Міністерство освіти і науки України  
Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра інформаційних систем та мереж

******

**ЗВІТ**

**ДО ЛАБОРАТОРНОЇ РОБОТИ №7  
з дисципліни  
«Спеціалізовані мови програмування»  
на тему:**  
«Робота з API та веб-сервісами»

***Виконала:*** *студентка групи РІ-31  
 Діана ВІЙЧУК*

***Прийняв:****к. т. н., доцент* *Сергій ЩЕРБАК*

## МЕТА РОБОТИ

## Створення консольного об’єктно - орієнтованого додатка з використанням API та патернів проектування

## ЗАВДАННЯ

Завдання 1: Вибір провайдера API та патернів проектування

Виберіть надійний API, який надає через HTTP необхідні дані для віддаленого зберігання, вивантаження або реалізуйте свій. Для прикладу це може бути jsonplaceholder.org. Крім того, оберіть 2-3 патерна проектування для реалізаціі імплементаціі цієї лабораторноі роботи. Для прикладу, це може бути патерн Unit of Work та Repository

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.

## ПРОГРАМНА РЕАЛІЗАЦІЯ

Файл */runner.py* .

## import sys

## import os

## lab7\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..", ".."))

## sys.path.append(lab7\_root)

## from src.main import main

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

## main()

Файл */src/bll/classes/api/ApiHandler.py* .

import requests

class APIHandler:

BASE\_URL = "https://jsonplaceholder.typicode.com"

\_instance = None

def \_\_new\_\_(cls):

if cls.\_instance is None:

cls.\_instance = super(APIHandler, cls).\_\_new\_\_(cls)

return cls.\_instance

def get\_data(self, endpoint):

response = requests.get(f"{self.BASE\_URL}/{endpoint}")

return response.json()

Файл */src/bll/classes/api/user\_repository.py*

from src.bll.classes.api.ApiHandler import APIHandler

class UserRepository:

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

self.api\_handler = api\_handler

def get\_users(self):

return self.api\_handler.get\_data("users")

def get\_posts(self):

return self.api\_handler.get\_data("posts")

Файл */src/bll/classes/api/interaction.py*

from src.bll.classes.api.ApiHandler import APIHandler

from src.bll.classes.api.user\_repository import UserRepository

from src.bll.classes.api.display import DisplayStrategy, DisplayFactory

from shered.constants.globalVariables import history

from src.bll.classes.api.utils import show\_history, save\_data, log\_request

def get\_user\_input():

action = input("Choose view format: 1. Table 2. List: ").strip()

if action == '1':

return 'table'

elif action == '2':

return 'list'

else:

print("Invalid input. Please try again.")

return get\_user\_input()

def interact\_with\_user(action):

api = APIHandler()

user\_repository = UserRepository(api)

display\_format = get\_user\_input()

print(f"Selected display format: {display\_format}")

display\_factory = DisplayFactory()

display\_strategy = display\_factory.get\_display\_strategy(display\_format) # Отримуємо стратегію

if action == 'get':

endpoint = input("Enter endpoint (e.g., posts, users): ")

if endpoint == "users":

data = user\_repository.get\_users()

elif endpoint == "posts":

data = user\_repository.get\_posts()

else:

print("Invalid endpoint.")

return

if data:

display\_strategy.display(data)

result = f"Data received successfully, displaying as {display\_format}."

else:

result = "No data received."

print(result)

log\_request("GET", endpoint, data, display\_format, result)

elif action == 'history':

show\_history()

elif action == 'save':

file\_format = input("Enter format to save data (json/csv/txt): ")

save\_data(history, file\_format)

Файл */src/bll/classes/api/display.py*

from src.bll.classes.api.ApiHandler import APIHandler

from src.bll.classes.api.user\_repository import UserRepository

from src.bll.classes.api.display import DisplayStrategy, DisplayFactory

from shered.constants.globalVariables import history

from src.bll.classes.api.utils import show\_history, save\_data, log\_request

def get\_user\_input():

action = input("Choose view format: 1. Table 2. List: ").strip()

if action == '1':

return 'table'

elif action == '2':

return 'list'

else:

print("Invalid input. Please try again.")

return get\_user\_input()

def interact\_with\_user(action):

api = APIHandler()

user\_repository = UserRepository(api)

display\_format = get\_user\_input()

print(f"Selected display format: {display\_format}")

display\_factory = DisplayFactory()

display\_strategy = display\_factory.get\_display\_strategy(display\_format) # Отримуємо стратегію

if action == 'get':

endpoint = input("Enter endpoint (e.g., posts, users): ")

if endpoint == "users":

data = user\_repository.get\_users()

elif endpoint == "posts":

data = user\_repository.get\_posts()

else:

print("Invalid endpoint.")

return

if data:

display\_strategy.display(data)

result = f"Data received successfully, displaying as {display\_format}."

else:

result = "No data received."

print(result)

log\_request("GET", endpoint, data, display\_format, result)

elif action == 'history':

show\_history()

elif action == 'save':

file\_format = input("Enter format to save data (json/csv/txt): ")

save\_data(history, file\_format)

Файл */src/bll/classes/api/utils.py*

import json

import csv

import re

from termcolor import colored

from tabulate import tabulate

from shered.constants.globalVariables import history

def display\_results(data):

if isinstance(data, list):

for item in data:

print(colored(f"{item}", "green"))

elif isinstance(data, dict):

for key, value in data.items():

print(colored(f"{key}: {value}", "yellow"))

def get\_user\_color\_choice():

colors = ["red", "green", "yellow", "blue", "magenta", "cyan", "white"]

print("Available colors for headers:")

for idx, color in enumerate(colors, 1):

print(f"{idx}. {color}")

choice = input("Choose a color for table headers by entering the number: ").strip()

if choice.isdigit() and 1 <= int(choice) <= len(colors):

return colors[int(choice) - 1]

else:

print("Invalid choice, defaulting to 'blue'.")

return "blue"

def display\_as\_table(data, headers=None,header\_color="blue"):

if data and isinstance(data, list) and all(isinstance(item, dict) for item in data):

if not headers:

all\_keys = set().union(\*(item.keys() for item in data))

headers = list(all\_keys)[:6]

header\_color = get\_user\_color\_choice()

filtered\_data = [

{key: item.get(key, '') for key in headers} for item in data

]

rows = [list(item.values()) for item in filtered\_data]

colored\_headers = [colored(header, header\_color) for header in headers]

print(tabulate(rows, headers=colored\_headers, tablefmt='pretty'))

else:

print("No data to display in table format.")

def display\_as\_list(data):

if data:

for i, item in enumerate(data, start=1):

if isinstance(item, dict):

print(f"{i}. {str(item)}")

else:

print(f"{i}. {item}")

else:

print("No data to display in list format.")

def validate\_date(date\_str):

pattern = r"^\d{4}-\d{2}-\d{2}$"

return bool(re.match(pattern, date\_str))

def validate\_phone(phone\_str):

pattern = r"^\+\d{1}-\d{3}-\d{3}-\d{4}$"

return bool(re.match(pattern, phone\_str))

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

try:

if file\_format == 'json':

with open('data.json', 'w') as f:

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

print("Data saved to data.json")

elif file\_format == 'csv':

with open('data.csv', 'w', newline='') as f:

writer = csv.writer(f)

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

headers = data[0].keys()

writer.writerow(headers)

for entry in data:

writer.writerow(entry.values())

print("Data saved to data.csv")

elif file\_format == 'txt':

with open('data.txt', 'w') as f:

for entry in data:

if isinstance(entry, dict):

for key, value in entry.items():

f.write(f"{key}: {value}\n")

else:

f.write(str(entry) + '\n')

print("Data saved to data.txt")

else:

print("Invalid format. Please use json/csv/txt.")

except Exception as e:

print(f"Error saving data: {e}")

def log\_request(action, endpoint, data, display\_format, result):

history.append({

"Action": action,

"Endpoint": endpoint,

"Data": data,

"DisplayFormat": display\_format,

"Result": result

})

print("Request logged to history.")

def show\_history():

if not history:

print("No history available.")

else:

print("\nHistory of requests:")

for i, entry in enumerate(history, 1):

if isinstance(entry, dict):

action = entry.get('Action', 'Unknown Action')

endpoint = entry.get('Endpoint', 'Unknown Endpoint')

data = entry.get('Data', 'No Data')

display\_format = entry.get('DisplayFormat', 'Unknown Format')

print(f"{i}. Action: {action}, Endpoint: {endpoint}, Data: {data}, Display Format: {display\_format}")

else:

print(f"{i}. {entry}")

Файл */src/main.py*

import sys

import os

lab7\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..", ".."))

sys.path.append(lab7\_root)

from src.ui.menu import display\_menu, get\_user\_choice

from src.bll.classes.api.interaction import interact\_with\_user

def main():

while True:

display\_menu()

choice = get\_user\_choice()

if choice == '1':

interact\_with\_user('get')

elif choice == '2':

interact\_with\_user('history')

elif choice == '3':

interact\_with\_user('save')

elif choice == '4':

print("Exiting...")

break

else:

print("Invalid choice. Please try again.")

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

main()

## ВИСНОВКИ

На цій лабораторній роботі я створила Unitt тести для додатка калькулятора на основі класів.

Посилання на GitHub: <https://github.com/Dinasi4ka/Calculator.git>