Python_통합_구현가이드

Python 통합 구현 가이드 (팀1 + 팀2)

프로젝트: EduMentor AI - Python 서버 (FastAPI)
포트: 8000 (단일 서버)
구성: 팀1 QA 모듈 + 팀2 문제 생성 모듈

📁 통합 프로젝트 구조

```
python-server/
├── main.py                     # FastAPI 메인 애플리케이션
├── config.py                   # 공통 설정
├── requirements.txt            # Python 의존성
├── .env                        # 환경 변수
│
├── shared/                     # 공통 모듈 (팀1+팀2 공유)
│   ├── __init__.py             # 패키지 초기화
│   ├── chroma_client.py        # ChromaDB 클라이언트 (싱글톤)
│   └── upstage_client.py       # Upstage API 클라이언트 (싱글톤)
│
├── paper_qa/                   # 팀1: QA 시스템 (/qa)
│   ├── __init__.py             # 패키지 초기화
│   ├── workflow.py             # LangGraph QA 워크플로우
│   ├── api.py                  # QA API 엔드포인트
│   ├── models.py               # Pydantic 모델
│   ├── parsers/
│   │   ├── __init__.py
│   │   ├── pdf_parser.py       # PDF 파싱 (Upstage Parse)
│   │   └── ppt_parser.py       # PPT 파싱
│   └── utils/
│       ├── __init__.py
│       └── cache.py            # QA 전용 캐싱 로직
│
└── paper_problem/              # 팀2: 문제 생성 (/problems)
    ├── __init__.py             # 패키지 초기화
    ├── workflow.py             # LangGraph 문제 생성 워크플로우
    ├── api.py                  # 문제 생성 API 엔드포인트
    ├── models.py               # Pydantic 모델
    ├── generators/
    │   ├── __init__.py
    │   ├── beginner.py         # 초급 문제 생성기
    │   ├── intermediate.py     # 중급 문제 생성기
    │   └── advanced.py         # 고급 문제 생성기
    ├── validators/
    │   ├── __init__.py
    │   └── problem_validator.py  # 문제 검증 로직
    └── utils/
        ├── __init__.py
        └── content_analyzer.py   # 문제생성 전용 내용 분석
```

디렉터리 역할

디렉터리	역할	공유 여부
shared/	ChromaDB, Upstage API 공통 클라이언트	팀1+팀2 공유
paper_qa/utils/	QA 전용 캐싱 로직	팀1 전용
paper_problem/utils/	학습 내용 분석 로직	팀2 전용

중요: `shared/`의 `chroma_client.py`와 `upstage_client.py`는 두 팀이 공유합니다.

---

1. 프로젝트 초기 설정

1.1 의존성 설치

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

# LangChain & LangGraph
langchain-core==0.3.7
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>=0.1
langchain-upstage>=0.1

# 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

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

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

1.2 환경 변수 설정

```
# config.py
from pydantic_settings import BaseSettings

class Settings(BaseSettings):
    # Upstage API
    UPSTAGE_API_KEY: str

    # ChromaDB
    CHROMA_HOST: str = "localhost"
    CHROMA_PORT: int = 8001

    # File Storage
    UPLOAD_DIR: str = "./uploads"

    # Cache
    CACHE_SIZE: int = 100

    class Config:
        env_file = ".env"

settings = Settings()
```

```
# .env
# Upstage API
UPSTAGE_API_KEY=your_upstage_api_key

# ChromaDB
CHROMA_HOST=localhost
CHROMA_PORT=8001

# File Storage
```

```
UPLOAD_DIR=./data/materials

# Cache
CACHE_SIZE=100
```

주의: Python 서버는 PostgreSQL을 사용하지 않습니다. PostgreSQL 환경변수는 Spring Boot의 `application.yml`에만 설정하세요.

---

2. 공통 모듈 (shared/)

2.1 init.py

```python
# shared/__init__.py
"""
공통 모듈 패키지
ChromaDB 클라이언트와 Upstage API 클라이언트를 제공합니다.
"""

from shared.chroma_client import chroma_client
from shared.upstage_client import upstage_client

__all__ = ['chroma_client', 'upstage_client']
```

2.2 ChromaDB 클라이언트

```python
# shared/chroma_client.py
import chromadb
from chromadb.config import Settings as ChromaSettings
from typing import List, Dict
from config import settings
import logging

logger = logging.getLogger(__name__)

class ChromaClient:
    """ChromaDB 클라이언트 (팀1, 팀2 공유)"""

    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls._instance._initialized = False
        return cls._instance

    def __init__(self):
        if self._initialized:
            return

        self.client = chromadb.HttpClient(
            host=settings.CHROMA_HOST,
            port=settings.CHROMA_PORT,
            settings=ChromaSettings(
                anonymized_telemetry=False
            )
        )
        self._initialized = True
        logger.info("ChromaDB client initialized")

    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"}
        )

    def add_documents(
        self,
```

```
        collection_name: str,
        documents: List[str],
        metadatas: List[Dict],
        ids: List[str],
        embeddings: List[List[float]] = None
    ):
        """문서 추가"""
        collection = self.get_or_create_collection(collection_name)

        if embeddings:
            collection.add(
                documents=documents,
                metadatas=metadatas,
                ids=ids,
                embeddings=embeddings
            )
        else:
            collection.add(
                documents=documents,
                metadatas=metadatas,
                ids=ids
            )

    def search(
        self,
        collection_name: str,
        query_texts: List[str] = None,
        query_embeddings: List[List[float]] = None,
        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,
            query_embeddings=query_embeddings,
            n_results=n_results,
            where=filter_dict
        )

# 싱글톤 인스턴스
chroma_client = ChromaClient()
```

2.3 Upstage 클라이언트

```
# shared/upstage_client.py
from upstage import Upstage
from langchain_upstage import UpstageEmbeddings, ChatUpstage
from config import settings
from typing import List
import asyncio
import logging

logger = logging.getLogger(__name__)

class UpstageClient:
    """Upstage API 클라이언트 (팀1, 팀2 공유)"""

    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls._instance._initialized = False
        return cls._instance

    def __init__(self):
        if self._initialized:
            return
```

```
        self.client = Upstage(api_key=settings.UPSTAGE_API_KEY)
        self.embeddings = UpstageEmbeddings(
            api_key=settings.UPSTAGE_API_KEY,
            model="embedding-query"
        )
        self._initialized = True
        logger.info("Upstage client initialized")

    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_query(self, text: str) -> List[float]:
        """쿼리 임베딩"""
        return await self.embeddings.aembed_query(text)

    async def embed_documents(self, texts: List[str]) -> List[List[float]]:
        """문서 리스트 임베딩"""
        embeddings = UpstageEmbeddings(
            api_key=settings.UPSTAGE_API_KEY,
            model="embedding-passage"
        )
        return await embeddings.aembed_documents(texts)

    def get_chat_model(self, model: str = "solar-1-mini-chat", temperature: float = 0.3):
        """Chat 모델 반환"""
        return ChatUpstage(
            api_key=settings.UPSTAGE_API_KEY,
            model=model,
            temperature=temperature
        )

# 싱글톤 인스턴스
upstage_client = UpstageClient()
```

---

3. 팀1: QA 시스템

3.1 init.py

```
# 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']
```

3.2 Pydantic 모델

```
# paper_qa/models.py
from pydantic import BaseModel
from typing import List, Dict

class UploadResponse(BaseModel):
    material_id: int
    status: str
    blocks_count: int

class QARequest(BaseModel):
    material_id: int
    question: str

class QAResponse(BaseModel):
    answer: str
    sources: List[Dict]
    response_time_ms: int
```

3.3 PDF 파서

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

logger = logging.getLogger(__name__)

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

        parsed = await upstage_client.parse_pdf(file_path)
        content_blocks = []

        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)
                })

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

pdf_parser = PDFParser()
```

3.4 QA 워크플로우

```
# paper_qa/workflow.py
from langgraph.graph import StateGraph, START, END
from typing import TypedDict, List, Dict
from shared.chroma_client import chroma_client
from shared.upstage_client import upstage_client
from paper_qa.parsers.pdf_parser import pdf_parser
from langchain.schema import HumanMessage
import logging
import time

logger = logging.getLogger(__name__)
```

```python
# 업로드 State
class UploadState(TypedDict):
    material_id: int
    file_path: str
    file_type: str
    parsed_blocks: List[Dict]
    status: str

# QA State
class QAState(TypedDict):
    question: str
    material_id: int
    retrieved_docs: List[Dict]
    answer: str
    sources: List[Dict]

# ============ 업로드 워크플로우 ============
async def parse_document_node(state: UploadState) -> dict:
    """문서 파싱"""
    file_path = state["file_path"]
    parsed_blocks = await pdf_parser.parse(file_path)
    return {"parsed_blocks": parsed_blocks}

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

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

    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,
        embeddings=embeddings
    )

    return {"status": "completed"}

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()

# ============ QA 워크플로우 ============
async def retrieve_node(state: QAState) -> dict:
    """ChromaDB에서 문서 검색 (0.2-0.3초 목표)"""
    start_time = time.time()

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

```
    # 질문 임베딩
    query_embedding = await upstage_client.embed_query(question)

    # ChromaDB 검색
    results = chroma_client.search(
        collection_name="learning_materials",
        query_embeddings=[query_embedding],
        n_results=3,
        filter_dict={"material_id": material_id}
    )

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

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

    return {"retrieved_docs": retrieved_docs}

async def generate_answer_node(state: QAState) -> dict:
    """답변 생성 (0.8-1.0초 목표)"""
    start_time = time.time()

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

    context = "\n\n---\n\n".join([
        f"[페이지 {doc['page']}]\n{doc['content']}"
        for doc in retrieved_docs
    ])

    llm = upstage_client.get_chat_model(
        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

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

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

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

def create_qa_workflow():
    graph = StateGraph(QAState)
    graph.add_node("retrieve", retrieve_node)
    graph.add_node("generate", generate_answer_node)
```

```
    graph.add_edge(START, "retrieve")
    graph.add_edge("retrieve", "generate")
    graph.add_edge("generate", END)
    return graph.compile()

# 워크플로우 인스턴스
upload_workflow = create_upload_workflow()
qa_workflow = create_qa_workflow()
```

3.5 QA API

```
# paper_qa/api.py
from fastapi import APIRouter, UploadFile, HTTPException
from paper_qa.models import UploadResponse, QARequest, QAResponse
from paper_qa.workflow import upload_workflow, qa_workflow
from config import settings
import shutil
import os
import time

router = APIRouter()
```

```
@router.post("/upload", response_model=UploadResponse)
async def upload_material(file: UploadFile, material_id: int):
    """학습자료 업로드 및 파싱"""

    file_ext = file.filename.split('.')[-1].lower()
    if file_ext not in ['pdf']:
        raise HTTPException(status_code=400, detail="Only PDF files are supported")

    os.makedirs(settings.UPLOAD_DIR, exist_ok=True)
    file_path = os.path.join(settings.UPLOAD_DIR, file.filename)

    with open(file_path, "wb") as buffer:
        shutil.copyfileobj(file.file, buffer)

    result = await upload_workflow.ainvoke({
        "material_id": material_id,
        "file_path": file_path,
        "file_type": "pdf"
    })

    return UploadResponse(
        material_id=material_id,
        status=result["status"],
        blocks_count=len(result["parsed_blocks"])
    )
```

```
@router.post("/ask", response_model=QAResponse)
async def ask_question(request: QARequest):
    """질의응답 (1-2초 목표)"""
    start_time = time.time()

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

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

    if response_time > 2000:
        print(f"⚠️ Slow response: {response_time}ms")
    else:
        print(f"✅ Response time: {response_time}ms")

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

4. 팀2: 문제 생성

4.1 init.py

```python
# paper_problem/__init__.py
"""
팀2: 문제 생성 모듈
난이도별 실습 문제 자동 생성 시스템을 제공합니다.
"""

from paper_problem.api import router as problem_router
from paper_problem.workflow import problem_workflow

__all__ = ['problem_router', 'problem_workflow']
```

```python
# paper_problem/generators/__init__.py
"""문제 생성기 모듈"""

from paper_problem.generators.beginner import beginner_generator
from paper_problem.generators.intermediate import intermediate_generator
from paper_problem.generators.advanced import advanced_generator

__all__ = ['beginner_generator', 'intermediate_generator', 'advanced_generator']
```

```python
# paper_problem/validators/__init__.py
"""문제 검증 모듈"""

from paper_problem.validators.problem_validator import problem_validator

__all__ = ['problem_validator']
```

```python
# paper_problem/utils/__init__.py
"""문제생성 전용 유틸리티 모듈"""

from paper_problem.utils.content_analyzer import content_analyzer

__all__ = ['content_analyzer']
```

4.2 Pydantic 모델

```python
# paper_problem/models.py
from pydantic import BaseModel
from typing import List, Dict, Literal, Optional

class Problem(BaseModel):
    question: str
    answer: str
    hints: List[str]
    difficulty_score: int
    problem_type: Literal["CODING", "SHORT_ANSWER"]
    test_cases: Optional[List[Dict]] = []

class ProblemRequest(BaseModel):
    material_id: int
    difficulty: Literal["BEGINNER", "INTERMEDIATE", "ADVANCED"]
    problem_count: int = 3

class ProblemResponse(BaseModel):
    problems: List[Problem]
    difficulty: str
    generated_count: int
    rejected_count: int
```

4.3 Generator (초급/중급/고급)

```python
# paper_problem/generators/beginner.py
from shared.upstage_client import upstage_client
from langchain.schema import HumanMessage
from paper_problem.models import Problem
from typing import List
import json
import logging

logger = logging.getLogger(__name__)

class BeginnerProblemGenerator:
    async def generate(self, context: str, count: int = 3) -> List[Problem]:
        """초급 문제 생성"""

        llm = upstage_client.get_chat_model(temperature=0.7)

        prompt = f"""다음 학습 내용을 바탕으로 **초급** 실습 문제를 {count}개 생성하세요.

**학습 내용**:
{context}

**초급 문제 요구사항**:
1. 기본 개념 이해 확인
2. 단순한 코드 작성 (5-10줄)
3. 명확한 정답

**출력 형식** (JSON):
[
    {{
        "question": "문제 내용",
        "answer": "정답 코드",
        "hints": ["힌트1", "힌트2"],
        "difficulty_score": 1-3,
        "problem_type": "CODING",
        "test_cases": [{{"input": "...", "expected": "..."}}]
    }}
]"""

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

        try:
            problems_data = json.loads(response.content)
            problems = [Problem(**p) for p in problems_data]
            logger.info(f"Generated {len(problems)} beginner problems")
            return problems
        except json.JSONDecodeError:
            logger.error("Failed to parse JSON")
            return []

beginner_generator = BeginnerProblemGenerator()
```

```python
# paper_problem/generators/intermediate.py 및 advanced.py도 동일한 패턴
# (온도와 프롬프트만 다름)
```

4.4 검증기

```python
# paper_problem/validators/problem_validator.py
from paper_problem.models import Problem
from typing import List, Tuple
import logging

logger = logging.getLogger(__name__)

class ProblemValidator:
    def validate(self, problem: Problem, difficulty: str) -> Tuple[bool, str]:
        """문제 유효성 검증"""

        if not problem.question or len(problem.question) < 50:
            return False, "Question too short"
```

```
        if not problem.answer or len(problem.answer) < 20:
            return False, "Answer too short"

        if len(problem.hints) < 2:
            return False, "Need at least 2 hints"

        difficulty_ranges = {
            "BEGINNER": (1, 3),
            "INTERMEDIATE": (4, 6),
            "ADVANCED": (7, 10)
        }

        min_score, max_score = difficulty_ranges[difficulty]
        if not (min_score <= problem.difficulty_score <= max_score):
            return False, f"Difficulty score must be {min_score}-{max_score}"

        return True, "Valid"

    def filter_valid_problems(
        self,
        problems: List[Problem],
        difficulty: str
    ) -> Tuple[List[Problem], List[str]]:
        """유효한 문제만 필터링"""

        valid_problems = []
        rejection_reasons = []

        for problem in problems:
            is_valid, reason = self.validate(problem, difficulty)
            if is_valid:
                valid_problems.append(problem)
            else:
                rejection_reasons.append(reason)

        return valid_problems, rejection_reasons

problem_validator = ProblemValidator()
```

4.5 문제 생성 워크플로우

```
# paper_problem/workflow.py
from langgraph.graph import StateGraph, START, END
from typing import TypedDict, List, Dict
from paper_problem.models import Problem
from shared.chroma_client import chroma_client
from paper_problem.generators.beginner import beginner_generator
from paper_problem.validators.problem_validator import problem_validator
import logging

logger = logging.getLogger(__name__)

class ProblemState(TypedDict):
    material_id: int
    difficulty: str
    problem_count: int
    learning_content: List[Dict]
    context: str
    generated_problems: List[Problem]
    validated_problems: List[Problem]
    rejection_reasons: List[str]

async def analyze_content_node(state: ProblemState) -> dict:
    """학습 내용 추출"""
    material_id = state["material_id"]
    difficulty = state["difficulty"]

    results = chroma_client.search(
        collection_name="learning_materials",
        query_texts=["기본 개념 예제"],
```

```
        n_results=5,
        filter_dict={"material_id": material_id}
    )

    learning_content = []
    for i in range(len(results['documents'][0])):
        learning_content.append({
            'content': results['documents'][0][i],
            'page': results['metadatas'][0][i]['page']
        })

    return {"learning_content": learning_content}

async def build_context_node(state: ProblemState) -> dict:
    """컨텍스트 구성"""
    learning_content = state["learning_content"]

    context = "\n\n---\n\n".join([
        f"[페이지 {c['page']}]\n{c['content']}"
        for c in learning_content
    ])

    return {"context": context}
```

```
async def generate_problems_node(state: ProblemState) -> dict:
    """문제 생성"""
    difficulty = state["difficulty"]
    context = state["context"]
    problem_count = state["problem_count"]

    if difficulty == "BEGINNER":
        problems = await beginner_generator.generate(context, problem_count)
    # elif INTERMEDIATE, ADVANCED...

    return {"generated_problems": problems}

async def validate_problems_node(state: ProblemState) -> dict:
    """문제 검증"""
    problems = state["generated_problems"]
    difficulty = state["difficulty"]

    validated, rejected = problem_validator.filter_valid_problems(
        problems, difficulty
    )

    return {
        "validated_problems": validated,
        "rejection_reasons": rejected
    }
```

```
def create_problem_workflow():
    graph = StateGraph(ProblemState)

    graph.add_node("analyze", analyze_content_node)
    graph.add_node("build_context", build_context_node)
    graph.add_node("generate", generate_problems_node)
    graph.add_node("validate", validate_problems_node)

    graph.add_edge(START, "analyze")
    graph.add_edge("analyze", "build_context")
    graph.add_edge("build_context", "generate")
    graph.add_edge("generate", "validate")
    graph.add_edge("validate", END)

    return graph.compile()

problem_workflow = create_problem_workflow()
```

4.6 문제 생성 API

```python
# paper_problem/api.py
from fastapi import APIRouter, HTTPException
from paper_problem.models import ProblemRequest, ProblemResponse
from paper_problem.workflow import problem_workflow

router = APIRouter()

@router.post("/generate", response_model=ProblemResponse)
async def generate_problems(request: ProblemRequest):
    """난이도별 문제 생성"""

    try:
        result = await problem_workflow.ainvoke({
            "material_id": request.material_id,
            "difficulty": request.difficulty,
            "problem_count": request.problem_count
        })

        validated_problems = result["validated_problems"]

        if not validated_problems:
            raise HTTPException(
                status_code=500,
                detail="Failed to generate valid problems"
            )

        return ProblemResponse(
            problems=validated_problems,
            difficulty=request.difficulty,
            generated_count=len(validated_problems),
            rejected_count=len(result["rejection_reasons"])
        )

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

---

5. 메인 애플리케이션

```python
# main.py
from fastapi import FastAPI
from paper_qa.api import router as qa_router
from paper_problem.api import router as problem_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"
)

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

@app.get("/")
async def root():
    return {
        "service": "EduMentor AI Engine",
        "version": "1.0.0",
        "endpoints": {
            "qa": "/qa",
            "problems": "/problems",
```

```
            "docs": "/docs"
        }
    }

@app.get("/health")
async def health_check():
    return {
        "status": "healthy",
        "services": ["qa", "problems"]
    }

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

---

6. 실행 방법

6.1 로컬 개발 환경

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

# 2. Python 서버 실행
cd python-server
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate
pip install -r requirements.txt
python main.py
```

6.2 서버 접속

- **API 문서**: http://localhost:8000/docs
- **Health Check**: http://localhost:8000/health

---

7. API 사용 예시

7.1 자료 업로드

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

7.2 질문하기

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

7.3 문제 생성

```
curl -X POST http://localhost:8000/problems/generate \
  -H "Content-Type: application/json" \
  -d '{
    "material_id": 1,
    "difficulty": "BEGINNER",
    "problem_count": 3
  }'
```

---

8. 테스트

8.1 전체 플로우 테스트

```python
# test_integration.py
import requests

BASE_URL = "http://localhost:8000"

def test_full_flow():
    # 1. Health Check
    response = requests.get(f"{BASE_URL}/health")
    print(f"✅ Health: {response.json()}")

    # 2. 자료 업로드
    with open("test.pdf", "rb") as f:
        response = requests.post(
            f"{BASE_URL}/qa/upload",
            files={"file": f},
            data={"material_id": 1}
        )
    print(f"✅ Upload: {response.json()}")

    # 3. QA 테스트
    response = requests.post(
        f"{BASE_URL}/qa/ask",
        json={
            "material_id": 1,
            "question": "JPA Entity란?"
        }
    )
    print(f"✅ QA: {response.json()}")

    # 4. 문제 생성 테스트
    response = requests.post(
        f"{BASE_URL}/problems/generate",
        json={
            "material_id": 1,
            "difficulty": "BEGINNER",
            "problem_count": 3
        }
    )
    print(f"✅ Problems: {response.json()}")

if __name__ == "__main__":
    test_full_flow()
```

9. Docker 배포

```dockerfile
# Dockerfile
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

```yaml
# docker-compose.yml
version: '3.8'

services:
  chromadb:
```

```
    image: chromadb/chroma:latest
    ports:
      - "8001:8000"
    volumes:
      - chroma_data:/chroma/data

  python-server:
    build: ./python-server
    ports:
      - "8000:8000"
    environment:
      - UPSTAGE_API_KEY=${UPSTAGE_API_KEY}
      - CHROMA_HOST=chromadb
      - CHROMA_PORT=8000
    depends_on:
      - chromadb

volumes:
  chroma_data:
```

✅ 체크리스트

공통 모듈

- [ ] ChromaDB 클라이언트 구현
- [ ] Upstage 클라이언트 구현
- [ ] 환경 설정 파일

팀1 (QA)

- [ ] PDF 파서
- [ ] 업로드 워크플로우
- [ ] QA 워크플로우 (1-2초)
- [ ] API 엔드포인트

팀2 (문제 생성)

- [ ] Generator (초급/중급/고급)
- [ ] 검증기
- [ ] 문제 생성 워크플로우
- [ ] API 엔드포인트

통합

- [ ] main.py 구현
- [ ] 라우터 등록
- [ ] 통합 테스트

작성일: 2025-10-28
단일 Python 서버: Port 8000