МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ КОМП’ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ



Лабораторна робота №7

з дисципліни “Спеціалізовані мови програмування”

Виконала:

студентка групи ІТ-31

Анастасія КАМІНОВСЬКА

Прийняв:

доцент кафедри ІСМ

Сергій ЩЕРБАК

**Тема роботи:** Робота з API та веб-сервісами

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

Хід 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:

Bot.py:

""" Bot main"""

import os

import json

from datetime import datetime

from prettytable import PrettyTable

from telegram import Update

from telegram.constants import ParseMode

from telegram.ext import Application,CommandHandler,MessageHandler,filters

from telegram.ext import ContextTypes,ConversationHandler

from Data.data import user\_data, AGE, HOBBIES, FAVORITE\_COLOR , EMAIL

from .validator import Validator

DATA\_FOLDER = 'Data'

validator =  Validator()

#bottest = TestBot()

def get\_user\_history\_filename(chat\_id):

    """get user history """

    return os.path.join(DATA\_FOLDER, f'history\_{chat\_id}.json')

def get\_users\_filename():

    """ get user info"""

    return os.path.join(DATA\_FOLDER, 'users.json')

def save\_history(chat\_id, message, answer):

    """ save user history"""

    filename = get\_user\_history\_filename(chat\_id)

    try:

        with open(filename, 'r+',encoding = 'UTF-8') as file:

            history\_file = json.load(file)

    except (FileNotFoundError, json.JSONDecodeError):

        history\_file = []

    history\_file.append({

        'message': message,

        'answer': answer,

        'timestamp': datetime.now().isoformat()

    })

    os.makedirs(DATA\_FOLDER, exist\_ok=True)

    with open(filename, 'w',encoding = 'UTF-8') as file:

        json.dump(history\_file, file, indent=4)

# Команда для перегляду історії користувача

async def history(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> None:

    """ Load user history"""

    chat\_id = update.effective\_chat.id

    filename = get\_user\_history\_filename(chat\_id)

    try:

        with open(filename, 'r',encoding = 'UTF-8') as file:

            history\_file = json.load(file)

            message = "Ваша історія запитів:\n\n"

            os.makedirs(DATA\_FOLDER, exist\_ok=True)

            for item in history\_file:

                message += f"{item['timestamp']}: Запит - {item['message']}\nВідповідь - {item['answer']}\n\n"

            await update.message.reply\_text(message)

    except FileNotFoundError:

        await update.message.reply\_text("Історія запитів відсутня.")

# Команда для початку введення завдань

async def start\_task\_input(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    """ Start inpunt option"""

    chat\_id = update.effective\_chat.id

    username = update.effective\_user.username

    first\_name = update.effective\_user.first\_name

    last\_name = update.effective\_user.last\_name

    user\_info = {

        'username': username,

        'first\_name': first\_name,

        'last\_name': last\_name

    }

    user\_data[chat\_id] = user\_info

    # Ask for the first piece of information

    await update.message.reply\_text("Hello! Let's get started. Please tell me your age.")

    return AGE

# Handlers for each state in the conversation

async def age\_input(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    """ Age input"""

    age = update.message.text

    chat\_id = update.effective\_chat.id

    if validator.validate\_age(age):

        # Якщо вік валідний, запитати улюблений колір

        user\_data[chat\_id]['age'] = age

        save\_history(chat\_id, f"Age entered: {age}", "Age is valid and saved.")

        await update.message.reply\_text("Вік прийнято. Будь ласка, введіть ваш улюблений колір:")

        return FAVORITE\_COLOR

    else:

        # Якщо вік невалідний, попросити ввести знову

        save\_history(chat\_id, f"Age entered: {age}", "Invalid age format.")

        await update.message.reply\_text("Невірний вік. Введіть вік ще раз:")

        return AGE

async def favorite\_color\_input(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    """ color input"""

    chat\_id = update.effective\_chat.id

    color = update.message.text

    if validator.validate\_color(color):

        user\_data[chat\_id]['fav\_color'] = color

        save\_history(chat\_id, f"Color entered: {color}", "Color is valid and saved.")

        await update.message.reply\_text("Interesting! What are your hobbies?")

        return HOBBIES

    else:

        save\_history(chat\_id, f"Color entered: {color}", "Invalid color format.")

        await update.message.reply\_text("Невірна палітра кольорів. Введіть ще раз:")

        return FAVORITE\_COLOR

async def hobbie\_input(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    """ hobbie input"""

    chat\_id = update.effective\_chat.id

    hobbies = update.message.text

    # Split the hobbies by comma and remove leading/trailing whitespace

    hobbies\_list = [[hobbie.strip()] for hobbie in hobbies.split(',')]

    if all(validator.validate\_hobbie(hobbie) for hobbie in hobbies\_list):

        user\_data[chat\_id]['hobbies'] = hobbies\_list  # Save hobbies as a list

        save\_history(chat\_id, f"hobbies entered: {hobbies}", "hobbies is valid and saved.")

        await update.message.reply\_text("I need to try that! And last one - what is your email?")

        return EMAIL

    else:

        save\_history(chat\_id, f"Hobbies entered: {hobbies}", "Invalid hobbies format.")

        await update.message.reply\_text("There might be a mistake in your hobbies. Please enter them again separated by commas.")

        return HOBBIES

async def email\_input(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    """ email input"""

    chat\_id = update.effective\_chat.id

    email = update.message.text

    if validator.validate\_email(email):

        user\_data[chat\_id]['email'] = email

        save\_history(chat\_id, f"Email entered: {email}", "email is valid and saved.")

        save\_user\_data(user\_data[chat\_id])

        await update.message.reply\_text("Well done,now type /print to see all information!")

        return ConversationHandler.END

    else:

        save\_history(chat\_id, f"Email entered: {email}", "Invalid email format.")

        await update.message.reply\_text("Ви допустили помилку! Спробуйте ще раз")

        return EMAIL

# Зберегти введені дані користувача

def save\_user\_data(user\_data):

    """ save user data"""

    try:

        # Try to open the file and read existing data

        with open(get\_users\_filename(), 'r+',encoding = 'UTF-8') as file:

            os.makedirs(DATA\_FOLDER, exist\_ok=True)

            data = json.load(file)

            # Append new user data

            data.append(user\_data)

            # Move file pointer to the beginning for overwriting

            file.seek(0)

            # Write updated data

            json.dump(data, file, indent=4)

    except (FileNotFoundError, json.JSONDecodeError):

        # If file doesn't exist or is empty, create it and write the data

        with open(get\_users\_filename(), 'w',encoding = 'UTF-8') as file:

            json.dump([user\_data], file, indent=4)

async def print\_user\_data(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> None:

    try:

        with open(get\_users\_filename(), 'r',encoding = 'UTF-8') as file:

            os.makedirs(DATA\_FOLDER, exist\_ok=True)

            users = json.load(file)

            message = "<b>User Information:</b>\n<pre>"

            message +="Username | First Name | Last Name | Age | Favorite Color | Hobbies | Email\n"

            message += "-" \* 70 + "\n"  # Розділова лінія

            for user in users:

                hobbies\_str = ', '.join(hobby for sublist in user['hobbies'] for hobby in sublist)

                message += f"{user['username']} | {user['first\_name']} | {user['last\_name']} | {user['age']} | {user['fav\_color']} | {hobbies\_str} | {user['email']}\n"

            message += "</pre>"

            await update.message.reply\_text(message, parse\_mode=ParseMode.HTML)

    except FileNotFoundError:

        await update.message.reply\_text("No user information available.")

def display\_data\_in\_console(users):

    """ display data in console"""

    table = PrettyTable()

    table.field\_names=["Username","First Name","Last Name","Age","Favorite Color","Hobbies","Email"]

    for user in users:

        hobbies\_str = ', '.join(hobby for hobby\_list in user['hobbies'] for hobby in hobby\_list)

        table.add\_row([user['username'], user['first\_name'], user['last\_name'],

        user['age'], user['fav\_color'], hobbies\_str, user['email']])

    print(table)

async def print\_user\_data\_console(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> None:

    """ Output user data into console"""

    try:

        with open(get\_users\_filename(), 'r',encoding = 'UTF-8') as file:

            os.makedirs(DATA\_FOLDER, exist\_ok=True)

            users = json.load(file)

            display\_data\_in\_console(users)

    except FileNotFoundError:

        print("No user information available.")

# Command to cancel the conversation

async def cancel(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> int:

    """ cancel inputing values"""

    await update.message.reply\_text('Operation cancelled.')

    return ConversationHandler.END

def main() -> None:

    """ main method"""

    application = Application.builder().token('5655207446:AAFV0v3R8scQtxhuuApOgGCnWkx7Tp1mXAM').build()

    conv\_handler = ConversationHandler(

        entry\_points=[CommandHandler('start', start\_task\_input)],

        states={

            AGE: [MessageHandler(filters.TEXT & ~filters.COMMAND, age\_input)],

            FAVORITE\_COLOR: [MessageHandler(filters.TEXT & ~filters.COMMAND, favorite\_color\_input)],

            HOBBIES: [MessageHandler(filters.TEXT & ~filters.COMMAND, hobbie\_input)],

            EMAIL: [MessageHandler(filters.TEXT & ~filters.COMMAND, email\_input)]

               },

               fallbacks=[CommandHandler('cancel', cancel)],

    )

    application.add\_handler(conv\_handler)

    application.add\_handler(CommandHandler("print", print\_user\_data))

    application.add\_handler(CommandHandler("print\_console", print\_user\_data\_console))

    application.add\_handler(CommandHandler('history', history))

    application.run\_polling()

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

    main()



Рис.1 Результат виконання роботи

**Висновок**: Виконуючи лабораторну роботу я створила проект, який надав цінний досвід роботи з API, дизайну користувацького інтерфейсу, валідації введення, обробки помилок та тестування.