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

Виконала:

ст. гр. РІ-31

Оксана ЛЕСЮК

Прийняв:

Сергій Сергійович ЩЕРБАК

Львів-2024

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

**Завдання:**

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.

Виконавши ці завдання, ви створите проект, який надасть вам цінний досвід роботи з API, дизайну користувацького інтерфейсу, валідації введення, обробки помилок та тестування.

**Хід роботи**

**APIClient.py:**

class APIClient:

\_instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if not cls.\_instance:

cls.\_instance = super(APIClient, cls).\_\_new\_\_(cls, \*args, \*\*kwargs)

cls.\_instance.api\_url = 'https://catfact.ninja/facts'

return cls.\_instance

def fetch\_data(self):

if not self.api\_url:

raise ValueError("API URL is not configured")

try:

response = requests.get(self.api\_url)

if response.status\_code == 200:

return response.json()['data']

else:

raise Exception(f"Failed to fetch data: {response.status\_code}")

except Exception as e:

raise Exception(f"Error fetching data: {e}")

**Application.py**:

class Application:

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

self.api\_client = APIClient()

self.history\_logger = HistoryLogger()

def fetch\_and\_display\_data(self, strategy\_type):

try:

data = self.api\_client.fetch\_data()

display\_strategy = DisplayFactory().get\_display\_strategy(strategy\_type)

display\_strategy.display(data)

self.history\_logger.log\_query("fetch\_and\_display\_data", data)

except Exception as e:

print(f"Error: {e}")

def save\_data(self, data, format\_type):

try:

if format\_type == 'json':

with open('data.json', 'w') as f:

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

elif format\_type == 'csv':

with open('data.csv', 'w', newline='') as f:

writer = csv.writer(f)

writer.writerow(['Fact'])

for item in data:

writer.writerow([item['fact']])

elif format\_type == 'txt':

with open('data.txt', 'w') as f:

for item in data:

f.write(f"{item['fact']}\n")

print(f"Data saved to {format\_type.upper()} format.")

except Exception as e:

print(f"Error saving data: {e}")

def show\_history(self):

self.history\_logger.show\_history()

def parse\_input(input\_str, pattern):

match = re.match(pattern, input\_str)

if match:

return match.groups()

else:

return None

def validate\_input(self, prompt, pattern):

user\_input = input(prompt)

while not self.parse\_input(user\_input, pattern):

print("Invalid input. Please try again.")

user\_input = input(prompt)

return user\_input

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

app = Application()

**DisplayFactory.py:**

class DisplayFactory:

def get\_display\_strategy(self, strategy\_type):

if strategy\_type == "table":

return TableDisplay()

elif strategy\_type == "list":

return ListDisplay()

else:

raise ValueError("Invalid display strategy")

**HistoryLogger.py:**

class HistoryLogger:

@staticmethod

def log\_query(query, result):

with open('history.txt', 'a') as history\_file:

history\_file.write(f"Query: {query}\nResult: {result}\n\n")

@staticmethod

def show\_history():

with open('history.txt', 'r') as history\_file:

print(history\_file.read())

**ListDisplay.py:**

class ListDisplay(DisplayStrategy):

def display(self, data):

print(colored("Cat Facts List", "green"))

for item in data:

print(f"{item['fact']}")

**TableDisplay.py:**

class TableDisplay(DisplayStrategy):

def display(self, data):

print(colored("Cat Facts Table", "blue"))

for item in data:

print(f"Fact: {item['fact']}")

**TestAPIIntegration.py:**

class TestAPIIntegration(unittest.TestCase):

def test\_fetch\_data(self):

api\_client = APIClient()

data = api\_client.fetch\_data()

self.assertTrue(isinstance(data, list))

self.assertTrue('fact' in data[0])

def test\_invalid\_input(self):

app = Application()

result = app.validate\_input("Enter a date (YYYY-MM-DD): ", r"\d{4}-\d{2}-\d{2}")

self.assertIsNotNone(result)

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

app = Application()

# Displaying data in table format

app.fetch\_and\_display\_data('table')

# Saving data in JSON format

app.save\_data([{'fact': "Cats sleep for 70% of their lives."}], 'json')

# Showing query history

app.show\_history()

# Running Unit Tests

unittest.main()

**Посилання на гіт-репозиторій:**

https://github.com/lesiukoksana/SMP-labs

**Висновок:**

У результаті виконання завдань було створено консольний додаток, який:

1. Інтегрується з API.
2. Відображає дані у зручному для користувача форматі.
3. Надає функції збереження, обробки помилок та ведення історії.
4. Забезпечує перевірку введення користувача для підвищення надійності.

Виконана робота дозволила отримати практичний досвід роботи з API, використання патернів програмування, розробки інтерфейсів та тестування. Створений додаток може бути легко розширений або інтегрований у більші системи.