МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №7

з дисципліни

«Спеціалізовані мови приграмування»

на тему

«Робота з API та веб-сервісами»

Виконав:

ст. гр. РІ-32

Сергій БИХНЮК

Прийняв:

Сергій Щербак

Львів – 2024

**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.

**Код програми:**

class Logger:

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

self.query\_history = []

def log\_query(self, query, result):

self.query\_history.append({"query": query, "result": result})

def show\_history(self):

if not self.query\_history:

print("No queries have been logged yet.")

else:

print("\nQuery History:")

for i, entry in enumerate(self.query\_history, 1):

print(f"{i}. Query: {entry['query']}, Result: {entry['result']}")

class WeatherData:

def \_\_init\_\_(self, city, temperature, description):

self.city = city

self.temperature = temperature

self.description = description

def as\_dict(self):

"""Returns the weather data as a dictionary for easy logging."""

return {

"city": self.city,

"temperature": self.temperature,

"description": self.description

}

Config

API\_URL = "https://api.openweathermap.org/data/2.5/weather"

API\_KEY = "bd89d8d6bee62af403385dff3b2b490a"

import requests

from lab7.Constant.Config import API\_URL, API\_KEY

from lab7.Classes.WeatherData import WeatherData

def get\_weather(city):

try:

params = {

'q': city,

'appid': API\_KEY,

'units': 'metric'

}

response = requests.get(API\_URL, params=params)

if response.status\_code == 200:

data = response.json()

# Return a WeatherData object instead of a dictionary

return WeatherData(

city=data['name'],

temperature=data['main']['temp'],

description=data['weather'][0]['description'].capitalize()

)

else:

print(f"Error {response.status\_code}: City '{city}' not found!")

return None

except requests.exceptions.RequestException as e:

print(f"Error: Unable to connect to the API. Details: {e}")

return None

from abc import ABC, abstractmethod

class ILogger(ABC):

@abstractmethod

def log\_query(self, query, result):

pass

class ILogger(ABC):

@abstractmethod

def log\_query(self, query, result):

pass

# Concrete Logger classes

class ConsoleLogger(ILogger):

def log\_query(self, query, result):

print(f"Query: {query}, Result: {result}")

class FileLogger(ILogger):

def log\_query(self, query, result):

with open("log.txt", "a") as file:

file.write(f"Query: {query}, Result: {result}\n")

class Logger(ConsoleLogger):

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

self.query\_history = []

def log\_query(self, query, result):

super().log\_query(query, result) # Calls ConsoleLogger to print the query

self.query\_history.append({"query": query, "result": result})

def show\_history(self):

if not self.query\_history:

print("No queries have been logged yet.")

else:

print("\nQuery History:")

for i, entry in enumerate(self.query\_history, 1):

print(f"{i}. Query: {entry['query']}, Result: {entry['result']}")

from lab7.Classes.Logger import Logger

from lab7.Functions.WeatherFunctions import get\_weather

def main7():

logger = Logger()

while True:

city = input("Enter the city: ")

if not city:

print("Please enter a valid city name.")

continue

weather = get\_weather(city)

if weather:

print(f"Weather in {weather['city']}: {weather['temperature']}°C, {weather['description']}")

logger.log\_query(city, weather)

else:

print(f"Could not fetch weather for {city}.")

# Ask user if they want to check another city

choice = input("Do you want to check another city? (y/n): ").lower()

if choice != 'y':

break

# Show the query history

logger.show\_history()

from lab7.Functions.WeatherFunctions import get\_weather

from lab7.Classes.Logger import Logger

def main7():

logger = Logger() # Initialize a logger instance

while True:

city = input("Enter the city: ")

if not city:

print("Please enter a valid city name.")

continue

weather = get\_weather(city)

if weather:

print(f"Weather in {weather.city}: {weather.temperature}°C, {weather.description}")

# Log using the dictionary representation of WeatherData

logger.log\_query(city, weather.as\_dict())

else:

print(f"Could not fetch weather for {city}.")

# Ask user if they want to check another city

choice = input("Do you want to check another city? (y/n): ").lower()

if choice != 'y':

break

# Show the query history at the end

logger.show\_history()

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

main7()

import unittest

from lab7.Classes.Logger import Logger

class TestLogger(unittest.TestCase):

def setUp(self):

self.logger = Logger()

def test\_log\_query(self):

self.logger.log\_query("New York", {"temperature": 20})

self.assertEqual(len(self.logger.query\_history), 1)

self.assertEqual(self.logger.query\_history[0]['query'], "New York")

self.assertEqual(self.logger.query\_history[0]['result']['temperature'], 20)

def test\_empty\_history(self):

self.assertEqual(len(self.logger.query\_history), 0)

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

unittest.main()

import unittest

from lab7.Functions.WeatherFunctions import get\_weather, WeatherData

class TestWeatherFunctions(unittest.TestCase):

def test\_get\_weather\_success(self):

city = "London"

weather = get\_weather(city)

# Check if the returned object is an instance of WeatherData

self.assertIsNotNone(weather)

self.assertIsInstance(weather, WeatherData)

# Access attributes of WeatherData directly

self.assertIsInstance(weather.city, str)

self.assertIsInstance(weather.temperature, (int, float))

self.assertIsInstance(weather.description, str)

def test\_get\_weather\_invalid\_city(self):

city = "InvalidCityName"

weather = get\_weather(city)

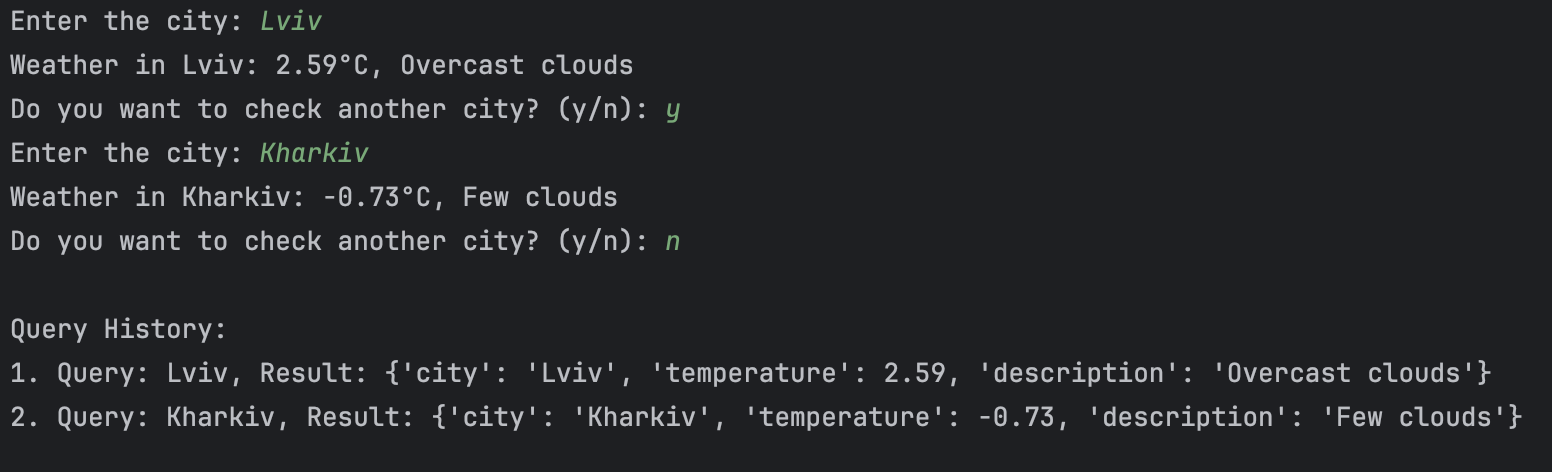
# Ensure that an invalid city returns None

self.assertIsNone(weather)

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

unittest.main()

**Виконання:**



**Висновок:**

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