Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Звіт про виконання лабораторної роботи №7

з дисципліни «Спеціалізовані мови програмування»

на тему «Робота з API та веб-сервісами»

Виконала:  
студентка групи РІ-31  
Заплетнюк Ю.І.

Прийняв:  
Щербак С.С.

**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 програмної реалізації**

**user\_repository.py**

import requests

class UserRepository:

API\_URL = "https://jsonplaceholder.typicode.com/users"

@staticmethod

def fetch\_all\_users():

response = requests.get(UserRepository.API\_URL)

response.raise\_for\_status()

return response.json()

**user\_service.py**

from repository.user\_repository import UserRepository

class UserService:

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

self.repository = UserRepository()

self.users = []

def load\_users(self):

self.users = self.repository.fetch\_all\_users()

def get\_users(self):

return self.users

**tests.py**

from repository.user\_repository import UserRepository

class UserService:

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

self.repository = UserRepository()

self.users = []

def load\_users(self):

self.users = self.repository.fetch\_all\_users()

def get\_users(self):

return self.users

**console\_interface.py**

import json

import csv

from tabulate import tabulate

from colorama import Fore, init

from utils.email\_validator import EmailValidator

def initialize\_colorama():

init(autoreset=True)

class ConsoleInterface:

def \_\_init\_\_(self, uow, history\_manager):

self.uow = uow

self.history\_manager = history\_manager

self.font\_color = None

def set\_colors(self):

print("\nCustomize Colors:")

user\_color = input("Enter font color: ").strip().upper()

if user\_color:

if user\_color in dir(Fore):

self.font\_color = getattr(Fore, user\_color)

initialize\_colorama()

else:

print("Invalid color input, defaulting to no color.")

self.font\_color = None

else:

print("No color applied, using default formatting.")

self.font\_color = None

def display\_users\_table(self):

headers = ["ID", "Name", "Username", "Email", "Website"]

table = [

[user['id'], user['name'], user['username'], user['email'], user['website']]

for user in self.uow.get\_users()

]

if self.font\_color:

headers = [self.font\_color + header for header in headers]

result = tabulate(table, headers=headers, tablefmt="grid")

print(result)

self.history\_manager.add\_to\_history("Display users as table", result)

def display\_users\_list(self):

result = ""

for user in self.uow.get\_users():

result += f"ID: {user['id']}\n"

result += f"Name: {user['name']}\n"

result += f"Username: {user['username']}\n"

result += f"Email: {user['email']}\n"

result += f"Website: {user['website']}\n"

result += "-" \* 40 + "\n"

print(result)

self.history\_manager.add\_to\_history("Display users as list", result)

def validate\_user\_email(self):

try:

user\_id = int(input("Enter the user ID to validate email: "))

user = next(user for user in self.uow.get\_users() if user['id'] == user\_id)

email = user['email']

is\_valid = EmailValidator.is\_valid\_email(email)

if is\_valid:

result = f"Email '{email}' for user ID {user\_id} is valid."

else:

result = f"Email '{email}' for user ID {user\_id} is invalid."

print(result)

self.history\_manager.add\_to\_history(f"Validate email for user ID {user\_id}", result)

except StopIteration:

result = f"User with ID {user\_id} not found."

print(result)

self.history\_manager.add\_to\_history(f"Validate email for user ID {user\_id}", result)

except ValueError:

result = "Invalid input. Please enter a numeric user ID."

print(result)

self.history\_manager.add\_to\_history(f"Validate email for user ID {user\_id}", result)

def save\_data(self, format):

filename = f"users.{format}"

users = self.uow.get\_users()

if format == "json":

with open(filename, "w", encoding="utf-8") as file:

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

elif format == "csv":

with open(filename, "w", newline="", encoding="utf-8") as file:

writer = csv.writer(file)

writer.writerow(["ID", "Name", "Username", "Email", "Website"])

for user in users:

writer.writerow([user['id'], user['name'], user['username'], user['email'], user['website']])

elif format == "txt":

with open(filename, "w", encoding="utf-8") as file:

for user in users:

file.write(f"ID: {user['id']}\n")

file.write(f"Name: {user['name']}\n")

file.write(f"Username: {user['username']}\n")

file.write(f"Email: {user['email']}\n")

file.write(f"Website: {user['website']}\n")

file.write("-" \* 40 + "\n")

result = f"Data saved to {filename}"

print(result)

self.history\_manager.add\_to\_history(f"Save data to {filename}", result)

**email\_validator.py**

import re

class EmailValidator:

EMAIL\_REGEX = r"^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"

@staticmethod

def is\_valid\_email(email):

return re.match(EmailValidator.EMAIL\_REGEX, email)

**history\_manager.py**

class HistoryManager:

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

self.history = []

def add\_to\_history(self, action, result):

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

def display\_history(self):

if not self.history:

print("No history available.")

else:

for idx, entry in enumerate(self.history, 1):

print(f"\nQuery #{idx}: {entry['action']}")

print(f"Result: {entry['result']}\n")

**app.py**

import sys

import os

sys.path.append(os.path.join(os.path.dirname(os.path.abspath(\_\_file\_\_))))

from service.user\_service import UserService

from utils.history\_manager import HistoryManager

from ui.console\_interface import ConsoleInterface

def main():

uow = UserService()

uow.load\_users()

history\_manager = HistoryManager()

interface = ConsoleInterface(uow, history\_manager)

while True:

print(f"\n1. Customize font colors")

print(f"2. Display users as table")

print(f"3. Display users as list")

print(f"4. Save data to file (JSON, CSV, TXT)")

print(f"5. Validate email of a user by ID")

print(f"6. Display query history")

print(f"7. Exit")

choice = input("Enter your choice: ")

if choice == "1":

interface.set\_colors()

elif choice == "2":

interface.display\_users\_table()

elif choice == "3":

interface.display\_users\_list()

elif choice == "4":

format = input("Enter file format (json, csv, txt): ").strip().lower()

if format in ["json", "csv", "txt"]:

interface.save\_data(format)

else:

print("Invalid format. Try again.")

elif choice == "5":

interface.validate\_user\_email()

elif choice == "6":

history\_manager.display\_history()

elif choice == "7":

break

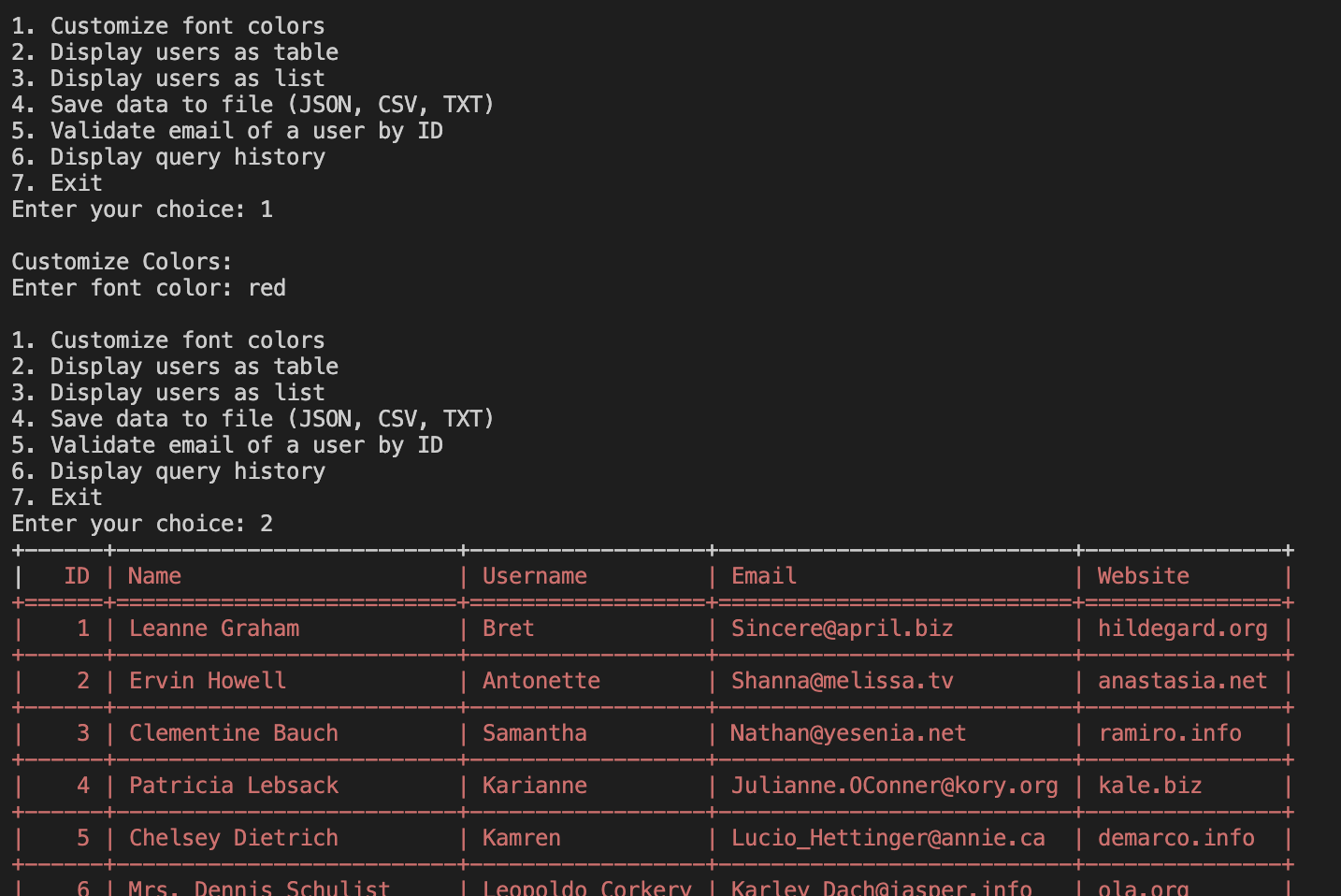
else:

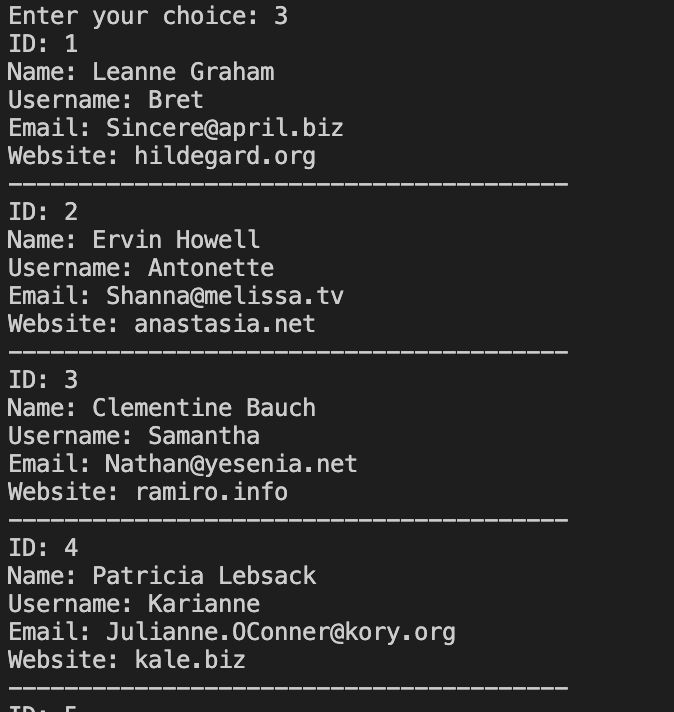
print("Invalid choice. Try again.")

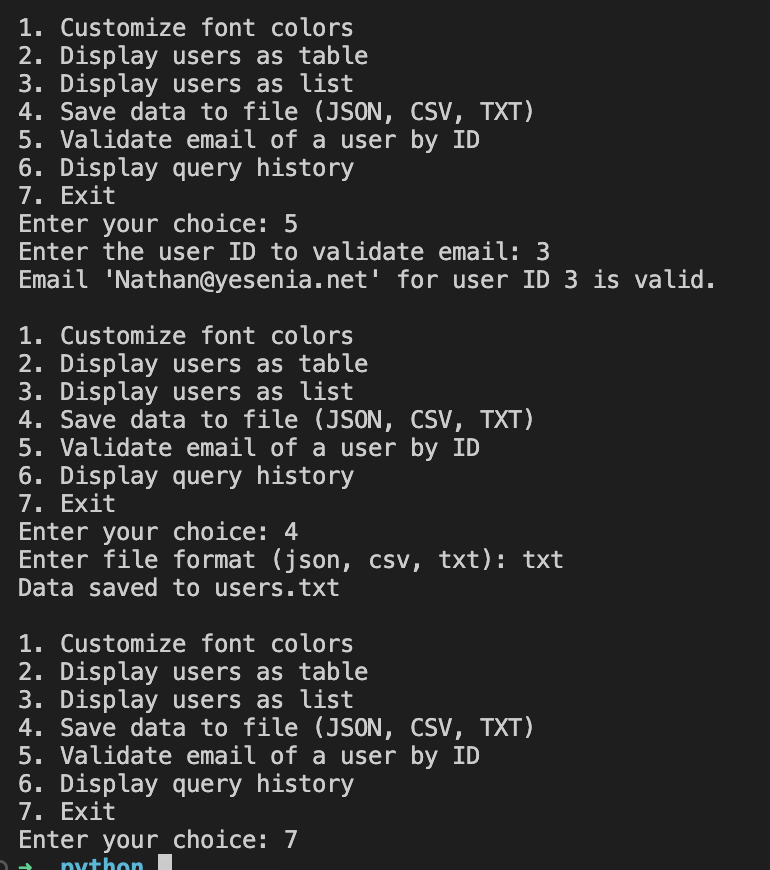
if \_\_name\_\_ == "\_\_main\_\_":

main()

На рис. 1-4 показано результат виконання завдання.







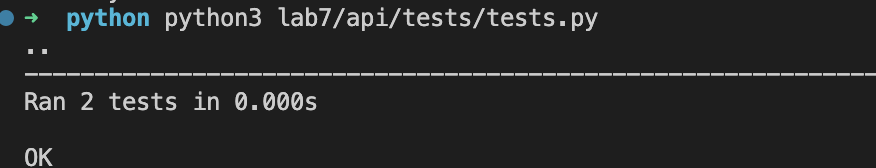


Рис. 1-4. Результат виконання завдання

**Висновки:** виконавши лабораторну роботу, було cтворено проєкт, який надав мені цінний досвід роботи з API, дизайну користувацького інтерфейсу, валідації введення, обробки помилок та тестування.