Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Лабораторна роботи №7

з дисципліни «Спеціалізовані мови програмування»

на тему

«Робота з API та веб-сервісами»

Виконав:

Бабич О. Ю.

Перевірив:

Щербак С. С.

Львів 2024

**Мета:** Створення консольного об’єктно - орієнтованого додатка з використанням API та патернів проектування

**План роботи**

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.

**Реалізація:**

**Папка api\_client, файл api\_client.py:**  
import requests

class APIClient:

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

def get\_data(self, parsed\_input):

if isinstance(parsed\_input, tuple):

resource\_name, resource\_id = parsed\_input

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

else:

resource\_name = parsed\_input

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

if response.status\_code == 404:

raise ValueError(f"Error: {response.status\_code} - Запитаний ресурс не знайдено.")

try:

data = response.json()

return data

except ValueError:

raise ValueError("Невідомий формат даних.")

**Папка app, файл app.py:**

from api\_client.api\_client import APIClient

from input\_parser.input\_parser import InputParser

from data\_visualizer.data\_visualizer import DataVisualizer

from data\_storage.data\_storage import DataStorage

from error\_handler.error\_handler import ErrorHandler

from query\_history.query\_history import QueryHistory

def main():

api\_client = APIClient()

input\_parser = InputParser()

data\_visualizer = DataVisualizer()

data\_storage = DataStorage()

error\_handler = ErrorHandler()

query\_history = QueryHistory()

resource\_counts = {

"posts": 100,

"comments": 500,

"users": 10,

"albums": 100,

"photos": 5000,

"todos": 200,

}

print("Доступні ресурси та їх кількість:")

for resource, count in resource\_counts.items():

print(f"{resource} - {count}")

while True:

try:

user\_input = input("Введіть ваш запит у форматі 'назва запиту' - якщо хочете побачити всі запити, або 'назва запиту/id запиту' - для конкретного запиту (або 'exit' для виходу): ")

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

break

parsed\_input = input\_parser.parse(user\_input)

data = api\_client.get\_data(parsed\_input)

data\_visualizer.visualize(data)

data\_storage.save(data)

query\_history.record(user\_input, data)

except Exception as e:

error\_handler.handle(e)

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

main()

**Папка data\_storage, файл data\_storage.py:**import json

class DataStorage:

def save(self, data):

try:

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

json.dump(data, json\_file, indent=4) # Записуємо дані у JSON форматі

except Exception as e:

print(f"Помилка при збереженні даних: {e}")

**Папка data\_visualizer, файл data\_visualizer.py:**

class DataVisualizer:

def visualize(self, data):

if isinstance(data, list):

self.display\_paginated(data) # Стратегія для відображення списків

elif isinstance(data, dict):

self.display\_item(data) # Стратегія для одного елемента

else:

print("Невідомий формат даних.")

def display\_paginated(self, data):

# Кількість записів на сторінці

items\_per\_page = 10

total\_items = len(data)

total\_pages = (total\_items + items\_per\_page - 1) // items\_per\_page # Обчислюємо загальну кількість сторінок

while True:

# Відображення кількості сторінок

print(f"\nКількість сторінок: {total\_pages}")

# Запит на введення номера сторінки

user\_input = input(f"Введіть номер сторінки (1-{total\_pages}) або 'back' для повернення: ")

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

print("Повертаємося до основного меню...")

# Виводимо доступні запити та їх кількість

self.show\_available\_requests()

return # Повертаємося до основного меню

try:

page\_number = int(user\_input)

if page\_number < 1 or page\_number > total\_pages:

print("Помилка: Неправильний номер сторінки. Спробуйте ще раз.")

continue

# Виводимо елементи на вибраній сторінці

start\_index = (page\_number - 1) \* items\_per\_page

end\_index = min(start\_index + items\_per\_page, total\_items)

page\_items = data[start\_index:end\_index]

for item in page\_items:

self.display\_item(item)

print("\n" + "-" \* 40) # Відокремлюємо кожен елемент

except ValueError:

print("Помилка: Введіть коректний номер сторінки.")

def display\_item(self, item):

# Метод для виведення окремого елемента

print(item)

def show\_available\_requests(self):

resource\_counts = {

"posts": 100,

"comments": 500,

"users": 10,

"albums": 100,

"photos": 5000,

"todos": 200,

}

print("Доступні запити та їх кількість:")

for resource, count in resource\_counts.items():

print(f"{resource} - {count}")

**Папка error\_handler, файл error\_handler.py:**

class ErrorHandler:

def handle(self, error):

print(f"Помилка: {error}")

**Папка input\_parser, файл input\_parser.py:**

class InputParser:

def parse(self, user\_input):

if not user\_input.strip(): # Перевірка на пустий рядок

raise ValueError("Неправильний формат запиту.")

parts = user\_input.split('/')

if len(parts) == 1:

return parts[0] # Повертає тільки назву запиту

elif len(parts) == 2:

resource\_name, resource\_id = parts

if not resource\_id.isdigit(): # Перевірка, що ID є числом

raise ValueError("ID запиту має бути числом.")

return (resource\_name, resource\_id) # Повертає кортеж з назви запиту та ID

else:

raise ValueError("Неправильний формат запиту.")

**Папка query\_history, файл query\_history.py:**

class QueryHistory:

\_instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if cls.\_instance is None:

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

return cls.\_instance

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

if not hasattr(self, 'history'): # Ініціалізуємо тільки один раз

self.history = []

def record(self, query, result):

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

def show\_history(self):

for entry in self.history:

print(f"Запит: {entry['query']}, Результат: {entry['result']}")

**Файл tests.py:**

import unittest

from input\_parser.input\_parser import InputParser

class TestInputParser(unittest.TestCase):

def setUp(self):

self.parser = InputParser()

def test\_parse\_invalid\_input(self):

with self.assertRaises(ValueError):

self.parser.parse("") # Тестуємо на пустий рядок

self.parser.parse("invalid\_format/") # Тестуємо на неправильний формат

self.parser.parse("posts/abc") # Тестуємо на недійсне ID (не число)

def test\_parse\_valid\_input(self):

result = self.parser.parse("posts/1")

self.assertEqual(result, ("posts", "1"))

result = self.parser.parse("comments")

self.assertEqual(result, "comments")

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

unittest.main()

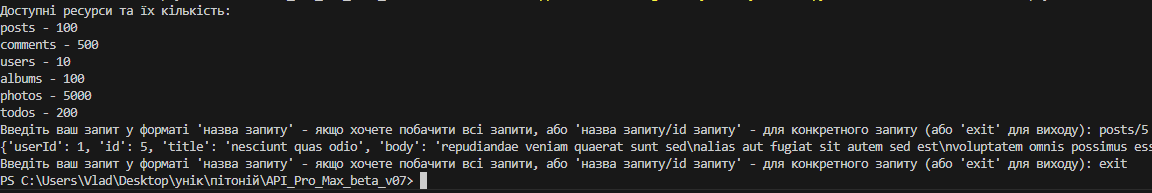
**Файл main.py:**

from app.app import main

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

main()

**Результат виконання:**

****

**Висновок:** на цій лабораторній роботі я створив API-проект використовуючи сторонній ресурс з даними