МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

Лабораторна робота №7

з дисципліни

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

на тему

Робота з API та веб-сервісами

Виконав:

ст. гр. РІ-21сп

Наконечний В.Б.

Прийняв:

Щербак С.С.

Львів-2024

**Мета лабораторної роботи:** Створення консольного об’єктно - орієнтованого додатка з використанням API та патернів проектування

**Хід 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.

**Виконання роботи**

**Папка Classes:**

**Вміст файлу adder.py:**

from Classes.data\_factory import DataFactory

class Adder:

def \_\_init\_\_(self, api\_repository, local\_users, local\_posts, history\_manager):

self.api\_repository = api\_repository

self.local\_users = local\_users

self.local\_posts = local\_posts

self.history\_manager = history\_manager

def add\_user(self, name, email):

user\_data = self.api\_repository.add\_user(name, email)

new\_user = DataFactory.create\_user(user\_data)

self.local\_users.append(new\_user)

self.history\_manager.log("Add user", f"User {name} ({email}) added.")

print("New user added:", user\_data)

def add\_post(self, title, body, user\_id):

post\_data = self.api\_repository.add\_post(title, body, user\_id)

new\_post = DataFactory.create\_post(post\_data)

self.local\_posts.append(new\_post)

self.history\_manager.log("Add post", f"Post {title} added.")

print("New post added:", post\_data)

**Вміст файлу api\_repository.py:**

import requests

class APIRepository:

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

def get\_users(self):

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

response.raise\_for\_status()

return response.json()

def get\_posts(self):

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

response.raise\_for\_status()

return response.json()

def add\_user(self, name, email):

payload = {"name": name, "email": email}

response = requests.post(f"{self.BASE\_URL}/users", json=payload)

response.raise\_for\_status()

return response.json()

def add\_post(self, title, body, user\_id):

payload = {"title": title, "body": body, "userId": user\_id}

response = requests.post(f"{self.BASE\_URL}/posts", json=payload)

response.raise\_for\_status()

return response.json()

**Вміст файлу console\_app.py:**

from tabulate import tabulate

from Classes.api\_repository import APIRepository

from Classes.data\_factory import DataFactory

from Classes.history\_manager import HistoryManager

from Classes.deleter import Deleter

from Classes.adder import Adder

from Classes.file\_manager import DataSaver

class ConsoleApp:

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

self.api\_repository = APIRepository()

self.local\_users = []

self.local\_posts = []

self.history\_manager = HistoryManager()

self.deleter = Deleter(self.local\_users, self.local\_posts, self.history\_manager)

self.adder = Adder(self.api\_repository, self.local\_users, self.local\_posts, self.history\_manager)

def show\_users(self):

users\_data = self.api\_repository.get\_users()

users = [DataFactory.create\_user(user) for user in users\_data]

users.extend(self.local\_users)

table = [[user.user\_id, user.name, user.email] for user in users]

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

self.history\_manager.log("Show users", "All users.")

def show\_posts(self):

posts\_data = self.api\_repository.get\_posts()

posts = [DataFactory.create\_post(post) for post in posts\_data]

posts.extend(self.local\_posts)

table = [[post.post\_id, post.title, post.body, post.user\_id] for post in posts]

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

self.history\_manager.log("Show posts", "All posts.")

def add\_user(self):

name = input("Enter name for new user: ")

email = input("Enter Email for new user: ")

self.adder.add\_user(name, email)

def add\_post(self):

title = input("Enter title: ")

body = input("Enter text: ")

user\_id = int(input("Enter user ID: "))

self.adder.add\_post(title, body, user\_id)

def delete\_user(self):

user\_id = int(input("Enter user ID for deletion: "))

self.deleter.delete\_user(user\_id)

def delete\_post(self):

post\_id = int(input("Enter post ID for deletion: "))

self.deleter.delete\_post(post\_id)

def save\_users(self):

users\_data = self.api\_repository.get\_users()

users = [DataFactory.create\_user(user) for user in users\_data]

users.extend(self.local\_users)

data = [user.\_\_dict\_\_ for user in users]

DataSaver.select\_save\_format(data, "users")

def save\_posts(self):

posts\_data = self.api\_repository.get\_posts()

posts = [DataFactory.create\_post(post) for post in posts\_data]

posts.extend(self.local\_posts)

data = [post.\_\_dict\_\_ for post in posts]

DataSaver.select\_save\_format(data, "posts")

def show\_history(self):

self.history\_manager.show\_history()

**Вміст файлу data\_factory.py:**

from Classes.user import User

from Classes.post import Post

class DataFactory:

@staticmethod

def create\_user(data):

return User(data["id"], data["name"], data["email"])

@staticmethod

def create\_post(data):

return Post(data["id"], data["title"], data["body"], data["userId"])

**Вміст файлу deleter.py:**

class Deleter:

def \_\_init\_\_(self, local\_users, local\_posts, history\_manager):

self.local\_users = local\_users

self.local\_posts = local\_posts

self.history\_manager = history\_manager

def delete\_user(self, user\_id):

for user in self.local\_users:

if user.user\_id == user\_id:

self.local\_users.remove(user)

self.history\_manager.log("Delete user", f"User with ID {user\_id} was deleted.")

print(f"User with ID {user\_id} was deleted.")

return

print(f"Error: User with ID {user\_id} is not found.")

def delete\_post(self, post\_id):

for post in self.local\_posts:

if post.post\_id == post\_id:

self.local\_posts.remove(post)

self.history\_manager.log("Delete post", f"Post with ID {post\_id} was deleted.")

print(f"Post with ID {post\_id} was deleted.")

return

print(f"Error: Post with ID {post\_id} is not found.")

**Вміст файлу file\_manager.py:**

import json

import csv

import os

class DataSaver:

@staticmethod

def save\_to\_json(data, filename):

os.makedirs("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Lab7/Data", exist\_ok=True)

filepath = os.path.join("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Lab7/Data", filename)

with open(filepath, "w") as file:

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

print(f"Data was saved in {filepath}")

@staticmethod

def save\_to\_csv(data, filename):

os.makedirs("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Lab7/Data", exist\_ok=True)

filepath = os.path.join("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Lab7/Data", filename)

if data:

keys = data[0].keys()

with open(filepath, "w", newline='') as file:

writer = csv.DictWriter(file, fieldnames=keys)

writer.writeheader()

writer.writerows(data)

print(f"Data was saved in {filepath}")

@staticmethod

def save\_to\_txt(data, filename):

os.makedirs("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Lab7/Data", exist\_ok=True)

filepath = os.path.join("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Lab7/Data", filename)

with open(filepath, "w") as file:

for entry in data:

file.write(f"{entry}\n")

print(f"Data was saved in {filepath}")

@staticmethod

def select\_save\_format(data, data\_type):

print("\nSelect format to save:")

print("1. JSON")

print("2. CSV")

print("3. TXT")

choice = input("Select option: ")

filename = input("Enter file name, without extencion: ")

if choice == "1":

filename += ".json"

DataSaver.save\_to\_json(data, filename)

elif choice == "2":

filename += ".csv"

DataSaver.save\_to\_csv(data, filename)

elif choice == "3":

filename += ".txt"

DataSaver.save\_to\_txt(data, filename)

else:

print("Error; Incorrect option! Please, try again!")

DataSaver.select\_save\_format(data, data\_type)

**Вміст файлу history\_manager.py:**

from datetime import datetime

from tabulate import tabulate

class HistoryManager:

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

self.\_\_history = []

def log(self, operation, details=""):

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

self.\_\_history.append({"timestamp": timestamp, "operation": operation, "details": details})

def show\_history(self):

print("\nHistory:")

table = [[entry["timestamp"], entry["operation"], entry["details"]] for entry in self.\_\_history]

print(tabulate(table, headers=["Time", "Operation", "Details"], tablefmt="grid"))

**Вміст файлу post.py:**

class Post:

def \_\_init\_\_(self, post\_id, title, body, user\_id):

self.post\_id = post\_id

self.title = title

self.body = body

self.user\_id = user\_id

**Вміст файлу user.py:**

class User:

def \_\_init\_\_(self, user\_id, name, email):

self.user\_id = user\_id

self.name = name

self.email = email

**Папка Tests:**

**Вміст файлу test\_adder.py:**

import unittest

from unittest.mock import MagicMock

from Classes.adder import Adder

class TestAdder(unittest.TestCase):

def setUp(self):

# Створюємо об'єкти, які будемо використовувати в тестах

self.api\_repository = MagicMock()

self.local\_users = []

self.local\_posts = []

self.history\_manager = MagicMock()

self.adder = Adder(self.api\_repository, self.local\_users, self.local\_posts, self.history\_manager)

def test\_add\_user(self):

# Налаштовуємо мок API для повернення певних даних користувача

self.api\_repository.add\_user.return\_value = {"id": 1, "name": "Test User", "email": "test@example.com"}

# Виконуємо метод

self.adder.add\_user("Test User", "test@example.com")

# Перевіряємо, чи користувача додано в локальний список

self.assertEqual(len(self.local\_users), 1)

self.assertEqual(self.local\_users[0].name, "Test User")

self.assertEqual(self.local\_users[0].email, "test@example.com")

# Перевіряємо, чи викликано метод для запису історії

self.history\_manager.log.assert\_called\_once\_with("Add user", " User Test User (test@example.com) added.")

def test\_add\_post(self):

# Налаштовуємо мок API для повернення певних даних поста

self.api\_repository.add\_post.return\_value = {"id": 1, "title": "Test Post", "body": "Post body", "userId": 1}

# Виконуємо метод

self.adder.add\_post("Test Post", "Post body", 1)

# Перевіряємо, чи пост додано в локальний список

self.assertEqual(len(self.local\_posts), 1)

self.assertEqual(self.local\_posts[0].title, "Test Post")

self.assertEqual(self.local\_posts[0].body, "Post body")

# Перевіряємо, чи викликано метод для запису історії

self.history\_manager.log.assert\_called\_once\_with("Add post", "Post Test Post added.")

**Вміст файлу test\_api\_repository.py:**

import unittest

from unittest.mock import patch

from Classes.api\_repository import APIRepository # Змініть шлях відповідно до структури проєкту

class TestAPIRepository(unittest.TestCase):

@patch("requests.get")

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

mock\_response = mock\_get.return\_value

mock\_response.raise\_for\_status.return\_value = None

mock\_response.json.return\_value = [{"id": 1, "name": "John Doe"}]

api\_repo = APIRepository()

result = api\_repo.get\_users()

mock\_get.assert\_called\_once\_with(f"{APIRepository.BASE\_URL}/users")

self.assertEqual(result, [{"id": 1, "name": "John Doe"}])

@patch("requests.get")

def test\_get\_posts(self, mock\_get):

mock\_response = mock\_get.return\_value

mock\_response.raise\_for\_status.return\_value = None

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

api\_repo = APIRepository()

result = api\_repo.get\_posts()

mock\_get.assert\_called\_once\_with(f"{APIRepository.BASE\_URL}/posts")

self.assertEqual(result, [{"id": 1, "title": "Sample Post"}])

@patch("requests.post")

def test\_add\_user(self, mock\_post):

mock\_response = mock\_post.return\_value

mock\_response.raise\_for\_status.return\_value = None

mock\_response.json.return\_value = {"id": 101, "name": "Jane Doe", "email": "jane@example.com"}

api\_repo = APIRepository()

result = api\_repo.add\_user(name="Jane Doe", email="jane@example.com")

mock\_post.assert\_called\_once\_with(

f"{APIRepository.BASE\_URL}/users",

json={"name": "Jane Doe", "email": "jane@example.com"}

)

self.assertEqual(result, {"id": 101, "name": "Jane Doe", "email": "jane@example.com"})

@patch("requests.post")

def test\_add\_post(self, mock\_post):

mock\_response = mock\_post.return\_value

mock\_response.raise\_for\_status.return\_value = None

mock\_response.json.return\_value = {"id": 101, "title": "New Post", "body": "Post content", "userId": 1}

api\_repo = APIRepository()

result = api\_repo.add\_post(title="New Post", body="Post content", user\_id=1)

mock\_post.assert\_called\_once\_with(

f"{APIRepository.BASE\_URL}/posts",

json={"title": "New Post", "body": "Post content", "userId": 1}

)

self.assertEqual(result, {"id": 101, "title": "New Post", "body": "Post content", "userId": 1})

**Вміст файлу test\_consol\_app.py:**

import unittest

from unittest.mock import patch, MagicMock

from Classes.console\_app import ConsoleApp

class TestConsoleApp(unittest.TestCase):

def setUp(self):

# Ініціалізуємо екземпляр ConsoleApp перед кожним тестом

self.app = ConsoleApp()

# Мокаємо залежні об'єкти

self.app.api\_repository = MagicMock()

self.app.history\_manager = MagicMock()

self.app.deleter = MagicMock()

self.app.adder = MagicMock()

self.app.local\_users = []

self.app.local\_posts = []

# Мокаємо зовнішні методи, щоб не виконували реальні HTTP-запити та інші дії

self.app.api\_repository.get\_users.return\_value = [{"id": 1, "name": "John Doe", "email": "john@example.com"}]

self.app.api\_repository.get\_posts.return\_value = [

{"id": 1, "title": "Post 1", "body": "Content of post 1", "userId": 1}]

@patch('builtins.print')

def test\_show\_users(self, mock\_print):

# Тестуємо метод show\_users

self.app.show\_users()

# Перевіряємо, що викликано правильно виведення на екран через print

mock\_print.assert\_called\_with(

'+------+--------------------------+---------------------------+\n| ID | Name | Email |\n+======+==========================+===========================+\n| 1 | John Doe | john@example.com |\n+------+--------------------------+---------------------------+'

)

self.app.history\_manager.log.assert\_called\_with("Show users", "All users.")

@patch('builtins.print')

def test\_show\_posts(self, mock\_print):

# Тестуємо метод show\_posts

self.app.show\_posts()

# Перевіряємо, що викликано правильно виведення на екран через print

mock\_print.assert\_called\_with(

'+------+---------------------------------------------------------------------------------+----------------------------------------------------------------------------------+-----------+\n| ID | Title | Text | User ID |\n+======+=================================================================================+==================================================================================+===========+\n| 1 | Post 1 | Content of post 1 | 1 |'

)

self.app.history\_manager.log.assert\_called\_with("Show posts", "All posts.")

@patch('builtins.input', side\_effect=["Alice", "alice@example.com"])

def test\_add\_user(self, mock\_input):

# Тестуємо метод add\_user

self.app.add\_user()

self.app.adder.add\_user.assert\_called\_with("Alice", "alice@example.com")

@patch('builtins.input', side\_effect=["New Post", "This is a test post", "1"])

def test\_add\_post(self, mock\_input):

# Тестуємо метод add\_post

self.app.add\_post()

self.app.adder.add\_post.assert\_called\_with("New Post", "This is a test post", 1)

@patch('builtins.input', side\_effect=["1"])

def test\_delete\_user(self, mock\_input):

# Тестуємо метод delete\_user

self.app.delete\_user()

self.app.deleter.delete\_user.assert\_called\_with(1)

@patch('builtins.input', side\_effect=["1"])

def test\_delete\_post(self, mock\_input):

# Тестуємо метод delete\_post

self.app.delete\_post()

self.app.deleter.delete\_post.assert\_called\_with(1)

@patch('Classes.data\_factory.DataSaver.select\_save\_format')

def test\_save\_users(self, mock\_save\_format):

# Тестуємо метод save\_users

self.app.save\_users()

self.app.api\_repository.get\_users.assert\_called\_once()

mock\_save\_format.assert\_called\_once()

@patch('Classes.data\_factory.DataSaver.select\_save\_format')

def test\_save\_posts(self, mock\_save\_format):

# Тестуємо метод save\_posts

self.app.save\_posts()

self.app.api\_repository.get\_posts.assert\_called\_once()

mock\_save\_format.assert\_called\_once()

@patch('builtins.print')

def test\_show\_history(self, mock\_print):

# Тестуємо метод show\_history

self.app.show\_history()

self.app.history\_manager.show\_history.assert\_called\_once()

**Вміст файлу test\_deleter.py:**

import unittest

from unittest.mock import MagicMock

from Classes.deleter import Deleter

from Classes.user import User

from Classes.post import Post

class TestDeleter(unittest.TestCase):

def setUp(self):

# Створюємо тестові дані

self.local\_users = [User(1, "User1", "user1@example.com")]

self.local\_posts = [Post(1, "Title1", "Body1", 1)]

self.history\_manager = MagicMock()

self.deleter = Deleter(self.local\_users, self.local\_posts, self.history\_manager)

def test\_delete\_user\_found(self):

# Виконуємо видалення користувача

self.deleter.delete\_user(1)

# Перевіряємо, чи користувача видалено

self.assertEqual(len(self.local\_users), 0)

# Перевіряємо, чи викликано метод для запису історії

self.history\_manager.log.assert\_called\_once\_with("Delete user", "User with ID 1 was deleted.")

def test\_delete\_user\_not\_found(self):

# Виконуємо видалення користувача, якого немає

self.deleter.delete\_user(2)

# Перевіряємо, чи кількість користувачів не змінилася

self.assertEqual(len(self.local\_users), 1)

# Перевіряємо, чи не викликано метод для запису історії

self.history\_manager.log.assert\_not\_called()

def test\_delete\_post\_found(self):

# Виконуємо видалення поста

self.deleter.delete\_post(1)

# Перевіряємо, чи пост видалено

self.assertEqual(len(self.local\_posts), 0)

# Перевіряємо, чи викликано метод для запису історії

self.history\_manager.log.assert\_called\_once\_with("Delete post", "Post with ID 1 was deleted.")

def test\_delete\_post\_not\_found(self):

# Виконуємо видалення поста, якого немає

self.deleter.delete\_post(2)

# Перевіряємо, чи кількість постів не змінилася

self.assertEqual(len(self.local\_posts), 1)

# Перевіряємо, чи не викликано метод для запису історії

self.history\_manager.log.assert\_not\_called()

**Вміст файлу test\_file\_manager.py:**

import unittest

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

from Classes.file\_manager import DataSaver # Шлях змінити відповідно до вашої структури проєкту

import json

import csv

class TestDataSaver(unittest.TestCase):

@patch("os.makedirs")

@patch("builtins.open", new\_callable=mock\_open)

def test\_save\_to\_json(self, mock\_file\_open, mock\_makedirs):

data = {"key": "value"}

filename = "test.json"

DataSaver.save\_to\_json(data, filename)

mock\_makedirs.assert\_called\_once\_with("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Data", exist\_ok=True)

mock\_file\_open.assert\_called\_once\_with("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Data/test.json", "w")

handle = mock\_file\_open()

handle.write.assert\_called\_once\_with(json.dumps(data, indent=4))

@patch("os.makedirs")

@patch("builtins.open", new\_callable=mock\_open)

def test\_save\_to\_csv(self, mock\_file\_open, mock\_makedirs):

data = [{"name": "Alice", "age": 30}, {"name": "Bob", "age": 25}]

filename = "test.csv"

DataSaver.save\_to\_csv(data, filename)

mock\_makedirs.assert\_called\_once\_with("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Data", exist\_ok=True)

mock\_file\_open.assert\_called\_once\_with("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Data/test.csv", "w", newline='')

handle = mock\_file\_open()

writer = csv.DictWriter(handle, fieldnames=["name", "age"])

writer.writeheader()

writer.writerows(data)

@patch("os.makedirs")

@patch("builtins.open", new\_callable=mock\_open)

def test\_save\_to\_txt(self, mock\_file\_open, mock\_makedirs):

data = ["Line 1", "Line 2", "Line 3"]

filename = "test.txt"

DataSaver.save\_to\_txt(data, filename)

mock\_makedirs.assert\_called\_once\_with("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Data", exist\_ok=True)

mock\_file\_open.assert\_called\_once\_with("/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Data/test.txt", "w")

handle = mock\_file\_open()

expected\_calls = [call("Line 1\n"), call("Line 2\n"), call("Line 3\n")]

handle.write.assert\_has\_calls(expected\_calls, any\_order=False)

@patch("builtins.input", side\_effect=["1", "test\_file"])

@patch("Classes.data\_saver.DataSaver.save\_to\_json") # Замінити на правильний шлях до DataSaver

def test\_select\_save\_format\_json(self, mock\_save\_to\_json, mock\_input):

data = {"key": "value"}

DataSaver.select\_save\_format(data, "json")

mock\_save\_to\_json.assert\_called\_once\_with(data, "test\_file.json")

@patch("builtins.input", side\_effect=["2", "test\_file"])

@patch("Classes.data\_saver.DataSaver.save\_to\_csv") # Замінити на правильний шлях до DataSaver

def test\_select\_save\_format\_csv(self, mock\_save\_to\_csv, mock\_input):

data = [{"name": "Alice"}, {"name": "Bob"}]

DataSaver.select\_save\_format(data, "csv")

mock\_save\_to\_csv.assert\_called\_once\_with(data, "test\_file.csv")

@patch("builtins.input", side\_effect=["3", "test\_file"])

@patch("Classes.data\_saver.DataSaver.save\_to\_txt") # Замінити на правильний шлях до DataSaver

def test\_select\_save\_format\_txt(self, mock\_save\_to\_txt, mock\_input):

data = ["Line 1", "Line 2"]

DataSaver.select\_save\_format(data, "txt")

mock\_save\_to\_txt.assert\_called\_once\_with(data, "test\_file.txt")

@patch("builtins.input", side\_effect=["4", "test\_file"])

@patch("builtins.print")

def test\_select\_save\_format\_invalid\_option(self, mock\_print, mock\_input):

data = ["Line 1", "Line 2"]

with self.assertRaises(RecursionError):

DataSaver.select\_save\_format(data, "txt")

mock\_print.assert\_called\_with("Error: Incorrect option! Please, try again!")

**Вміст файлу test\_user\_interface.py:**

import unittest

from unittest.mock import patch, MagicMock

from Classes.console\_app import ConsoleApp

from UI.user\_interface import main

class TestConsoleApp(unittest.TestCase):

def setUp(self):

pass

@patch.object(ConsoleApp, 'show\_users')

@patch('builtins.input', side\_effect=["1", "10"])

def test\_show\_users\_option(self, mock\_input, mock\_show\_users):

# Перевірка, чи викликається show\_users при виборі опції 1

main()

mock\_show\_users.assert\_called\_once()

@patch.object(ConsoleApp, 'show\_posts')

@patch('builtins.input', side\_effect=["2", "10"])

def test\_show\_posts\_option(self, mock\_input, mock\_show\_posts):

main()

mock\_show\_posts.assert\_called\_once()

@patch.object(ConsoleApp, 'add\_user')

@patch('builtins.input', side\_effect=["3", "10"])

def test\_add\_user\_option(self, mock\_input, mock\_add\_user):

main()

mock\_add\_user.assert\_called\_once()

@patch.object(ConsoleApp, 'add\_post')

@patch('builtins.input', side\_effect=["4", "10"])

def test\_add\_post\_option(self, mock\_input, mock\_add\_post):

main()

mock\_add\_post.assert\_called\_once()

@patch.object(ConsoleApp, 'delete\_user')

@patch('builtins.input', side\_effect=["5", "10"])

def test\_delete\_user\_option(self, mock\_input, mock\_delete\_user):

main()

mock\_delete\_user.assert\_called\_once()

@patch.object(ConsoleApp, 'delete\_post')

@patch('builtins.input', side\_effect=["6", "10"])

def test\_delete\_post\_option(self, mock\_input, mock\_delete\_post):

main()

mock\_delete\_post.assert\_called\_once()

@patch.object(ConsoleApp, 'save\_users')

@patch('builtins.input', side\_effect=["7", "10"])

def test\_save\_users\_option(self, mock\_input, mock\_save\_users):

main()

mock\_save\_users.assert\_called\_once()

@patch.object(ConsoleApp, 'save\_posts')

@patch('builtins.input', side\_effect=["8", "10"])

def test\_save\_posts\_option(self, mock\_input, mock\_save\_posts):

main()

mock\_save\_posts.assert\_called\_once()

@patch.object(ConsoleApp, 'show\_history')

@patch('builtins.input', side\_effect=["9", "10"])

def test\_show\_history\_option(self, mock\_input, mock\_show\_history):

main()

mock\_show\_history.assert\_called\_once()

@patch('builtins.input', side\_effect=["11", "10"])

def test\_invalid\_option(self, mock\_input):

# Тест для некоректного вибору опції

with patch('builtins.print') as mock\_print:

main()

mock\_print.assert\_any\_call("Error: Incorrect option! Please, try adain!")

**Папка UI:**

**Вміст файлу user\_interface.py:**

import os

import sys

lab7\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), ".."))

sys.path.append(lab7\_root)

from Classes.console\_app import ConsoleApp

def main():

app = ConsoleApp()

while True:

print("\nMain menu:")

print("1. Show users.")

print("2. Show posts.")

print("3. Add new user.")

print("4. Add new post.")

print("5. Delete user.")

print("6. Delete post.")

print("7. Save users.")

print("8. Save posts.")

print("9. Show history.")

print("0. Exit.")

choice = input("Select option: ")

match choice:

case "1":

app.show\_users()

case "2":

app.show\_posts()

case "3":

app.add\_user()

case "4":

app.add\_post()

case "5":

app.delete\_user()

case "6":

app.delete\_post()

case "7":

app.save\_users()

case "8":

app.save\_posts()

case "9":

app.show\_history()

case "0":

print("Exit!")

break

case \_:

print("Error: Incorrect option! Please, try again!")

**Вміст файлу runner.py:**

from UI.user\_interface import main

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

main()

**Вміст файлу runner\_tests.py:**

import coverage

import unittest

import os

def run\_tests():

cov = coverage.Coverage(source=['Classes', 'UI'])

cov.start()

loader = unittest.TestLoader()

test\_folder = os.path.abspath('/Users/admin/Desktop/ДЗ/3 семестр/Спеціалізовані мови програмування/Final/Lab7/Tests')

tests = loader.discover(test\_folder)

test\_runner = unittest.TextTestRunner()

test\_runner.run(tests)

cov.stop()

cov.save()

print("\n--- Coverage Report ---")

cov.report(show\_missing=True)

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

run\_tests()

Результат виконання деяких команд, а також тестування програми зображено на рис. 1–рис. 5.

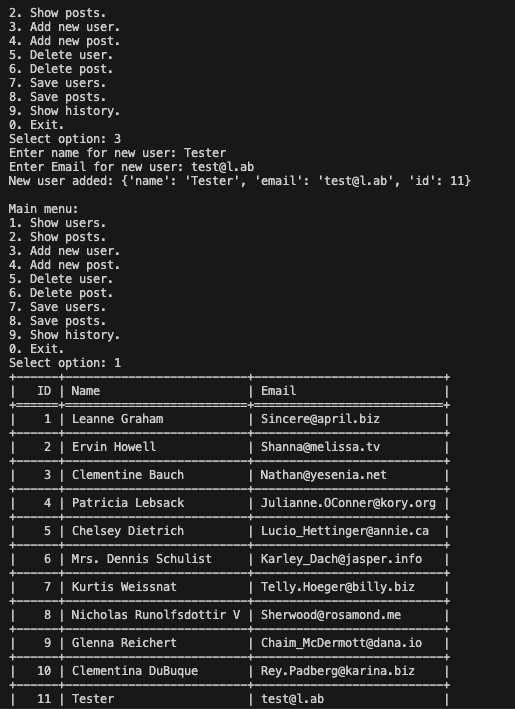


Рис. 1 – Додавання користувача



Рис. 2 – Видалення користувача

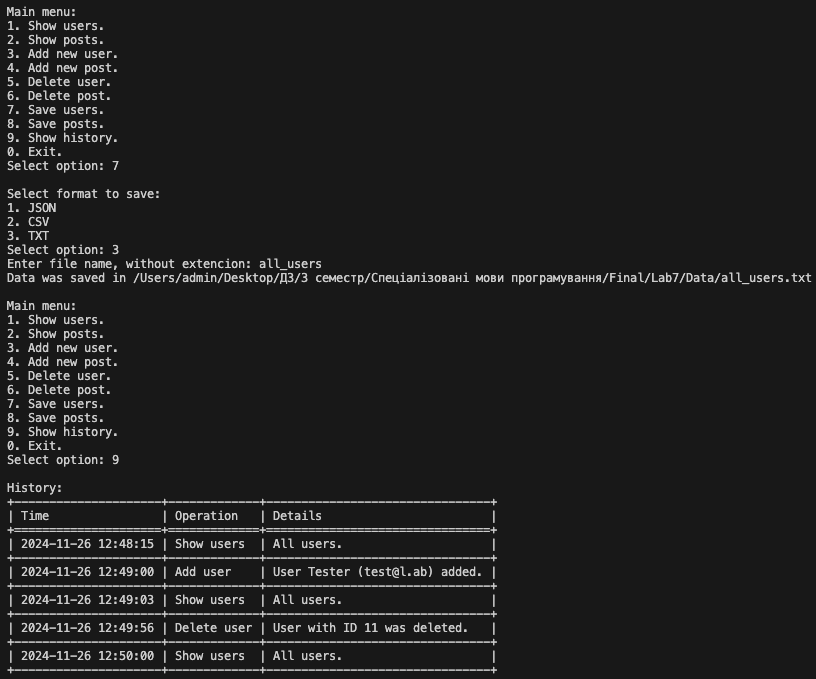


Рис. 3 – Збереження користувачів у файл

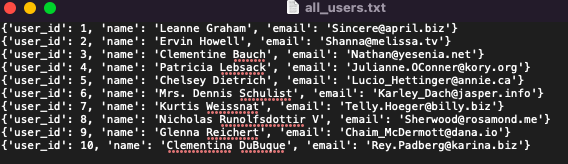


Рис. 4 – Зображення користувачів у файлі

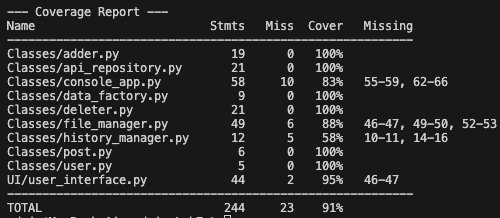


Рис. 5 – Результат тестування

**Висновок:** У ході виконання ЛР я створив проект, який надав цінний досвід роботи з API, дизайну користувацького інтерфейсу, валідації введення, обробки помилок та тестування. Цей проект виконує різні дії з використанням API, а також тестує додаток та виводить Coverage Report.