МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ КОМП’ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ



Лабораторна робота №7

з дисципліни “Спеціалізовані мови програмування”

на тему

“Робота з API та веб-сервісами

Виконала:

студентка групи РІ-32

Мар’ян МІЛЯНЕЦЬ

Прийняв:

к.т.н.,

доц. кафедри ІСМ

Сергій ЩЕРБАК

Львів-2023

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

**План роботи**

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.

Програмний код:

**from abc import ABC, abstractmethod**

import requests

from tabulate import tabulate

import json

import csv

# Command Interface

class Command(ABC):

@abstractmethod

def execute(self):

pass

# Concrete Command

class WeatherCommand(Command):

def \_\_init\_\_(self, api\_key, base\_url, city):

self.api\_key = api\_key

self.base\_url = base\_url

self.city = city

def execute(self):

params = {'q': self.city, 'key': self.api\_key}

response = requests.get(self.base\_url, params=params)

# Print debugging information

print(f"Request URL: {response.url}")

print(f"Status Code: {response.status\_code}")

try:

response.raise\_for\_status()

data = response.json()

print(f"Response Data: {data}")

return data

except requests.exceptions.HTTPError as http\_err:

print(f"HTTP error occurred: {http\_err}")

except json.JSONDecodeError as json\_err:

print(f"Error decoding JSON response: {json\_err}")

except requests.RequestException as req\_err:

print(f"An error occurred during the request: {req\_err}")

return None

# Invoker

class WeatherInvoker:

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

self.command = None

def set\_command(self, command):

self.command = command

def execute\_command(self):

if self.command:

return self.command.execute()

else:

print("No command set.")

class WeatherApp:

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

self.base\_url = base\_url

self.api\_key = api\_key

self.history = []

self.invoker = WeatherInvoker()

def get\_weather(self, city):

command = WeatherCommand(self.api\_key, self.base\_url, city)

self.invoker.set\_command(command)

result = self.invoker.execute\_command()

# Check if the response is successful

if result is not None and 'error' not in result:

# Extract relevant information from the response

current\_data = result.get('current', {})

temperature = current\_data.get('temp\_c', 'N/A')

description = current\_data.get('condition', {}).get('text', 'N/A')

weather\_info = {'city': city, 'temperature': temperature, 'description': description}

self.history.append(weather\_info)

return weather\_info

else:

print(f"Error: Unable to retrieve weather information for {city}.")

return None

def display\_table(self, data):

headers = ['City', 'Temperature (°C)', 'Description']

data\_list = [

[data['city'], data['temperature'], data['description']]

]

print(tabulate(data\_list, headers=headers, tablefmt='grid'))

def save\_history(self, filename, file\_format):

if file\_format == 'json':

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

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

elif file\_format == 'txt':

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

for entry in self.history:

txt\_file.write(f"{entry['city']}: {entry['temperature']}°C, {entry['description']}\n")

elif file\_format == 'csv':

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

headers = list(self.history[0].keys()) if self.history else []

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

writer.writeheader()

writer.writerows(self.history)

def display\_history(self):

if not self.history:

print("No history available.")

else:

print("Weather History:")

headers = ['City', 'Temperature (°C)', 'Description']

data\_list = [

[entry['city'], entry['temperature'], entry['description']] for entry in self.history

]

print(tabulate(data\_list, headers=headers, tablefmt='grid'))

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

api\_key = '676bfc001e434e959b4143404231312'

base\_url = 'http://api.weatherapi.com/v1/current.json'

weather\_app = WeatherApp(api\_key, base\_url)

while True:

user\_input = input('Enter city name (or type "history" to view history, "save" to save history, or "exit" to quit): ')

if user\_input.lower() == 'exit':

break

elif user\_input.lower() == 'history':

weather\_app.display\_history()

elif user\_input.lower() == 'save':

save\_format = input('Enter save format (json, txt, or csv): ')

save\_filename = input('Enter save filename: ')

if not save\_filename.endswith('.'):

save\_filename += '.'

save\_filename += save\_format

weather\_app.save\_history(save\_filename, save\_format)

print(f"History saved to {save\_filename} in {save\_format} format.")

else:

weather\_data = weather\_app.get\_weather(user\_input)

if weather\_data is not None:

weather\_app.display\_table(weather\_data)

**import unittest**

from unittest.mock import Mock, patch

from sources.lab7.lab7 import WeatherCommand

class TestWeatherCommand(unittest.TestCase):

def setUp(self):

self.api\_key = '676bfc001e434e959b4143404231312'

self.base\_url = 'http://api.weatherapi.com/v1/current.json'

self.city = 'Lviv'

@patch('lab7.app.requests.get')

def test\_execute\_successful\_request(self, mock\_requests\_get):

# Mock the requests.get method to return a successful response

mock\_response = Mock()

mock\_response.status\_code = 200

mock\_response.json.return\_value = {'current': {'temp\_c': 25, 'condition': {'text': 'Sunny'}}}

mock\_requests\_get.return\_value = mock\_response

weather\_command = WeatherCommand(self.api\_key, self.base\_url, self.city)

result = weather\_command.execute()

# Assertions

self.assertIsNotNone(result)

self.assertEqual(result['current']['temp\_c'], 25)

self.assertEqual(result['current']['condition']['text'], 'Sunny')

@patch('lab7.app.requests.get')

def test\_execute\_unsuccessful\_request(self, mock\_requests\_get):

mock\_response = Mock()

mock\_response.status\_code = 404

mock\_requests\_get.return\_value = mock\_response

weather\_command = WeatherCommand(self.api\_key, self.base\_url, self.city)

result = weather\_command.execute()

# Assertions

self.assertIsNotNone(result)

mock\_requests\_get.assert\_called\_once\_with(

self.base\_url, params={'q': self.city, 'key': self.api\_key}

)

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

unittest.main()

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