МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра ІСМ

******

Звіт

до лабораторної роботи №7

На тему “Робота з API та веб-сервісами”

З дисципліни “Спеціалізовані мови програмування”

*Виконав:*

*ст. гр. ІТ-31*

*Лисецький Володимир*

*Прийняв:*

*Щербак С. С.*

*Львів - 2023*

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

**Хід роботи**

**Програмний код:**

/handlers/APIHandler.py

import requests

class APIHandler:

def \_\_init\_\_(self, api\_url, api\_key=None):

self.api\_url = api\_url

self.api\_key = api\_key

def set\_api\_key(self, api\_key):

self.api\_key = api\_key

def fetch\_data(self, endpoint, params=None):

try:

headers = {'Authorization': f'Bearer {self.api\_key}'} if self.api\_key else None

response = requests.get(f'{self.api\_url}/{endpoint}', params=params, headers=headers)

response.raise\_for\_status() # Raise an error for non-200 status codes

return response.json()

except requests.RequestException as e:

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

return None

def post\_data(self, endpoint, data=None):

try:

headers = {'Authorization': f'Bearer {self.api\_key}'} if self.api\_key else None

response = requests.post(f'{self.api\_url}/{endpoint}', json=data, headers=headers)

response.raise\_for\_status()

return response.json()

except requests.RequestException as e:

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

return None

def patch\_data(self, endpoint, data=None):

try:

headers = {'Authorization': f'Bearer {self.api\_key}'} if self.api\_key else None

response = requests.patch(f'{self.api\_url}/{endpoint}', json=data, headers=headers)

response.raise\_for\_status()

return response.json()

except requests.RequestException as e:

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

return None

def delete\_data(self, endpoint):

try:

headers = {'Authorization': f'Bearer {self.api\_key}'} if self.api\_key else None

response = requests.delete(f'{self.api\_url}/{endpoint}', headers=headers)

response.raise\_for\_status()

return "Data deleted successfully"

except requests.RequestException as e:

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

return None

/dataprocess/DataParse.py

import json

from datetime import datetime

class JSONParser:

def \_\_init\_\_(self, json\_data):

self.json\_data = json\_data

def json\_output(self, data=None, level=0):

"""

Recursively formats and prints JSON data in a pretty way with dynamic indentation depth.

Parameters:

data (dict or list): The JSON data to format and print. If not provided, uses the data from the instance.

level (int): Current indentation level.

"""

data = data if data is not None else self.json\_data

indent = 2 # Default indentation width

if level > 10: # Add a limit to recursion depth to avoid infinite loops

print("Warning: Maximum recursion depth reached.")

return

if isinstance(data, dict):

# Print curly braces for dictionaries

print("{")

for key, value in data.items():

# Indentation for each key-value pair

print(" " \* (level + 1) \* indent + f'"{key}": ', end="")

# Recursive call for nested structures

self.json\_output(value, level + 1)

print(",")

# Closing curly brace

print(" " \* level \* indent + "}")

elif isinstance(data, list):

# Print square brackets for lists

print("[")

for item in data:

# Indentation for each list item

print(" " \* (level + 1) \* indent, end="")

# Recursive call for nested structures

self.json\_output(item, level + 1)

print(",")

# Closing square bracket

print(" " \* level \* indent + "]")

else:

# Print primitive values

print(f'"{data}"' if isinstance(data, str) else str(data), end="")

def parse\_timeline\_items(self):

try:

data = self.json\_data.get("data", [])

for item in data:

item\_id = item.get("id")

item\_type = item.get("type")

attributes = item.get("attributes", {})

item\_type\_attr = attributes.get("item\_type")

at = attributes.get("at")

content = attributes.get("data", {}).get("content")

links = attributes.get("data", {}).get("links")

# Convert the timestamp to a datetime object

at\_datetime = datetime.strptime(at, "%Y-%m-%dT%H:%M:%S.%fZ")

# Display or process the extracted information as needed

print(f"Item ID: {item\_id}")

print(f"Item Type: {item\_type}")

print(f"Item Type Attribute: {item\_type\_attr}")

print(f"At: {at\_datetime}")

print(f"Content: {content}")

print(f"Links: {links}")

print("\n")

except Exception as e:

print(f"Error parsing JSON: {e}")

def parse\_monitor\_data(self):

try:

# Parse JSON data

parsed\_data = self.json\_data

# Extract monitors

monitors = parsed\_data.get('data', [])

# Format and return readable output

output = ""

for monitor in monitors:

attributes = monitor.get('attributes', {})

monitor\_id = monitor.get('id', '')

monitor\_type = attributes.get('monitor\_type', '')

status = attributes.get('status', '')

url = attributes.get('url', '')

output += f"Monitor ID: {monitor\_id}\n"

output += f"Monitor Type: {monitor\_type}\n"

output += f"Status: {status}\n"

output += f"URL: {url}\n"

output += "-" \* 30 + "\n"

print(output)

return output

except json.JSONDecodeError as e:

return f"Error decoding JSON: {e}"

def parse\_incidents(self):

incidents = self.json\_data.get("data", [])

output = ""

for incident in incidents:

incident\_id = incident.get("id", "N/A")

incident\_type = incident.get("type", "N/A")

incident\_attributes = incident.get("attributes", {})

name = incident\_attributes.get("name", "N/A")

url = incident\_attributes.get("url", "N/A")

http\_method = incident\_attributes.get("http\_method", "N/A")

cause = incident\_attributes.get("cause", "N/A")

started\_at = incident\_attributes.get("started\_at", "N/A")

acknowledged\_at = incident\_attributes.get("acknowledged\_at", "N/A")

acknowledged\_by = incident\_attributes.get("acknowledged\_by", "N/A")

resolved\_at = incident\_attributes.get("resolved\_at", "N/A")

resolved\_by = incident\_attributes.get("resolved\_by", "N/A")

response\_url = incident\_attributes.get("response\_url", "N/A")

output += f"""

Incident ID: {incident\_id}

Type: {incident\_type}

Name: {name}

URL: {url}

HTTP Method: {http\_method}

Cause: {cause}

Started At: {started\_at}

Acknowledged At: {acknowledged\_at}

Acknowledged By: {acknowledged\_by}

Resolved At: {resolved\_at}

Resolved By: {resolved\_by}

Response URL: {response\_url}

"""

print(output)

return output.strip()

def parse\_monitor\_sla(self):

try:

data = self.json\_data.get("data", {})

attributes = data.get("attributes", {})

monitor\_sla = {

"id": data.get("id", "N/A"),

"type": data.get("type", "N/A"),

"availability": attributes.get("availability", "N/A"),

"total\_downtime": attributes.get("total\_downtime", "N/A"),

"number\_of\_incidents": attributes.get("number\_of\_incidents", "N/A"),

"longest\_incident": attributes.get("longest\_incident", "N/A"),

"average\_incident": attributes.get("average\_incident", "N/A"),

}

print("Parsed Monitor SLA:")

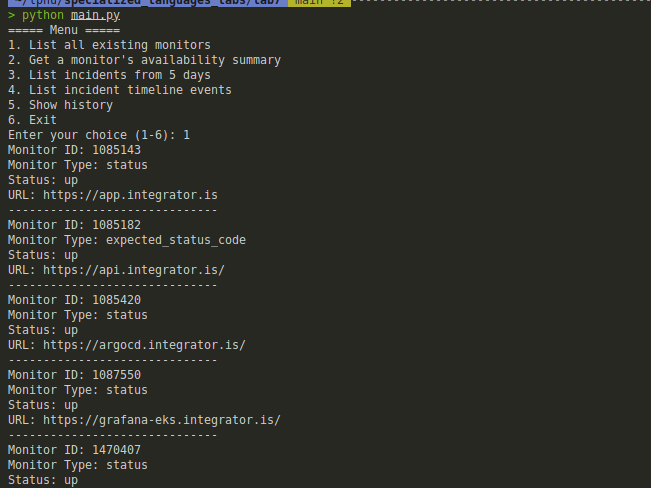
for key, value in monitor\_sla.items():

print(f"{key}: {value}")

except Exception as e:

print(f"Error parsing Monitor SLA JSON: {e}")

Результат виконання програми:



**Висновок:** Під час виконання даної лабораторної роботи було створено консольний об’єктно - орієнтований додаток÷ з використанням API.