

README

EduMentor AI - 학습자료 기반 QA 및 실습 문제 생성 시스템

프로젝트 기간: 2주 (10 working days)

팀 구성: 4명 (팀1: QA 시스템 2명, 팀2: 문제 생성 2명)

목표: 학습자료(PDF/PPT) 기반 1-2초 응답 QA + 난이도별 실습 문제 자동 생성

🎯 프로젝트 개요

핵심 기능

- 학습자료 업로드 및 파싱 (Upstage Document Parse)
- 즉각 응답 **QA** 시스템 (1-2초 응답, LangChain RAG)
- 난이도별 실습 문제 생성 (초급/중급/고급, LangGraph Agent)
- 학습 관리 및 추적 (Spring Boot Backend)

필수 요구사항

요구사항	구현 방법
✅ LangChain Framework	RAG 파이프라인 구축
✅ Upstage API	Parse, Embeddings, Chat
✅ ChromaDB	Vector Database
✅ 1-2초 응답	최적화된 Retrieve + Generate
✅ LangGraph	Agent Workflow 구축
✅ RAG 아키텍처	다이어그램 작성

📁 문서 구성

1. [팀1QA시스템구현가이드.md](#)

- 담당: 개발자 A, B (2명)
- 내용: 자료 업로드, Upstage Parse, 임베딩, ChromaDB 저장, QA RAG 파이프라인
- 목표: Retrieve 포함 1-2초 응답

주요 구현 내용:

- Upstage Document Parse로 PDF/PPT 파싱
- Upstage Embeddings로 벡터화
- ChromaDB 저장 및 검색
- LangGraph QA Agent (Retrieve → Generate)
- 응답 속도 최적화 (캐싱, 멀티 쿼리)

2. [팀2문제생성구현가이드.md](#)

- 담당: 개발자 C, D (2명)
- 내용: 학습 내용 분석, 난이도별 문제 생성, 검증
- 목표: 난이도별(초급/중급/고급) 실습 문제 자동 생성

주요 구현 내용:

- 학습 내용 분석 (ChromaDB 검색)
- 난이도별 Generator (Beginner/Intermediate/Advanced)
- 문제 검증 시스템 (최소 길이, 난이도 점수, 테스트 케이스)
- LangGraph Problem Agent (Analyze → Generate → Validate)

- 재생성 로직

3. [SpringBoot연동가이드.md](#)

- 담당: 공통 (통합 단계)
- 내용: Python AI Engine ↔ Spring Boot Backend 연동
- 기술: Java 21, Spring Boot 3.2.0, WebClient

주요 구현 내용:

- WebClient 설정 (Python 서버 통신)
- 도메인 엔티티 (User, Material, QASession, Problem)
- PythonClient 구현
- REST API Controller
- PostgreSQL 연동

4. [전체시스템아키텍처.md](#)

- 내용: 전체 시스템 개요, 데이터 흐름, API 명세
- 다이어그램: Mermaid 기반 아키텍처 다이어그램

주요 내용:

- 시스템 구조도
- 데이터 흐름도 (업로드, QA, 문제 생성)
- 데이터베이스 스키마
- API 명세서
- Docker Compose 설정

5. [Docker환경설정가이드.md](#)

- 담당: 공통 (인프라)
- 내용: Docker를 활용한 데이터베이스 및 시스템 관리
- 기술: Docker, Docker Compose, PostgreSQL, ChromaDB

주요 구현 내용:

- PostgreSQL Docker 설정 및 초기화
- ChromaDB Docker 설정
- 데이터 영속성 관리 (Volumes)
- 백업 및 복원 전략
- 트러블슈팅 가이드

6. [RAG파이프라인다이어그램.md](#)

- 담당: 공통 (아키텍처)
- 내용: 팀별 RAG 파이프라인 상세 구조 및 데이터 흐름
- 다이어그램: Mermaid 기반 파이프라인 시각화

주요 내용:

- 전체 시스템 RAG 아키텍처
- 팀1 QA RAG 파이프라인 (자료 파싱 → 임베딩 → 검색 → 생성)
- 팀2 문제 생성 RAG 파이프라인 (내용 분석 → 생성 → 검증)
- 난이도별 Generator 상세 흐름
- 성능 최적화 전략

기술 스택

Backend

- **Spring Boot:** 3.2.0 (Java 21)
- **Database:** PostgreSQL 15

- **Security:** Spring Security + JWT
- **HTTP Client:** WebClient (WebFlux)

AI Engine

- **Framework:** FastAPI
- **LLM:** Upstage Solar API
- **Orchestration:** LangChain + LangGraph
- **Vector DB:** ChromaDB
- **PDF/PPT Parsing:** PyMuPDF, python-pptx

Infrastructure

- **Container:** Docker + Docker Compose
- **Version Control:** Git

2주 개발 일정

Week 1: 기반 구축 + 핵심 기능

날짜	팀1 (QA)	팀2 (문제 생성)	공통
Day 1	환경 설정, ChromaDB 연동	환경 설정, Upstage API 테스트	DB 설계, Spring Boot 프로젝트
Day 2	자료 업로드 + Upstage Parse	난이도 분류 로직 설계	Python FastAPI 서버 구축
Day 3	임베딩 + ChromaDB 저장	문제 생성 Agent 구조 설계	Spring ↔ Python 연동
Day 4	QA RAG 파이프라인 구현	초급 문제 생성 구현	-
Day 5	응답 속도 최적화 (1-2초)	중급/고급 문제 생성 구현	통합 테스트

Week 2: 통합 + 최적화 + 문서화

날짜	팀1 (QA)	팀2 (문제 생성)	공통
Day 6	LangGraph Agent 워크플로우	LangGraph Agent 워크플로우	-
Day 7	출처 표시 기능	문제 검증 로직	-
Day 8	캐싱 및 성능 개선	난이도 자동 조정	통합 테스트
Day 9	전체 QA 테스트	전체 문제 생성 테스트	UI 연동
Day 10	버그 수정, 문서화	버그 수정, 문서화	RAG 아키텍처 다이어그램

빠른 시작

1. 저장소 클론

```
git clone https://github.com/your-team/edumentor.git
cd edumentor
```

2. 환경 변수 설정

```
# Python 서버 디렉토리로 이동
cd python-server

# .env 파일 생성
cp .env.example .env

# 필수 환경 변수 입력
vim .env # 또는 선호하는 에디터 사용
```

필수 설정:

```
UPSTAGE_API_KEY=your_upstage_api_key      # Upstage API 키
POSTGRES_PASSWORD=your_password            # PostgreSQL 비밀번호 (운영 환경에서는 강력한 비밀번호 사용)
JWT_SECRET=your_jwt_secret                 # JWT 서명 키
```

3. 데이터베이스 Docker 실행

중요: PostgreSQL과 ChromaDB는 Docker로 관리됩니다.

```
# PostgreSQL + ChromaDB 시작
docker-compose up -d postgres chromadb

# 데이터베이스 초기화 확인 (최초 실행 시 자동 초기화)
docker-compose logs postgres | grep "database system is ready"

# 샘플 데이터 확인
docker exec -it edumentor-postgres psql -U admin -d edumentor -c "SELECT * FROM users;"
```

자동 초기화 내용:

- 테이블 생성 (users, learning_materials, qa_sessions, practice_problems)
- 인덱스 생성 (성능 최적화)
- 샘플 사용자 생성 (instructor1, student1 / password: password123)

4. Docker로 전체 실행

```
# 전체 시스템 빌드 및 실행
docker-compose up --build

# 백그라운드 실행
docker-compose up -d

# 상태 확인
docker-compose ps
```

5. 개별 서비스 실행 (로컬 개발)

```
# PostgreSQL & ChromaDB
docker-compose up -d postgres chromadb

# Python 통합 서버 (Port 8000)
cd python-server
pip install -r requirements.txt
python main.py

# Spring Boot (Port 8080)
cd spring-backend
./gradlew bootRun
```

6. 접속 확인

- Spring Boot API:** http://localhost:8080
- Python AI API:** http://localhost:8000/docs

API 사용 예시

1. 학습자료 업로드

```
curl -X POST http://localhost:8080/api/materials/upload \
  -H "Authorization: Bearer {token}" \
  -F "file=@spring_guide.pdf" \
  -F "title=Spring Boot 입문"
```

2. 질문하기 (QA)

```
curl -X POST http://localhost:8080/api/qa/ask \
-H "Authorization: Bearer {token}" \
-H "Content-Type: application/json" \
-d '{
  "materialId": 1,
  "question": "JPA Entity란 무엇인가요?"
}'
```

응답 예시:

```
{
  "answer": "JPA Entity는 데이터베이스 테이블과 매핑되는 Java 클래스입니다. @Entity 어노테이션을 사용하여 정의하며...",
  "sources": [
    {
      "page": 23,
      "excerpt": "JPA Entity는 데이터베이스..."
    }
  ],
  "responseTimeMs": 1230
}
```

3. 실습 문제 생성

```
curl -X POST http://localhost:8080/api/problems/generate \
-H "Authorization: Bearer {token}" \
-H "Content-Type: application/json" \
-d '{
  "materialId": 1,
  "difficulty": "BEGINNER",
  "problemCount": 3
}'
```

응답 예시:


```
{
  "problems": [
    {
      "question": "User 엔티티 클래스를 작성하세요...",
      "answer": "@Entity\npublic class User {...}",
      "hints": ["@Entity 어노테이션 사용", "@Id로 기본키 지정"],
      "difficultyScore": 2,
      "problemType": "CODING",
      "testCases": [
        {"input": "...", "expected": "..."}
      ]
    }
  ],
  "difficulty": "BEGINNER",
  "generatedCount": 3,
  "rejectedCount": 0
}
```


테스트

성능 테스트 (QA 응답 시간)

```
# 팀1 성능 테스트
cd python-ai/paper_qa
python tests/test_performance.py
```

기대 결과:

Question: JPA Entity란 무엇인가요?
응답 시간: 1.234s –  PASS

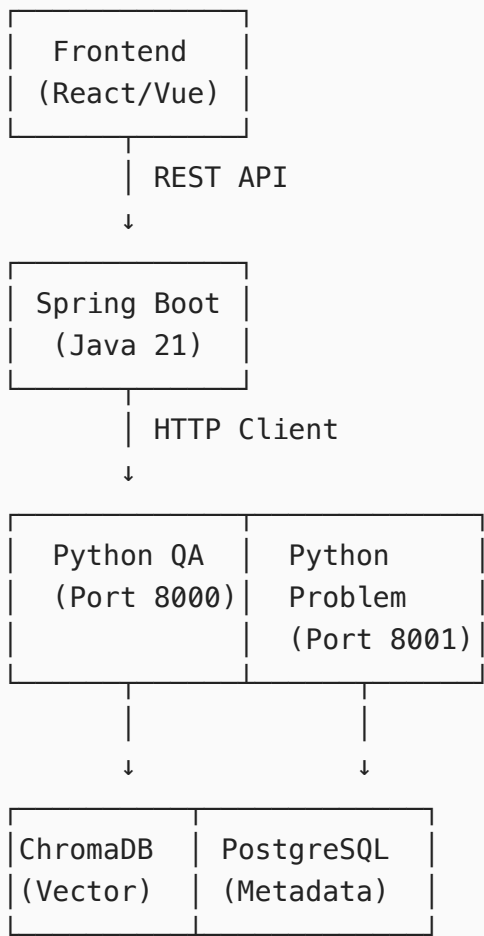
Question: @Transactional은 언제 사용하나요?
응답 시간: 1.567s -  PASS

총 테스트: 3
통과: 3/3
평균 응답 시간: 1.401s

문제 생성 테스트

```
# 팀2 문제 생성 테스트
cd python-ai/paper_problem
python tests/test_problem_generation.py
```

시스템 아키텍처



완성 체크리스트

핵심 기능

- ☐ Upstage API 연동 (Parse, Embeddings, Chat)
- ☐ ChromaDB 벡터 저장 및 검색
- ☐ LangGraph QA Agent (1-2초 응답)
- ☐ LangGraph Problem Agent (난이도별 생성)
- ☐ Spring Boot Backend (Java 21)
- ☐ REST API 구현

성능 목표

- ☐ QA 응답 시간 < 2초
- ☐ 문제 생성 시간 < 30초
- ☐ 파싱 성공률 > 95%
- ☐ 문제 검증 통과율 > 80%

문서화

- ☐ RAG 아키텍처 다이어그램
- ☐ API 문서 (Swagger)

- ☐ 팀별 구현 가이드
- ☐ README.md

테스트

- ☐ 단위 테스트
- ☐ 통합 테스트
- ☐ 성능 테스트
- ☐ E2E 테스트

🤝 팀 역할

팀1: QA 시스템 (2명)

- 개발자 A: 자료 업로드, Upstage Parse, 임베딩, ChromaDB 저장
- 개발자 B: QA RAG 파이프라인, LangGraph Agent, 응답 속도 최적화

팀2: 문제 생성 (2명)

- 개발자 C: 난이도 분석, 학습 내용 추출, LangGraph Agent 구조
- 개발자 D: 문제 생성 로직, 검증, 답안 생성

공통 작업

- Spring Boot 통합 (전체)
- 통합 테스트 (전체)
- 문서화 (전체)

☎ 문의 및 지원

- 이슈 트래커: GitHub Issues
- 이메일: team@edumentor.ai
- 문서: [전체 시스템 아키텍처.md](#)

📖 라이선스

MIT License

프로젝트 시작: 2025-10-28

목표 완료: 2025-11-08 (2주)

팀: 4명 (2 + 2)