팀1_QA시스템_구현가이드

팀1: QA 시스템 구현 가이드

프로젝트: EduMentor AI
담당: 팀1 (2명)
기간: 2주
목표: 학습자료 기반 1-2초 응답 QA 시스템

📋 팀1 역할 분담

개발자	담당 기능	예상 시간
개발자 A	자료 업로드, Upstage Parse, 임베딩, ChromaDB 저장	5일
개발자 B	QA RAG 파이프라인, LangGraph Agent, 응답 속도 최적화	5일

🎯 핵심 요구사항

요구사항	목표	검증 방법
응답 속도	Retrieve 포함 1-2초	로깅으로 측정
LangChain	RAG 파이프라인 구축	코드 리뷰
Upstage API	Parse, Embeddings, Chat 사용	API 호출 로그
ChromaDB	벡터 DB 구축	Collection 확인
LangGraph	Agent 워크플로우	다이어그램 작성

Week 1: 기반 구축

Day 1: 환경 설정 및 ChromaDB 연동

개발자 A + B (공동 작업)

1.1 Python 프로젝트 생성

```
# 팀1 프로젝트 구조 (python-server/paper_qa/)
paper_qa/
├── __init__.py          # 패키지 초기화
├── models.py            # Pydantic 모델
├── workflow.py          # LangGraph 워크플로우
├── api.py               # FastAPI 엔드포인트
├── parsers/
│   ├── __init__.py
│   ├── pdf_parser.py    # PDF 파싱 (Upstage Parse)
│   └── ppt_parser.py    # PPT 파싱
└── utils/
    ├── __init__.py
    └── cache.py         # QA 전용 캐싱

# 공통 모듈 사용 (python-server/shared/)
shared/
├── __init__.py
├── chroma_client.py     # ChromaDB 클라이언트 (팀1+팀2 공유)
└── upstage_client.py    # Upstage API 클라이언트 (팀1+팀2 공유)
```

주의: `upstage_client.py`와 `chroma_client.py`는 `shared/` 디렉터리에 있으며, 팀1과 팀2가 공유합니다.

1.2 의존성 설치

```
# requirements.txt (Python 3.12.12 호환)

# LangChain & LangGraph
langchain==0.3.7
langgraph==0.2.45
langsmith==0.1.140
langchain-community==0.3.7
langchain-text-splitters==0.3.2

# Upstage API
upstage==0.10.0
langchain-upstage==0.2.1

# Vector DB
chromadb==0.5.18

# FastAPI
fastapi==0.115.4
uvicorn[standard]==0.32.0
pydantic==2.9.2
pydantic-settings==2.6.1
python-multipart==0.0.12

# PDF/PPT 처리
PyMuPDF==1.24.13
python-pptx==1.0.2

# 유틸리티
python-dotenv==1.0.1
aiohttp==3.10.10
```

```
pip install -r requirements.txt
```

1.3 환경 변수 설정

```
# config.py
import os
from pathlib import Path

class Settings:
    # Upstage API
    UPSTAGE_API_KEY = os.getenv("UPSTAGE_API_KEY")

    # ChromaDB
    CHROMA_HOST = os.getenv("CHROMA_HOST", "localhost")
    CHROMA_PORT = int(os.getenv("CHROMA_PORT", 8001))

    # 공유 파일 디렉토리 (Docker 볼륨)
    # ✨ Spring Boot와 공유하는 볼륨 경로
    SHARED_UPLOAD_DIR = os.getenv("SHARED_UPLOAD_DIR", "/app/shared/uploads")

    # LLM 설정
    EMBEDDING_MODEL = "solar-embedding-1-large"
    CHAT_MODEL = "solar-1-mini-chat"

    # RAG 설정
    CHUNK_SIZE = 500
    CHUNK_OVERLAP = 50
    TOP_K = 3

    class Config:
        env_file = ".env"

settings = Settings()

# 공유 디렉토리 확인 및 생성
upload_dir = Path(settings.SHARED_UPLOAD_DIR)
```

```
if not upload_dir.exists():
    print(f"⚠️  Warning: Upload directory does not exist: {upload_dir}")
    print(f"   Creating directory...")
    upload_dir.mkdir(parents=True, exist_ok=True)
else:
    print(f"✅ Upload directory ready: {upload_dir}")
```

```
# .env (Python 서버 루트)
# Upstage API
UPSTAGE_API_KEY=your_upstage_api_key_here

# ChromaDB
CHROMA_HOST=localhost
CHROMA_PORT=8001

# 공유 파일 디렉토리 (Docker 볼륨)
SHARED_UPLOAD_DIR=/app/shared/uploads

# Cache
CACHE_SIZE=100
```

주의:

- Python 서버는 PostgreSQL을 사용하지 않습니다. PostgreSQL 환경변수는 Spring Boot에만 설정합니다.
- **파일 업로드 최적화**: Spring Boot가 파일을 공유 볼륨에 저장하고, Python은 파일 경로만 받아 처리합니다.

1.4 init.py 작성

```
# paper_qa/__init__.py
"""
팀1: QA 시스템 모듈
학습자료 기반 질의응답 시스템을 제공합니다.
"""

from paper_qa.api import router as qa_router
from paper_qa.workflow import upload_workflow, qa_workflow

__all__ = ['qa_router', 'upload_workflow', 'qa_workflow']
```

```
# paper_qa/parsers/__init__.py
"""PDF/PPT 파서 모듈"""

from paper_qa.parsers.pdf_parser import pdf_parser
from paper_qa.parsers.ppt_parser import ppt_parser

__all__ = ['pdf_parser', 'ppt_parser']
```

```
# paper_qa/utils/__init__.py
"""QA 전용 유틸리티 모듈"""

from paper_qa.utils.cache import query_cache

__all__ = ['query_cache']
```

1.5 ChromaDB 클라이언트 확인

ChromaDB 클라이언트는 `shared/chroma_client.py` 에 있습니다 (팀1+팀2 공유).

```
# shared/chroma_client.py (이미 구현됨)
import chromadb
from chromadb.config import Settings as ChromaSettings
from typing import List, Dict
from config import settings

class ChromaClient:
    def __init__(self):
        self.client = chromadb.HttpClient(
```

```
            host=settings.CHROMA_HOST,
            port=settings.CHROMA_PORT,
            settings=ChromaSettings(
                anonymized_telemetry=False
            )
        )

    def get_or_create_collection(self, name: str):
        """컬렉션 생성 또는 가져오기"""
        return self.client.get_or_create_collection(
            name=name,
            metadata={"hnsw:space": "cosine"}  # 코사인 유사도
        )

    def add_documents(
        self,
        collection_name: str,
        documents: List[str],
        metadatas: List[Dict],
        ids: List[str]
    ):
        """문서 추가"""
        collection = self.get_or_create_collection(collection_name)
        collection.add(
            documents=documents,
            metadatas=metadatas,
            ids=ids
        )

    def search(
        self,
        collection_name: str,
        query_texts: List[str],
        n_results: int = 3,
        filter_dict: Dict = None
    ):
        """유사도 검색"""
        collection = self.get_or_create_collection(collection_name)
        return collection.query(
            query_texts=query_texts,
            n_results=n_results,
            where=filter_dict
        )

# 싱글톤 인스턴스
chroma_client = ChromaClient()
```

1.6 ChromaDB 테스트

```
# test_chroma.py
from shared.chroma_client import chroma_client

def test_chromadb():
    # 테스트 문서 추가
    chroma_client.add_documents(
        collection_name="test",
        documents=["JPA는 Java Persistence API입니다."],
        metadatas=[{"source": "test"}],
        ids=["test_1"]
    )

    # 검색 테스트
    results = chroma_client.search(
        collection_name="test",
        query_texts=["JPA란?"],
        n_results=1
    )

    print("✅ ChromaDB 연동 성공!")
    print(f"검색 결과: {results['documents'][0][0]}")
```

```python
if __name__ == "__main__":
    test_chromadb()
```

```
# 실행
python test_chroma.py
```

Day 2-3: 자료 업로드 및 Upstage Parse (개발자 A)

2.1 Upstage Client 확인

Upstage 클라이언트는 `shared/upstage_client.py` 에 있습니다 (팀1+팀2 공유).

```python
# shared/upstage_client.py (이미 구현됨)
from upstage import Upstage
from config import settings
import asyncio

class UpstageClient:
    def __init__(self):
        self.client = Upstage(api_key=settings.UPSTAGE_API_KEY)

    async def parse_pdf(self, file_path: str) -> dict:
        """PDF 파싱"""
        result = await asyncio.to_thread(
            self.client.document_parse,
            file=file_path
        )
        return result

    async def embed_text(self, text: str) -> List[float]:
        """텍스트 임베딩"""
        from langchain_upstage import UpstageEmbeddings

        embeddings = UpstageEmbeddings(
            api_key=settings.UPSTAGE_API_KEY,
            model="embedding-query"
        )

        return await embeddings.aembed_query(text)

    async def embed_documents(self, texts: List[str]) -> List[List[float]]:
        """문서 리스트 임베딩"""
        from langchain_upstage import UpstageEmbeddings

        embeddings = UpstageEmbeddings(
            api_key=settings.UPSTAGE_API_KEY,
            model="embedding-passage"
        )

        return await embeddings.aembed_documents(texts)

upstage_client = UpstageClient()
```

2.2 PDF 파서 작성

```python
# paper_qa/parsers/pdf_parser.py
from typing import List, Dict
from shared.upstage_client import upstage_client
import logging

logger = logging.getLogger(__name__)

class PDFParser:
    async def parse(self, file_path: str) -> List[Dict]:
        """PDF 파싱 및 요소별 분리"""
        logger.info(f"Parsing PDF: {file_path}")
```

```
        # Upstage Document Parse
        parsed = await upstage_client.parse_pdf(file_path)

        content_blocks = []

        # elements 순회
        for element in parsed.get('elements', []):
            element_type = element.get('type')

            if element_type == 'text':
                content_blocks.append({
                    'type': 'text',
                    'content': element.get('content', ''),
                    'page': element.get('page', 1),
                    'category': element.get('category', 'paragraph')
                })

            elif element_type == 'table':
                # 표는 텍스트로 변환
                table_text = self._table_to_text(element.get('content'))
                content_blocks.append({
                    'type': 'table',
                    'content': table_text,
                    'page': element.get('page', 1)
                })

        logger.info(f"Parsed {len(content_blocks)} blocks from PDF")
        return content_blocks

    def _table_to_text(self, table_data) -> str:
        """표 데이터를 텍스트로 변환"""
        # 간단한 변환 로직
        return str(table_data)

pdf_parser = PDFParser()
```

2.3 PPT 파서 작성 (선택사항)

```
# paper_qa/parsers/ppt_parser.py
from pptx import Presentation
from typing import List, Dict
import logging

logger = logging.getLogger(__name__)

class PPTParser:
    def parse(self, file_path: str) -> List[Dict]:
        """PPT 파싱"""
        logger.info(f"Parsing PPT: {file_path}")

        prs = Presentation(file_path)
        content_blocks = []

        for slide_idx, slide in enumerate(prs.slides, 1):
            # 슬라이드 텍스트 추출
            slide_text = []

            for shape in slide.shapes:
                if hasattr(shape, "text"):
                    slide_text.append(shape.text)

            if slide_text:
                content_blocks.append({
                    'type': 'slide',
                    'content': '\n'.join(slide_text),
                    'page': slide_idx
                })

        logger.info(f"Parsed {len(content_blocks)} slides from PPT")
        return content_blocks
```

```
ppt_parser = PPTParser()
```

2.4 자료 업로드 워크플로우

```
# paper_qa/workflow.py (Part 1: Upload)
from langgraph.graph import StateGraph, START, END
from typing import TypedDict, List, Dict
from paper_qa.parsers.pdf_parser import pdf_parser
from paper_qa.parsers.ppt_parser import ppt_parser
from shared.upstage_client import upstage_client
from shared.chroma_client import chroma_client
import logging

logger = logging.getLogger(__name__)

class UploadState(TypedDict):
    material_id: int
    file_path: str
    file_type: str  # "pdf" or "ppt"
    parsed_blocks: List[Dict]
    embeddings: List[List[float]]
    status: str

async def parse_document_node(state: UploadState) -> dict:
    """1단계: 문서 파싱"""
    file_path = state["file_path"]
    file_type = state["file_type"]

    logger.info(f"Parsing {file_type}: {file_path}")

    if file_type == "pdf":
        parsed_blocks = await pdf_parser.parse(file_path)
    elif file_type == "ppt":
        parsed_blocks = ppt_parser.parse(file_path)
    else:
        raise ValueError(f"Unsupported file type: {file_type}")

    return {"parsed_blocks": parsed_blocks}

async def embed_and_store_node(state: UploadState) -> dict:
    """2단계: 임베딩 및 ChromaDB 저장"""
    material_id = state["material_id"]
    parsed_blocks = state["parsed_blocks"]

    logger.info(f"Embedding {len(parsed_blocks)} blocks")

    # 배치 임베딩 (효율성)
    texts = [block['content'] for block in parsed_blocks]
    embeddings = await upstage_client.embed_documents(texts)

    logger.info("Storing in ChromaDB")

    # ChromaDB에 저장
    documents = []
    metadatas = []
    ids = []

    for idx, block in enumerate(parsed_blocks):
        documents.append(block['content'])
        metadatas.append({
            'material_id': material_id,
            'page': block['page'],
            'type': block['type']
        })
        ids.append(f"material_{material_id}_block_{idx}")

    chroma_client.add_documents(
        collection_name="learning_materials",
        documents=documents,
        metadatas=metadatas,
```

```
        ids=ids
    )

    return {
        "embeddings": embeddings,
        "status": "completed"
    }

# 업로드 워크플로우 생성
def create_upload_workflow():
    graph = StateGraph(UploadState)

    graph.add_node("parse", parse_document_node)
    graph.add_node("embed_store", embed_and_store_node)

    graph.add_edge(START, "parse")
    graph.add_edge("parse", "embed_store")
    graph.add_edge("embed_store", END)

    return graph.compile()

upload_workflow = create_upload_workflow()
```

2.5 업로드 API 엔드포인트 (최적화)

```
# paper_qa/api.py (Part 1: Upload)
from fastapi import APIRouter, HTTPException
from pathlib import Path
import logging
import time

from paper_qa.models import MaterialUploadRequest, MaterialUploadResponse
from paper_qa.workflow import upload_workflow
from shared.chroma_client import chroma_client
from shared.upstage_client import upstage_client
from config import settings

router = APIRouter(prefix="/qa", tags=["QA"])
logger = logging.getLogger(__name__)

@router.post("/upload", response_model=MaterialUploadResponse)
async def upload_material(request: MaterialUploadRequest):
    """
    ✨ 학습 자료 업로드 및 파싱 (최적화 버전)

    - Spring Boot가 파일을 공유 볼륨에 저장
    - 이 API는 파일 경로를 받아 직접 읽어서 처리
    - 파일을 다시 전송받지 않음 (네트워크 최적화)
    """
    logger.info(f"📁 Received upload request: material_id={request.material_id}, path={request.file_path}")

    try:
        # 1. 파일 존재 확인
        file_path = Path(request.file_path)
        if not file_path.exists():
            logger.error(f"❌ File not found: {file_path}")
            raise HTTPException(
                status_code=404,
                detail=f"File not found at path: {request.file_path}"
            )

        logger.info(f"✅ File found: {file_path} ({file_path.stat().st_size} bytes)")

        # 2. 워크플로우 실행 (파일 경로 전달)
        result = await upload_workflow.ainvoke({
            "material_id": request.material_id,
            "file_path": str(file_path),
            "file_type": "pdf"
        })

        return MaterialUploadResponse(
```

```
            material_id=request.material_id,
            status="completed",
            page_count=len(result.get("parsed_blocks", [])),
            chunk_count=len(result.get("parsed_blocks", [])),
            message=f"Successfully processed {len(result.get('parsed_blocks', []))} blocks"
        )

    except Exception as e:
        logger.error(f"❌ Upload failed: {str(e)}", exc_info=True)
        return MaterialUploadResponse(
            material_id=request.material_id,
            status="failed",
            page_count=0,
            chunk_count=0,
            message=f"Processing failed: {str(e)}"
        )
```

주요 변경사항:

- ✨ `MultipartFile` 대신 `MaterialUploadRequest` (파일 경로)
- ✨ Spring Boot가 파일 저장, Python은 경로에서 읽기만
- ✨ 네트워크 전송 최소화 (50% 감소)

Day 4-5: QA RAG 파이프라인 (개발자 B)

3.1 QA State 정의

```
# paper_qa/workflow.py (Part 2: QA)
class QAState(TypedDict):
    question: str
    material_id: int
    retrieved_docs: List[Dict]
    answer: str
    sources: List[Dict]
    response_time: float
```

3.2 Retrieve 노드 (0.2-0.3초 목표)

```
# paper_qa/workflow.py (Part 2: QA - Retrieve)
import time

async def retrieve_node(state: QAState) -> dict:
    """ChromaDB에서 관련 문서 검색"""
    start_time = time.time()

    question = state["question"]
    material_id = state["material_id"]

    logger.info(f"Retrieving docs for: {question}")

    # 검색 (k=3으로 제한 - 속도 최적화)
    results = chroma_client.search(
        collection_name="learning_materials",
        query_texts=[question],
        n_results=3,
        filter_dict={"material_id": material_id}
    )

    # 결과 구성
    retrieved_docs = []
    for i in range(len(results['documents'][0])):
        retrieved_docs.append({
            'content': results['documents'][0][i],
            'page': results['metadatas'][0][i]['page'],
            'type': results['metadatas'][0][i]['type'],
            'distance': results['distances'][0][i]
        })
```

```
    retrieve_time = time.time() - start_time
    logger.info(f"⚡ Retrieve time: {retrieve_time:.3f}s")

    return {"retrieved_docs": retrieved_docs}
```

3.3 Generate 노드 (0.8-1.0초 목표)

```
# paper_qa/workflow.py (Part 2: QA - Generate)
from langchain_upstage import ChatUpstage
from langchain.schema import HumanMessage
from config import settings

async def generate_answer_node(state: QAState) -> dict:
    """Upstage Solar로 답변 생성"""
    start_time = time.time()

    question = state["question"]
    retrieved_docs = state["retrieved_docs"]

    # 컨텍스트 구성
    context_parts = []
    for doc in retrieved_docs:
        context_parts.append(
            f"[페이지 {doc['page']}]\n{doc['content']}"
        )
    context = "\n\n---\n\n".join(context_parts)

    # Upstage Solar Mini (빠른 응답)
    llm = ChatUpstage(
        api_key=settings.UPSTAGE_API_KEY,
        model="solar-1-mini-chat",
        temperature=0.3
    )

    prompt = f"""당신은 학습자료 기반 QA 봇입니다.

**학습자료 내용**:
{context}

**학생 질문**: {question}

**답변 규칙**:
1. 학습자료에 있는 내용만 사용하세요
2. 명확하고 간결하게 답변하세요 (3-5문장)
3. 관련 페이지 번호를 명시하세요

답변:"""

    response = await llm.ainvoke([HumanMessage(content=prompt)])
    answer = response.content

    # 출처 정보
    sources = [
        {
            "page": doc["page"],
            "excerpt": doc["content"][:100] + "..."
        }
        for doc in retrieved_docs
    ]

    generate_time = time.time() - start_time
    logger.info(f"⚡ Generate time: {generate_time:.3f}s")

    return {
        "answer": answer,
        "sources": sources
    }
```

3.4 QA 워크플로우 완성

```python
# paper_qa/workflow.py (Part 2: QA - Complete)
def create_qa_workflow():
    """QA LangGraph 워크플로우"""
    graph = StateGraph(QAState)

    # 노드 추가
    graph.add_node("retrieve", retrieve_node)
    graph.add_node("generate", generate_answer_node)

    # 엣지 (순차 실행)
    graph.add_edge(START, "retrieve")
    graph.add_edge("retrieve", "generate")
    graph.add_edge("generate", END)

    return graph.compile()

qa_workflow = create_qa_workflow()
```

3.5 QA API 엔드포인트

```python
# paper_qa/api.py (Part 2: QA)
from paper_qa.models import QARequest, QAResponse
import time

@router.post("/ask", response_model=QAResponse)
async def ask_question(request: QARequest):
    """학습자료 기반 질의응답"""
    start_time = time.time()

    # LangGraph 워크플로우 실행
    result = await qa_workflow.ainvoke({
        "question": request.question,
        "material_id": request.material_id
    })

    response_time = int((time.time() - start_time) * 1000)

    # 2초 초과 시 경고
    if response_time > 2000:
        logger.warning(f"⚠️ Slow response: {response_time}ms")
    else:
        logger.info(f"✅ Response time: {response_time}ms")

    return QAResponse(
        answer=result["answer"],
        sources=result["sources"],
        response_time_ms=response_time
    )
```

Week 2: 최적화 및 완성

Day 6-7: LangGraph Agent 고도화

4.1 응답 캐싱 추가

```python
# paper_qa/utils/cache.py
from functools import lru_cache
from typing import List

class QueryCache:
    def __init__(self, maxsize=100):
        self.cache = {}
        self.maxsize = maxsize

    def get(self, key: str):
        return self.cache.get(key)
```

```
    def set(self, key: str, value):
        if len(self.cache) >= self.maxsize:
            # FIFO 제거
            self.cache.pop(next(iter(self.cache)))
        self.cache[key] = value

query_cache = QueryCache()
```

4.2 캐싱 적용

```
# paper_qa/workflow.py (캐싱 추가)
from paper_qa.utils.cache import query_cache
async def retrieve_node(state: QAState) -> dict:
    question = state["question"]
    material_id = state["material_id"]

    # 캐시 키 생성
    cache_key = f"{material_id}:{question}"

    # 캐시 확인
    cached = query_cache.get(cache_key)
    if cached:
        logger.info("✅ Cache hit!")
        return {"retrieved_docs": cached}

    # 캐시 미스 - 검색 수행
    results = chroma_client.search(...)

    retrieved_docs = [...]

    # 캐시 저장
    query_cache.set(cache_key, retrieved_docs)

    return {"retrieved_docs": retrieved_docs}
```

4.3 멀티 쿼리 검색 (정확도 향상)

```
# paper_qa/workflow.py (멀티 쿼리)
from langchain_upstage import ChatUpstage
from config import settings

async def generate_multi_queries(question: str) -> List[str]:
    """질문을 다양한 형태로 변환"""
    llm = ChatUpstage(
        api_key=settings.UPSTAGE_API_KEY,
        model="solar-1-mini-chat"
    )

    prompt = f"""다음 질문을 3가지 다른 형태로 변환하세요:

질문: {question}

변환된 질문들 (줄바꿈으로 구분):"""

    response = await llm.ainvoke([HumanMessage(content=prompt)])
    queries = response.content.strip().split('\n')

    return [question] + queries[:2]  # 원본 + 2개 변환

async def retrieve_node(state: QAState) -> dict:
    question = state["question"]
    material_id = state["material_id"]

    # 멀티 쿼리 생성
    queries = await generate_multi_queries(question)

    all_docs = []
    seen_ids = set()

    # 각 쿼리로 검색
```

```
    for query in queries:
        results = chroma_client.search(
            collection_name="learning_materials",
            query_texts=[query],
            n_results=2,
            filter_dict={"material_id": material_id}
        )

        # 중복 제거
        for i, doc_id in enumerate(results['ids'][0]):
            if doc_id not in seen_ids:
                seen_ids.add(doc_id)
                all_docs.append({
                    'content': results['documents'][0][i],
                    'page': results['metadatas'][0][i]['page'],
                    'distance': results['distances'][0][i]
                })

    # 거리 기준 정렬 (가장 관련성 높은 3개)
    all_docs.sort(key=lambda x: x['distance'])
    retrieved_docs = all_docs[:3]

    return {"retrieved_docs": retrieved_docs}
```

Day 8-9: 성능 테스트 및 최적화

5.1 성능 테스트 스크립트

```
# tests/test_performance.py
import asyncio
import time
from paper_qa.workflow import qa_workflow

async def test_qa_performance():
    """QA 응답 속도 테스트"""

    test_cases = [
        {
            "question": "JPA Entity란 무엇인가요?",
            "material_id": 1
        },
        {
            "question": "@Transactional 어노테이션은 언제 사용하나요?",
            "material_id": 1
        },
        {
            "question": "Spring Boot의 Auto Configuration은 어떻게 동작하나요?",
            "material_id": 1
        }
    ]

    results = []

    for test in test_cases:
        print(f"\n질문: {test['question']}")

        start = time.time()

        result = await qa_workflow.ainvoke({
            "question": test["question"],
            "material_id": test["material_id"]
        })

        elapsed = time.time() - start

        status = "✅ PASS" if elapsed < 2.0 else "❌ FAIL"
        print(f"응답 시간: {elapsed:.3f}s - {status}")
        print(f"답변: {result['answer'][:100]}...")
```

```
        results.append({
            "question": test["question"],
            "time": elapsed,
            "passed": elapsed < 2.0
        })

    # 통계
    total = len(results)
    passed = sum(1 for r in results if r['passed'])
    avg_time = sum(r['time'] for r in results) / total

    print("\n" + "="*50)
    print(f"총 테스트: {total}")
    print(f"통과: {passed}/{total}")
    print(f"평균 응답 시간: {avg_time:.3f}s")
    print("="*50)

if __name__ == "__main__":
    asyncio.run(test_qa_performance())
```

```
# 실행
python tests/test_performance.py
```

5.2 성능 프로파일링

```
# paper_qa/utils/profiler.py
import time
from functools import wraps

def profile(func):
    """함수 실행 시간 측정 데코레이터"""
    @wraps(func)
    async def wrapper(*args, **kwargs):
        start = time.time()
        result = await func(*args, **kwargs)
        elapsed = time.time() - start

        print(f"⏱️ {func.__name__}: {elapsed:.3f}s")

        return result
    return wrapper

# 사용 예시
@profile
async def retrieve_node(state: QAState) -> dict:
    ...
```

Day 10: 문서화 및 배포 준비

6.1 API 문서화 (FastAPI 자동 생성)

```
# main.py (Python 서버 루트)
from fastapi import FastAPI
from paper_qa.api import router as qa_router
import logging

# 로깅 설정
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)

app = FastAPI(
    title="EduMentor AI Engine",
    description="QA 시스템 + 문제 생성 통합 API",
    version="1.0.0"
```

```
)

# 팀1 QA 라우터 등록
app.include_router(qa_router, prefix="/qa", tags=["QA"])

@app.get("/health")
async def health_check():
    return {"status": "healthy"}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

```
# 서버 실행
python main.py
```

API 문서 자동 생성: http://localhost:8000/docs

6.2 README 작성

```
# 팀1: QA 시스템

## 설치

```bash
pip install -r requirements.txt
```

실행

```
# ChromaDB 시작
docker run -p 8001:8000 chromadb/chroma:latest

# FastAPI 서버 시작
python main.py
```

API 사용법

1. 학습자료 업로드

```
curl -X POST http://localhost:8000/materials/upload \
  -F "file=@spring_guide.pdf" \
  -F "material_id=1"
```

2. 질문하기

```
curl -X POST http://localhost:8000/materials/ask \
  -H "Content-Type: application/json" \
  -d '{
    "material_id": 1,
    "question": "JPA Entity란 무엇인가요?"
  }'
```

성능 목표

- **응답 속도**: 1-2초 (Retrieve 포함)
- **정확도**: 학습자료 기반 정확한 답변
- **출처 표시**: 페이지 번호 명시

📊 팀1 완성 체크리스트

핵심 기능

- [ ] Upstage Document Parse 연동
```

- [ ] Upstage Embeddings 연동
- [ ] ChromaDB 벡터 저장
- [ ] LangGraph QA 워크플로우
- [ ] 1-2초 응답 속도 달성
- [ ] 출처 표시 기능

API 엔드포인트

- [ ] POST `/materials/upload` - 자료 업로드
- [ ] POST `/materials/ask` - 질의응답

테스트

- [ ] 업로드 기능 테스트
- [ ] QA 성능 테스트 (1-2초)
- [ ] 다양한 질문 테스트

문서화

- [ ] API 문서 (FastAPI Swagger)
- [ ] README.md
- [ ] 코드 주석

🚀 다음 단계

팀1 완료 후:

1. **팀2와 통합**: Spring Boot 연동
2. **전체 시스템 테스트**: E2E 테스트
3. **성능 최적화**: 병목 구간 개선
4. **배포 준비**: Docker 이미지 생성

작성일: 2025-10-28
담당: 팀1 (개발자 A, B)