an artist.

- search\_track: Searches for a track.

- search\_album: Searches for an album.

- player\_menu: Displays the artist menu.

- get\_recommendations: Gets track recommendations based on user input.

- get\_artist\_top\_tracks: Gets an artist's top tracks.

- get\_artist\_albums: Gets an artist's albums.

- choose\_menu: Displays the print or save menu.

- save\_menu: Displays the save menu.

- save\_json: Saves the data as JSON.

- save\_csv: Saves the data as CSV.

- save\_txt: Saves the data as TXT.

- print\_menu: Displays the print menu.

- print\_table: Prints the data as a table.

- print\_list: Prints the data as a list.

- settings\_menu: Displays the settings menu.

- print\_settings: Prints the current settings.

- change\_color: Changes the color settings.

"""

def \_\_init\_\_(self):

"""

Initializes an instance of APIMenu.

This method sets up the initial state of the object, including:

- Setting `data` attribute to None.

- Initializing the `history` attribute with an instance of the History class,

loading historical data from the specified file path.

- Initializing the `data\_visualization` attribute with an instance of the DataVisualization class.

Parameters:

None

Returns:

None

"""

self.data = None

self.history = History(HISTORY\_FILE\_PATH)

self.data\_visualization = DataVisualization()

def history\_menu(self):

"""

Displays a menu for interacting with the history feature.

This method creates a menu with options to view and clear history.

The user's choice triggers corresponding actions and updates the program state.

Parameters:

None

Returns:

None

"""

history\_menu = Menu("\nHistory Menu")

history\_menu.set\_color("green")

history\_menu.add\_item(Item("1", "View History", self.history.print\_history))

history\_menu.add\_item(Item("2", "Clear History", self.history.clear\_history))

history\_menu.add\_item(Item("0", "Back"))

history\_menu.run()

def search\_menu(self):

"""

Displays a menu for searching artists, tracks, and albums.

This method creates a menu with options to search for artists, tracks, and albums.

User choices trigger specific search functions and update the program state.

Parameters:

None

Returns:

None

"""

search\_menu = Menu("\nSearch Menu")

search\_menu.set\_color("green")

search\_menu.add\_item(Item("1", "Search Artist", self.search\_artist))

search\_menu.add\_item(Item("2", "Search Track", self.search\_track))

search\_menu.add\_item(Item("3", "Search Album", self.search\_album))

search\_menu.add\_item(Item("0", "Back"))

search\_menu.run()

def search\_artist(self):

"""

Searches for an artist and updates the program state with the artist's information.

This method prompts the user to enter an artist's name and retrieves information about the artist.

The artist's data is formatted and stored, and the action is logged in the history.

Parameters:

None

Returns:

None

"""

artist\_name = get\_name("artist")

artist = Artist()

artist.init\_artist(artist\_name)

self.data = artist.get\_artist\_formatted\_json()

self.history.add\_event(f"Get Artist {artist\_name}")

self.choose\_menu()

def search\_track(self):

"""

Searches for a track and updates the program state with the track's information.

This method prompts the user to enter a track's name and retrieves information about the track.

The track's data is formatted and stored, and the action is logged in the history.

Parameters:

None

Returns:

None

"""

track\_name = get\_name("track")

track = Track()

track.init\_track(track\_name)

self.data = track.get\_track\_formatted\_json()

self.history.add\_event(f"Get Track {track\_name}")

self.choose\_menu()

def search\_album(self):

"""

Searches for an album and updates the program state with the album's information.

This method prompts the user to enter an album's name and retrieves information about the album.

The album's data is formatted and stored, and the action is logged in the history.

Parameters:

None

Returns:

None

"""

album\_name = get\_name("album")

album = Album()

album.init\_album(album\_name)

self.data = album.get\_album\_formatted\_json()

self.history.add\_event(f"Get Album {album\_name}")

self.choose\_menu()

def player\_menu(self):

"""

Displays a menu for interacting with artist-related features.

This method creates a menu with options to get an artist's top tracks, albums, and recommendations.

User choices trigger specific actions related to artists and update the program state.

Parameters:

None

Returns:

None

"""

player\_menu = Menu("\nArtist Menu")

player\_menu.set\_color("green")

player\_menu.add\_item(Item("1", "Get Artist's Top Tracks", self.get\_artist\_top\_tracks))

player\_menu.add\_item(Item("2", "Get Artist's Albums", self.get\_artist\_albums))

player\_menu.add\_item(Item("3", "Get Recommendations", self.get\_recommendations))

player\_menu.add\_item(Item("0", "Back"))

player\_menu.run()

def get\_recommendations(self):

"""

Retrieves track recommendations based on user input and updates the program state.

This method prompts the user for input, generates track recommendations, and stores the recommendations.

The action is logged in the history.

Parameters:

None

Returns:

None

"""

user\_input = get\_user\_input\_recommendations()

user\_rec = Recommendation(seed\_artists=user\_input.get("artist"), seed\_genres=user\_input.get("genre"), seed\_tracks=user\_input.get("track"))

self.data = user\_rec.get\_track\_recommendation\_formatted\_json()

self.history.add\_event("Get recommendations")

self.choose\_menu()

def get\_artist\_top\_tracks(self):

"""

Retrieves an artist's top tracks and updates the program state.

This method prompts the user to enter an artist's name, retrieves the top tracks,

and stores the tracks' data. The action is logged in the history.

Parameters:

None

Returns:

None

"""

artist\_name = get\_name("artist")

artist = DataByArtist()

artist.init\_artist(artist\_name)

self.data = artist.get\_tracks\_formatted\_json()

self.history.add\_event(f"Get {artist\_name} Top Tracks")

self.choose\_menu()

def get\_artist\_albums(self):

"""

Retrieves an artist's albums and updates the program state.

This method prompts the user to enter an artist's name, retrieves the albums,

and stores the albums' data. The action is logged in the history.

Parameters:

None

Returns:

None

"""

artist\_name = get\_name("artist")

artist = DataByArtist()

artist.init\_artist(artist\_name)

self.data = artist.get\_albums\_formatted\_json()

self.history.add\_event(f"Get {artist\_name} Albums")

self.choose\_menu()

def choose\_menu(self):

"""

Displays a menu for choosing between printing, saving, or going back.

This method creates a menu with options to print, save, or go back to the previous menu.

User choices trigger specific actions and update the program state.

Parameters:

None

Returns:

None

"""

choose\_menu = Menu("\nPrint or Save")

choose\_menu.set\_color("green")

choose\_menu.add\_item(Item("1", "Print", self.print\_menu))

choose\_menu.add\_item(Item("2", "Save", self.save\_menu))

choose\_menu.add\_item(Item("0", "Back"))

choose\_menu.run()

def save\_menu(self):

"""

Displays a menu for saving data in different formats.

This method creates a menu with options to save data in JSON, CSV, or TXT formats.

User choices trigger specific saving actions.

Parameters:

None

Returns:

None

"""

save\_menu = Menu("\nSave Menu")

save\_menu.set\_color("green")

save\_menu.add\_item(Item("1", "Save JSON", self.save\_json))

save\_menu.add\_item(Item("2", "Save CSV", self.save\_csv))

save\_menu.add\_item(Item("3", "Save TXT", self.save\_txt))

save\_menu.add\_item(Item("0", "Back", self.choose\_menu))

save\_menu.run()

def save\_json(self):

"""

Saves the current data to a JSON file.

This method initializes a DataSaver object with the current data and

triggers the save\_to\_json method to save the data in JSON format.

Parameters:

None

Returns:

None

"""

data\_saver = DataSaver(self.data)

data\_saver.save\_to\_json()

def save\_csv(self):

"""

Saves the current data to a CSV file.

This method initializes a DataSaver object with the current data and

triggers the save\_to\_csv method to save the data in CSV format.

Parameters:

None

Returns:

None

"""

data\_saver = DataSaver(self.data)

data\_saver.save\_to\_csv()

def save\_txt(self):

"""

Saves the current data to a TXT file.

This method initializes a DataSaver object with the current data and

triggers the save\_to\_txt method to save the data in TXT format.

Parameters:

None

Returns:

None

"""

data\_saver = DataSaver(self.data)

data\_saver.save\_to\_txt()

def print\_menu(self):

"""

Displays a menu for printing data.

This method creates a menu with options to print data as a table, list, view settings, or change color.

User choices trigger specific print or settings actions.

Parameters:

None

Returns:

None

"""

print\_menu = Menu("\nPrint Menu")

print\_menu.set\_color("green")

print\_menu.add\_item(Item("1", "Print Table", self.print\_table))

print\_menu.add\_item(Item("2", "Print List", self.print\_list))

print\_menu.add\_item(Item("3", "Settings", self.settings\_menu))

print\_menu.add\_item(Item("0", "Back", self.choose\_menu))

print\_menu.run()

def print\_table(self):

"""

Prints the current data as a table.

This method sets the data for visualization, and then triggers the visualization

of the data in table format using the DataVisualization class.

Parameters:

None

Returns:

None

"""

self.data\_visualization.set\_data(self.data)

self.data\_visualization.visualize\_as\_table()

def print\_list(self):

"""

Prints the current data as a list.

This method sets the data for visualization, and then triggers the visualization

of the data in list format using the DataVisualization class.

Parameters:

None

Returns:

None

"""

self.data\_visualization.set\_data(self.data)

self.data\_visualization.visualize\_as\_list()

def settings\_menu(self):

"""

Displays a menu for managing visualization settings.

This method creates a menu with options to view settings, change color, or go back.

User choices trigger specific settings actions.

Parameters:

None

Returns:

None

"""

settings\_menu = Menu("\nSettings Menu")

settings\_menu.set\_color("green")

settings\_menu.add\_item(Item("1", "View Settings", self.print\_settings))

settings\_menu.add\_item(Item("2", "Change Color", self.change\_color))

settings\_menu.add\_item(Item("0", "Back", self.print\_menu))

settings\_menu.run()

def print\_settings(self):

"""

Displays the current application settings.

This method retrieves and displays the current settings for visualization,

allowing users to view and adjust the application's display settings.

Parameters:

None

Returns:

None

"""

self.data\_visualization.set\_data(self.data)

self.data\_visualization.view\_settings()

def change\_color(self):

"""

Allows the user to change the color settings of the application.

This method prompts the user to input new color preferences, which are then applied

to the visualization components of the application.

Parameters:

None

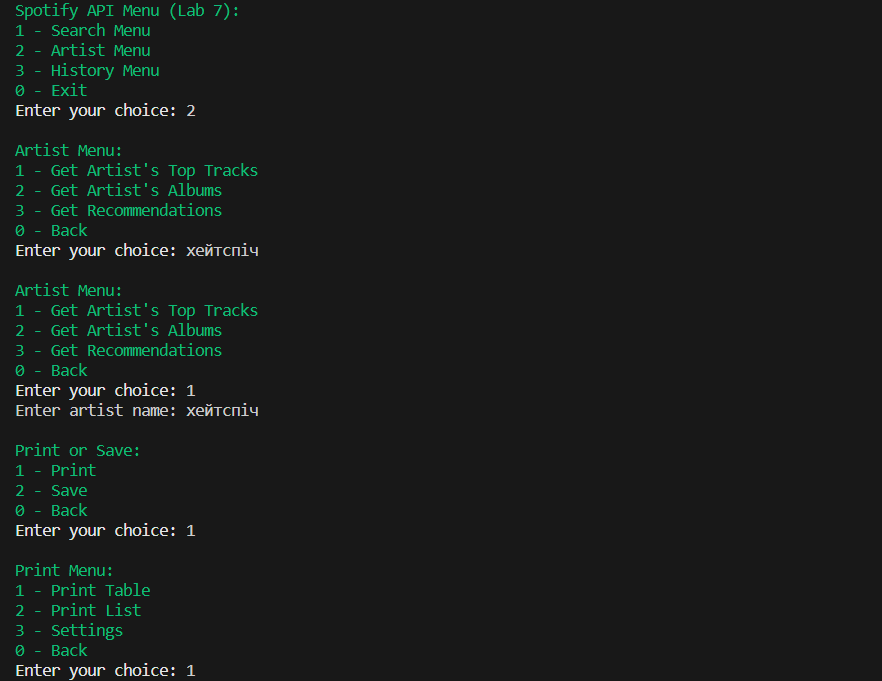
Returns:

None

"""

self.data\_visualization.set\_data(self.data)

self.data\_visualization.settings()

*Рис. 1 Результат виконання завдання*

*Зображення, що містить текст, знімок екрана

Автоматично згенерований опис*

*Рис. 2 Результат виконання завдання*

**Висновок:** Виконавши ці завдання, я створила проект, який надав цінний досвід роботи з API, дизайном користувацького інтерфейсу, валідації введення, обробки помилок та тестування.