Queue Implementation Report

Introduction

The purpose of implementing a queue in our document analyzer system is to manage multiple tasks concurrently. This report outlines how queues are implemented in the [**upload.py**] script to handle PDF and text document processing and integrate these tasks seamlessly into our API.

1. Queue Implementation for PDF Analysis

To handle the processing of PDF files without stalling user interactions, we employ a queue mechanism. This is achieved by:

Creating a Task Queue: When a PDF file is uploaded, the file path and task details are enqueued in a task_queue. This queue holds all the pending tasks that need to be processed.

Starting a Processing Thread: A new thread is initiated specifically for processing each PDF file. This thread is responsible for extracting text from the PDF using the pdfplumber library. Once the text extraction is complete, the extracted text is enqueued in a result_queue, ready for the next stage of processing.

```
if filename.lower().endswith('.pdf'):
    # 启动一个线程来处理PDF
    # Starts a thread to process the PDF
    threading.Thread(target=process_pdf, args=(file_path, task_queue)).start()
elif filename.lower().endswith('.txt'):
    threading.Thread(target=process_text, args=(file_path, task_queue)).start()
```

2. Queue Implementation for NLP Analysis

Following the text extraction, the next step involves analyzing the text using Natural Language Processing (NLP):

Retrieving Text from Task Queue: The main thread waits to retrieve the extracted text from the task_queue. This blocking call ensures that the thread does not proceed until text is available.

Starting NLP Analysis Thread: Once text is retrieved, an NLP analysis thread is launched to process this text. The analysis involves calling a custom function analyze_document which performs the necessary computation and analysis.

```
text = task_queue.get()  # 等待PDF处理完成
# Waits for the PDF processing to complete

# 启动另一个线程进行NLP分析
# Starts another thread for NLP analysis
threading.Thread(target=process_nlp, args=(text, result_queue)).start()
```

3. Integration into the API

The final step involves integrating this asynchronous processing mechanism into our API:

Waiting for Analysis Results: The API endpoint waits for the NLP analysis to complete by retrieving results from the result_queue.

Response Preparation: Once the analysis is complete, the results are formatted into a JSON response, which is then returned to the client.

File Cleanup: Post-processing, temporary files are cleaned up to free resources and maintain the system's integrity.

```
analysis_result = result_queue.get()  # 等待NLP分析完成
# Waits for the NLP analysis to complete

    # 保存结果到文件
    # Saves the results to a file
save_analysis_results(analysis_result, filename)  # 调用函数保存结果 # Calls function to save results

    # 清理操作：删除临时文件
    # Cleanup operation: delete the temporary file
os.remove(file_path)

return jsonify(analysis_result)
```

Conclusion

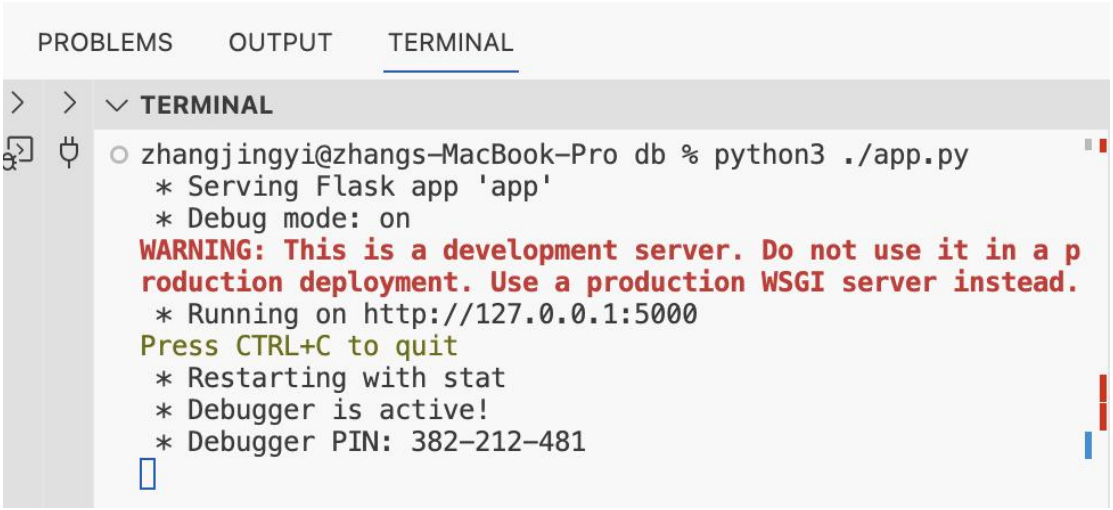
By using queues and threading in our document processing system, we manage to keep the API responsive and capable of handling multiple simultaneous requests efficiently. This setup ensures that our system can scale and handle heavy loads, especially when processing large documents or when multiple users are interacting with the system concurrently.

I will also use some screenshot to demonstrate you how two users use the nowclient.py at same time to upload documents.

Usage Demonstration

Step 0: **download all code in main branch**

Step 1 open [**app.py**] : input **python3 ./app.py** in **terminal A** to run the app (the back end).



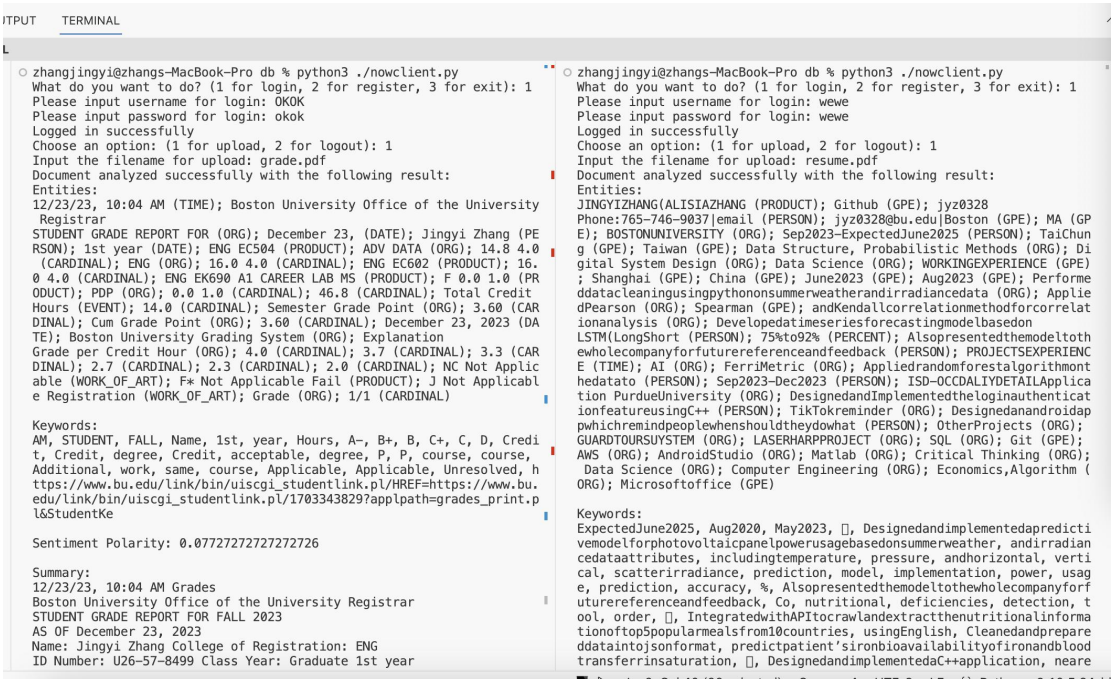
Step 2 client 1 usage : input **python3 ./nowclient.py** in **terminal B** to run the app for client 1 (the font end).

Then input 1 to login, then input username and password, then input 1 to be ready for uploading document.

Step 3 client 2 usage : input **python3 ./nowclient.py** in **terminal C** to run the app for client 2 (the font end).

Then input 1 to login, then input username and password, then input 1 to be ready for uploading document, similar with client 1 does.

Step 4 upload document for both clients: I upload [grade.pdf] for client 1, and upload [resume.pdf] for client2, so we see both analyzing report can be formed. The analyzing report is long , you can see [result_of_grade.txt] and [result_of_example.txt] since these documents print out the analyzing result of these two documents. You can use other pdf and txt documents to test.



Step 5: input 2 and 3 to exit, or upload other document you want to test.