МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

Лабораторна робота №7

з дисципліни

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

на тему

РОБОТА З API ТА ВЕБ-СЕРВІСОМ

Виконала:

ст. гр. ІТ-21сп

Борисов П.Р.

Прийняв:

Щербак С.С.

Львів-2023

**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.

**Код програми:**  
from colorama import Fore  
from colorama import Style  
  
  
class APIApp:  
 def \_\_init\_\_(self, weather\_api\_client, reader\_writer, validator, history, error\_handler, results\_display,  
 input\_parser, data\_saver):  
 self.weather\_api\_client = weather\_api\_client  
 self.reader\_writer = reader\_writer  
 self.validator = validator  
 self.history = history  
 self.error\_handler = error\_handler  
 self.results\_display = results\_display  
 self.input\_parser = input\_parser  
 self.data\_saver = data\_saver  
  
 def get\_weather\_by\_city(self, city):  
 weather\_data = self.weather\_api\_client.get\_weather\_by\_city(city)  
 if 'main' in weather\_data:  
 temp = weather\_data['main']['temp']  
 feels\_like = weather\_data['main']['feels\_like']  
 return {'temp': temp, 'feels\_like': feels\_like}  
 return {}  
  
 def get\_wind\_info(self, city):  
 weather\_data = self.weather\_api\_client.get\_weather\_by\_city(city)  
 if 'wind' in weather\_data:  
 return weather\_data['wind']  
 return {}  
  
 def check\_snow(self, city):  
 weather\_data = self.weather\_api\_client.get\_weather\_by\_city(city)  
 return 'snow' in weather\_data  
  
 def get\_user\_input(self):  
 data\_type = self.validator.validate\_input("Enter the desired information ( temp, wind , is snowing): ",  
 ['phone', 'email', 'payment card', 'date'])  
 user\_input = self.reader\_writer.read\_input(f"Enter {data\_type}: ")  
 return data\_type, user\_input  
  
 def run(self, data\_type, city):  
 try:  
 # Task 1: Selecting the API provider  
 data\_from\_api = None  
  
 if data\_type == '1':  
 data\_from\_api = self.get\_weather\_by\_city(city)  
 elif data\_type == '2':  
 data\_from\_api = self.get\_wind\_info(city)  
 elif data\_type == '3':  
 data\_from\_api = self.check\_snow(city)  
  
 # Adding the user's request and the result to the history  
 self.history.add\_to\_history(city, (data\_from\_api))  
  
 # Task 5: Displaying the results  
 self.results\_display.display\_data(data\_from\_api, Fore.BLUE, Style.BRIGHT)  
  
 # Task 6: Saving the data  
 self.data\_saver.save\_data(data\_from\_api, 'json')  
  
 except Exception as e:  
 print(f"Error: {e}")

import json  
import pandas as pd  
  
  
# Class for saving data  
class DataSaver:  
 def save\_data(self, data, format):  
 if format == 'json':  
 with open('data.json', 'w') as f:  
 json.dump(data, f, indent=4)  
 elif format == 'csv':  
 if isinstance(data, dict):  
 data = [data] # convert dictionary to list of dictionaries  
 df = pd.DataFrame(data) # create DataFrame  
 df.to\_csv('data.csv')  
 elif format == 'txt':  
 with open('data.txt', 'w') as f:  
 for item in data:  
 f.write("%s\n" % item)

# Class for error handling  
class ErrorHandler:  
 def handle\_error(self, func):  
 def wrapper(\*args, \*\*kwargs):  
 try:  
 return func(\*args, \*\*kwargs)  
 except Exception as e:  
 print(f"Error: {e}")  
 return [], [], [], [] # return empty lists  
  
 return wrapper

class History:  
 def \_\_init\_\_(self):  
 self.history = []  
 self.query\_id = 0  
  
 def add\_to\_history(self, query, result):  
 self.query\_id += 1  
 self.history.append((self.query\_id, query, result))  
  
 def view\_history(self):  
 for item in self.history:  
 print(f"{item[0]}. Query: {item[1]}, Result: {item[2]}")

import json  
import os  
import re  
  
  
# Class for handling user input and parsing  
class InputParser:  
 def \_\_init\_\_(self):  
 with open(os.path.join('..', 'Config', 'default.json')) as f:  
 data = json.load(f)  
 self.date\_pattern = data['patterns']['date']  
 self.phone\_pattern = data['patterns']['phone']  
 self.email\_pattern = data['patterns']['email']  
 self.credit\_card\_pattern = data['patterns']['credit\_card']  
  
  
 def parse\_user\_input(self, user\_input):  
 date\_tuples = re.findall(self.date\_pattern, user\_input)  
 dates = ['/'.join(date\_tuple) for date\_tuple in date\_tuples]  
 phones = re.findall(self.phone\_pattern, user\_input)  
 emails = [email.rstrip(',') for email in re.findall(self.email\_pattern, user\_input)]  
 credit\_cards = re.findall(self.credit\_card\_pattern, user\_input)  
  
 return dates, phones, emails, credit\_cards

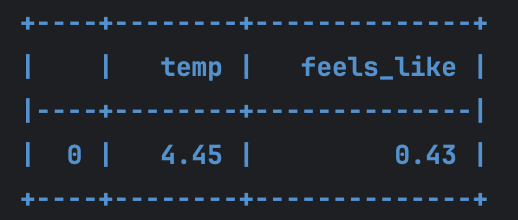
import pandas as pd  
from tabulate import tabulate  
from colorama import Style  
import numpy as np  
  
  
# Class for displaying results  
class ResultsDisplay:  
 def display\_data(self, data, headers\_color, font\_style):  
 if np.isscalar(data):  
 df = pd.DataFrame([data]) # if data is a scalar value, create DataFrame from list  
 elif isinstance(data, dict):  
 df = pd.DataFrame([data]) # if data is a dictionary, create DataFrame from list of dictionaries  
 else:  
 df = pd.DataFrame(data) # otherwise, create DataFrame as usual  
 formatted\_table = tabulate(df, headers='keys', tablefmt='psql')  
 print(self.apply\_styles(formatted\_table, headers\_color, font\_style))  
  
 def apply\_styles(self, text, color, style):  
 return f"{color}{style}{text}{Style.RESET\_ALL}"

import unittest  
from unittest.mock import patch, Mock  
  
from BLL.Lab7.APIClient import APIClient  
from BLL.Lab7.InputParser import InputParser  
from Shared.Validator import Validator  
  
  
# Unit tests for the API  
class UnitTestsAPI(unittest.TestCase):  
 def setUp(self):  
 self.api\_client = APIClient('https://jsonplaceholder.typicode.com/posts/1')  
 self.input\_parser = InputParser()  
 self.validator = Validator(self.input\_parser)  
  
 @patch('requests.get')  
 def test\_get\_data(self, mock\_get):  
 mock\_response = Mock()  
 expected\_dict = {"title": "foo", "body": "bar", "userId": 1}  
 mock\_response.json.return\_value = expected\_dict  
 mock\_get.return\_value = mock\_response  
 self.assertEqual(self.api\_client.get\_data(), expected\_dict)  
  
 def test\_validate\_phone(self):  
 self.assertTrue(self.validator.validate\_phone('+380631234567'))  
 self.assertFalse(self.validator.validate\_phone('1234567'))  
  
 def test\_validate\_email(self):  
 self.assertTrue(self.validator.validate\_email('test@example.com'))  
 self.assertFalse(self.validator.validate\_email('testexample.com'))  
  
 def test\_validate\_credit\_card(self):  
 self.assertTrue(self.validator.validate\_credit\_card('1234-5678-9012-3456'))  
 self.assertFalse(self.validator.validate\_credit\_card('1234-5678-9012'))  
  
 def test\_validate\_date(self):  
 self.assertTrue(self.validator.validate\_date('31/12/2023'))  
 self.assertFalse(self.validator.validate\_date('31/13/2023'))  
  
 def test\_parse\_user\_input(self):  
 dates, phones, emails, credit\_cards = self.input\_parser.parse\_user\_input('31/12/2023, +380631234567, test@example.com, 1234-5678-9012-3456')  
 self.assertEqual(dates, ['31/12/2023'])  
 self.assertEqual(phones, ['+380631234567'])  
 self.assertEqual(emails, ['test@example.com'])  
 self.assertEqual(credit\_cards, ['1234-5678-9012-3456'])  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

import requests  
  
  
class WeatherAPIClient:  
 def \_\_init\_\_(self, api\_key):  
 self.api\_key = api\_key  
 self.base\_url = "http://api.openweathermap.org/data/2.5/weather"  
  
 def get\_weather\_by\_city(self, city\_name, days=1):  
 params = {'q': city\_name, 'appid': self.api\_key, 'units': 'metric'}  
 response = requests.get(self.base\_url, params=params)  
 return response.json()

# Клас для запуску програми  
from BLL.lab7.APIApp import APIApp  
from BLL.lab7.DataSaver import DataSaver  
from BLL.lab7.InputParser import InputParser  
from BLL.lab7.WeatherAPIClient import WeatherAPIClient  
from SharedAccess.ReaderWriter import ReaderWriter  
from SharedAccess.Validator import Validator  
from BLL.lab7.History import History  
from BLL.lab7.ErrorHandler import ErrorHandler  
from BLL.lab7.ResultsDisplay import ResultsDisplay  
from UI.Menu.MenuLab7 import MenuLab7  
  
  
class RunnerLab7:  
 def run\_program(self):  
 api\_key = "49d965b7ac6933d5cc545a73b1751b78"  
 weather\_api\_client = WeatherAPIClient(api\_key)  
  
 reader\_writer = ReaderWriter()  
 input\_parser = InputParser() # Creating an instance of the InputParser class  
 validator = Validator(input\_parser) # Passing the instance of the InputParser class to the Validator class  
 history = History()  
 error\_handler = ErrorHandler()  
 results\_display = ResultsDisplay()  
 data\_saver = DataSaver()  
  
 api\_app = APIApp(weather\_api\_client, reader\_writer, validator, history, error\_handler, results\_display, input\_parser,  
 data\_saver)  
  
 menu\_lab7 = MenuLab7(api\_app, validator, reader\_writer)  
 menu\_lab7.run()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 RunnerLab7().run\_program()

На рисунку 1 зображено результат програми.



**Рис. 1** Результат виконання програми(запит)

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