Міністерство освіти і науки України

Національний університет «Львівська політехніка»



**ЗВІТ**

про виконання лабораторної роботи № 7

з курсу:

«Спеціалізовані мови програмування»

**Виконала:**

студентка гр. ІТ-31

Ірина ПЕРХУН

**Прийняв:**

Сергій ЩЕРБАК

Львів 2023

**Тема:** 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.*

**api\_handler.py**

import requests

from tkinter import messagebox

from .message\_box\_wrapper import MessageBoxWrapper

class APIHandler:

    """

    A class for handling API requests.

    """

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

    @staticmethod

    def get\_users():

        """

        Get a list of users from the API.

        Returns:

            list: A list of user data dictionaries.

        """

        try:

            response = requests.get(APIHandler.BASE\_URL, timeout=10)

            response.raise\_for\_status()

            return response.json()

        except requests.exceptions.RequestException as e:

            MessageBoxWrapper.show\_error("API Error", f'Error getting data from API: {e}')

            return []

    @staticmethod

    def get\_user\_by\_id(user\_id):

        """

        Get user data by user ID from the API.

        Args:

            user\_id (int): The ID of the user to retrieve.

        Returns:

            dict: A dictionary containing user data.

        """

        try:

            url = f'{APIHandler.BASE\_URL}/{user\_id}'

            response = requests.get(url, timeout=10)

            response.raise\_for\_status()

            return response.json()

        except requests.exceptions.RequestException as e:

            MessageBoxWrapper.show\_error("API Error", f'Error getting user data from API: {e}')

            return {}

**data\_saver.py**

from .file\_operations import FileOperations

import json

import csv

from datetime import datetime

from tkinter import messagebox

history\_entries = []

class DataSaver:

    """Class for saving data and history entries."""

    @staticmethod

    def save\_data(users\_data, file\_format):

        """

        Save user data in the specified file format.

        Args:

            users\_data (list): List of user data.

            file\_format (str): File format (json, csv, txt).

        Raises:

            ValueError: If an unsupported file format is provided.

            Exception: If an error occurs during saving.

        """

        try:

            file\_format = file\_format.lower()

            if file\_format == "json":

                json\_representation = json.dumps(users\_data, ensure\_ascii=False, indent=4)

                FileOperations.save\_representation(json\_representation, "JSON", file\_format)

            elif file\_format == "csv":

                with open('data.csv', 'w', newline='', encoding='utf-8') as csv\_file:

                    csv\_writer = csv.DictWriter(csv\_file, fieldnames=users\_data[0].keys())

                    csv\_writer.writeheader()

                    csv\_writer.writerows(users\_data)

                with open('data.csv', 'r', encoding='utf-8') as csv\_file:

                    csv\_representation = csv\_file.read()

                FileOperations.save\_representation(csv\_representation, "CSV", file\_format)

            elif file\_format == "txt":

                txt\_representation = "\n".join(str(line) for line in users\_data)

                FileOperations.save\_representation(txt\_representation, "TXT", file\_format)

            else:

                raise ValueError("Unsupported file format.")

        except FileNotFoundError as e:

            messagebox.showerror("File Not Found", f'Error saving data: {e}')

        except Exception as e:

            messagebox.showerror("Error", f'Error saving data: {e}')

    @staticmethod

    def save\_history\_entry(user\_input):

        """

        Save user input as a history entry.

        Args:

            user\_input (str): User input.

        Raises:

            Exception: If an error occurs during saving.

        """

        try:

            timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

            new\_entry = {'timestamp': timestamp, 'user\_input': user\_input}

            # Load existing history from the file

            history\_data = FileOperations.load\_history\_from\_file()

            # Append the new entry to the existing history

            history\_data.append(new\_entry)

            # Save the updated history back to the file

            FileOperations.save\_history\_to\_file(history\_data)

            # Update the in-memory history\_entries list

            history\_entries.append(new\_entry)

        except FileNotFoundError as e:

            messagebox.showerror("File Not Found", f'Error saving history entry: {e}')

        except Exception as e:

            messagebox.showerror("Error", f'Error saving history entry: {e}')

**display\_handler.py**

import tkinter as tk

from tkinter import ttk, messagebox

class DisplayHandler:

    """

    DisplayHandler is a utility class for creating and managing the display

    of tables and lists in Tkinter GUI applications.

    Methods:

    - display\_table(root, data, header\_color, table\_bg\_color):

        Display a table using Tkinter Treeview.

    - display\_list(root, data, item\_color, list\_bg\_color):

        Display a list using Tkinter Listbox.

    - sort\_treeview(tree, col, reverse):

        Sort a Tkinter Treeview by a specific column.

    Usage:

    - Create an instance of DisplayHandler and use its methods to display tables and lists.

    Example:

    ```python

    root = tk.Tk()

    # Example data for table

    table\_data = [{"Name": "John", "Age": 30, "City": "New York"},

                  {"Name": "Alice", "Age": 25, "City": "London"}]

    DisplayHandler.display\_table(root, table\_data, 'blue', 'white')

    # Example data for list

    list\_data = ["Item 1", "Item 2", "Item 3"]

    DisplayHandler.display\_list(root, list\_data, 'black', 'lightgrey')

    root.mainloop()

    ```

    """

    @staticmethod

    def display\_table(root, data, header\_color, table\_bg\_color):

        """

        Display a table using Tkinter Treeview.

        Args:

        - root: Tkinter root window.

        - data (list): List of dictionaries representing table data.

        - header\_color (str): Color of table headers.

        - table\_bg\_color (str): Background color of the table.

        Raises:

        - Exception: If an error occurs during table display.

        """

        try:

            if not data:

                messagebox.showinfo("Result", "No data to display.")

                return

            headers = list(data[0].keys())

            # Create Tkinter window for the table

            table\_window = tk.Toplevel(root)

            table\_window.title("Table Data")

            # Maximize the window to full screen

            table\_window.state('zoomed')

            # Create Treeview for the table

            tree = ttk.Treeview(table\_window, columns=headers, show='headings', height=10,

                                xscrollcommand=lambda x: scrollbar\_x.set(x))

            tree.pack(fill=tk.BOTH, expand=True)

            # Add horizontal scrollbar

            scrollbar\_x = tk.Scrollbar(tree, orient=tk.HORIZONTAL, command=tree.xview)

            scrollbar\_x.pack(side=tk.BOTTOM, fill=tk.X)

            tree.configure(xscrollcommand=scrollbar\_x.set)

            # Set table headers

            for header in headers:

                tree.heading(header, text=header, anchor=tk.CENTER,

                             command=lambda h=header: DisplayHandler.sort\_treeview(tree, h, False))

                tree.column(header, anchor=tk.CENTER)

            # Add a tag for headers

            tree.tag\_configure('header', foreground=header\_color, background=table\_bg\_color)

            # Insert data into the table

            for row in data:

                values = [row[header] for header in headers]

                tree.insert("", tk.END, values=values, tags=('header',))

        except Exception as e:

            messagebox.showerror("Error", f'Error displaying table: {e}')

    @staticmethod

    def display\_list(root, data, item\_color, list\_bg\_color):

        """

        Display a list using Tkinter Listbox.

        Args:

        - root: Tkinter root window.

        - data (list): List of items.

        - item\_color (str): Color of list items.

        - list\_bg\_color (str): Background color of the list.

        Raises:

        - Exception: If an error occurs during list display.

        """

        try:

            if not data:

                messagebox.showinfo("Result", "No data to display.")

                return

            # Create Tkinter window for the list

            list\_window = tk.Toplevel(root)

            list\_window.title("List Data")

            # Maximize the window to full screen

            list\_window.state('zoomed')

            # Create Listbox for the list

            listbox = tk.Listbox(list\_window, bg=list\_bg\_color, fg=item\_color, width=80)

            # Add horizontal scrollbar

            scrollbar\_x = tk.Scrollbar(list\_window, orient=tk.HORIZONTAL, command=listbox.xview)

            scrollbar\_x.pack(side=tk.BOTTOM, fill=tk.X)

            listbox.configure(xscrollcommand=scrollbar\_x.set)

            # Insert data into the list

            for item in data:

                listbox.insert(tk.END, item)

            listbox.pack(fill=tk.BOTH, expand=True)

        except Exception as e:

            messagebox.showerror("Error", f'Error displaying list: {e}')

    @staticmethod

    def sort\_treeview(tree, col, reverse):

        """

        Sort a Tkinter Treeview by a specific column.

        Args:

        - tree: Tkinter Treeview.

        - col: Column to sort.

        - reverse (bool): Whether to sort in reverse order.

        Note:

        - This method modifies the Treeview in-place.

        Raises:

        - None

        """

        data = [(tree.set(child, col), child) for child in tree.get\_children('')]

        data.sort(reverse=reverse)

        for ix, item in enumerate(data):

            tree.move(item[1], '', ix)

        tree.heading(col, command=lambda: DisplayHandler.sort\_treeview(tree, col, not reverse))

**file\_operations.py**

import tkinter as tk

from tkinter import filedialog, messagebox

import json

import os

class FileOperations:

    """

    FileOperations is a utility class for handling file-related operations,

    including saving representations and managing history data.

    Methods:

    - save\_representation(representation, representation\_type, file\_format):

        Save a textual representation to a file.

    - load\_history\_from\_file():

        Load history data from a file.

    - save\_history\_to\_file(history\_data):

        Save history data to a file.

    Usage:

    - Create an instance of FileOperations to perform file-related operations.

    Example:

    ```python

    file\_ops = FileOperations()

    # Save a representation to a file

    representation = "Sample data"

    file\_ops.save\_representation(representation, "Sample", "txt")

    # Load history data from a file

    history\_data = file\_ops.load\_history\_from\_file()

    # Save updated history data to a file

    file\_ops.save\_history\_to\_file(history\_data)

    ```

    """

    @staticmethod

    def save\_representation(representation: str, representation\_type: str, file\_format: str):

        """

        Save a textual representation to a file.

        Args:

        - representation (str): The textual representation to save.

        - representation\_type (str): Type of the representation (e.g., "API data").

        - file\_format (str): Format of the file to save (e.g., "json", "txt").

        Raises:

        - PermissionError: If there is a permission issue while saving the file.

        - FileNotFoundError: If the specified file path is not found.

        """

        try:

            root = tk.Tk()

            root.withdraw()

            # Open a dialog for selecting the file path to save the representation

            file\_path = filedialog.asksaveasfilename(

                defaultextension=f".{file\_format}",

                filetypes=[(f"{file\_format.upper()} files", f"\*.{file\_format}"), ("All files", "\*.\*")],

                title=f"Save {representation\_type} representation as",

                initialdir="data/lab7", initialfile="api\_data"

            )

            if not file\_path:

                return

            # Write the representation to the selected file

            with open(file\_path, "w", encoding='utf-8') as file:

                file.write(representation)

            messagebox.showinfo("Success", f"{representation\_type} representation saved successfully to {file\_path}")

        except (PermissionError, FileNotFoundError) as e:

            # Show an error message if there is an issue saving the file

            messagebox.showerror("Error", str(e))

        finally:

            root.destroy()

    @staticmethod

    def load\_history\_from\_file():

        """

        Load history data from a file.

        Returns:

        - list: List containing history data.

        Raises:

        - json.JSONDecodeError: If there is an issue decoding JSON from the history file.

        """

        # Get the path to the history file

        history\_file\_path = "data/lab7/history.json"

        try:

            if os.path.exists(history\_file\_path):

                with open(history\_file\_path, 'r', encoding='utf-8') as history\_file:

                    history\_data = json.load(history\_file)

            else:

                history\_data = []

        except json.JSONDecodeError as e:

            messagebox.showerror("Error", f'Error decoding JSON from history file: {e}')

            history\_data = []

        return history\_data

    @staticmethod

    def save\_history\_to\_file(history\_data):

        """

        Save history data to a file.

        Args:

        - history\_data (list): List containing history data.

        Raises:

        - Exception: If there is an issue saving the history file.

        """

        # Get the path to the history file

        history\_file\_path = "data/lab7/history.json"

        try:

            with open(history\_file\_path, 'w', encoding='utf-8') as history\_file:

                json.dump(history\_data, history\_file, ensure\_ascii=False, indent=4)

        except Exception as e:

            messagebox.showerror("Error", f'Error saving history file: {e}')

**file\_operations.py**

from .data\_saver import DataSaver

import re

from datetime import datetime

class InputParser:

    """

    InputParser is a utility class for parsing user input and determining its format.

    Methods:

    - parse\_user\_input(user\_input):

        Parse the given user input and determine its format, such as full name, phone number, date, etc.

    Usage:

    - Create an instance of InputParser to parse user input.

    Example:

    ```python

    input\_parser = InputParser()

    # Parse user input

    user\_input = "John Doe"

    result = input\_parser.parse\_user\_input(user\_input)

    print(result)

    ```

    Result:

    ```

    Entered full name: John Doe

    ```

    Note: The result may vary based on the input format.

    """

    @staticmethod

    def parse\_user\_input(user\_input):

        """

        Parse the given user input and determine its format.

        Args:

        - user\_input (str): The user input to be parsed.

        Returns:

        - str: A string indicating the format of the user input.

        Example:

        ```python

        input\_parser = InputParser()

        user\_input = "John Doe"

        result = input\_parser.parse\_user\_input(user\_input)

        print(result)

        ```

        Result:

        ```

        Entered full name: John Doe

        ```

        """

        # Define regular expression patterns for various input formats

        full\_name\_pattern = r'^[A-Z][a-z]+ [A-Z][a-z]+$'

        phone\_pattern = r'^\(\d{3}\) \d{3}-\d{4}$'

        date\_pattern = r'\d{4}-\d{2}-\d{2}$'

        datetime\_pattern = r'^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}$'

        email\_pattern = r'^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'

        website\_pattern = r'^(http|https)://[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

        # Check if input matches any of the patterns

        if re.match(full\_name\_pattern, user\_input):

            result = f"Entered full name: {user\_input}"

        elif re.match(phone\_pattern, user\_input):

            result = f"Entered phone number: {user\_input}"

        elif re.match(date\_pattern, user\_input):

            try:

                parsed\_date = datetime.strptime(user\_input, '%Y-%m-%d').date()

                result = f"Entered date: {parsed\_date}"

            except ValueError:

                result = "Failed to parse the date."

        elif re.match(datetime\_pattern, user\_input):

            try:

                parsed\_datetime = datetime.strptime(user\_input, '%Y-%m-%d %H:%M:%S')

                result = f"Entered date and time: {parsed\_datetime}"

            except ValueError:

                result = "Failed to parse the date and time."

        elif re.match(email\_pattern, user\_input):

            result = f"Entered email: {user\_input}"

        elif re.match(website\_pattern, user\_input):

            result = f"Entered website: {user\_input}"

        else:

            result = "Failed to determine the input format."

        # Save the user input to the history

        DataSaver.save\_history\_entry(user\_input)

        return result

**main.py**

import tkinter as tk

from tkinter import messagebox

import unittest

import time

from .api\_handler import APIHandler

from .data\_saver import DataSaver

from .display\_handler import DisplayHandler

from .input\_parser import InputParser

from .file\_operations import FileOperations

from .tests import TestAPIHandler

from shared.runnable import Runnable

class App(Runnable):

    """

    Initializes the App instance.

    Args:

    - root (Tk): The root Tkinter window.

    """

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Data Display Menu")

        # Expand the window to full screen

        self.root.state('zoomed')

        # Initialize Tkinter variables

        self.init\_tkinter\_vars()

        self.create\_interface()

    def init\_tkinter\_vars(self):

        """

        Initializes Tkinter variables used for GUI customization.

        """

        self.header\_color\_var = tk.StringVar()

        self.header\_color\_var.set("black")

        self.table\_bg\_color\_var = tk.StringVar()

        self.table\_bg\_color\_var.set("white")

        self.item\_color\_var = tk.StringVar()

        self.item\_color\_var.set("black")

        self.list\_bg\_color\_var = tk.StringVar()

        self.list\_bg\_color\_var.set("white")

        self.file\_format\_var = tk.StringVar()

        self.file\_format\_var.set("json")

    def create\_interface(self):

        """

        Creates the main graphical user interface.

        """

        display\_button = tk.Button(self.root, text="Display as Table", command=self.on\_display\_table)

        display\_button.pack()

        # Button for displaying a list

        display\_button = tk.Button(self.root, text="Display as List", command=self.on\_display\_list)

        display\_button.pack()

        # Dropdown menu for table header colors

        header\_color\_label = tk.Label(self.root, text="Select color for table headers:")

        header\_color\_label.pack()

        header\_color\_menu = tk.OptionMenu(self.root, self.header\_color\_var, "black", "red", "green", "blue")

        header\_color\_menu.pack()

        # Dropdown menu for table background colors

        table\_bg\_color\_label = tk.Label(self.root, text="Select table background color:")

        table\_bg\_color\_label.pack()

        table\_bg\_color\_menu = tk.OptionMenu(self.root, self.table\_bg\_color\_var, "white", "lightgray", "lightblue", "lightgreen")

        table\_bg\_color\_menu.pack()

        # Dropdown menu for list item colors

        item\_color\_label = tk.Label(self.root, text="Select color for list items:")

        item\_color\_label.pack()

        item\_color\_menu = tk.OptionMenu(self.root, self.item\_color\_var, "black", "red", "green", "blue")

        item\_color\_menu.pack()

        # Dropdown menu for list background colors

        list\_bg\_color\_label = tk.Label(self.root, text="Select list background color:")

        list\_bg\_color\_label.pack()

        list\_bg\_color\_menu = tk.OptionMenu(self.root, self.list\_bg\_color\_var, "white", "lightgray", "lightblue", "lightgreen")

        list\_bg\_color\_menu.pack()

        user\_input\_label = tk.Label(self.root, text="Enter user input:")

        user\_input\_label.pack()

        self.user\_input\_entry = tk.Entry(self.root)

        self.user\_input\_entry.pack()

        parse\_button = tk.Button(self.root, text="Parse Input", command=self.on\_parse)

        parse\_button.pack()

        self.result\_text = tk.Text(self.root, wrap=tk.WORD, height=10, width=80)

        self.result\_text.pack()

        # Add an entry field for user ID

        self.entry\_user\_id = tk.Entry(self.root)

        self.entry\_user\_id.pack()

        # Add a button to call the get\_user\_by\_id function

        btn\_get\_user\_by\_id = tk.Button(self.root, text="Get User by ID", command=self.get\_user\_by\_id)

        btn\_get\_user\_by\_id.pack()

        # Add a button that calls the show\_history function

        show\_history\_button = tk.Button(self.root, text="Show History", command=self.show\_history)

        show\_history\_button.pack()

        # Dropdown menu for selecting the format

        format\_label = tk.Label(self.root, text="Select format for saving:")

        format\_label.pack()

        file\_format\_menu = tk.OptionMenu(self.root, self.file\_format\_var, "json", "csv", "txt")

        file\_format\_menu.pack()

        # Button to save data

        save\_button = tk.Button(self.root, text="Save", command=self.on\_save)

        save\_button.pack()

        # Button to run tests

        run\_tests\_button = tk.Button(self.root, text="Run Tests", command=self.run\_tests)

        run\_tests\_button.pack()

        exit\_button = tk.Button(self.root, text="Exit", command=self.root.destroy)

        exit\_button.pack()

    def on\_display\_table(self):

        """

        Displays data in a table format using DisplayHandler.

        """

        try:

            users\_data = APIHandler.get\_users()

            if not users\_data:

                raise ValueError("Немає даних для відображення.")

            DisplayHandler.display\_table(self.root, users\_data, self.header\_color\_var.get(), self.table\_bg\_color\_var.get())

        except Exception as e:

            # Обробка помилок, виведення інформативного повідомлення

            self.show\_result(f"Помилка при відображенні даних: {e}")

    def on\_display\_list(self):

        try:

            users\_data = APIHandler.get\_users()

            if not users\_data:

                raise ValueError("Немає даних для відображення.")

            DisplayHandler.display\_list(self.root, users\_data, self.item\_color\_var.get(), self.list\_bg\_color\_var.get())

        except Exception as e:

            # Обробка помилок, виведення інформативного повідомлення

            self.show\_result(f"Помилка при відображенні даних: {e}")

    def on\_parse(self):

        user\_input = self.user\_input\_entry.get()

        result = InputParser.parse\_user\_input(user\_input)

        messagebox.showinfo("Result", result)

    def on\_save(self):

        users\_data = APIHandler.get\_users()

        selected\_format = self.file\_format\_var.get()

        DataSaver.save\_data(users\_data, selected\_format)

    def show\_history(self):

        history\_data = FileOperations.load\_history\_from\_file()

        if not history\_data:

            messagebox.showinfo("Історія", "Історія порожня.")

        else:

            formatted\_history = ""

            for entry in history\_data:

                formatted\_history += f"{entry['timestamp']} - {entry['user\_input']}\n"

            messagebox.showinfo("Історія", formatted\_history)

    # Function to get a user by ID and display the result

    def get\_user\_by\_id(self):

        user\_id = self.entry\_user\_id.get()

        if user\_id.isdigit():

            user\_data = APIHandler.get\_user\_by\_id(int(user\_id))

            self.show\_result(user\_data)

        else:

            messagebox.showerror("Error", "Please enter a valid numeric user ID.")

    def show\_result(self, result):

        self.result\_text.config(state=tk.NORMAL)  # Дозволити редагування текстового поля

        self.result\_text.delete(1.0, tk.END)  # Очистити текстове поле перед виведенням нових результатів

        self.result\_text.insert(tk.END, result)

        self.result\_text.config(state=tk.DISABLED)  # Заборонити редагування текстового поля

    def run\_tests(self):

        # Запуск тестів

        # Збереження часу початку тестування

        start\_time = time.time()

        loader = unittest.TestLoader()

        suite = loader.loadTestsFromTestCase(TestAPIHandler)

        runner = unittest.TextTestRunner()

        result = runner.run(suite)

        # Обчислення часу виконання тестів

        end\_time = time.time()

        execution\_time = end\_time - start\_time

        # Викликайте show\_test\_results з об'єктом TestResult та часом виконання

        self.show\_test\_results(result, execution\_time)

    def show\_test\_results(self, result, execution\_time):

        if result.wasSuccessful():

            message = f"Всі тести успішно пройдено!\n"

        else:

            message = f"{result.errors} помилок, {result.failures} невдач\n"

        message += f"Пройдено тестів: {result.testsRun}\n"

        message += f"Час виконання тестів: {execution\_time:.3f} сек"

        tk.messagebox.showinfo("Результати тестів", message)

    def run(self):

        """

        Runs the Tkinter main loop.

        """

        self.root.mainloop()

def main():

    """

    Main entry point for the application.

    """

    root = tk.Tk()

    app = App(root)

    app.run()

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

    main()

**message\_box\_wrapper.py**

from tkinter import messagebox

class MessageBoxWrapper:

    @staticmethod

    def show\_error(title, message):

        messagebox.showerror(title, message)

**tests.py**

import unittest

import requests.exceptions

import os

from unittest.mock import Mock, patch, mock\_open

from .api\_handler import APIHandler

from .message\_box\_wrapper import MessageBoxWrapper

from .display\_handler import DisplayHandler

from .input\_parser import InputParser

from .file\_operations import FileOperations

class TestAPIHandler(unittest.TestCase):

    @patch('.api\_handler.requests.get')

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

        # Arrange

        expected\_data = [{'id': 1, 'firstname': 'John'}, {'id': 2, 'firstname': 'Jane'}]

        mock\_get.return\_value.json.return\_value = expected\_data

        # Act

        result = APIHandler.get\_users()

        # Assert

        self.assertEqual(result, expected\_data)

    @patch('.api\_handler.requests.get')

    @patch('message\_box\_wrapper.MessageBoxWrapper.show\_error')

    def test\_get\_users\_failure(self, mock\_show\_error, mock\_get):

        # Arrange

        mock\_get.side\_effect = requests.exceptions.RequestException("Fake Exception")

        # Act

        result = APIHandler.get\_users()

        # Assert

        self.assertEqual(result, [])

        mock\_show\_error.assert\_called\_once\_with("API Error", "Error getting data from API: Fake Exception")

    @patch('.api\_handler.requests.get')

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

        # Arrange

        user\_id = 1

        expected\_data = {'id': user\_id, 'firstname': 'John'}

        mock\_get.return\_value.json.return\_value = expected\_data

        # Act

        result = APIHandler.get\_user\_by\_id(user\_id)

        # Assert

        self.assertEqual(result, expected\_data)

    @patch('.api\_handler.requests.get')

    @patch('message\_box\_wrapper.MessageBoxWrapper.show\_error')

    def test\_get\_user\_by\_id\_failure(self, mock\_show\_error, mock\_get):

        # Arrange

        user\_id = 1

        mock\_get.side\_effect = requests.exceptions.RequestException("Fake Exception")

        # Act

        result = APIHandler.get\_user\_by\_id(user\_id)

        # Assert

        self.assertEqual(result, {})

        mock\_show\_error.assert\_called\_once\_with("API Error", f'Error getting user data from API: Fake Exception')

    def test\_display\_table\_success(self):

        # Arrange

        root\_mock = Mock()

        data = [{'id': 1, 'firstname': 'John'}, {'id': 2, 'firstname': 'Jane'}]

        header\_color = "black"

        table\_bg\_color = "white"

        # Act

        with patch('.display\_handler.messagebox') as messagebox\_mock, \

             patch('.display\_handler.tk.Toplevel') as toplevel\_mock:

            DisplayHandler.display\_table(root\_mock, data, header\_color, table\_bg\_color)

        # Assert

        messagebox\_mock.showinfo.assert\_not\_called()

        toplevel\_mock.assert\_called\_once()

    def test\_display\_table\_empty\_data(self):

        # Arrange

        root\_mock = Mock()

        data = []

        header\_color = "black"

        table\_bg\_color = "white"

        # Act

        with patch('display\_handler.messagebox') as messagebox\_mock, \

             patch('display\_handler.tk.Toplevel') as toplevel\_mock:

            DisplayHandler.display\_table(root\_mock, data, header\_color, table\_bg\_color)

        # Assert

        messagebox\_mock.showinfo.assert\_called\_once()

        toplevel\_mock.assert\_not\_called()

    def test\_display\_list\_success(self):

        # Arrange

        root\_mock = Mock()

        data = ['User 1', 'User 2']

        item\_color = "green"

        list\_bg\_color = "lightblue"

        # Act

        with patch('display\_handler.messagebox') as messagebox\_mock, \

             patch('display\_handler.tk.Toplevel') as toplevel\_mock:

            DisplayHandler.display\_list(root\_mock, data, item\_color, list\_bg\_color)

        # Assert

        messagebox\_mock.showinfo.assert\_not\_called()

        toplevel\_mock.assert\_called\_once()

    def test\_display\_list\_empty\_data(self):

        # Arrange

        root\_mock = Mock()

        data = []

        item\_color = "black"

        list\_bg\_color = "white"

        # Act

        with patch('display\_handler.messagebox') as messagebox\_mock, \

             patch('display\_handler.tk.Toplevel') as toplevel\_mock:

            DisplayHandler.display\_list(root\_mock, data, item\_color, list\_bg\_color)

        # Assert

        messagebox\_mock.showinfo.assert\_called\_once()

        toplevel\_mock.assert\_not\_called()

    def test\_parse\_full\_name(self):

        result = InputParser.parse\_user\_input("John Doe")

        self.assertEqual(result, "Введено повне ім'я: John Doe")

    def test\_parse\_phone\_number(self):

        result = InputParser.parse\_user\_input("(123) 456-7890")

        self.assertEqual(result, "Введено номер телефону: (123) 456-7890")

    def test\_parse\_date(self):

        result = InputParser.parse\_user\_input("2023-01-01")

        self.assertEqual(result, "Введено дату: 2023-01-01")

    def test\_parse\_datetime(self):

        result = InputParser.parse\_user\_input("2023-01-01 12:34:56")

        self.assertEqual(result, "Введено дату та час: 2023-01-01 12:34:56")

    def test\_parse\_email(self):

        result = InputParser.parse\_user\_input("test@example.com")

        self.assertEqual(result, "Введено електронну пошту: test@example.com")

    def test\_parse\_website(self):

        result = InputParser.parse\_user\_input("http://example.com")

        self.assertEqual(result, "Введено веб-сайт: http://example.com")

    @patch('data\_saver.DataSaver.save\_history\_entry')

    def test\_save\_history\_entry\_called(self, mock\_save\_history\_entry):

        user\_input = "John Doe"

        InputParser.parse\_user\_input(user\_input)

        mock\_save\_history\_entry.assert\_called\_with(user\_input)

    @patch("tkinter.filedialog.asksaveasfilename")

    @patch("tkinter.messagebox.showinfo")

    def test\_save\_representation(self, mock\_showinfo, mock\_file\_dialog):

        representation = "Test representation"

        representation\_type = "Test Type"

        file\_format = "txt"

        # Configure mock\_file\_dialog to return a file path

        mock\_file\_dialog.return\_value = "test\_file.txt"

        # Run the code that writes to the file

        FileOperations.save\_representation(representation, representation\_type, file\_format)

        # Check if the function called the correct methods with the file

        mock\_file\_dialog.assert\_called\_once\_with(

            defaultextension=f".{file\_format}",

            filetypes=[(f"{file\_format.upper()} files", f"\*.{file\_format}"), ("All files", "\*.\*")],

            title=f"Save {representation\_type} representation as"

        )

        # Check if the showinfo method was not called

        mock\_showinfo.assert\_called\_once\_with("Success", f"{representation\_type} representation saved successfully to test\_file.txt")

    def test\_load\_history\_from\_file(self):

        # Determine the expected path based on the current test directory

        test\_directory = os.path.dirname(os.path.abspath(\_\_file\_\_))

        expected\_path = os.path.join(test\_directory, 'history.json')

        # Create an empty history file

        with open(expected\_path, 'w', encoding='utf-8') as history\_file:

            history\_file.write('[]')

        # Now run the test

        history\_data = FileOperations.load\_history\_from\_file()

        # Perform assertions on history\_data as needed

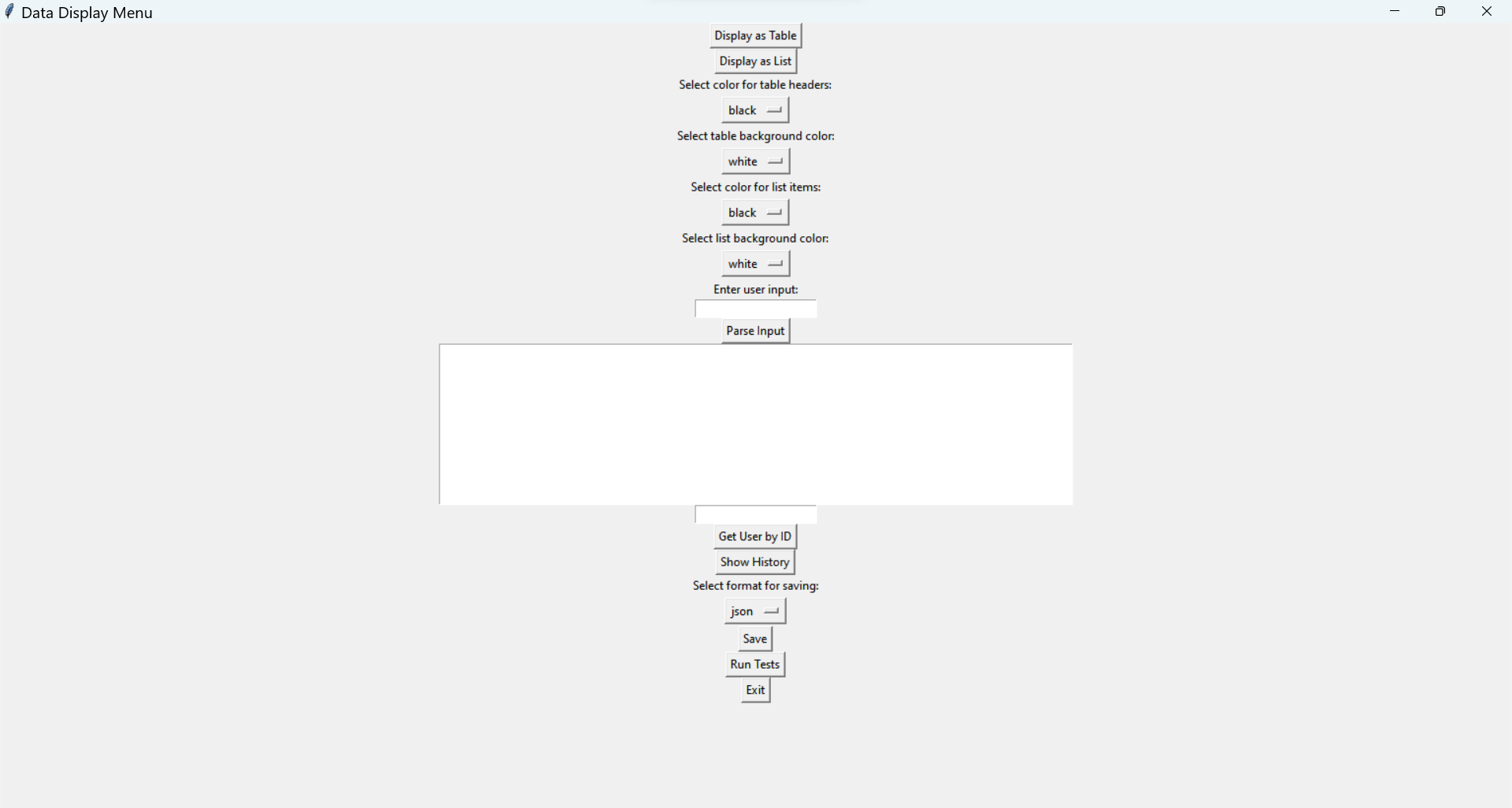
        self.assertEqual(history\_data, [])

        # Clean up: remove the history file after the test

        os.remove(expected\_path)

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

    unittest.main()



*Рис. 1 – Результат створення*

**Висновок:** на даній лабораторній роботі я створила консольний об’єктно - орієнтований додаток з використанням API.