Міністерство освіти і науки України

Національний університет «Львівська політехніка»



Лабораторна робота №7

з навчальної дисципліни

«Спеціалізовані мови програмування»

на тему

«Робота з API та веб-сервісами»

*Інститут комп’ютерних наук*

*та інформаційних технологій*

*Кафедри інформаційних*

*систем та мереж*

***Виконав:***

*Студент групи РІ-31*

*Олександр ЛУЧКЕВИЧ*

***Прийняла:***

*викладач*

*Cергій ЩЕРБАК*

*Львів – 2024*

**Мета роботи:** 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.

**Код програми:**

Файл APIClient.py:

import requests

from requests.exceptions import RequestException

class APIClient:

"""Client for interacting with the JSONPlaceholder API."""

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

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

"""Initialize the API client."""

pass

def \_get(self, endpoint: str):

"""Helper method to send a GET request to the API."""

try:

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

response.raise\_for\_status()

return response.json()

except RequestException as e:

print(f"Error while fetching data from {endpoint}: {e}")

return None

def get\_data(self, endpoint: str):

"""Get data from a specified endpoint."""

return self.\_get(endpoint)

def get\_data\_by\_id(self, endpoint: str, item\_id: int):

"""Get data from a specified endpoint by ID."""

return self.\_get(f"{endpoint}/{item\_id}")

def get\_all\_users(self):

"""Get all users."""

return self.get\_data("users")

def get\_user\_by\_id(self, user\_id: int):

"""Get a user by their ID."""

return self.get\_data\_by\_id("users", user\_id)

def get\_all\_comments(self):

"""Get all comments."""

return self.get\_data("comments")

def get\_comment\_by\_id(self, comment\_id: int):

"""Get a comment by its ID."""

return self.get\_data\_by\_id("comments", comment\_id)

Файл DataDisplay.py:

from typing import List, Dict

from tabulate import tabulate

from colorama import Fore, Style

class DataDisplay:

"""Class for displaying different types of data in tabular format."""

@staticmethod

def show\_table(data: List[Dict]) -> None:

"""Displays a table of posts."""

table = [[item.get('id', 'N/A'), item.get('title', 'N/A')] for item in data]

print(Fore.GREEN + Style.BRIGHT + "Posts" + Style.RESET\_ALL)

print(tabulate(table, headers=["ID", "Title"], tablefmt="grid"))

@staticmethod

def show\_users(users: List[Dict]) -> None:

"""Displays a table of users."""

table = []

for user in users:

table.append([

user.get("id", 'N/A'),

user.get("name", 'N/A'),

user.get("username", 'N/A'),

user.get("email", 'N/A'),

f"{user.get('address', {}).get('street', 'N/A')}, {user.get('address', {}).get('suite', 'N/A')}, "

f"{user.get('address', {}).get('city', 'N/A')}, {user.get('address', {}).get('zipcode', 'N/A')}",

user.get("phone", 'N/A'),

user.get("website", 'N/A'),

user.get("company", {}).get("name", 'N/A')

])

print(Fore.CYAN + Style.BRIGHT + "Users" + Style.RESET\_ALL)

print(tabulate(table, headers=["ID", "Name", "Username", "Email", "Address", "Phone", "Website", "Company"], tablefmt="grid"))

@staticmethod

def show\_comments(comments: List[Dict]) -> None:

"""Displays a table of comments."""

table = []

for comment in comments:

# Fallback to 'body' if 'comment' key does not exist

comment\_text = comment.get('comment') or comment.get('body', 'N/A')

table.append([

comment.get('id', 'N/A'),

comment.get('postId', 'N/A'),

comment.get('userId', 'N/A'),

comment\_text

])

print(Fore.YELLOW + Style.BRIGHT + "Comments" + Style.RESET\_ALL)

print(tabulate(table, headers=["ID", "Post ID", "User ID", "Comment"], tablefmt="grid"))

Файл ErrorHandler.py:

class ErrorHandler:

@staticmethod

def handle\_error(e):

print(f"Помилка: {str(e)}")

Файл HistoryLogger.py:

from typing import List, Dict

class HistoryLogger:

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

self.history: List[Dict[str, str]] = []

def log(self, command: str, result: str) -> None:

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

def show\_history(self) -> None:

for record in self.history:

print(f"Команда: {record['command']}")

print(f"Результат: {record['result']}")

Файл InputParser.py:

import re

from typing import Optional

class InputParser:

@staticmethod

def parse\_date(input\_str: str) -> str:

if re.match(r'^\d{4}-\d{2}-\d{2}$', input\_str):

return "Дата розпізнана!"

return "Невірний формат дати"

@staticmethod

def parse\_phone(input\_str: str) -> str:

if re.match(r'^\+?\d[\d -]{7,12}\d$', input\_str):

return "Телефонний номер розпізнано!"

return "Невірний формат телефону"

Файл main.py:

from typing import List, Dict

from tabulate import tabulate

from colorama import Fore, Style

class DataDisplay:

"""Class for displaying different types of data in tabular format."""

@staticmethod

def show\_table(data: List[Dict]) -> None:

"""Displays a table of posts."""

table = [[item.get('id', 'N/A'), item.get('title', 'N/A')] for item in data]

print(Fore.GREEN + Style.BRIGHT + "Posts" + Style.RESET\_ALL)

print(tabulate(table, headers=["ID", "Title"], tablefmt="grid"))

@staticmethod

def show\_users(users: List[Dict]) -> None:

"""Displays a table of users."""

table = []

for user in users:

table.append([

user.get("id", 'N/A'),

user.get("name", 'N/A'),

user.get("username", 'N/A'),

user.get("email", 'N/A'),

f"{user.get('address', {}).get('street', 'N/A')}, {user.get('address', {}).get('suite', 'N/A')}, "

f"{user.get('address', {}).get('city', 'N/A')}, {user.get('address', {}).get('zipcode', 'N/A')}",

user.get("phone", 'N/A'),

user.get("website", 'N/A'),

user.get("company", {}).get("name", 'N/A')

])

print(Fore.CYAN + Style.BRIGHT + "Users" + Style.RESET\_ALL)

print(tabulate(table, headers=["ID", "Name", "Username", "Email", "Address", "Phone", "Website", "Company"], tablefmt="grid"))

@staticmethod

def show\_comments(comments: List[Dict]) -> None:

"""Displays a table of comments."""

table = []

for comment in comments:

# Fallback to 'body' if 'comment' key does not exist

comment\_text = comment.get('comment') or comment.get('body', 'N/A')

table.append([

comment.get('id', 'N/A'),

comment.get('postId', 'N/A'),

comment.get('userId', 'N/A'),

comment\_text

])

print(Fore.YELLOW + Style.BRIGHT + "Comments" + Style.RESET\_ALL)

print(tabulate(table, headers=["ID", "Post ID", "User ID", "Comment"], tablefmt="grid"))

Файл lab\_calculator/functions:

from DAL.DataSaver import DataSaver

from Lab\_7.APIClient import APIClient

from Lab\_7.DataDisplay import DataDisplay

from Lab\_7.ErrorHandler import ErrorHandler

from Lab\_7.HistoryLogger import HistoryLogger

from UI.MenuItem import MenuItem

from UI.MenuBuilder import MenuBuilder

def show\_all\_posts(client, display, logger):

"""Fetch and display all posts."""

try:

posts = client.get\_data("posts")

display.show\_table(posts)

logger.log("показати всі пости", posts)

except Exception as error:

ErrorHandler.handle\_error(error)

def show\_post\_by\_id(client, display, logger):

"""Fetch and display a post by its ID."""

try:

post\_id = input("Введіть ID поста: ")

post = client.get\_data\_by\_id("posts", post\_id)

display.show\_table([post])

logger.log(f"показати пост з ID {post\_id}", post)

except Exception as error:

ErrorHandler.handle\_error(error)

def save\_data(client, saver):

"""Save data to a specified file format."""

format\_choice = input("Введіть формат збереження (json/csv/txt): ").strip().lower()

filename = input("Введіть ім'я файлу: ").strip()

if format\_choice in ['json', 'csv', 'txt']:

try:

saver\_method = {

'json': saver.save\_as\_json,

'csv': saver.save\_as\_csv,

'txt': saver.save\_as\_txt

}[format\_choice]

posts = client.get\_data("posts") # Fetch posts before saving

saver\_method(posts, filename)

print(f"Дані збережено у {filename}.{format\_choice}")

except Exception as error:

ErrorHandler.handle\_error(error)

else:

print("Невідомий формат.")

def show\_all\_users(client, display, logger):

"""Fetch and display all users."""

try:

users = client.get\_all\_users()

display.show\_users(users)

logger.log("показати всіх користувачів", users)

except Exception as error:

ErrorHandler.handle\_error(error)

def show\_user\_by\_id(client, display, logger):

"""Fetch and display a user by its ID."""

try:

user\_id = input("Введіть ID користувача: ")

user = client.get\_user\_by\_id(user\_id)

display.show\_users([user])

logger.log(f"показати користувача з ID {user\_id}", user)

except Exception as error:

ErrorHandler.handle\_error(error)

def show\_all\_comments(client, display, logger):

"""Fetch and display all comments."""

try:

comments = client.get\_all\_comments()

display.show\_comments(comments)

logger.log("показати всі коментарі", comments)

except Exception as error:

ErrorHandler.handle\_error(error)

def show\_comment\_by\_id(client, display, logger):

"""Fetch and display a comment by its ID."""

try:

comment\_id = input("Введіть ID коментаря: ")

comment = client.get\_comment\_by\_id(comment\_id)

display.show\_comments([comment])

logger.log(f"показати коментар з ID {comment\_id}", comment)

except Exception as error:

ErrorHandler.handle\_error(error)

def show\_history(logger):

"""Show the history of logs."""

logger.show\_history()

def exit\_program():

"""Exit the program."""

print("Вихід...")

return

def run():

"""Main function to run the program."""

client = APIClient()

display = DataDisplay()

saver = DataSaver()

logger = HistoryLogger()

menu\_items = [

MenuItem("1", "Показати всі пости", lambda: show\_all\_posts(client, display, logger)),

MenuItem("2", "Показати пост за ID", lambda: show\_post\_by\_id(client, display, logger)),

MenuItem("3", "Зберегти дані", lambda: save\_data(client, saver)),

MenuItem("4", "Показати всіх користувачів", lambda: show\_all\_users(client, display, logger)),

MenuItem("5", "Показати користувача за ID", lambda: show\_user\_by\_id(client, display, logger)),

MenuItem("6", "Показати всі коментарі", lambda: show\_all\_comments(client, display, logger)),

MenuItem("7", "Показати коментар за ID", lambda: show\_comment\_by\_id(client, display, logger)),

MenuItem("8", "Показати історію", lambda: show\_history(logger)),

MenuItem("0", "Вийти", exit\_program),

]

menu = MenuBuilder(menu\_items)

while True:

menu.initialize()

Файл TestAPIClient.py:

import unittest

from unittest.mock import patch

from APIClient import APIClient

class TestAPIClient(unittest.TestCase):

def setUp(self):

"""Sets up the APIClient instance for testing."""

self.client = APIClient()

@patch('requests.get')

def test\_get\_data\_success(self, mock\_get):

"""Tests the success of the get\_data method."""

mock\_get.return\_value.status\_code = 200

mock\_get.return\_value.json.return\_value = [{"id": 1, "title": "Test Post"}]

result = self.client.get\_data("posts")

self.assertIsNotNone(result)

self.assertEqual(result[0]["id"], 1)

self.assertEqual(result[0]["title"], "Test Post")

@patch('requests.get')

def test\_get\_data\_by\_id\_success(self, mock\_get):

"""Tests the success of the get\_data\_by\_id method."""

mock\_get.return\_value.status\_code = 200

mock\_get.return\_value.json.return\_value = {"id": 1, "title": "Test Post"}

result = self.client.get\_data\_by\_id("posts", 1)

self.assertIsNotNone(result)

self.assertEqual(result["id"], 1)

self.assertEqual(result["title"], "Test Post")

@patch('requests.get')

def test\_get\_all\_users\_success(self, mock\_get):

"""Tests the success of the get\_all\_users method."""

mock\_get.return\_value.status\_code = 200

mock\_get.return\_value.json.return\_value = [{"id": 1, "name": "Leanne Graham"}]

result = self.client.get\_all\_users()

self.assertIsNotNone(result)

self.assertEqual(result[0]["id"], 1)

self.assertEqual(result[0]["name"], "Leanne Graham")

@patch('requests.get')

def test\_get\_user\_by\_id\_success(self, mock\_get):

"""Tests the success of the get\_user\_by\_id method."""

mock\_get.return\_value.status\_code = 200

mock\_get.return\_value.json.return\_value = {"id": 1, "name": "Leanne Graham"}

result = self.client.get\_user\_by\_id(1)

self.assertIsNotNone(result)

self.assertEqual(result["id"], 1)

self.assertEqual(result["name"], "Leanne Graham")

@patch('requests.get')

def test\_get\_all\_comments\_success(self, mock\_get):

"""Tests the success of the get\_all\_comments method."""

mock\_get.return\_value.status\_code = 200

mock\_get.return\_value.json.return\_value = [{"id": 1, "postId": 1, "userId": 1, "comment": "Sample comment"}]

result = self.client.get\_all\_comments()

self.assertIsNotNone(result)

self.assertEqual(result[0]["id"], 1)

self.assertEqual(result[0]["postId"], 1)

self.assertEqual(result[0]["comment"], "Sample comment")

@patch('requests.get')

def test\_get\_comment\_by\_id\_success(self, mock\_get):

"""Tests the success of the get\_comment\_by\_id method."""

mock\_get.return\_value.status\_code = 200

mock\_get.return\_value.json.return\_value = {"id": 1, "postId": 1, "userId": 1, "comment": "Sample comment"}

result = self.client.get\_comment\_by\_id(1)

self.assertIsNotNone(result)

self.assertEqual(result["id"], 1)

self.assertEqual(result["postId"], 1)

self.assertEqual(result["comment"], "Sample comment")

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

unittest.main()

**Висновок.**

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