

# 세부 Component Design (Ver1.0)

---

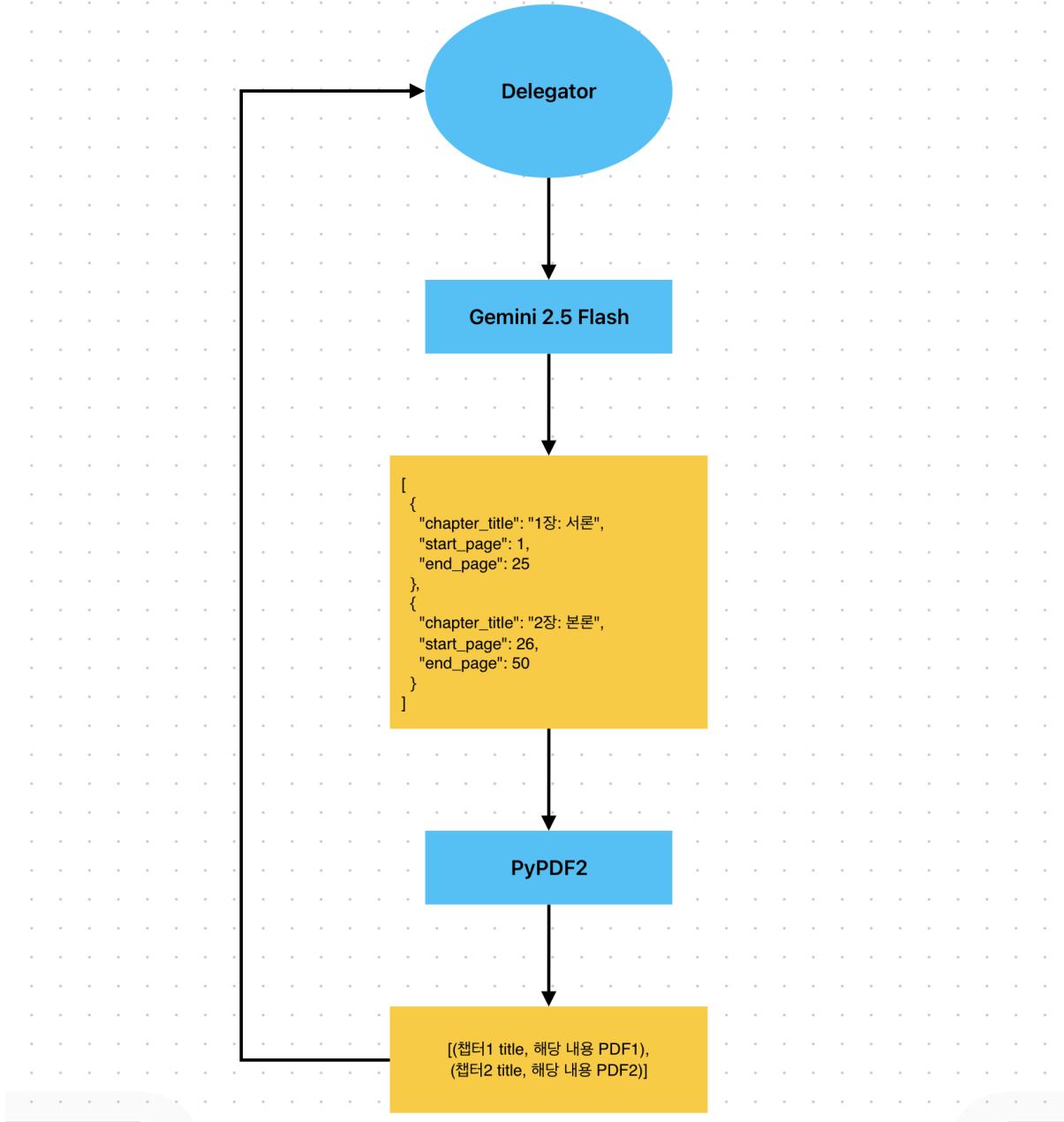
## 1) Coordinator

- 원래는 전체 흐름을 제어하는 역할
- Ver1인 지금은 별도의 구현 X
- 시스템의 전원 버튼 역할

## 2) PDF Analysis Agent

- 목표: PDF 내용을 전체적으로 분석후, 개념을 챕터 별로 나누고, 챕터별 PDF페이지를 나누어서 별도의 파일로 나누어야 함
- 입력: PDF 파일
- 출력: [(챕터1, 슬라이싱된 PDF1), (챕터2, 슬라이싱된 PDF2), ...]

### 2-1) 세부 구현 Diagram



1. Gemini 2.5 Flash에게 강의안 PDF파일을 제공
2. Gemini가 Json 형태로 → 각 챕터 title과 PDF내 해당 챕터 내용 시작 page와 끝 page를 기록함
3. PyPDF가 이를 받아, PDF파일 내용을 챕터별 페이지에 맞게 슬라이싱함
4. 슬라이싱된 파일은 별도 DB에 저장하고 이를 필요할때마다 꺼내쓸 수 있게 한다
  - 현재 구현에선, 내부 컴퓨터에 저장후 해당 PDF의 경로를 반환하게 됨

## 2-2) Gemini 2.5 Flash의 System Prompt

당신은 PDF 파일의 구조를 분석하는 전문 AI 에이전트입니다.

주어진 PDF 콘텐츠를 분석하여 각 챕터의 제목과, 해당 챕터가 시작하고 끝나는 정확한 페이지 번호를 식별하는 것이 당신의 임무입니다.

문서의 목차, 제목 스타일, 전체 구조를 종합적으로 분석하여 챕터를 구분하세요. "서론", "결론", "부록" 등도 하나하나의 챕터로 취급해야 합니다.

분석 결과는 반드시 각 챕터의 `"chapter\_title"`, `"start\_page"`, `"end\_page"`를 포함하는 JSON 배열 형식으로 반환해야 합니다. 응답에 다른 어떤 설명도 포함하지 마세요.

출력 형식 예시:

```
```json
[
  {
    "chapter_title": "1장: 서론",
    "start_page": 1,
    "end_page": 25
  },
  {
    "chapter_title": "2장: 본론",
    "start_page": 26,
    "end_page": 50
  }
]
```

- 25년 10월 28일 기준, 잘 동작하는 것을 확인

## 2-3) PyPDF 알고리즘 (코드)

- gemini의 출력이 → gemini\_ouput이라는 변수에 저장되었다고 가정
1. 각 chapter 별로 chapter\_title과 start\_page, end\_page를 가져온다
  2. 해당 내용을 바탕으로, PDF파일을 분리한다
  3. 이후 출력으로 [(챕터1 title, 챕터1PDF 경로), (챕터2 title, 챕터2 PDF 경로), ...]

### ▼ 코드 구현

```

from PyPDF2 import PdfReader, PdfWriter
import os
import re

# 상대 경로로 PDF 파일 경로 설정
pdf_path = "2025-2학기 캡스톤디자인 예산 사용 매뉴얼 3.pdf"

# PDF 파일 읽기
reader = PdfReader(pdf_path)

# 파일명에서 사용할 수 없는 특수문자 제거 함수
def sanitize_filename(filename):
    # 파일 시스템에서 사용할 수 없는 문자들을 '_'로 대체
    return re.sub(r'[^<>:\\"/?*]', '_', filename)

# PDF 파일명에서 확장자를 제거하고 폴더명 생성
pdf_name_without_ext = os.path.splitext(os.path.basename(pdf_path))[0]
output_folder = f"{pdf_name_without_ext}_split_pdf"
os.makedirs(output_folder, exist_ok=True)
print(f"저장 폴더: {output_folder}/")

# 챕터별로 PDF 슬라이싱
chapter_pdfs = []

for chapter in gemini_output:
    chapter_title = chapter["chapter_title"]
    start_page = chapter["start_page"] - 1 # 0-indexed로 변환
    end_page = chapter["end_page"] # end_page는 포함되므로 그대로 사용

    # 새로운 PDF writer 생성
    writer = PdfWriter()

    # 해당 챕터의 페이지들을 추가
    for page_num in range(start_page, end_page):
        writer.add_page(reader.pages[page_num])

```

```
# 파일명 생성 (특수문자 처리)
safe_title = sanitize_filename(chapter_title[:50])
output_filename = f"chapter_{chapter['start_page']}_{safe_title}.pdf"
output_path = os.path.join(output_folder, output_filename)

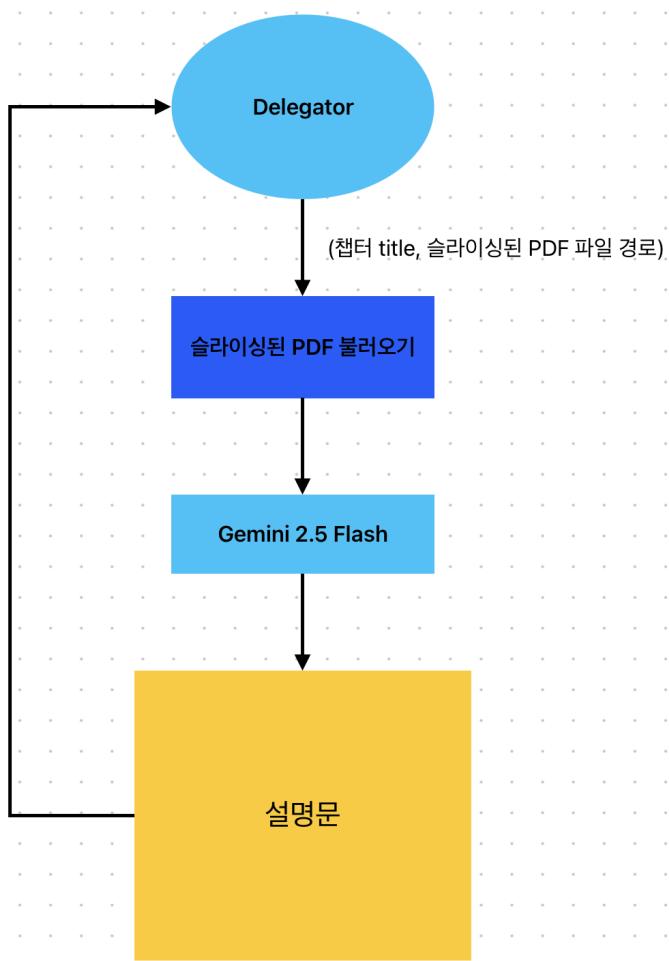
with open(output_path, "wb") as output_file:
    writer.write(output_file)

# (챕터 제목, PDF 경로) 튜플로 저장
chapter_pdfs.append((chapter_title, output_path))

print(f"생성완료: {chapter_title} (페이지 {chapter['start_page']}-{chapter['end_page']})")

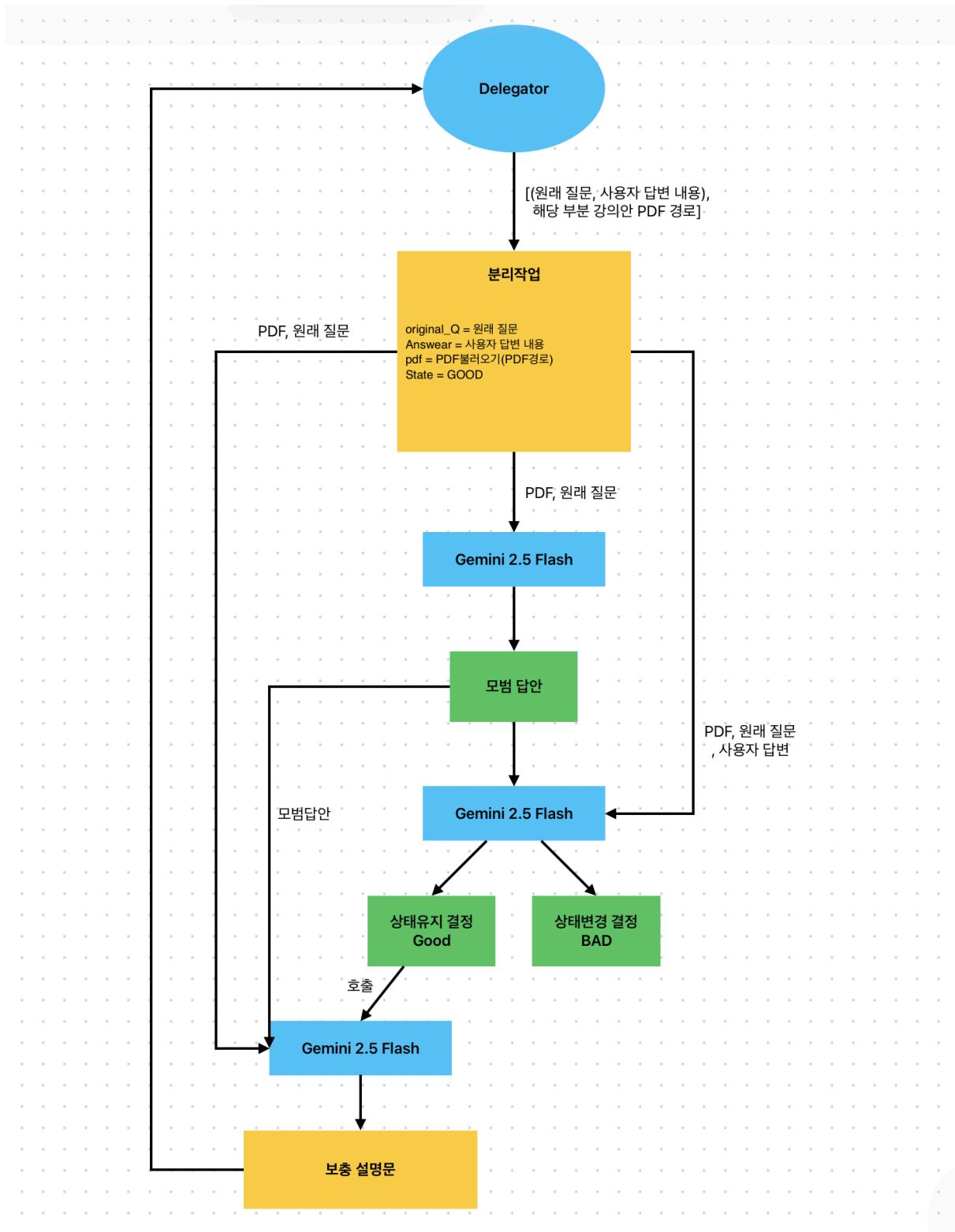
print(f"\n총 {len(chapter_pdfs)}개의 챕터 PDF가 생성되었습니다.")
print(f"저장 위치: {output_folder}/ 폴더")
print("\n결과:")
for title, path in chapter_pdfs:
    print(f"- {title}: {path}")
```

### 3) Main Lecture Agent



1. Delegator로 부터 (챕터 title, 슬라이싱된 PDF 파일 경로) 를 받는다
2. 슬라이싱된 PDF 파일 경로로부터 PDF를 읽는다
3. (챕터 title, 슬라이싱된 PDF 파일, 시스템 프롬포트) 를 Gemini 에게 입력한다
4. Gemini가 설명문을 출력한다
5. 출력한 설명문을 변수에 저장하여 Delegator에게 전달한다 (메모리 상으로 전달)
  - 향후 필요시 별도 저장

## 4) Main Q&A Agent Good Mode



1. Delegator로 부터 **[(원래 질문, 사용자 답변 내용), 해당 부분 강의안 PDF 경로]** 를 받는다

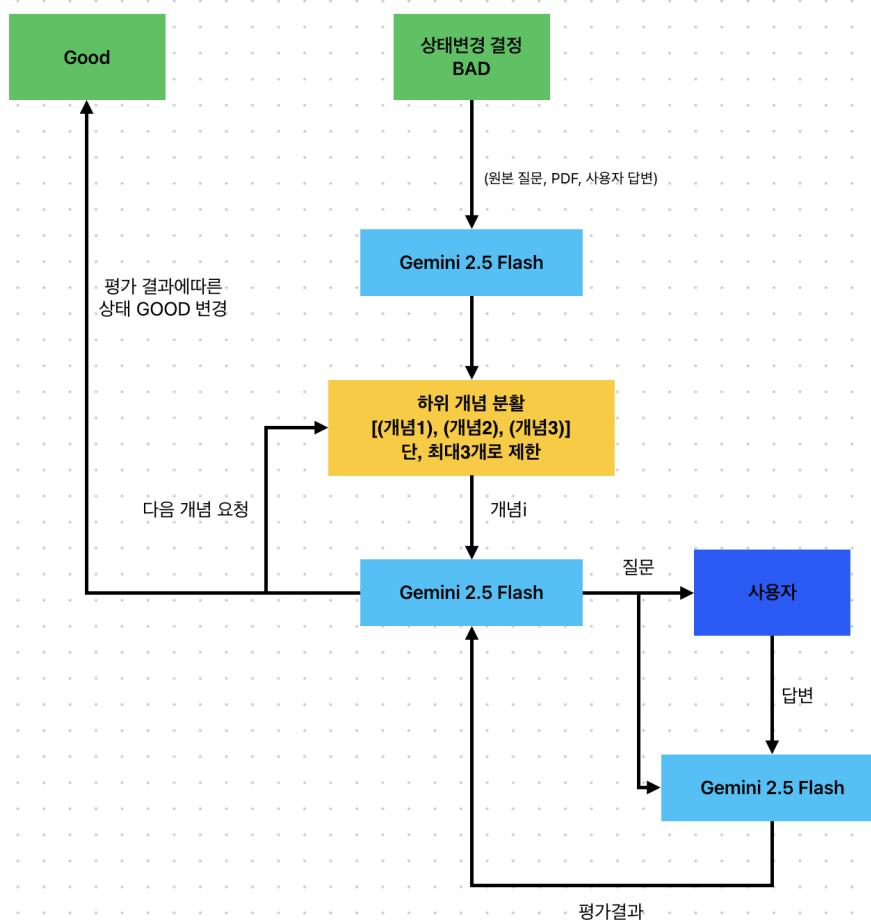
- 다만, 더 좋은 퀄리티를 위해 → 해당 부분 강의안 PDF 경로에서 전체 PDF로 바꾸고, 거기다가, 현재 개념 부분이 어디 페이지 어디인지 추가 데이터 첨부

2. 분리 작업을 수행한다

- a. original\_Q = 원래 질문
- b. Answear = 사용자 답변 내용
- c. pdf = PDF불러오기(PDF경로)
- d. State = GOOD → 향후 MQA의 방향성을 결정할 State를 GOOD으로 기본 설정

3. Gemini에게 pdf, original\_Q 를 입력하여 모범 답변을 생성하도록 한다
4. 이후 다른 Gemini에게 모범답안, pdf, Answear, original\_Q 을 입력하여 → 사용자의 답변이 괜찮다면 Good유지, 별로라면 Bad로 구분한다
5. Good모드 유지할시 → 다른 Gemini에게, pdf, originalQ, 모범답안 을 제공하여 보충 설명문을 제작한뒤 Delegator에게 전달한다

## 5) Main Q&A Agent BAD Mode



### 1. 시작 신호

- 상태가 **BAD**로 바뀌면 즉시 시작.

## 2. 개념 쪼개기

- LLM에 원본 질문, PDF, 사용자 답변을 건넨다.
- LLM이 질문을 풀기 위한 하위 개념 1~3개를 순서대로 뽑아낸다.
- 결과 예시 형태: [(개념1), (개념2), (개념3)]
- 이 목록을 개념 큐로 저장한다.

## 3. 첫 질문 띄우기

- 큐에서 개념1을 꺼낸다.
- “이걸 이해했나?”를 확인하는 진단용 질문 1개를 만들고 바로 화면에 표시한다.

## 4. 사용자 답변 받기

- 사용자가 답한다

## 5. 이해도 평가

- 질문 + 사용자 답변으로 LLM이 상/중/하로 판단한다.

## 6. 설명 + 다음 질문 동시 출력

- 방금 다른 개념1을 평가 수준에 맞춰 짧게 설명한다.
- 큐에 \*\*다음 개념(개념2)\*\*이 있으면, 그 개념에 대한 다음 질문을 같이 만든다.
- 사용자 화면에는 \*\*[개념1 설명] + [개념2 질문]\*\*이 동시에 뜬다.

## 7. 반복

- 사용자가 개념2에 답 → 평가 → 개념2 설명 + 개념3 질문...
- 큐가 빌 때까지 \*\*6)~7)\*\*을 반복한다.
- 원칙은 변하지 않는다: 설명은 항상 직전 개념, 질문은 항상 다음 개념.

## 8. 마지막 회차 처리

- 더 꺼낼 다음 개념이 없으면(“end”), 마지막 개념의 설명을 즉시 보여주고,
- 동시에 상태를 GOOD으로 전환한다. (같은 타이밍, 자연 금지)

## 9. GOOD 모드로 이어 달리기

- 이후부터는 시스템의 **GOOD 모드 로직**을 그대로 수행한다.

# 6) Delegator



## 1. 입력 단계

1. 사용자에게 PDF 경로를 입력받는다.
2. 그 경로를 PDFAnalysis에 넣어서 호출한다.

## 2. 분석 단계

1. PDFAnalysis는 PDF를 챕터 단위로 분리한다.
2. 분리된 각 챕터에 대해 동시에 실행해서, 각 챕터의 title과 그 title에 맞는 설명문을 만든다.

## 3. 설명 취합·출력 단계

1. 만들어진 설명문들을 리스트로 모아두고, 나중에 zip으로 순서대로 꺼내서 사용자에게 출력한다.

## 4. 질문 처리 단계

1. 출력 중에 [질문] 질문 내용 [/질문] 토큰이 나오면, 토큰 안에 있는 질문 내용만 사용자에게 보여준다.
2. 사용자에게서 그 질문에 대한 답변을 받는다.
3. 받은 답변과 원래 질문, 그리고 그 부분의 PDF 경로를 Main Q&A Agent에게 넘긴다.
4. Main Q&A Agent가 보충 설명문을 주면 그걸 먼저 출력한다.
5. 그다음에 원래 [질문] [/질문] 뒤에 이어지던 내용을 계속 출력한다.
6. 다시 질문 토큰이 나오면 6번부터 다시 한다.

## 5. 종료 단계

1. 모든 챕터에 대한 출력이 끝나면 종료한다.