МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ КОМП’ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ



Лабораторна робота №7

з дисципліни “Спеціалізовані мови програмування”

Виконав:

студент групи ІТ-31

Роман ХРАПЧУН

Прийняв:

доцент кафедри ІСМ

Сергій ЩЕРБАК

Львів-2023

**Тема роботи:** Робота з API та веб-сервісами

**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:**

**Main.py**import requests

import json

import pandas as pd

import logging

from prettytable import PrettyTable

import os

logging.basicConfig(filename='api\_client.log', level=logging.INFO, format='%(asctime)s - %(levelname)s: %(message)s')

class FileStorage:

@staticmethod

def store\_data(data, file\_format):

"""

Зберігає дані у вказаному форматі у папці Data на одному рівні з Core.

Parameters:

data (list): Дані для збереження.

file\_format (str): Формат збереження ('json', 'csv', 'txt').

"""

# Отримуємо шлях до файлу, який виконується

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

# Виходимо на один рівень вгору, щоб потім зайти в папку Data

base\_dir = os.path.dirname(current\_dir)

data\_dir = os.path.join(base\_dir, 'Data')

# Створюємо шлях до файлу у папці Data

filename = f'saved\_data.{file\_format}'

file\_path = os.path.join(data\_dir, filename)

# Зберігаємо дані в зазначеному форматі

if file\_format == 'json':

with open(file\_path, 'w') as file:

json.dump(data, file, indent=4)

elif file\_format == 'csv':

dataframe = pd.DataFrame(data)

dataframe.to\_csv(file\_path, index=False)

elif file\_format == 'txt':

with open(file\_path, 'w') as file:

file.write(str(data))

print(f"Дані збережено у файлі: {file\_path}")

class ApiClient:

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

"""

Ініціалізація об'єкта ApiClient.

Parameters:

api\_url (str): URL API-провайдера.

"""

self.api\_url = api\_url

def make\_request(self, endpoint, params=None):

"""

Виконує HTTP GET-запит до API.

Parameters:

endpoint (str): Ім'я ендпоінта API.

params (dict): Параметри запиту.

Returns:

dict or None: Результат запиту або None у випадку помилки.

"""

try:

headers = {'Accept': 'application/json'}

response = requests.get(f'{self.api\_url}/{endpoint}', headers=headers, params=params)

response.raise\_for\_status()

return response.json()

except requests.RequestException as err:

logging.error(f'Помилка виклику API: {err}')

return None

def add\_todo(self, todo\_data):

"""

Додає новий елемент справи до API.

Parameters:

todo\_data (dict): Дані нової справи.

Returns:

dict or None: Результат запиту або None у випадку помилки.

"""

try:

endpoint = 'todos'

headers = {'Content-Type': 'application/json'}

response = requests.post(f'{self.api\_url}/{endpoint}', headers=headers, json=todo\_data)

response.raise\_for\_status()

return response.json()

except requests.RequestException as err:

logging.error(f'Помилка виклику API: {err}')

return None

class UserInterface:

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

"""

Ініціалізація об'єкта UserInterface.

Parameters:

api\_client (ApiClient): Об'єкт для взаємодії з API.

"""

self.api\_client = api\_client

self.history = []

def display\_data(self, data):

"""

Виводить дані у форматі PrettyTable.

Parameters:

data (list): Список об'єктів для виведення.

"""

if data:

headers = data[0].keys()

rows = [d.values() for d in data]

table = PrettyTable()

table.field\_names = headers

table.add\_rows(rows)

print(table)

else:

print('Немає даних для відображення.')

def display\_menu(self):

"""

Виводить меню та отримує вибір користувача.

Returns:

str: Вибір користувача.

"""

print("Виберіть опцію:")

print("1. Переглянути список справ")

print("2. Зберегти дані")

print("3. Додати нову справу")

print("exit. Вийти з програми")

user\_input = input('Введіть свій вибір: ')

return user\_input

def process\_command(self, command):

"""

Обробляє команди користувача.

Parameters:

command (str): Команда користувача.

"""

try:

if command == '1':

endpoint = 'todos'

params = None

api\_response = self.api\_client.make\_request(endpoint, params)

self.history.append({'command': command, 'data': api\_response})

self.display\_data(api\_response)

elif command == '2':

if self.history:

last\_response = self.history[-1]['data']

self.save\_data(last\_response, 'json')

self.save\_data(last\_response, 'csv')

self.save\_data(last\_response, 'txt')

print("Дані успішно збережено.")

else:

print("Немає даних для збереження.")

elif command == '3':

self.add\_new\_todo()

elif command.lower() == 'exit':

print("Вихід з програми.")

else:

print('Невірна команда. Будь ласка, введіть правильний номер опції або "exit".')

except Exception as e:

logging.error(f'Помилка обробки команди {command}: {e}')

def add\_new\_todo(self):

"""

Додає нову справу.

"""

title = input('Введіть назву нової справи: ')

completed = input('Чи завершена справа? (True/False): ').lower() == 'true'

new\_todo = {'userId': 1, 'title': title, 'completed': completed}

self.history.append({'command': 'add\_todo', 'data': new\_todo})

print('Нову справу успішно додано.')

def save\_data(self, data, file\_format):

"""

Зберігає дані у вказаному форматі.

Parameters:

data (list): Дані для збереження.

file\_format (str): Формат збереження ('json', 'csv', 'txt').

"""

FileStorage.store\_data(data, file\_format)

def run(self):

"""

Запускає інтерфейс користувача.

"""

while True:

command = self.display\_menu()

if command.lower() == 'exit':

break

self.process\_command(command)

def run\_lab7():

api\_url = 'https://jsonplaceholder.typicode.com'

api\_client = ApiClient(api\_url)

interface = UserInterface(api\_client)

interface.run()

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

run\_lab7()

**unit.py**

import unittest

from unittest.mock import patch

from unittest.mock import MagicMock, patch

from main import ApiClient

from io import StringIO

from main import ApiClient, UserInterface

class TestApiClient(unittest.TestCase):

@patch('main.requests.get')

def test\_make\_request\_failure(self, mock\_get):

# Створення об'єкта MagicMock, який імітує властивість raise\_for\_status

response\_mock = MagicMock()

response\_mock.raise\_for\_status.side\_effect = Exception("API error")

# Налаштування поведінки MagicMock для об'єкта requests.get

mock\_get.return\_value = response\_mock

# Створення об'єкта ApiClient та виклик методу, який викликає requests.get

api\_client = ApiClient(api\_url='https://example.com')

with self.assertRaises(Exception, msg="API error"):

api\_client.make\_request('todos')

class TestUserInterface(unittest.TestCase):

def test\_add\_new\_todo(self):

api\_url = 'https://jsonplaceholder.typicode.com'

api\_client = ApiClient(api\_url)

interface = UserInterface(api\_client)

# Ми використовуємо unittest.mock.patch, щоб вказати введені дані для тестування.

with patch('builtins.input', side\_effect=['Test Todo', 'true']):

# Викликаємо add\_new\_todo.

interface.add\_new\_todo()

# Перевіряємо, чи є нова справа у списку історії.

self.assertEqual(interface.history[-1]['data']['title'], 'Test Todo')

self.assertEqual(interface.history[-1]['data']['completed'], True)

def test\_display\_menu(self):

api\_url = 'https://jsonplaceholder.typicode.com'

api\_client = ApiClient(api\_url)

interface = UserInterface(api\_client)

# Використовуємо unittest.mock.patch для тестування введення з клавіатури.

with patch('builtins.input', return\_value='1'):

# Викликаємо display\_menu та перевіряємо, чи виведено правильне меню.

with patch('sys.stdout', new\_callable=StringIO) as mock\_stdout:

interface.display\_menu()

self.assertEqual(mock\_stdout.getvalue().strip(), 'Виберіть опцію:\n1. Переглянути список справ\n2. Зберегти дані\n3. Додати нову справу\nexit. Вийти з програми')

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

unittest.main(exit=False)

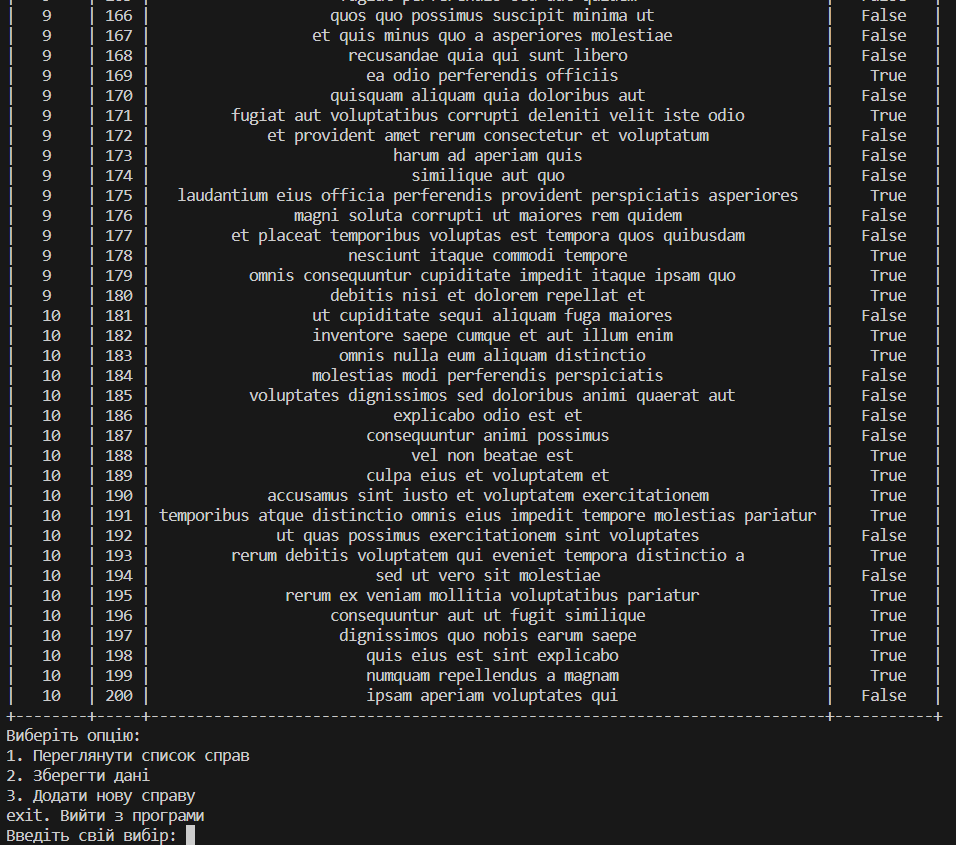


Рис.1 Результат виконання роботи

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