МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №7

з курсу

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

Виконав студент

групи ІТ-21сп

**Романишин.Д.О.**

Прийняв

**Щербак С.С.**

Львів - 2023

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

**План 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: Відображення результатів**

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.

**ConsoleAPI.py**

class ConsoleAPI:

def run(self):

pass

**ConsoleInterface.py**

from sty import fg, ef

from Classes.lab\_7.RequestsAPI import RequestsAPI

from sty import rs

import requests

# site = 'https://api.openweathermap.org/data/2.5/weather'

# q = 'Варшава'

# appid = "94214c464205a73bcf054c4c2d071905"

# params={'q': q, 'appid': appid}

#

# qa = (requests.get(site, params=params).json())

#

# # response1 = RequestsAPI(site=site, params=params)

# # response1.data = qa

# response2 = RequestsAPI(site='https://jsonplaceholder.typicode.com/users')

#

#

dct = [

{'header': 'name', 'color': fg.red, 'font': ef.bold},

{'header': 'email', 'color': fg.blue, 'font': ef.underl},

{'header': 'phone', 'color': fg.green, 'font': ef.dim},

]

# response2.get\_table(dct)

# print(response2.visualize\_table(dct))

class ConsoleRequestsApi:

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

self.site = ''

self.params = {}

self.headers = []

self.history = []

def configuration(self):

self.site = input("Input site: ")

if input("Do you want to input params(1/0): ") == '1':

self.set\_params()

response = RequestsAPI(site=self.site, params=self.params)

if input("Do you want to custom style for headers(1/0): ") == '1':

self.set\_headers\_style(response)

# result = response.visualize\_table()

result = response.get\_table()

self.history.append(result)

if input("Do you want to custom headers output(1/0): ") == '1':

self.set\_custom\_headers(response)

result = response.visualize\_table(self.headers)

if input("Do you want to save your data: ") == '1':

self.save(response)

if input("Do you want to view your history: ") == '1':

for history in self.history:

print(history)

return result

def set\_headers\_style(self, response):

response.color = self.select\_color(response.COLORS)

response.font = self.select\_font(response.FONTS)

return response

def save(self, response):

save = input("Input format (txt, csv, json): ")

filename = input("Input name: ")

if save == 'txt':

response.save\_to\_txt(filename)

elif save == 'csv':

response.save\_to\_csv(filename)

elif save == 'json':

response.save\_to\_json(filename)

else:

ext = input("Incorrect format: Try again?(1/0)")

if ext == '1':

self.save(response)

def set\_custom\_headers(self, response: RequestsAPI):

stop = '1'

while stop == '1':

self.set\_header\_dict(response)

stop = input("Do you have any parameters?(1/0)")

def set\_header\_dict(self, response):

header = self.select\_header(response.get\_headers())

color = self.select\_color(response.COLORS)

font = self.select\_font(response.FONTS)

self.headers.append({'header': header, 'color': color, 'font': font})

@staticmethod

def select\_color(colors):

for key, value in colors.items():

print(f"{key}: {value + rs.all}")

value = input(f"Select color: ")

current\_value = colors.get(value, None)

if current\_value:

return current\_value

else:

return ''

@staticmethod

def select\_font(fonts):

for key, value in fonts.items():

print(f"{key}: {value + rs.all}")

value = input(f"Select font: ")

current\_value = fonts.get(value, None)

if current\_value:

return current\_value

else:

return ''

@staticmethod

def select\_header(headers):

header = input(f"Select header with {headers}: ")

if header in headers:

return header

else:

return ''

def set\_params(self):

stop = '1'

while stop == '1':

self.set\_dict()

stop = input("Do you have any parameters?(1/0)")

def set\_dict(self):

key = input("key: ")

value = input("value: ")

self.params[key] = value

def run(self):

stop = ''

while stop != 'f':

try:

result = self.configuration()

print(result)

except Exception as e:

print(e)

finally:

stop = input("Press 'f' if you want to exit: ")

# site = input("Input site: ")

# params = {}

#

#

# for i in range(2):

# key = input("key: ")

# value = input("value: ")

# params[key] = value

#

# response = RequestsAPI(site=site, params=params)

#

# print(response1.visualize\_table())

# print(response2.visualize\_table())

# # print(response.visualize\_table())

**RequestsAPI.py**

import csv

import json

import requests

from prettytable import PrettyTable

from sty import fg, ef, rs

class RequestsAPI:

site = ''

params = {}

color = fg.blue

font = ef.italic

reset = rs.all

COLORS = {

'BLACK': fg.black,

'RED': fg.red,

'GREEN': fg.green,

'YELLOW': fg.yellow,

'BLUE': fg.blue,

'MAGENTA': fg.magenta,

'CYAN': fg.cyan,

'WHITE': fg.white,

'RESET': reset,

}

FONTS = {

'italic': ef.italic,

'bold': ef.bold,

'dim': ef.dim,

'underl': ef.underl,

'blink': ef.blink,

'hidden': ef.hidden,

'inverse': ef.inverse,

'strike': ef.strike,

'RESET': reset,

}

def \_\_init\_\_(self, \*\*kwargs):

self.result = PrettyTable()

if 'site' in kwargs:

self.site = kwargs['site']

if 'params' in kwargs:

self.params = kwargs['params']

self.response = requests.get(self.site)

self.data = self.response.json()

def get\_headers(self, data=None):

if not data:

data = self.data

if isinstance(data, list) and len(data) > 0 and isinstance(data[0], dict):

return data[0].keys()

elif isinstance(data, dict):

return data.keys()

else:

raise ValueError("Непідтримуваний формат даних")

def get\_table(self):

table = PrettyTable()

field\_names = []

if isinstance(self.data, list) and len(self.data) > 0 and isinstance(self.data[0], dict):

headers = self.data[0].keys()

for row in self.data:

table.add\_row(row.values(), divider=True)

elif isinstance(self.data, dict):

headers = self.data.keys()

table.add\_row(self.data.values(), divider=True)

else:

raise ValueError("Непідтримуваний формат даних")

table.field\_names = headers

for field\_name in table.field\_names:

field\_names.append(self.color + self.font + field\_name + self.reset)

table.field\_names = field\_names

return table

def get\_custom\_headers(self, custom\_headers):

headers = []

for custom\_header in custom\_headers:

headers.append(custom\_header.get('header'))

return headers

def get\_custom\_table(self, custom\_headers):

table = PrettyTable()

new\_headers = self.get\_custom\_headers(custom\_headers)

new\_data = []

if isinstance(self.data, list) and len(self.data) > 0 and isinstance(self.data[0], dict):

for data in self.data:

dicts = {}

for key, value in data.items():

if key in new\_headers:

dicts[key] = value

new\_data.append(dicts)

for row in new\_data:

table.add\_row(row.values(), divider=True)

headers = new\_data[0].keys()

elif isinstance(self.data, dict):

headers = self.data.keys()

table.add\_row(self.data.values(), divider=True)

else:

raise ValueError("Непідтримуваний формат даних")

table.field\_names = headers

return table, new\_data

def set\_custom\_headers(self, custom\_headers: list[dict], data):

new\_headers = []

user\_headers = [item.get('header') for item in custom\_headers]

old\_headers = self.get\_headers(data)

for old\_header in old\_headers:

if old\_header in user\_headers:

new\_header = self.get\_custom\_header(old\_header, custom\_headers)

else:

new\_header = old\_header

new\_headers.append(new\_header)

return new\_headers

def get\_custom\_header(self, old\_header, custom\_headers):

new\_header = ''

for header in custom\_headers:

if header.get('header') == old\_header:

new\_header = self.set\_style(header.get('header'), header.get('color'), header.get('font'))

break

return new\_header

def set\_style(self, header, color, font):

return color + font + header + self.reset

def visualize\_table(self, custom\_headers=None):

table, data = self.get\_custom\_table(custom\_headers)

field\_names = []

if custom\_headers:

field\_names = self.set\_custom\_headers(custom\_headers, data)

# print(field\_names)

else:

for field\_name in table.field\_names:

field\_names.append(self.color + self.font + field\_name + self.reset)

table.field\_names = field\_names

return table

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

filename = self.\_add\_extension(filename, "json")

with open(filename, "w") as json\_file:

json.dump(self.data, json\_file, indent=2)

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

filename = self.\_add\_extension(filename, "csv")

csv\_columns = self.data[0].keys()

with open(filename, "w", newline="") as csv\_file:

writer = csv.DictWriter(csv\_file, fieldnames=csv\_columns)

writer.writeheader()

for row in self.data:

writer.writerow(row)

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

filename = self.\_add\_extension(filename, "txt")

with open(filename, "w") as txt\_file:

for row in self.data:

txt\_file.write(f"{row}\n")

@staticmethod

def \_add\_extension(filename, default\_extension):

if not filename.endswith(f".{default\_extension}"):

filename += f".{default\_extension}"

return filename

# site = 'https://jsonplaceholder.typicode.com/users'

# params = {}

#

# response = RequestsAPI(site=site, params=params)

#

# print(response.get\_table())

# print(response.visualize\_table())

**Main.py**

from Classes.lab\_7.ConsoleInterface import ConsoleRequestsApi

req = ConsoleRequestsApi()

req.run()

**Tests.py**

import unittest

from unittest.mock import patch

from io import StringIO

from prettytable import PrettyTable

from Classes.lab\_7.RequestsAPI import RequestsAPI

class TestRequestsAPI(unittest.TestCase):

def setUp(self):

self.site = 'https://jsonplaceholder.typicode.com/users'

self.params = {}

self.response = RequestsAPI(site=self.site, params=self.params)

def test\_get\_headers(self):

headers = self.response.get\_headers()

self.assertIsInstance(list(headers), list)

self.assertTrue(all(isinstance(header, str) for header in headers))

def test\_get\_table(self):

table = self.response.get\_table()

self.assertIsInstance(table, PrettyTable)

def test\_get\_custom\_headers(self):

custom\_headers = [{'header': 'name', 'color': 'RED', 'font': 'bold'}]

headers = self.response.get\_custom\_headers(custom\_headers)

self.assertEqual(headers, ['name'])

def test\_get\_custom\_table(self):

custom\_headers = [{'header': 'name', 'color': 'RED', 'font': 'bold'}]

table, data = self.response.get\_custom\_table(custom\_headers)

self.assertIsInstance(table, PrettyTable)

self.assertIsInstance(data, list)

def test\_set\_custom\_headers(self):

custom\_headers = [{'header': 'name', 'color': 'RED', 'font': 'bold'}]

new\_headers = self.response.set\_custom\_headers(custom\_headers, self.response.data)

expected\_headers = ['id', 'name', 'username', 'email', 'address', 'phone', 'website', 'company']

# Set style only for the specified custom header

expected\_styled\_headers = [self.response.set\_style(header, 'RED', 'bold') if header == 'name' else header for

header in expected\_headers]

self.assertEqual(new\_headers, expected\_styled\_headers)

def test\_save\_to\_json(self):

filename = 'test\_json\_output'

self.response.save\_to\_json(filename)

# Add assertions to check if the file is created and contains expected data

def test\_save\_to\_csv(self):

filename = 'test\_csv\_output'

self.response.save\_to\_csv(filename)

# Add assertions to check if the file is created and contains expected data

def test\_save\_to\_txt(self):

filename = 'test\_txt\_output'

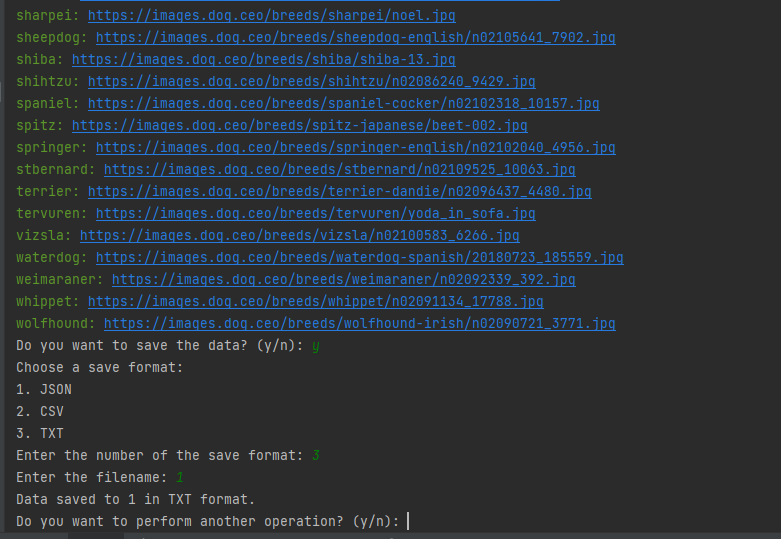
self.response.save\_to\_txt(filename)

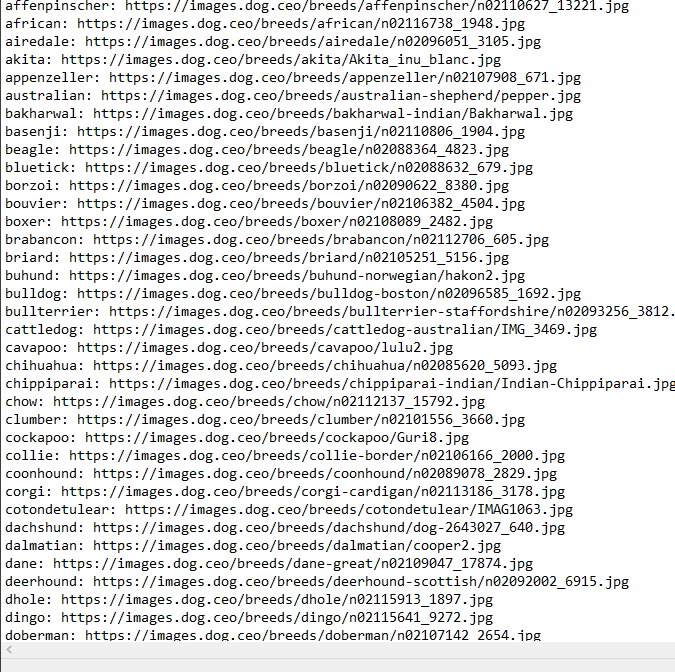
# Add assertions to check if the file is created and contains expected data

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

unittest.main()

**Результат виконання програми**





**Висновок:** під час виконання лабораторної роботи я навчився cтворювати консольного об’єктно - орієнтованого додатка з використанням API