OTT 콘텐츠 Q&A 서비스 프로젝트 기획 및 실행 계획서

1. 프로젝트 기획서

1.1. 프로젝트명

AI 기반 OTT 콘텐츠 대화형 검색 서비스

1.2. 프로젝트 목표 및 비전

- 목표: 흩어져 있는 OTT 콘텐츠의 객관적 정보와 전문적인 평가를 한곳에 모아, 사용자가 자연어 질문을 통해 원하는 정보를 즉각적으로 얻을 수 있는 MVP(최소 기능 제품) 웹 서비스를 개발한다.
- 비전: 단순 정보 검색을 넘어, 사용자의 취향과 질문의 맥락을 이해하여 새로운 콘텐츠를 발견하고 탐색하는 즐거움을 제공하는 AI 기반의 'OTT 콘텐츠 전문 가이드'로 발전시킨다.

1.3. 핵심 기능

1. 자연어 **Q&A**: "이 드라마 줄거리 뭐야?", "이 영화 평론가 평점 어때?" 등 자유로운 형태의 질문에 대해 RAG 파이프라인을 통해 답변 생성
2. 신뢰도 기반 정보 제공: OTT 공식 사이트의 사실 정보와 네이버 영화 등 전문 사이트의 평가를 기반으로 답변하여 신뢰도 확보
3. 직관적인 웹 인터페이스: 사용자가 쉽게 질문을 입력하고 답변을 확인할 수 있는 최소한의 기능을 갖춘 채팅형 웹 UI 제공

1.4. 주요 기술 스택

- **AI/ML**: RAG, LangChain, LangGraph
- **Backend**: Python, FastAPI
- **Frontend**: React (혹은 MVP로 streamlit)
- **Vector Store**: ChromaDB (파일 기반)
- **Deployment**: AWS (Lambda, API Gateway, S3, Amplify), Docker

2. 상세 프로젝트 실행 계획

과정 **1 :** 프로젝트 준비 및 계획 수립 **(1**주차**)**

- 목표: 프로젝트의 성공적인 시작을 위한 명확한 역할 분담과 구체적인 작업 계획 수립.
- 주요 활동:
  1. 역할 정의: 각 팀원의 주력 분야(Backend, Frontend, Data/AI)를 정하고 담당 영역을 명확히 합니다.
  2. **WBS** 작성: 본 문서를 기반으로 각 단계 별 상세 기능을 분배하고 담당자와 예상 소요 시간을 할당합니다.
  3. 개발 환경 통일: Python 버전, 주요 라이브러리 버전을 통일하여 개발 환경 차이로 인한 문제를 예방합니다.
  4. **Git** 협업 전략 수립: 브랜치 전략(예: Git-flow), 커밋 메시지, 컨벤션 등 협업 규칙을 정합니다.
- 고민해야 할 것:
  - 의사소통 채널: 빠른 소통을 위한 주 채널(Discord)과 정기 회의 규칙을 어떻게 정할 것인가?
  - 리스크 관리: 일정 지연, 기술적 난관 등 예상되는 리스크에 대한 대응 계획을 미리 논의했는가?
- 예상되는 어려움:
  - 프론트, AI 에이전트 부분에서 초기 단계에서 팀원 간의 기술 스택 이해도 차이가 있을 수 있을 것 같습니다.
- 산출물:
  - WBS가 작성된 프로젝트 관리 보드
  - Git Repository 및 협업 규칙 문서

과정 **2:** 데이터 수집 및 전처리 **(1-2**주차**)**

- 목표: RAG 파이프라인의 원천 재료가 될 고품질의 텍스트 데이터를 안정적으로 수집하고 정제하는 시스템 구축.
- 주요 활동:
  1. 크롤링 대상 확정: 1차 수집 대상으로 할 OTT 사이트 1~2개(예: 넷플릭스)와 전문 리뷰 사이트 1개(예: 네이버 영화)를 최종 선정합니다.
  2. 데이터 스키마 정의: 수집할 정보(제목, 줄거리, 감독, 배우, 장르, 평점, 리뷰 본문, URL 등)를 구조화된 JSON 형식으로 정의합니다.
  3. 크롤러 개발: Selenium(동적 페이지 대응)과 BeautifulSoup을 조합하여 대상 사이트의 데이터를 수집하는 Python 스크립트를 개발합니다.
  4. 데이터 정제 및 저장: 수집된 데이터에서 불필요한 HTML 태그, 광고, 특수 문자를 제거하고 정의된 스키마에 맞춰 JSON 파일로 저장합니다.
  5. 텍스트 분할 전략 수립: LangChain의 RecursiveCharacterTextSplitter를 사용하여 문서를 의미 있는 단위 청크로 분할합니다. 청크 사이즈와 오버랩 크기를 테스트하며 최적값을 찾습니다.
- 고민해야 할 것:
  - 크롤링 윤리 및 정책: 사이트의 robots.txt를 준수하고, 과도한 요청으로 서버에 부하를 주지 않도록 요청 간에 time.sleep()을 설정하는 등 윤리적 크롤링 수행 방법?
  - 데이터 업데이트 주기: OTT 콘텐츠는 계속 추가되는데, 언제마다 새로운 데이터를 크롤링하여 업데이트할 것인가? (MVP 단계에서는 1회성 수집으로 진행)
- 예상되는 어려움 :
  - 안티 크롤링 기술: 사이트에서 IP 차단, Captcha 등 안티 크롤링 기술을 사용할 경우, 이를 우회하기 위한 추가적인 기술(Proxy, User-Agent 변경 등)이 필요할 수 있습니다.
  - 비정형 데이터: 리뷰나 줄거리의 형식이 일정하지 않아 원하는 정보만 정확히 추출하는 데 많은 예외 처리가 필요할 수 있습니다.
- 산출물:
  - 크롤링 Python 스크립트
  - 정제된 원본 데이터 JSON 파일
  - 텍스트 분할 로직이 포함된 전처리 스크립트

과정 **3 :** 핵심 기능 개발 **- RAG** 파이프라인 구축 **(3**주차**)**

- 목표 : 프로젝트의 심장인 Q&A 생성 파이프라인을 구축하고, 스크립트 레벨에서 기능이 동작하는 것을 확인.
- 주요 활동:
  1. 임베딩 모델 선정: 한국어 성능이 검증된 오픈소스 모델(예: ko-sbert-nli)을 찾은 후 우선 테스트하고, 필요시 OpenAI의 text-embedding-ada-002 사용을 고려합니다.
  2. **Vector Store** 구축: 전처리 된 텍스트 청크를 임베딩하여 ChromaDB에 저장합니다. 이때, 검색 효율을 높이기 위한 메타데이터(출처, 장르 등)를 함께 저장합니다.
  3. 검색기**(Retriever)** 구현: ChromaDB를 기반으로 사용자 질문과 가장 유사한 상위 K개 문서를 검색하는 검색기를 LangChain으로 구현합니다.
  4. **LLM** 선정 및 프롬프트 엔지니어링: 답변 생성에 사용할 LLM(예: GPT-4.0 / GPT-4.0 mini)을 정하고, 검색된 문서를 기반으로 정확하고 일관된 답변을 생성하도록 시스템 프롬프트를 설계합니다.
  5. **RAG Chain** 통합: 검색기, 프롬프트, LLM을 LangChain Expression Language (LCEL)를 사용하여 하나의 체인으로 연결하고, 스크립트 상에서 Q&A 테스트를 진행합니다.
- 고민해야 할 것:
  - 성능과 비용의 **Trade-off**: 오픈소스 임베딩/LLM 모델은 무료이지만 직접 서버를 운영해야 하는 부담이 있고, OpenAI API는 사용이 간편하지만 비용이 발생할 때 선택을 어떻게 할 것인가?
  - 답변 품질 평가 기준: "좋은 답변"이란 무엇인가? 답변의 정확성, 유용성, 일관성 등을 평가할 테스트 질문 목록과 평가 기준을 미리 정의해야 합니다.
- 예상되는 어려움:
  - 환각 : LLM이 주어진 정보에 없는 내용을 지어내어 답변하는 현상이 발생할 수 있습니다. 이를 최소화하기 위해 정교한 프롬프트 설계와 후처리 과정이 필요합니다.
  - 맥락 이해 부족: "그 영화 감독의 다른 작품 알려줘"와 같이 대명사가 포함된 후속 질문의 맥락을 이해하고 답변하기 위해서는 채팅 히스토리를 관리하는 기능이 추가로 필요
- 산출물:
  - Vector Store가 저장된 폴더
  - 임베딩 및 RAG Chain 실행 Python 스크립트
  - 시스템 프롬프트 초안

과정 **4 :** 서비스 구현 **- API** 및 **UI** 개발 **(4**주차**)**

- 목표 : 사용자가 실제로 서비스를 이용할 수 있도록 백엔드 API 서버와 프론트엔드 웹 화면을 개발.
- 주요 활동:
  1. **API** 설계: FastAPI를 사용하여 /ask 엔드포인트를 설계합니다. Request Body로는 사용자 질문({"question": "..."}), Response Body로는 생성된 답변({"answer": "..."})을 주고받는 규격을 정의합니다.
  2. 백엔드 서버 개발: FastAPI 서버가 시작될 때 과정 3 에서 만든 RAG Chain(모델, Vector Store 포함)을 메모리에 로드하도록 구현합니다. /ask 요청이 오면 로드된 체인을 실행하여 답변을 반환하도록 개발합니다.
  3. **UI/UX** 디자인: 와이어프레임을 통해 채팅창, 전송 버튼, 로딩 인디케이터 등 사용자 화면의 기본 구조와 흐름을 디자인합니다.
  4. 프론트엔드 개발: React를 사용하여 디자인된 UI를 구현합니다. axios 라이브러리를 사용해 백엔드 API와 비동기 통신을 구현합니다.
- 고민해야 할 것:
  - 사용자 경험**(UX)**: 답변이 생성되는 데 시간이 걸릴 수 있는데, 사용자가 지루하지 않게 기다리도록 어떤 장치를 마련할 것인가? (예: "답변 생성 중..." 메시지 표시, 답변 스트리밍 기능)
  - 에러 처리: 백엔드 서버에서 오류가 발생하거나 API 호출에 실패했을 때, 사용자에게 어떤 메시지를 어떻게 보여줄 것인가?
- 예상되는 어려움:
  - **CORS(Cross-Origin Resource Sharing)** 이슈: 로컬에서 개발 시 React(예: 3000번 포트)와 FastAPI(예: 8000번 포트) 간의 포트가 달라 발생하는 CORS 오류를 해결해야 합니다. FastAPI에 미들웨어를 추가하여 처리해야 합니다.
  - 상태 관리: React에서 사용자의 질문 목록, 로딩 상태 등 클라이언트의 상태를 효율적으로 관리하기 위한 전략(예: useState, useEffect)이 필요합니다.
- 산출물:
  - FastAPI 기반의 백엔드 서버 코드
  - React 기반의 프론트엔드 코드
  - API 명세서

과정 **5 :** 통합**,** 배포 및 테스트 **(5**주차**)**

- 목표 : 개발된 모든 구성 요소를 클라우드 환경에 배포하고, 실제 서비스처럼 동작하는 것을 검증하며 안정성을 확보.
- 주요 활동 :
  1. **Docker**: FastAPI 서버를 Docker 이미지로 만들어 어떤 환경에서든 동일하게 실행될 수 있도록 컨테이너화합니다.
  2. **AWS** 인프라 구축:
     - **S3**: ChromaDB 벡터 데이터와 크롤링 원본 데이터를 업로드할 버킷을 생성합니다.
     - **Lambda & API Gateway**: Docker 이미지를 Lambda 함수로 배포하고, API Gateway와 연결하여 외부에서 접속 가능한 HTTPS 엔드포인트를 생성합니다.
     - **Amplify**: React 빌드 파일을 배포하여 프론트엔드 웹사이트를 호스팅합니다.
  3. 환경 변수 설정: OpenAI API 키 등 민감한 정보는 AWS의 환경 변수 관리 기능을 통해 안전하게 주입합니다.
  4. 통합 테스트: 배포된 웹사이트에서 질문을 입력했을 때, API Gateway -> Lambda -> RAG Chain -> LLM을 거쳐 다시 사용자 화면에 답변이 정상적으로 표시되는지 전체 흐름을 테스트합니다.
- 고민해야 할 것:
  - 비용 최적화: AWS 프리티어 범위를 최대한 활용할 수 있는 방법은 무엇인가? Lambda의 메모리 할당량과 타임아웃 설정을 어떻게 최적화할 것인가?
  - 보안: API Gateway에 API 키나 인증 절차를 추가하여 무분별한 호출을 막을 필요가 있는가? (MVP 단계에서는 open으로 두되, 추후 고려)
- 예상되는 어려움:
  - 클라우드 환경 설정: AWS 서비스들의 설정(IAM 권한, 네트워크 등)이 복잡하여 초기 배포 시 많은 시행착오를 겪을 수 있습니다.
  - **Cold Start**: Lambda 를 사용할 때 시간이 지난 후 첫 호출할 때 딜레이가 발생합니다.
- 산출물:
  - Dockerfile 및 서버 Docker 이미지
  - 배포된 웹 서비스 URL
  - 테스트 케이스 및 결과 기록 문서