МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА

ІНСТИТУТ КОМП’ЮТЕРНИХ НАУК ТА ІНОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра ІСМ



ЗВІТ

Про виконання лабораторної роботи №7

«Робота з API та веб-сервісами»

З дисципліни

«Спеціалізовані мови програмування»

Студента групи РІ – 31

Тураша Івана Павловича

Прийняв викладач

Щербак С.С.

**Лабораторна робота № 7. Робота з API та веб-сервісами.**

**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.

**Основний код програми:**

from API\_Layer.api\_client import APIClient

from Repository\_Layer.repository import Repository

from Service\_Layer.service import DataService

from Persistance\_Layer.history import HistoryManager

from Factory\_Layer.factory import Factory

from Service\_Layer.unit\_of\_work import UnitOfWork

from input\_parser import InputParser

def main():

    # Створюємо об'єкти через фабрику

    api\_client = Factory.create\_api\_client()

    repository = Factory.create\_repository(api\_client)

    service = Factory.create\_data\_service(repository)

    history\_manager = Factory.create\_history\_manager()

    unit\_of\_work = UnitOfWork(history\_manager)

    while True:

        print("\nChoose an option:")

        print("1. List posts")

        print("2. List users")

        print("3. View history")

        print("4. Parse input")

        print("5. Exit")

        choice = input("Enter choice: ")

        if choice == '1':

            service.list\_posts()

            unit\_of\_work.record\_action("Listed posts")

        elif choice == '2':

            service.list\_users()

            unit\_of\_work.record\_action("Listed users")

        elif choice == '3':

            history\_manager.show\_history()

        elif choice == '4':

            user\_input = input("Enter text to parse: ")

            parsed\_result = InputParser.parse\_input(user\_input)

            print("Parsed data:", parsed\_result)

            unit\_of\_work.record\_action(f"Parsed input: {parsed\_result}")

        elif choice == '5':

            unit\_of\_work.commit()

            print("Exiting program.")

            break

        else:

            print("Invalid choice. Try again.")

from Repository\_Layer.repository import Repository

class DataService:

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

        self.repository = repository

    def list\_posts(self):

        posts = self.repository.get\_posts()

        if posts:

            for post in posts:

                print(f"ID: {post['id']}, Title: {post['title']}")

        else:

            print("No posts available.")

    def list\_users(self):

        users = self.repository.get\_users()

        if users:

            for user in users:

                print(f"ID: {user['id']}, Name: {user['name']}")

        else:

            print("No users available.")

class UnitOfWork:

    def \_\_init\_\_(self, history\_manager):

        self.history\_manager = history\_manager

        self.\_actions = []

    def record\_action(self, action):

        self.\_actions.append(action)

    def commit(self):

        for action in self.\_actions:

            self.history\_manager.record(action)

        self.\_actions.clear()

    def rollback(self):

        self.\_actions.clear()

from API\_Layer.api\_client import APIClient

class Repository:

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

        self.api\_client = api\_client

        self.cache = {}

    def get\_posts(self):

        if 'posts' not in self.cache:

            self.cache['posts'] = self.api\_client.get\_data('posts')

        return self.cache['posts']

    def get\_users(self):

        if 'users' not in self.cache:

            self.cache['users'] = self.api\_client.get\_data('users')

        return self.cache['users']

import requests

class APIClient:

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

    def get\_data(self, endpoint):

        try:

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

            response.raise\_for\_status()

            return response.json()

        except requests.RequestException as e:

            print(f"API Error: {e}")

            return None

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