МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №7

з курсу

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

Виконав студент

групи ІТ-21сп

**Петровський Т.О.**

Прийняв

**Щербак С.С.**

Львів - 2023

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

**План роботи**

План роботи

Завдання 1: Вибір провайдера API

Виберіть надійний API, який надає через HTTP необхідні дані для віддаленого зберігання, вивантаження або реалізуйте свій. Для прикладу це може бути jsonplaceholder.org

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 виконаних завдань представлено нижче.

import logging

import requests

from requests import Response

from Lab7.http\_exception import HttpException

from Lab7.http\_method\_type import HttpMethod

# Represents HTTP client

class HttpClient:

# Http Client constructor

def \_\_init\_\_(self, base\_url: str):

logging.info('Initialize http client')

self.\_\_base\_url = base\_url

# Makes get request

def get(self, url\_params: str = None) -> Response:

return self.\_\_send\_request(method\_type=HttpMethod.GET, url\_params=url\_params)

# Makes post request

def post(self, url\_params: str = None, data=None) -> Response:

return self.\_\_send\_request(method\_type=HttpMethod.POST, url\_params=url\_params, data=data)

# Makes put request

def put(self, url\_params: str = None, data=None) -> Response:

return self.\_\_send\_request(method\_type=HttpMethod.PUT, url\_params=url\_params, data=data)

# Makes patch request

def patch(self, url\_params: str = None, data=None) -> Response:

return self.\_\_send\_request(method\_type=HttpMethod.PATCH, url\_params=url\_params, data=data)

# Makes delete request

def delete(self, url\_params: str = None, data=None) -> Response:

return self.\_\_send\_request(method\_type=HttpMethod.DELETE, url\_params=url\_params, data=data)

# Handles all requests

def \_\_send\_request(self, method\_type: HttpMethod, url\_params: str = None, data=None) -> Response:

logging.info(f'Making {method\_type} request with ({url\_params}) params')

request\_url = self.\_\_base\_url + (url\_params or "")

request\_methods = {

HttpMethod.GET: requests.get,

HttpMethod.POST: requests.post,

HttpMethod.PUT: requests.put,

HttpMethod.PATCH: requests.patch,

HttpMethod.DELETE: requests.delete,

}

if method\_type not in request\_methods:

raise ValueError(f"Invalid HttpMethod: {method\_type}")

request\_method = request\_methods[method\_type]

response = request\_method(request\_url, data=data)

if response.ok:

return response.json()

raise HttpException(f"Failed to make HTTP request\nHTTP method: {method\_type}, Url parameters:{url\_params}")

# Represents Http exception

import logging

class HttpException(Exception):

# Http exception constructor

def \_\_init\_\_(self, message, data=None):

logging.info('Initialize http exception')

super().\_\_init\_\_(message)

self.data = data

# Represents http method type

class HttpMethod(Enum):

GET = 1,

POST = 2,

PUT = 3,

PATCH = 4,

DELETE = 5

# Represents users repository

class UserRepository:

BASE\_URL = 'https://jsonplaceholder.org'

# User repo constructor

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

logging.info('Initialize user constructor')

self.\_\_http\_client = HttpClient(UserRepository.BASE\_URL)

# Gets all users

def get\_all\_users(self):

logging.info('Getting all users')

return self.\_\_http\_client.get('/users')

# Gets user by id

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

logging.info('Getting user by id')

return self.\_\_http\_client.get(f'/users/{user\_id}')

# Creates user

def create\_user(self, user):

logging.info('Creating user')

return self.\_\_http\_client.post('/users', data=user)

# Represents user interface

class UsersInterface:

# User interface constructor

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

logging.info('initialize user interface')

self.\_\_user\_repo = UserRepository()

self.\_\_display\_helper = DisplayHelper()

self.\_\_data\_saver = FileHandler()

self.\_\_colors = {

'1': Fore.RED,

'2': Fore.GREEN,

'3': Fore.YELLOW

}

# Runs user interface

def run(self):

logging.info('Running user interface')

while True:

option = UserInputHelper.get\_limited\_user\_input(

"Enter option:\n1-Print users\n2-Save users in file\n0-Exit\n",

['1', '2', '0'])

if option == '1':

self.\_\_display\_users()

elif option == '2':

self.\_\_save\_to\_file()

else:

break

# Saves data to file

def \_\_save\_to\_file(self):

logging.info('Saving data to file')

data = self.\_\_user\_repo.get\_all\_users()

file\_option = UserInputHelper.get\_limited\_user\_input("Enter file type to save(json, csv, txt)\n",

['json', 'csv', 'txt'])

file\_name = UserInputHelper.get\_user\_input("Enter file name\n")

if file\_option == 'json':

self.\_\_data\_saver.save\_to\_json(file\_name, data)

elif file\_option == 'csv':

self.\_\_data\_saver.save\_to\_csv(file\_name, data)

elif file\_option == 'txt':

self.\_\_data\_saver.save\_to\_csv(file\_name, data)

# Displays users

def \_\_display\_users(self):

logging.info('Displaying users')

color = self.\_\_get\_color()

input\_option = UserInputHelper.get\_limited\_user\_input("How do you want to display user:\n1 - List\n2 - Table\n",

['1', '2'])

users = self.\_\_user\_repo.get\_all\_users()

if input\_option == '1':

self.\_\_display\_helper.display\_list(color, users)

elif input\_option == '2':

headers = [Fore.RESET + color + "First name",

Fore.RESET + color + "Last name",

Fore.RESET + color + "Email"]

rows = [(color + user['firstname'], color + user['lastname'], color + user['email'])

for user in users]

self.\_\_display\_helper.display\_table(headers, rows)

# Gets color

def \_\_get\_color(self):

logging.info('Getting color')

color = UserInputHelper.get\_limited\_user\_input("Choose color:\n1-Red\n2-Green\n3-Yellow\n",

list(self.\_\_colors.keys()))

return self.\_\_colors[color]

class TestUserRepository(unittest.TestCase):

def setUp(self):

self.\_\_user\_repository = UserRepository()

self.\_\_file\_handler = FileHandler()

@patch('Lab7.HttpClient')

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

expected\_result = self.\_\_file\_handler.read\_json\_from\_file('test\_data/users.json')

mock\_http\_client.return\_value.get.return\_value = expected\_result

result = self.\_\_user\_repository.get\_all\_users()

mock\_http\_client.assert\_called\_with(UserRepository.BASE\_URL)

mock\_http\_client.return\_value.get.assert\_called\_with('/users')

self.assertEqual(result, expected\_result)

@patch('Lab7.HttpClient')

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

user\_id = 1

expected\_result = self.\_\_file\_handler.read\_json\_from\_file('test\_data/user.json')

mock\_http\_client.return\_value.get.return\_value = expected\_result

result = self.\_\_user\_repository.get\_user\_by\_id(user\_id)

mock\_http\_client.assert\_called\_with(UserRepository.BASE\_URL)

mock\_http\_client.return\_value.get.assert\_called\_with(f'/users/{user\_id}')

self.assertEqual(result, expected\_result)

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

unittest.main()

**Висновок:** Виконавши ці завдання, я створив проект, який надасть мені цінний досвід роботи з API, дизайну користувацького інтерфейсу, валідації введення, обробки помилок та тестування