# **МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА” ІНСТИТУТ КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ** **Кафедра ІСМ**

# **Звіт**

# **до лабораторної роботи №7**

# **З дисципліни “Спеціальні мови програмування”**

**Виконав:  
ст.гр.ІТ-31  
Шельвах Максим  
  
Прийняв:  
Щербак С.С.**

# **Львів — 2023**

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

**Мета роботи:** 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: 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

**Реалізований код:**

main.py

from .API.api import APIClient, prompt\_for\_endpoint, safe\_api\_request, display\_data, save\_data

from .history.history import CalculationHistory

def main():

api\_client = APIClient('https://jsonplaceholder.typicode.com')

history = CalculationHistory()

while True:

endpoint = prompt\_for\_endpoint()

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

break

data = safe\_api\_request(api\_client.get, f'/{endpoint}')

if data:

display\_data(data)

history.add(data)

if input("Would you like to save the data? (yes/no): ").lower() == 'yes':

format = input("Which format would you like to use? (json/csv): ").lower()

save\_data(data, format)

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

main()

unittests.py

import unittest

from .API.api import APIClient, parse\_input

class TestAPIClient(unittest.TestCase):

def setUp(self):

self.client = APIClient('https://jsonplaceholder.typicode.com')

def test\_get\_posts(self):

posts = self.client.get('/posts')

self.assertIsInstance(posts, list)

self.assertTrue(len(posts) > 0)

for post in posts:

self.assertIsInstance(post, dict)

self.assertIn('userId', post)

self.assertIn('id', post)

self.assertIn('title', post)

self.assertIn('body', post)

def test\_parse\_input(self):

self.assertEqual(parse\_input("01-01-2020"), ["01-01-2020"])

self.assertEqual(parse\_input("The date is 01-01-2020."), ["01-01-2020"])

self.assertEqual(parse\_input("No date here"), [])

def main():

test\_lab7 = unittest.TestLoader().loadTestsFromTestCase(TestAPIClient)

test\_loader = unittest.TextTestRunner()

test\_loader.run(test\_lab7)

input()

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

main()

api.py

import unittest

import requests

import re

import json

import csv

import textwrap

from prettytable import PrettyTable

import os

class APIClient:

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

self.base\_url = base\_url

def get(self, endpoint):

# Виконує GET-запит до вказаної точки (endpoint) API і повертає відповідь у форматі JSON

response = requests.get(self.base\_url + endpoint)

return response.json()

def prompt\_for\_endpoint():

# Перепитує користувача про введення точки (endpoint) API

return input("Please enter the endpoint ('posts', 'comments', etc.): ")

def parse\_input(input\_string):

# Використовує регулярний вираз для пошуку дат у введеному рядку

dates = re.findall(r'\b\d{2}-\d{2}-\d{4}\b', input\_string)

return dates

def display\_data(data):

if not data:

print("No data available to display.")

return

if isinstance(data[0], dict):

# Створює PrettyTable для відображення даних у вигляді таблички

table = PrettyTable(field\_names=data[0].keys())

else:

print("Data format is not recognized.")

return

for item in data:

# Заповнює табличку даними, обгортаючи значення для кращого відображення

row = [textwrap.fill(str(value), width=50) for value in item.values()]

table.add\_row(row)

print(table)

def save\_data(data, format):

# Визначаємо шлях до папки 'output', яка знаходиться на рівень вище

output\_dir = os.path.join(os.path.dirname(\_\_file\_\_), '..', '..', 'output')

# Створюємо папку, якщо вона ще не існує

if not os.path.exists(output\_dir):

os.makedirs(output\_dir)

# Визначаємо шлях до файлу у папці 'output' та зберігаємо дані у вказаному форматі (json або csv)

if format == 'json':

file\_path = os.path.join(output\_dir, 'data.json')

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

json.dump(data, f)

elif format == 'csv':

file\_path = os.path.join(output\_dir, 'data.csv')

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

writer = csv.DictWriter(f, fieldnames=data[0].keys())

writer.writeheader()

writer.writerows(data)

def safe\_api\_request(callable, \*args, \*\*kwargs):

try:

# Викликає функцію callable з переданими аргументами та оброблює винятки

return callable(\*args, \*\*kwargs)

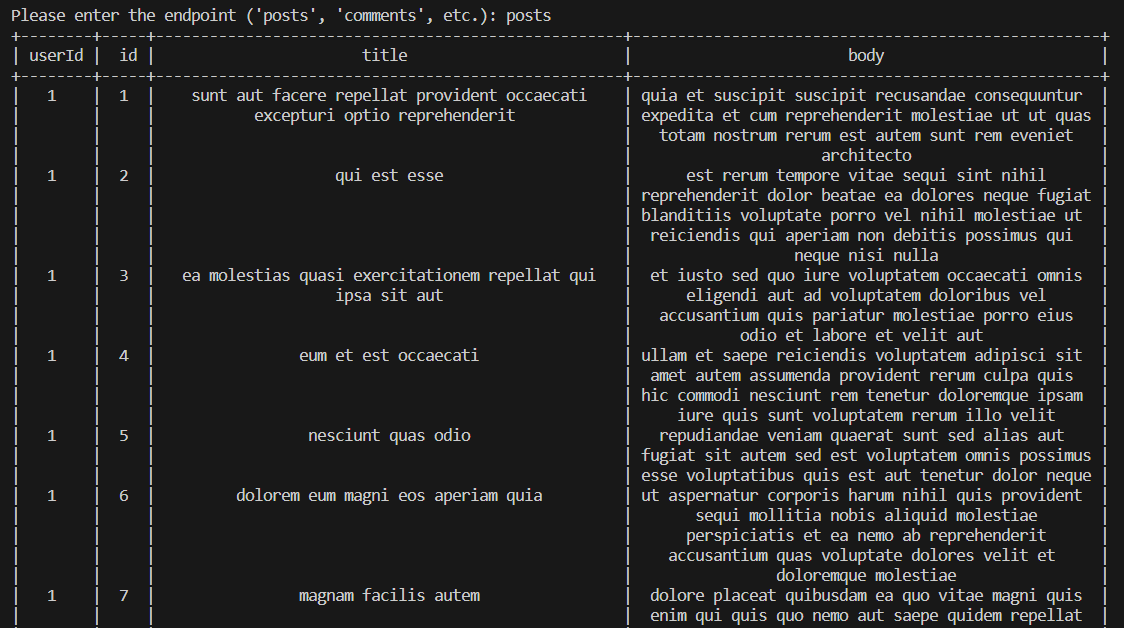
except requests.RequestException as e:

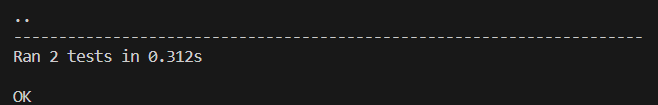
print(f"An error occurred during API request: {e}")

except Exception as e:

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

**Результат виконання програми:**





**Висновок:** виконавши ці завдання, я створив проект, який надав мені цінний досвід роботи з API, дизайну користувацького інтерфейсу, валідації введення, обробки помилок та тестування.