МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

Звіт

до лабораторної роботи №7

Робота з API та веб-сервісами

Виконала:

ст. гр. PI-32
Довгошия А.А

Балів	**Дата**

Прийняв:

асис. каф. ІСМ

Щербак С.С

Львів — 2024

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

Хід 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 реалізації імплементації цієї лабораторної 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.

Api_service.py

```
import requests
from src.const.const import BASE_URL

class ApiService:
    """Handles API requests and responses."""
    _instance = None

    def __new__(cls):
        if cls._instance is None:
```

```
            cls._instance = super().__new__(cls)
        return cls._instance

    def get_data(self, endpoint):
        """Fetches data from the specified API endpoint."""
        try:
            response = requests.get(f"{BASE_URL}/{endpoint}")
            response.raise_for_status()
            return response.json()
        except requests.exceptions.HTTPError as http_err:
            print(f"HTTP error occurred: {http_err}")
            return None
        except Exception as err:
            print(f"An error occurred: {err}")
            return None
```

history_logger.py

```
from datetime import datetime

class HistoryLogger:
    """Logs the history of user choices."""
    @staticmethod
    def log(user_choice):
        """Logs the user's choice."""
        with open('user_history.log', 'a') as log_file:
            log_file.write(f"{datetime.now()} - {user_choice}\n")
```

data_saver.py

```
import os
import json
import csv
from src.const.path_constants import FOLDER_PATH_RESULT_LAB7

class DataSaver:
    """Handles the saving of data in different formats."""
    @staticmethod
    def save_data(data, format_choice, filename):
        """Saves the data to a specified format and filename."""
        results_dir = FOLDER_PATH_RESULT_LAB7
        os.makedirs(results_dir, exist_ok=True)

        file_path = os.path.join(results_dir, filename)

        if format_choice == 'json':
            DataSaver.save_json(data, file_path)
        elif format_choice == 'csv':
            DataSaver.save_csv(data, file_path)
        elif format_choice == 'txt':
            DataSaver.save_txt(data, file_path)
        else:
            print("Unsupported format type.")

    @staticmethod
    def save_json(data, file_path):
        """Saves data in JSON format."""
        with open(file_path, 'w') as json_file:
```

```
            json.dump(data, json_file, indent=4)
        print(f"Data saved in JSON format to {file_path}")

    @staticmethod
    def save_csv(data, file_path):
        """Saves data in CSV format."""
        with open(file_path, 'w', newline='') as csv_file:
            writer = csv.writer(csv_file)
            if data:
                writer.writerow(data[0].keys())
                for item in data:
                    writer.writerow(item.values())
        print(f"Data saved in CSV format to {file_path}")

    @staticmethod
    def save_txt(data, file_path):
        """Saves data in plain text format."""
        with open(file_path, 'w') as txt_file:
            for item in data:
                txt_file.write(str(item) + "\n")
        print(f"Data saved in plain text format to {file_path}")
```

ui.py

```
"""
User Interface module for displaying and managing user interactions.

Provides functions to display options, get user input, and format data display.
"""
```

```
from tabulate import tabulate
from src.const.const import (
    DATA_TYPES, USER_CHOICE_PROMPT,
INVALID_CHOICE_MESSAGE, FORMAT_CHOICE_PROMPT,
    DATA_CHOICE_MESSAGE, SAVE_CHOICE_PROMPT,
FILE_FORMAT_PROMPT, FILE_NAME_PROMPT
)
from lib.user_input import UserInput



class UserInterface:
    """Class to handle user interactions and data display options."""

    # (rest of your methods)

    @staticmethod
    def display_options():
        """Displays the available data type options to the user."""
        print(f"\n{DATA_CHOICE_MESSAGE}")
        for data_type in DATA_TYPES.keys():
            print(f"\t {data_type}. {DATA_TYPES[data_type]}")

    @staticmethod
    def get_user_choice():
        """
        Prompts the user to choose a data type.
        Returns:
```

```
            str: Selected data type or None if exiting.
        """
        UserInterface.display_options()
        choice = UserInput.get_input(USER_CHOICE_PROMPT,
DATA_TYPES.keys())
        if choice == '0':
            print("BYEEEEEEEEEEEEEEEEEEEEEEEEEEEE")
            return None
        if choice in DATA_TYPES:
            return DATA_TYPES[choice]
        return None

    @staticmethod
    def display_list(data):
        """
        Displays data in a list format.
        Args:
            data (list): List of data items to display.
        """
        if not data:
            print("...empty :(")
            return
        for index, item in enumerate(data, start=1):
            print(f"{index}. {item}")

    @staticmethod
    def display_table(data):
        """
```

```
    Displays data in a table format using 'tabulate'.
    Args:
        data (list of dicts): List of data dictionaries to display.
    """
    headers = data[0].keys() if data else []
    table_data = [list(item.values()) for item in data]
    print(tabulate(table_data, headers, tablefmt="grid"))

@staticmethod
def get_display_format():
    """
    Prompts the user to choose a display format.
    Returns:
        str: The chosen display format (e.g., 'list' or 'table').
    """
    format_choice = UserInput.get_input(FORMAT_CHOICE_PROMPT)
    return format_choice.lower()

@staticmethod
def display_data(data):
    """
    Displays data based on the user-chosen format.
    Args:
        data (list): Data to display, either in list or table format.
    """
    format_choice = UserInterface.get_display_format()
    display_methods = {
        "list": UserInterface.display_list,
```

```
            "table": UserInterface.display_table,
        }
        display_method = display_methods.get(format_choice)

        if display_method:
            display_method(data)
        else:
            print(INVALID_CHOICE_MESSAGE)

    @staticmethod
    def ask_save_data():
        """
        Asks the user whether to save the data.
        Returns:
            bool: True if the user agrees to save, False otherwise.
        """
        response = UserInput.get_input(SAVE_CHOICE_PROMPT).lower()
        return response in ['yes', 'y']

    @staticmethod
    def get_save_options():
        """
        Prompts the user for save format and filename.
        Returns:
            tuple: Format choice (e.g., 'json') and filename.
        """
        format_choice = UserInput.get_input(FILE_FORMAT_PROMPT, ["json",
"csv", "txt"])
```

```
        filename = UserInput.get_input(FILE_NAME_PROMPT)
        return format_choice, filename
```

api_manager.py

```
"""
This module manages the creation of API clients for different data types
(posts, users, and comments) and provides a method to format data for output.
"""

from src.classes.lab7.comments_api import CommentsAPI
from src.classes.lab7.posts_api import PostsAPI
from src.classes.lab7.users_api import UsersAPI

class APIManager:
    """Manages the creation of API clients based on the data type."""
    client_classes = {
        'posts': PostsAPI,
        'users': UsersAPI,
        'comments': CommentsAPI
    }

    @classmethod
    def create_client(cls, data_type):
        """Creates and returns an API client for the specified data type."""
        try:
            return cls.client_classes[data_type]()
        except KeyError as exc:
            raise ValueError("! unknown type of data") from exc
```

```
    @staticmethod
    def format_data(data):
        """Formats data for output."""
        return "\n".join(map(str, data)) if isinstance(data, list) else str(data)
```

data_controller.py

```
from src.classes.lab7.api_manager import APIManager
from src.ui.lab7.ui import UserInterface

from lib import DataSaver, HistoryLogger

class DataController:
    """Controller that manages user interaction, data retrieval, and saving."""
    def __init__(self):
        self.ui = UserInterface()
        self.api_manager = APIManager()
        self.history_logger = HistoryLogger()

    def run(self):
        """Starts the application and processes user choices."""
        try:
            while True:
                user_choice = self.ui.get_user_choice()
                if user_choice is None:
                    break

                client = self.api_manager.create_client(user_choice)
```

```
            data = client.get_data()

            if data is not None:
                self.ui.display_data(data)
            else:
                print("Failed to retrieve data.")

            if self.ui.ask_save_data():
                format_choice, filename = self.ui.get_save_options()
                DataSaver.save_data(data, format_choice, filename)

            self.history_logger.log(user_choice)
        except Exception as e:
            print(f"An unexpected error occurred: {e}")
```

comments.api

```
from lib import ApiService

class CommentsAPI:
    """API client for fetching comment data."""
    def get_data(self):
        """Fetches comments data from the API."""
        client = ApiService()
        return client.get_data('comments')
```

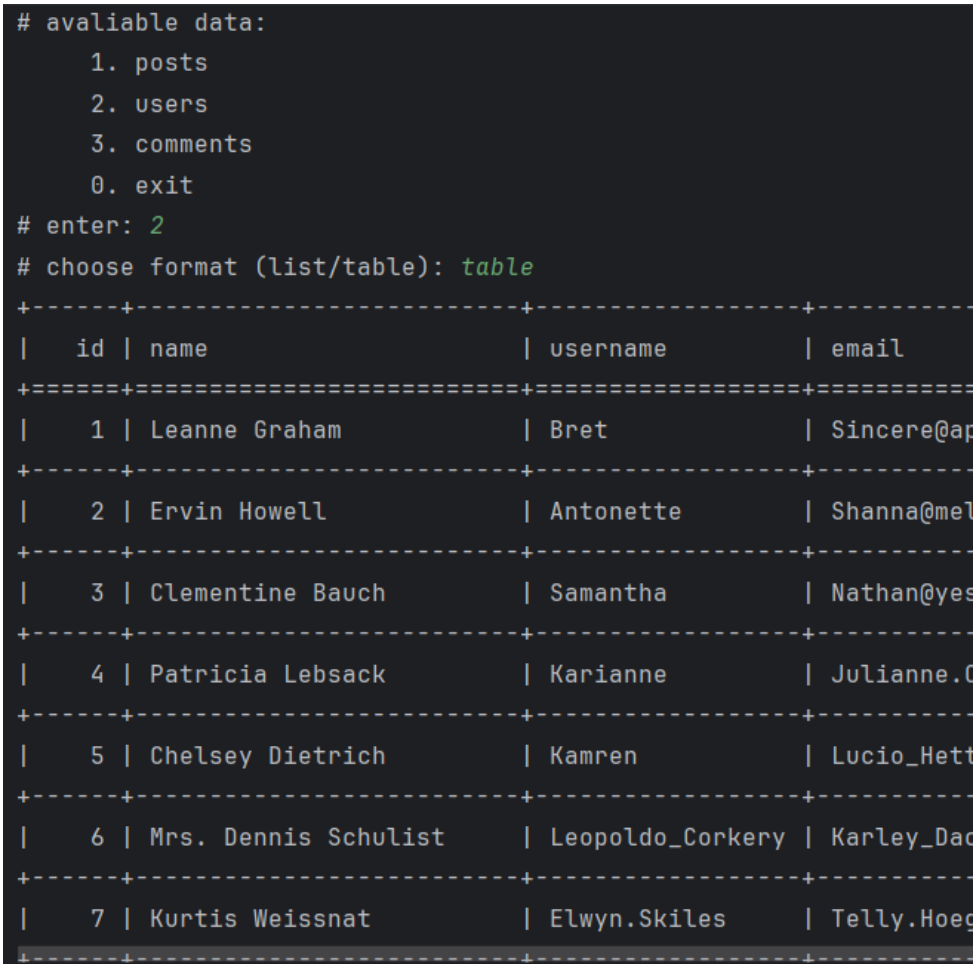

Рис.1. Результат програми.

Висновок: на цій лабораторній роботі я вивчила створення консольного об’єктно - орієнтованого додатка з використанням API та патернів проектування