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-32

Гірняк О. О.

Прийняв:

доц. каф. ІСМ

Щербак С. С.

Львів - 2024

**Мета**

Створення консольного об’єктно - орієнтованого додатка з використанням API та патернів проектування

**Хід виконання роботи**

**Завдання 1: Вибір провайдера API та патернів проектування**

Було обрано надійний API, який забезпечує доступ до даних через HTTP для віддаленого зберігання та вивантаження. Використовувався Google Books API. Для імплементації лабораторної роботи було обрано патерн проектування — Unit.

**Завдання 2: Інтеграція API**

Для інтеграції API в консольний додаток на Python було обрано бібліотеку для роботи з HTTP запитами — Requests. Було налаштовано необхідні облікові дані та API-ключ, а також інтегровано API у додаток після ознайомлення з його документацією.

**Завдання 3: Введення користувача**

Було розроблено користувальницький інтерфейс, який дозволяє користувачам візуалізувати всі доступні дані у табличному вигляді та у вигляді списку. Також реалізовано механізм збору та перевірки введених даних від користувача для подальшої обробки.

**Завдання 4: Розбір введення користувача**

Було створено розбірник для видобування та інтерпретації виразів користувача на основі регулярних виразів. Розбірник обробляє різні формати введення та надає зворотний зв'язок у разі виникнення помилок.

**Завдання 5: Відображення результатів**

Реалізовано логіку для візуалізації даних, отриманих через API, у консольному додатку. Результати виводилися у вигляді таблиць та списків, причому заголовки цих елементів виділялися кольором та шрифтом, який обирав користувач.

**Завдання 6: Збереження даних**

Було реалізовано можливість збереження даних у різних форматах, таких як JSON, CSV та TXT, забезпечуючи чітке і читабельне представлення даних.

**Завдання 7: Обробка помилок**

Розроблено надійний механізм обробки помилок для управління помилками API, некоректним введенням користувача та іншими можливими проблемами. Було забезпечено відображення інформативних повідомлень про помилки.

**Завдання 8: Ведення історії обчислень**

Було включено функцію реєстрації запитів користувача, зберігаючи введені запити та відповідні результати. Надано можливість користувачам переглядати та рецензувати історію своїх запитів.

**Завдання 9: Юніт-тести**

Було написано юніт-тести для перевірки функціональності додатку. Тестувалися різні операції, граничні випадки та сценарії помилок для забезпечення надійності роботи системи.

**Код програмного продукту**

from dotenv import load\_dotenv

load\_dotenv()

from labs.lab7.bll.Controller import Controller

import os

from labs.lab7.dal.JsonpickeHandler import handle

from labs.lab7.bll.GoogleBooksAPI import GoogleBooksAPI

from labs.lab7.ui.UserInterface import UserInterface

from labs.lab7.dal.SettingsModel import SettingsModel

from labs.lab7.dal.HistoryModel import HistoryModel

from labs.lab7.dal.UserSettingsModel import UserSettingsModel

from config.settings\_paths import settings\_path\_lab7

from shared.services.relative\_to\_absolute\_path import absolute

import logging

def set\_up\_logging(file\_path):

"""

Set up basic logging configuration.

:param file\_path: The path to the log file where log messages should be saved.

:return: None

"""

# Create a file handler

file\_handler = logging.FileHandler(file\_path)

file\_handler.setLevel(logging.INFO)

# Create a stream handler

stream\_handler = logging.StreamHandler()

stream\_handler.setLevel(logging.WARNING)

# Set the logging format

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

file\_handler.setFormatter(formatter)

stream\_handler.setFormatter(formatter)

# Configure the root logger

logging.basicConfig(level=logging.DEBUG,

handlers=[file\_handler, stream\_handler])

def set\_up\_models():

"""

Initializes and sets up the models required for the application.

:return: A tuple containing initialized instances of SettingsModel, HistoryModel, and UserSettingsModel

"""

settings = SettingsModel(settings\_path\_lab7)

relative\_history\_path = settings.get\_history\_path()

relative\_user\_settings\_path = settings.get\_user\_settings\_path()

results\_to\_save = settings.get\_results\_to\_save()

history\_path = absolute(relative\_history\_path)

user\_settings\_path = absolute(relative\_user\_settings\_path)

history = HistoryModel(history\_path, results\_to\_save)

user\_settings = UserSettingsModel(user\_settings\_path)

return settings, history, user\_settings

def main():

"""

Initializes the main components of the application and starts the user interface.

:return: None

"""

handle()

api\_key = os.getenv('GOOGLE\_BOOKS\_API\_KEY')

api = GoogleBooksAPI(api\_key)

settings, history, user\_settings = set\_up\_models()

logger\_path = settings.get\_logger\_path()

# Configure logging with the logger path from settings

controller = Controller(settings, history, user\_settings, api)

logger\_path = controller.get\_logger\_path()

set\_up\_logging(logger\_path)

ui = UserInterface(controller)

ui.show()

import requests

import logging

logger = logging.getLogger(\_\_name\_\_)

class GoogleBooksAPI:

"""

Class for interacting with the Google Books API.

Methods

-------

\_\_init\_\_(api\_key=None)

Initializes the GoogleBooksAPI class with an optional API key.

search(query, language='en', start\_index=0, max\_results\_count=40, fields=None)

Searches for books matching the query with optional parameters for language, start index, maximum results count, and specific fields to include in the response.

\_handle\_error(error)

Handles errors encountered during the request to the Google Books API and logs appropriate error messages.

"""

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

self.api\_key = api\_key

self.base\_url = "https://www.googleapis.com/books/v1/volumes"

def search(self, query, language='en', start\_index=0, max\_results\_count=40, fields=None):

fields = ','.join(fields) if fields else None

params = {

'q': query,

'langRestrict': language,

'startIndex': start\_index,

'maxResults': max\_results\_count,

'fields': fields

}

if self.api\_key:

params['key'] = self.api\_key

response = requests.get(self.base\_url, params=params)

try:

response.raise\_for\_status()

except requests.exceptions.RequestException as error:

return self.\_handle\_error(error)

return response.json(), response.url

def \_handle\_error(self, error):

if isinstance(error, requests.exceptions.HTTPError):

logger.error(error)

return {'error': str(error)}

elif isinstance(error, requests.exceptions.ConnectionError):

logger.critical("There is no internet connection or connection error:", error)

return {'error': 'No internet connection or connection error: ' + str(error)}

else:

logger.critical("Error not related to connection or HTTP:", error)

return {'error': str(error)}

import re

import logging

logger = logging.getLogger(\_\_name\_\_)

class InputParser:

"""

Class for parsing and querying data with support for nested fields and regular expressions.

"""

@staticmethod

def is\_regex(query\_string):

try:

re.compile(query\_string)

return True

except re.error:

logger.debug("Invalid regex")

return False

@staticmethod

def test\_is\_regex\_invalid\_regex(self):

self.assertFalse(InputParser.is\_regex("{[]{"))

@staticmethod

def parse\_query(data, field, query):

field\_list = field.split('.')

return [item for item in data if InputParser.\_matches\_query(item, field\_list, query)]

@staticmethod

def \_matches\_query(item, field\_list, query):

field\_value = InputParser.\_get\_nested\_value(item, field\_list)

return re.search(query, str(field\_value)) is not None

@staticmethod

def \_get\_nested\_value(data, fields):

for field in fields:

if isinstance(data, dict):

data = data.get(field)

else:

return None

return data

**Висновки**

Я створив проект, який надав досвід роботи з Google Book API, дизайну користувацького інтерфейсу, валідації введення, обробки помилок та тестування.