МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

**Лабораторна робота №7**

**з дисципліни**

**СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ**

**на тему**

**«Робота з API та веб-сервісами»**

Виконав:

ст. гр. ІТ-21сп

Одноріг Д.І.

Прийняв:

Щербак С.С.

Львів-2023

**Мета роботи:** 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.

Код програми:

dogs.py:

import requests

from PIL import Image

from io import BytesIO

from tabulate import tabulate

from prettytable import PrettyTable

import signal

import sys

from colorama import Fore, init

import json

import csv

init(autoreset=True)

class DogAPI:

base\_url = "https://api.thedogapi.com/v1"

def \_\_init\_\_(self, api\_key):

self.headers = {"x-api-key": api\_key}

@classmethod

def get\_all\_breeds(cls):

response = requests.get(f"{cls.base\_url}/breeds/list/all")

data = response.json()

breeds = list(data.get("message", {}).keys()) if isinstance(data, dict) else []

return breeds

def get\_random\_image(self, breed\_id=None):

url = f"{self.base\_url}/images/search"

if breed\_id:

url += f"?breed\_id={breed\_id}"

response = requests.get(url, headers=self.headers)

return response.json()

class DisplayDogApi:

BASE\_URL = "https://dog.ceo/api"

@classmethod

def get\_all\_breeds(cls):

response = requests.get(f"{cls.BASE\_URL}/breeds/list/all")

data = response.json()

breeds = data.get("message", {})

return breeds.keys()

@classmethod

def get\_random\_image(cls, breed):

response = requests.get(f"{cls.BASE\_URL}/breed/{breed}/images/random")

data = response.json()

image\_url = data.get("message", "")

return image\_url

class ConsoleInterface:

def display\_table(self, data, color):

headers = [Fore.RESET + color + "DogBreed", Fore.RESET + color + "PictureUrl"]

rows = [(color + breed, DisplayDogApi.get\_random\_image(breed)) for breed in data]

table = tabulate(rows, headers, tablefmt="grid")

print(table)

def display\_list(self, data, color):

for breed in data:

print(color + f"{breed}: {DisplayDogApi.get\_random\_image(breed)}")

def display\_image(self, image\_url):

response = requests.get(image\_url)

img = Image.open(BytesIO(response.content))

img.show()

def remove\_color\_codes(self, text):

while '\033[' in text:

start = text.find('\033[')

end = text.find('m', start)

if end != -1:

text = text[:start] + text[end+1:]

else:

break

return text

def save\_to\_json(self, data, filename):

data\_to\_save = [{"DogBreed": breed, "PictureUrl": DisplayDogApi.get\_random\_image(breed)} for breed in data]

with open(filename, 'w') as file:

json.dump(data\_to\_save, file, indent=2)

print(f"Data saved to {filename} in JSON format.")

self.history.append({

'request': 'save\_to\_json',

'filename': filename

})

def save\_to\_csv(self, data, filename):

with open(filename, 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow([Fore.RESET + "DogBreed", Fore.RESET + "PictureUrl"])

for breed in data:

writer.writerow([self.remove\_color\_codes(breed), self.remove\_color\_codes(DisplayDogApi.get\_random\_image(breed))])

print(f"Data saved to {filename} in CSV format.")

self.history.append({

'request': 'save\_to\_csv',

'filename': filename

})

def save\_to\_txt(self, data, filename):

with open(filename, 'w') as file:

for breed in data:

file.write(f"{self.remove\_color\_codes(breed)}: {self.remove\_color\_codes(DisplayDogApi.get\_random\_image(breed))}\n")

print(f"Data saved to {filename} in TXT format.")

self.history.append({

'request': 'save\_to\_txt',

'filename': filename

})

def signal\_handler(sig, frame):

print("You pressed Ctrl+C!")

sys.exit(0)

def choose\_display\_format():

while True:

print("Choose a display format:")

print("1. Table")

print("2. List")

format\_choice = input("Enter the number of the display format: ")

if format\_choice in ("1", "2"):

return format\_choice

else:

print("Invalid display format. Please enter 1 or 2.")

def choose\_color():

while True:

print("Choose a color:")

print("1. Red")

print("2. Green")

print("3. Yellow")

color\_choice = input("Enter the number of the color: ")

if color\_choice == "1":

return Fore.RED

elif color\_choice == "2":

return Fore.GREEN

elif color\_choice == "3":

return Fore.YELLOW

else:

print("Invalid color choice. Please enter 1, 2, or 3.")

def start\_menu():

print("Hello! Choose the option:")

print("1. Continue")

print("2. Exit")

option = input("Enter the number of the option: ")

return option

def main():

api\_key = "live\_erHxdDaDlKKkhVAn0IpEIPT9trh9HzJx"

dog\_api = DogAPI(api\_key)

interface = ConsoleInterface()

while True:

user\_option = start\_menu()

if user\_option == "1":

breeds = DisplayDogApi.get\_all\_breeds()

color = choose\_color()

format\_choice = choose\_display\_format()

if format\_choice == "1":

interface.display\_table(breeds, color)

elif format\_choice == "2":

interface.display\_list(breeds, color)

save\_choice = input("Do you want to save the data? (y/n): ")

if save\_choice.lower() == 'y':

save\_format = input("Choose a save format:\n1. JSON\n2. CSV\n3. TXT\nEnter the number of the save format: ")

save\_filename = input("Enter the filename: ")

if save\_format == "1":

interface.save\_to\_json(breeds, save\_filename)

elif save\_format == "2":

interface.save\_to\_csv(breeds, save\_filename)

elif save\_format == "3":

interface.save\_to\_txt(breeds, save\_filename)

else:

print("Invalid save format. Please enter 1, 2, or 3.")

repeat\_choice = input("Do you want to perform another operation? (y/n): ")

if repeat\_choice.lower() != 'y':

break

elif user\_option == "2":

print("Exiting the program. Goodbye!")

break

else:

print("Invalid option. Please enter 1 or 2.")

if \_\_name\_\_ == "\_\_main\_\_":

signal.signal(signal.SIGINT, signal\_handler)

main()

Ця програма використовує The Dog API, який надає через HTTP необхідні дані для віддаленого зберігання, а саме інформацію про породи собак та url-зображення кожної із відомих порід.

Зображення, що містить текст, собака, знімок екрана, Веб-сайт

Автоматично згенерований опис

*Рис. 1. The Dog API.*

Перед тим як обрати формат виведення, користувач обирає колір для заповнення своєї таблиці або списку:

Зображення, що містить текст, Шрифт, знімок екрана

Автоматично згенерований опис

*Рис. 2. Вибір бажаного кольору таблиці чи списку.*

Як було вказано вище, дані відображаються у вигляді таблиці або списку, де перелічені десятки відомих порід собак та посилання на їх зображення, які виводяться завдяки The Dog API:

Зображення, що містить текст, знімок екрана

Автоматично згенерований опис

*Рис. 3. Таблиця із породами собак та url-адресами їхніх зображень.*

*Зображення, що містить текст, знімок екрана, Шрифт, дизайн

Автоматично згенерований опис*

*Рис. 4. Спсиок із породами собак та url-адресами їхніх зображень.*

На відміну від таблиці, у списку виведення даних у консоль відбувається поштучно у режимі реального часу, у той час як при виборі табличного формату користувачеві потрібно зачекати на генерацію результату.

Виведені дані користувач може зберегти у файл (json/csv/txt). Перед збереженням користувач обирає формат файлу та вводить його назву, після чого відповідний файл зберігається:

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

*Рис. 5. Збережений список із інформацією у текстовому форматі.*

Для додатку було зроблено unit-тести, щоб переконатись у правильності роботи написаного коду.

Код програми:

import unittest

from unittest.mock import patch, MagicMock

from io import StringIO

from Lab7.dogs import \*

class TestDisplayDogApi(unittest.TestCase):

@patch('requests.get')

def test\_get\_all\_breeds(self, mock\_requests\_get):

mock\_response = MagicMock()

mock\_response.json.return\_value = {"message": {"breed1": {}, "breed2": {}}}

mock\_requests\_get.return\_value = mock\_response

breeds = DisplayDogApi.get\_all\_breeds()

self.assertEqual(list(breeds), ["breed1", "breed2"])

class TestConsoleInterface(unittest.TestCase):

@patch('builtins.input', side\_effect=['3'])

def test\_invalid\_display\_format(self, mock\_input):

with self.assertRaises((ValueError, StopIteration)):

choose\_display\_format()

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

Ця програма містить два набори юніт-тестів для перевірки функцій у коді мого додатку:

* TestDisplayDogApi: Перевіряє коректність роботи функції get\_all\_breeds в класі DisplayDogApi. Використовує "моковану" версію requests.get, щоб вирішити виклики до API і повертати визначений набір даних.
* TestConsoleInterface: Перевіряє, як програма обробляє некоректний вибір формату виводу, тобто введення, що не дорівнює "1" або "2". Симулює введення користувача та перевіряє, чи виникає відповідне виключення при некоректному введенні.

На рис. 6 зображено результат виконання unit-тестувань.

*Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис*

*Рис. 6. Результат успішного виконання тестувань.*

**Висновок:** виконуючи дану лабораторну роботу, я створив проект, який надав мені цінний досвід роботи з API, дизайну користувацького інтерфейсу, валідації введення, обробки помилок та тестування.