회의록 및 뉴스 추천 요약 시스템

10조
202011318 신홍규
202311394 추정민
202315305 곽인경

목차

팀원 구성 및 역할 분담

202011318 신홍규 (팀장)

전체 백엔드 시스템 아키텍처 설계 및 구현

202315305 곽인경 (팀원)

UI/UX 개발, 사용자 인터랙션 처리, 프론트엔드와 백엔드 연동

202311394 추정민 (팀원)

UI/UX 개발, 사용자 인터랙션 처리, 프론트엔드와 백엔드 연동

작품 소개

개요

회사에서 회의록을 입력받아 대규모 언어 모델(LLM)으로 내용을 요약하고, 이를 기반으로 관련된 뉴스를 크롤링 및 내용을 요약하여 사용자가 이를 보고 검색을 할 수 있는 시스템을 구축하고자 합니다.

선정 배경

회사 회의에서 논의된 내용을 바탕으로 경쟁 기업의 동향, 시장의 흐름, 최신 기술들을 빠르게 파악하기 위해, 가장 빠르게 세상의 정보를 보여주는 뉴스를 보여주어 발빠른 대처가 가능하도록 도와주는 시스템을 제안합니다.

사용 기술 소개 - FastAPI

개요

- 파이썬 표준 타입 힌트를 기반으로 API를 빠르고 효율적으로 빌드하기 위한 현대적인 웹 프레임워크

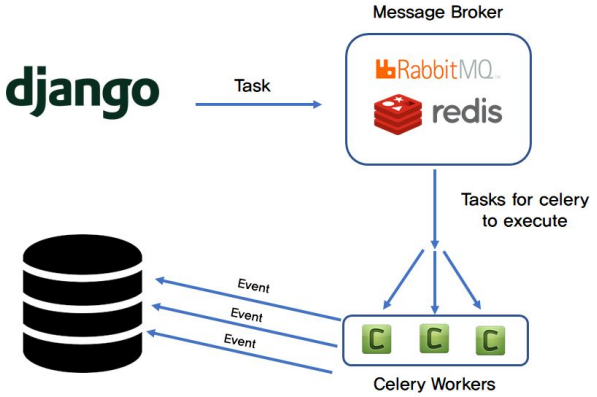
특징

- 높은 성능
- 빠른 개발 속도
- 유지보수성
- 자동 문서화
- 비동기 지원 및 간단한 코드

사용 기술 소개 - Celery

개요

- Python 기반 분산 작업 큐 시스템
- 웹 앱에서 비동기 백그라운드 작업 처리
- 실시간 응답성 + 스케쥴링 모두 지원



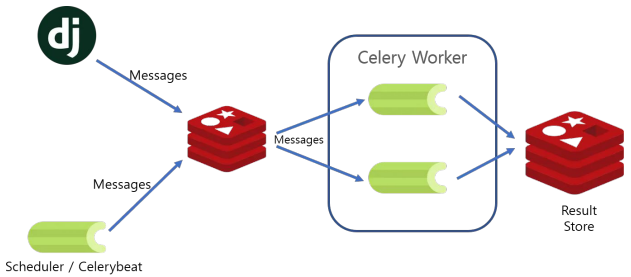
핵심 개념과 아키텍처

Client (Producer)	작업 요청
Broker	Redis or RabbitMQ
Worker (Consumer)	Celery 워커가 큐에서 작업을 꺼내와 수행
Result Backend	MySQL에 저장

사용 기술 소개 - Redis

개요

- 메모리 기반의 Key-Value 데이터 저장소, 비동기 큐 시스템으로 활용
- 본래는 캐시 서버로 사용되었으나, 높은 성능과 간단한 구조 덕분에 Celery의 Task Queue로 자주 활용



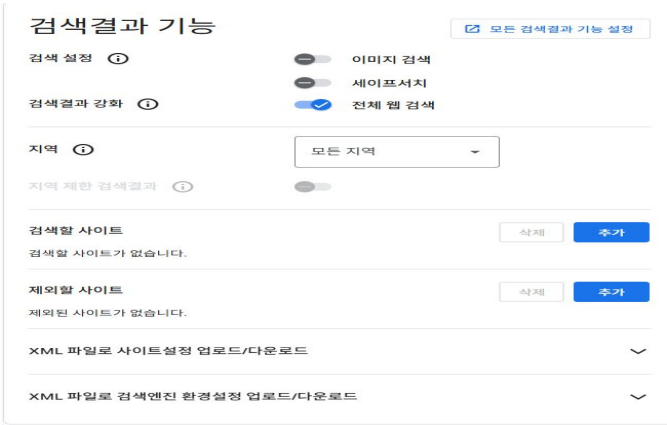
특징

- 메모리 기반으로 작동하여 매우 빠른 읽기/쓰기 속도 제공
- Key, List, Set, Hash, Sorted Set등 다양한 자료구조 지원
- pub.sub, list 기반 queue, stream 처리 기능 내장
- python의 celery 등과 연동이 쉽고 빠름

사용 기술 소개 - Google Custom Search JSON API

개요

- Google의 강력한 검색 엔진 기능을 API 형태로 제공하여, 검색 결과를 JSON 형식으로 받아볼 수 있는 서비스
- 웹 크롤링 없이도 Google의 검색 인덱스를 합법적이고 안정적으로 애플리케이션에 통합하여 사용할 수 있음



특징

- 구조화된 JSON 반환
- 고도의 검색 제어
- 쿼리 제한 및 할당량
- Python과의 간편한 연동

사용 기술 소개 - LLM / Google Gemini API

개요

- 멀티모달 LLM으로, 텍스트뿐 아니라 이미지, 코드, 오디오 등 다양한 입력을 동시에 이해하고 처리할 수 있는 AI 모델

특징

- 문맥 이해 능력이 뛰어남
- JSON 기반 구조화 출력 지원
- 다양한 입력 길이 처리 가능
- 높은 품질의 자연어 요약
- 빠른 속도

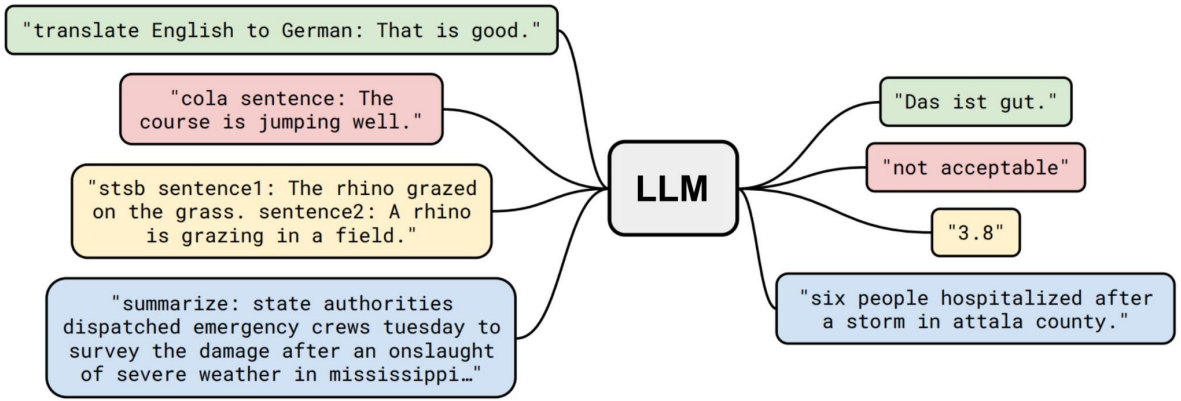
사용 기술 소개 - Prompt Engineering

개요

- 대규모 언어 모델(LLM)인 Gemini가 원하는 결과(정확한 요약, 관련 키워드 리스트)를 생성하도록 프롬프트를 최적화하고 구조화하는 과정

특징

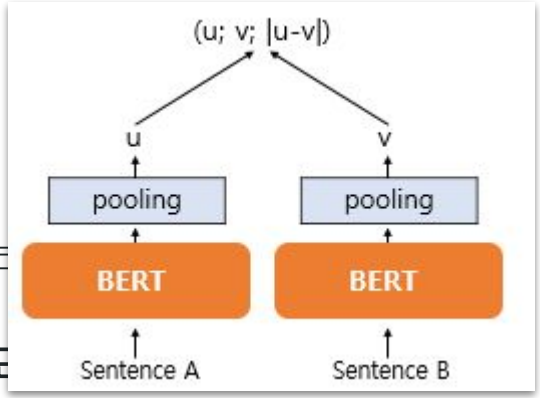
- 명확한 역할 부여
- 결과 형식 지정
- 일관된 품질 유지
- 추론 오류 감소 및 안정성 향상



사용 기술 소개 - S-BERT

사용 목적

- 회의록의 내용을 활용해 단순 키워드 매칭으로 검색 하면 단어가 같으나 문맥적으로 무관한 뉴스도 검색될 수 있다.
  ⇒ 의미기반으로 검색해 유효한 뉴스를 걸러낼 필터가 필요

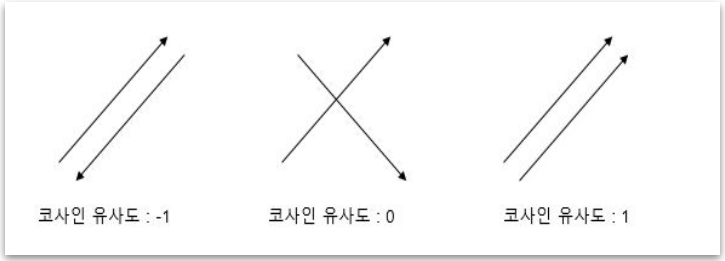


- 실시간으로 뉴스를 걸러내야 하므로 성능이 빨라야 하고, 뉴스에는 수많은 문장이 있으므로 이를 감당할 수 있어야 함
  ⇒ 속도 요구사항을 만족하면서 코사인 유사도 측정에 적합한 도
  ⇒ 이를 위해 Fine Tuning된 S-BERT 모델이 적합.

사용 기술 소개 - Cosine Similarity

개요

- 벡터화된 문장의 코사인 유사도를 비교하여 회의록과 관련이 있는 문장을 포함하는 뉴스를 선별



사용 예시

- 회의록: "3분기 차량용 반도체 공급 부족 대책 논의"
- 뉴스 기사: "글로벌 반도체 품귀 현상, 자동차 공장 가동 중단 위기"

⇒ 서로 다른 문장이지만, 맥락상 같은 상황에 대해 논하고 있음

사용 기술 소개 - Cosine Similarity

사용 예시

- 회의록: "3분기 차량용 반도체 공급 부족 대책 논의"
- 뉴스 기사: "글로벌 반도체 품귀 현상,
  자동차 공장 가동 중단 위기"

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

⇒ 두 문장을 SBERT로 벡터화 하면,
서로 유사한 값을 가지게 됨
회의록 벡터: `[ 0.88, 0.12, -0.35, 0.50, 0.04 ]`

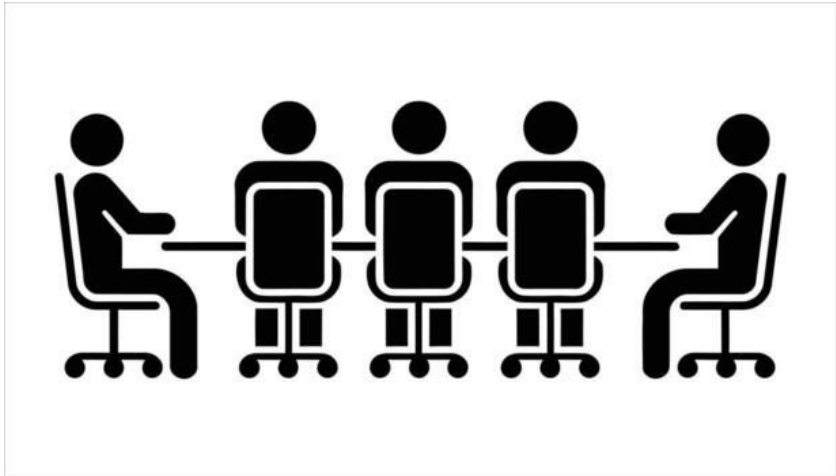
뉴스 벡터: `[ 0.85, 0.15, -0.30, 0.45, 0.10 ]`

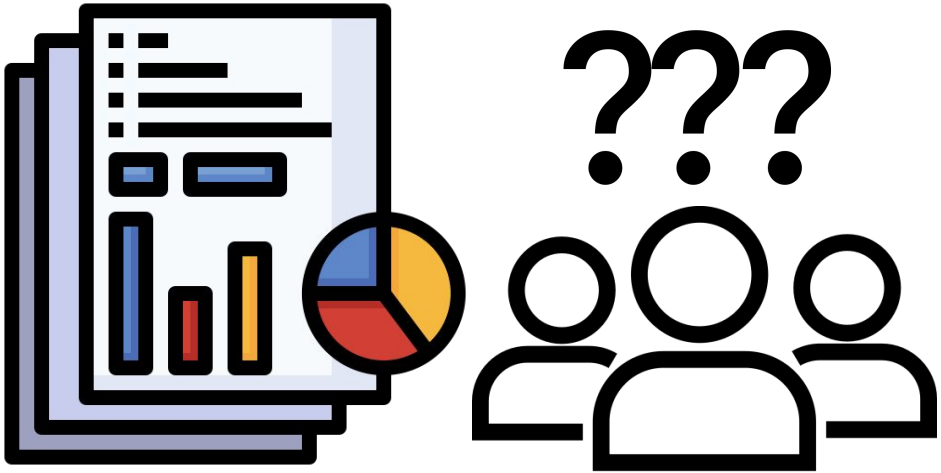
우측 공식을 이용해 유사도를 계산하면 ⇒ 0.982로, 높은 유사도값을 보임

핵심 문제 및 해결 방안

As-Is

- 회의록 분석의 비효율성과 최신 동향 탐색의 어려움
- 수작업 기반의 뉴스 추천 과정

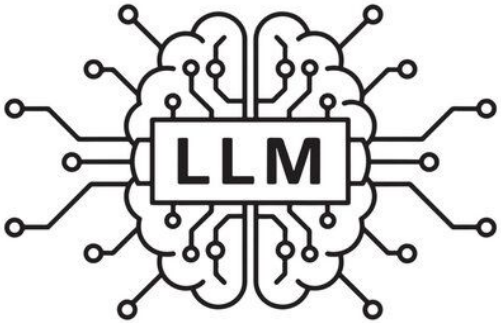


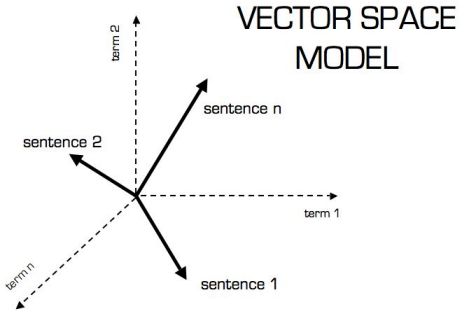


핵심 문제 및 해결 방안

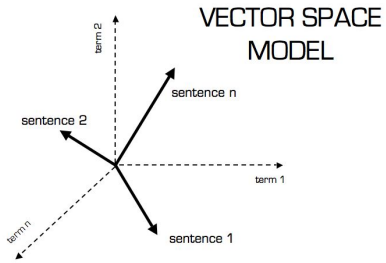
To-Be

- LLM 기반 자동 요약 및 키워드 추출
- 외부 뉴스 검색 자동화
- S-BERT 기반 코사인 유사도 측정



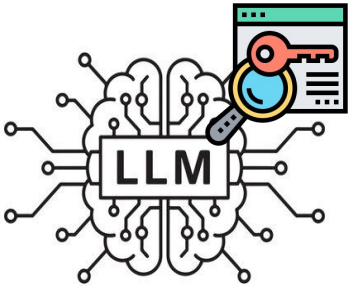


요구사항 - 기능 요구사항



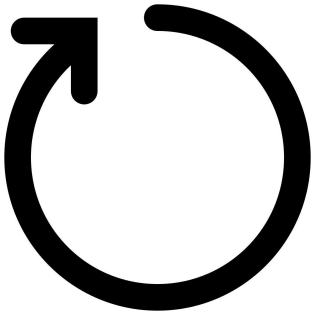
관련 뉴스 자동 수집 및 필터링 (SBERT 코사인 유사도 기반)

AI기반 회의록 요약 및 키워드 추출



추천 뉴스 요약 제공

관련 뉴스 재검색 기능



요구사항 - 비기능 요구사항

성능 및 효율성

시스템에 병목 현상을 일으키지 않도록 함

유지보수성 및 확장성

시스템이 미래의 변화에 유연하게 대응하고 관리가 용이하도록 함

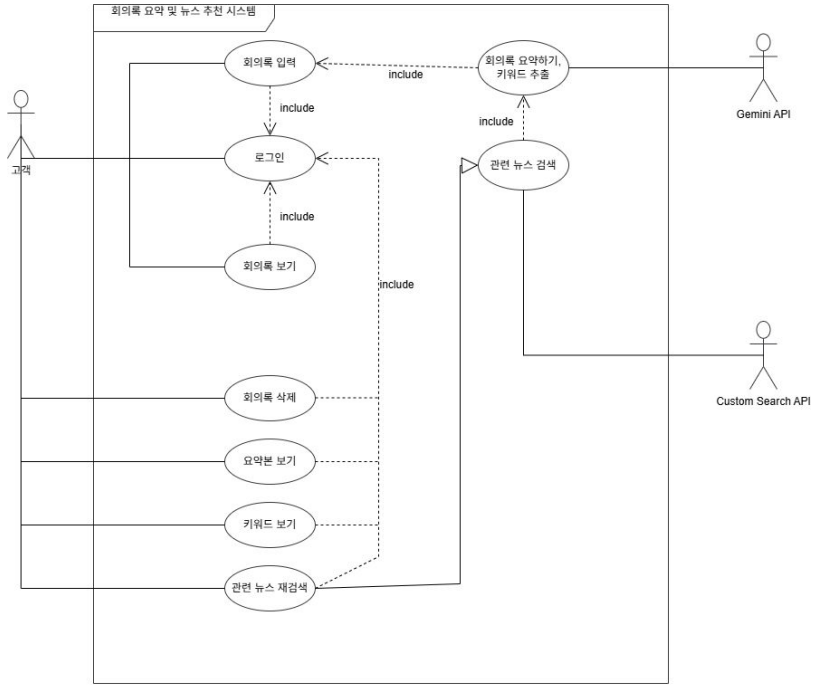
보안 및 안전성

사용자 데이터의 기밀성과 접근 통제

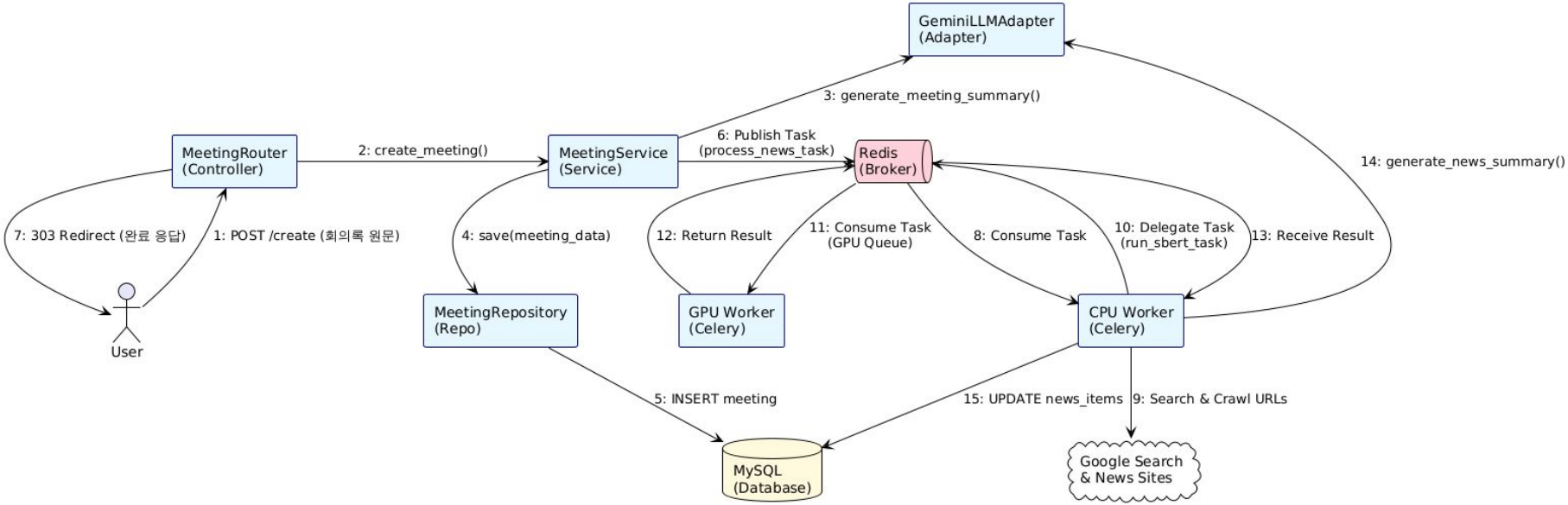
신뢰성

시스템이 오류 없이 일관되게 동작하고 데이터를 정확하게 관리

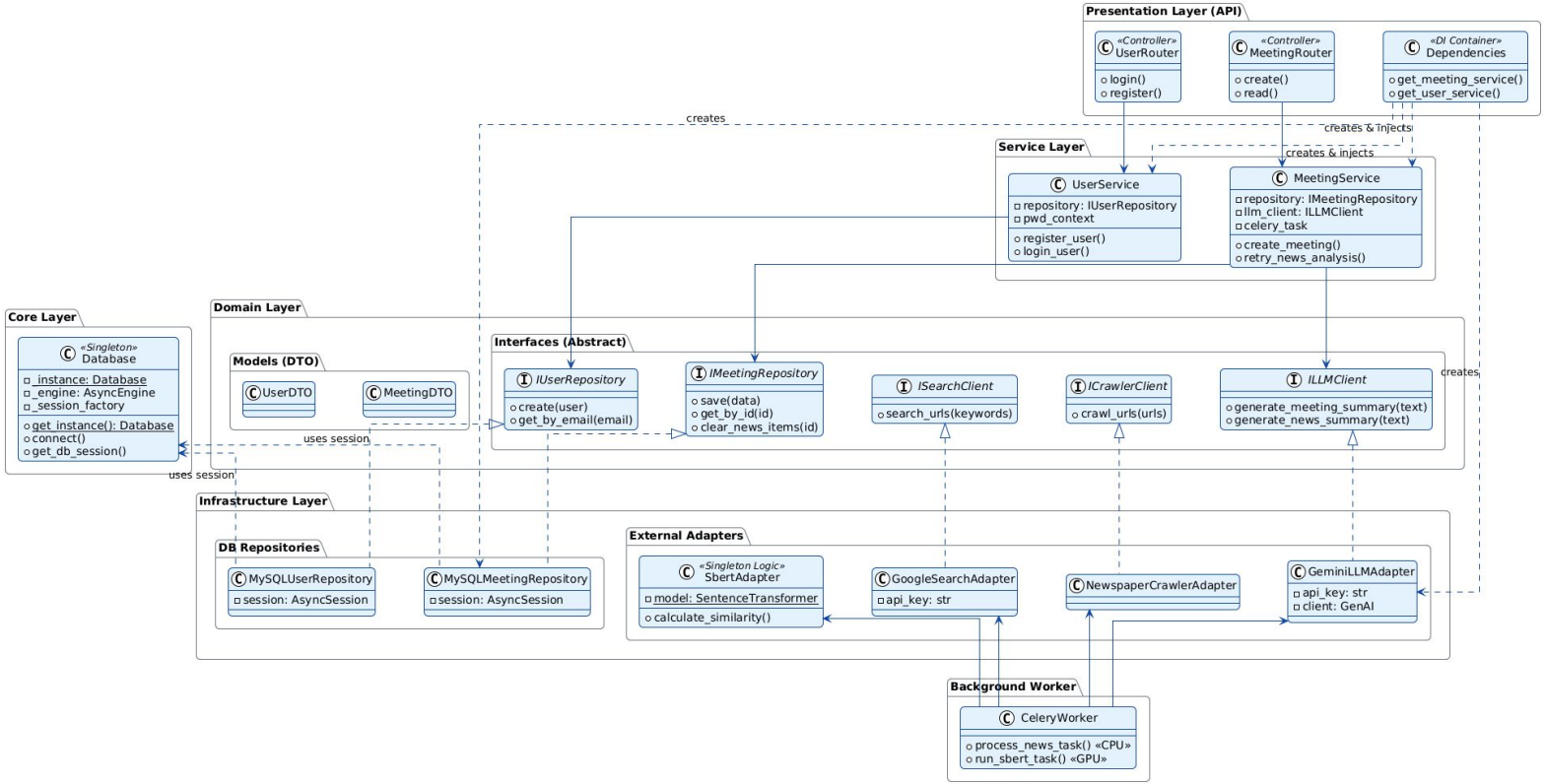
SW 설계 - Use Case Diagram



SW 설계 - Collaboration Diagram



SW 설계 - Class Diagram



SW 설계 - Class Diagram

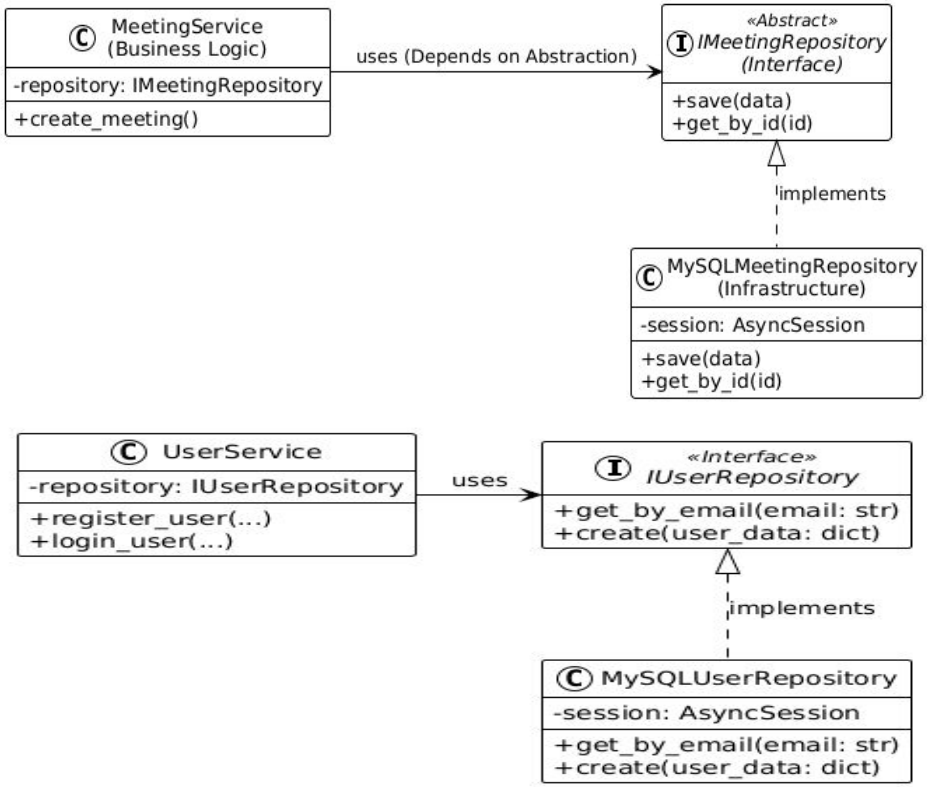
의존성 역전 원칙 (DIP) & 리포지토리 패턴

- 상위 모듈(Service)이 하위 모듈(MySQLRepository)에 의존하지 않고, 둘 다 추상화(Interface)에 의존

설계 이유

- 서비스가 SQL문을 직접 포함하고 있으면 DB 스키마가 바뀌거나 DB 종류를 바꾸려면 비즈니스 로직까지 다 뜯어 고쳐야함
- DIP를 적용함으로써 DB 기술이 변경되어도 비즈니스 로직은 보호된다.

```python
class MySQLMeetingRepository(IMeetingRepository):

    def __init__(self, session: AsyncSession):
        self.session = session


class MeetingService:
    def __init__(
        self,
        repository: IMeetingRepository,
        llm_client: ILLMClient,
        celery_task = None  # 순환 참조 방지를 위해 런타임에 주입하거나 래퍼 사용
    ):
        self.repository = repository
        self.llm_client = llm_client
        self.celery_task = celery_task
```



SW 설계 - Class Diagram

Singleton Pattern

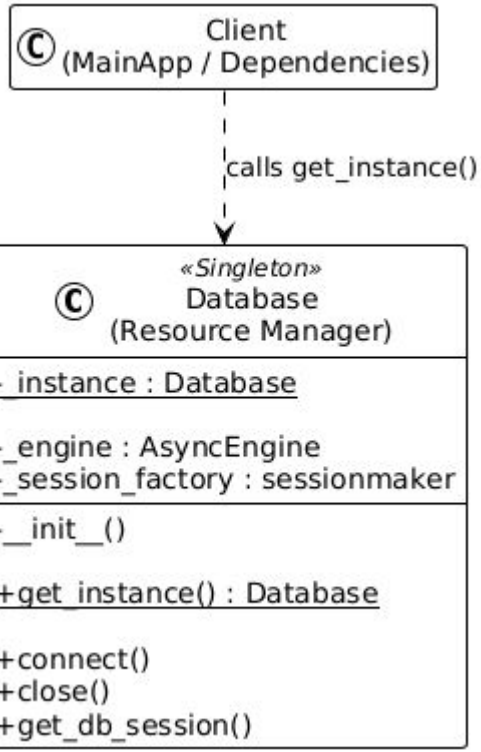
- 어떤 클래스의 인스턴스가 오직 하나임을 보장하고, 이에 대한 전역적인 접근점을 제공

설계 이유

- 데이터베이스 연결의 효율성:
  - DB 연결을 맺는 과정은 시간이 오래 걸리고 서버 리소스를 많이 소모
  - 커넥션 풀을 사용하여 DB 리소스 재사용
- S-BERT 로딩:
  - S-BERT 모델을 메모리에 로드하는데 오래 걸리고, 많은 메모리를 점유
  - 요청마다 로드하면 응답 속도가 느려지고 메모리가 터질 수 있어, 사용 시에만 로드

```python
class Database:
    _instance = None

    def __init__(self):
        self._engine = None
        self._session_factory = None
        self.db_url = os.getenv("DB
        if not self.db_url:
            raise ValueError("DB_CO

    @classmethod
    def get_instance(cls):
        if cls._instance is None:
            cls._instance = cls()
        return cls._instance
```

```python
sbert_model = None

def get_sbert_model():
    global sbert_model
    if sbert_model is None:
        print("[GPU Worker] S-BERT 모델 로드 중...")
        sbert_model = SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')
    return sbert_model
```



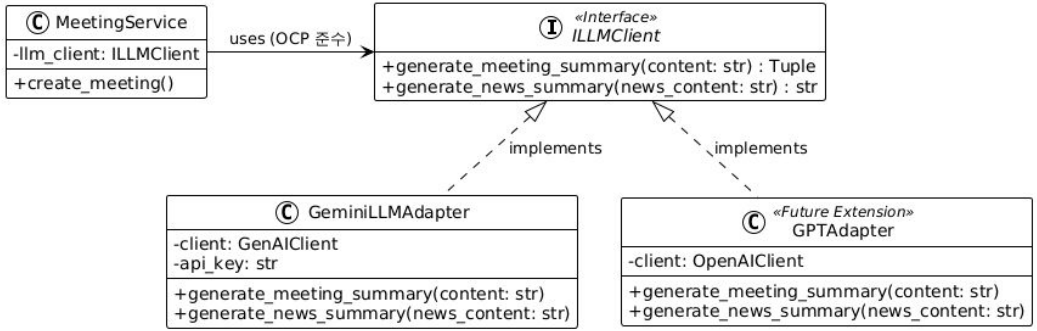
SW 설계 - Class Diagram

Adapter Pattern & 개방-폐쇄의 원칙 (OCP)

- Adapter Pattern: 인터페이스가 호환되지 않는 클래스(Gemini 라이브러리)를 사용자가 기대하는 인터페이스(`ILLMClient`)로 변환하여 함께 작동하게 만듦
- OCP (Open-Closed Principle): 소프트웨어 개체는 확장에는 열려 있어야 하고, 수정에는 닫혀 있어야 함

설계 이유

- 표준화: 서비스에 필요한 메서드명으로 정의해서 외부 라이브러리의 복잡한 사용법을 감출 수 있음
- 확장 가능성: 나중에 다른 LLMAPI를 사용해야할 경우 해당 클래스를 구현만 하면 쉽게 확장할 있음




```python
class ILLMClient(ABC):

    @abstractmethod
    async def generate_meeting_summary(self, content: str) -> Tuple[str, List[str]]:
        """회의록 원문을 받아 요약문과 키워드 리스트를 반환"""
        pass

    @abstractmethod
    async def generate_news_summary(self, news_content: str) -> str:
        """뉴스 원문을 받아 요약문을 반환"""
        pass
```

```python
class GeminiLLMAdapter(ILLMClient):
    def __init__(self, api_key: str):
        self.client = genai.Client(api_key=api_key)
        self.model_name = 'gemini-2.0-flash'

    async def generate_meeting_summary(self, content: str) -> Tuple[str, List[str]]:
```

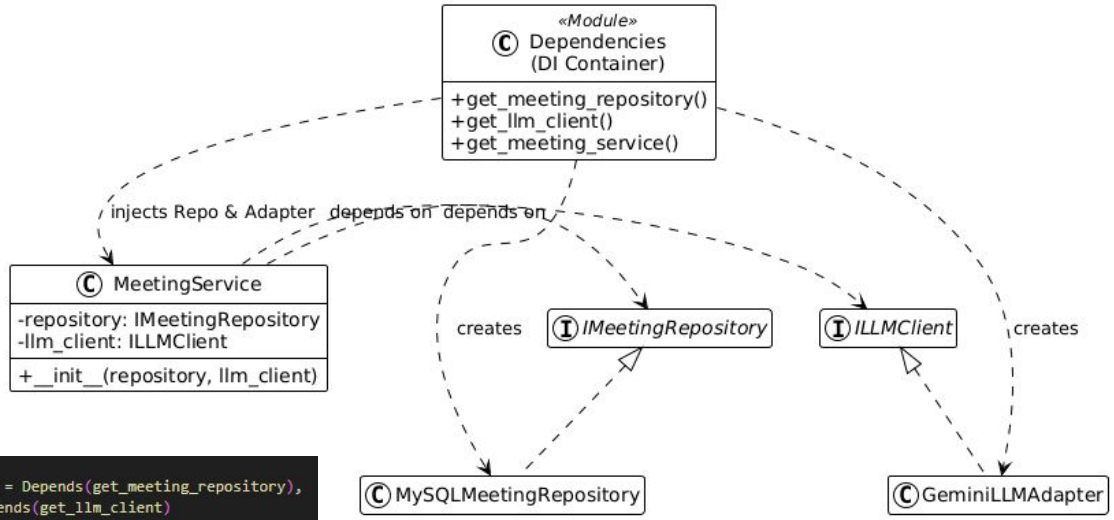
SW 설계 - Class Diagram

Dependency Injection & 제어의 역전 (IoC)

- IoC(Inversion of Control): 프로그램의 제어 프름 (객체의 생성과 생명주기 관리)을 코드로 작성하는 것이 아니라 외부에 위임. FastAPI의 Depends 시스템이 제어권을 가지도록 설계
- DI(Dependency Injection): 객체가 필요로하는 Repository, Adapter를 외부에서 주입

설계 이유

- 결합도 감소: MeetingService 안에서 MySQLMeetingRepository로 입력하면 결합도가 강해질 수 있음
- 테스트 용이성: 나중에 가짜 객체를 주입하여 실제 DB 연동이나 API 호출 없이 테스트를 진행할 수 있음




```python
def get_meeting_service(
    repository: MySQLMeetingRepository = Depends(get_meeting_repository),
    llm_client: GeminiLLMAdapter = Depends(get_llm_client)
) -> MeetingService:
    # Celery Task는 런타임에 가져오거나(순환참조 방지), None으로 처리
    try:
        from celery_worker import process_news_task
    except ImportError:
        process_news_task = None

    return MeetingService(
        repository=repository,
        llm_client=llm_client,
        celery_task=process_news_task
    )
```

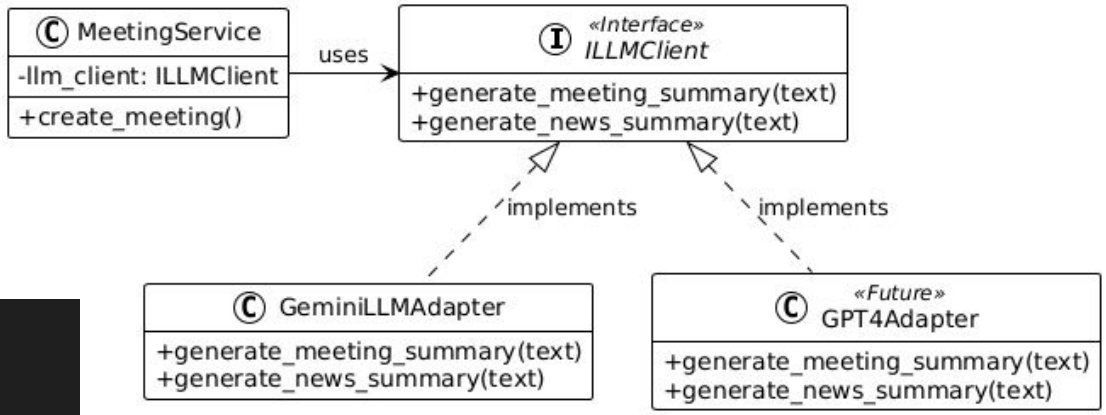
SW 설계 - Class Diagram

Strategy Pattern

- MeetingService: 전략을 사용하는 주체
- ILLMClient: 알고리즘의 공통적인 규격
- GeminiLLMAdapter: 실제 구현된 알고리즘

설계 이유

- 모델 교체의 용이성: 현재는 GeminiLLM만 사용하고 있지만 추후 GPT나 다른 LLMAPI를 사용할 수 있고, 이때 변경을 유연하게 할 수 있음




```python
class MeetingService:
    def __init__(
        self,
        repository: IMeetingRepository,
        llm_client: ILLMClient,
        celery_task = None  # 순환 참조 방지를 위해 런타임에 주입하거나 래퍼 사용
    ):
        self.repository = repository
        self.llm_client = llm_client
        self.celery_task = celery_task

    async def create_meeting(self, user_id: int, title: str, original_text: str) -> int:
        # 1. LLM 요약 수행
        try:
            summary, keywords = await self.llm_client.generate_meeting_summary(original_text)
        except Exception as e:
            # 로그 처리 필요
            raise HTTPException(
                status_code=status.HTTP_503_SERVICE_UNAVAILABLE,
                detail=f"회의록 분석 실패: {e}"
            )
```

SW 설계 - 성능과 효율성을 위한 비동기 아키텍처

```python
class MeetingService:
    def __init__(
        self,
        repository: IMeetingRepository,
        llm_client: ILLMClient,
        celery_task = None  # 순환 참조 방지를 위해 런타임에 주입하거나 래퍼
    ):
        self.repository = repository
        self.llm_client = llm_client
        self.celery_task = celery_task

    async def create_meeting(self, user_id: int, title: str, original_

    async def get_all_meetings(self, user_id: int) -> List[dict]: ···

    async def get_meeting_detail(self, meeting_id: int, user_id: int)

    async def delete_meeting(self, meeting_id: int): ···

    async def retry_news_analysis(self, meeting_id: int, user_id: int): ···
```

```python
async with httpx.AsyncClient(timeout=timeout) as client:
    for i in range(num_requests):
        start_index = 1 + i * news_per_request
        params = {
            "key": self.api_key,
            "cx": self.engine_id,
            "q": query,
            "num": news_per_request,
            "start": start_index,
            "sort": "date"
        }
        tasks.append(client.get(self.api_url, params=params))

    responses = await asyncio.gather(*tasks, return_exceptions=True)
```

- FastAPI의 Async 함수를 적극 활용하여 DB나 외부 API의 요청을 보내고 응답을 반환받을 동안 대기하지 않도록 설계
- httpx 라이브러리를 사용해 비동기 HTTP 요청보내고 이를 병렬로 처리

SW 설계 - 성능과 효율성을 위한 작업 큐 리

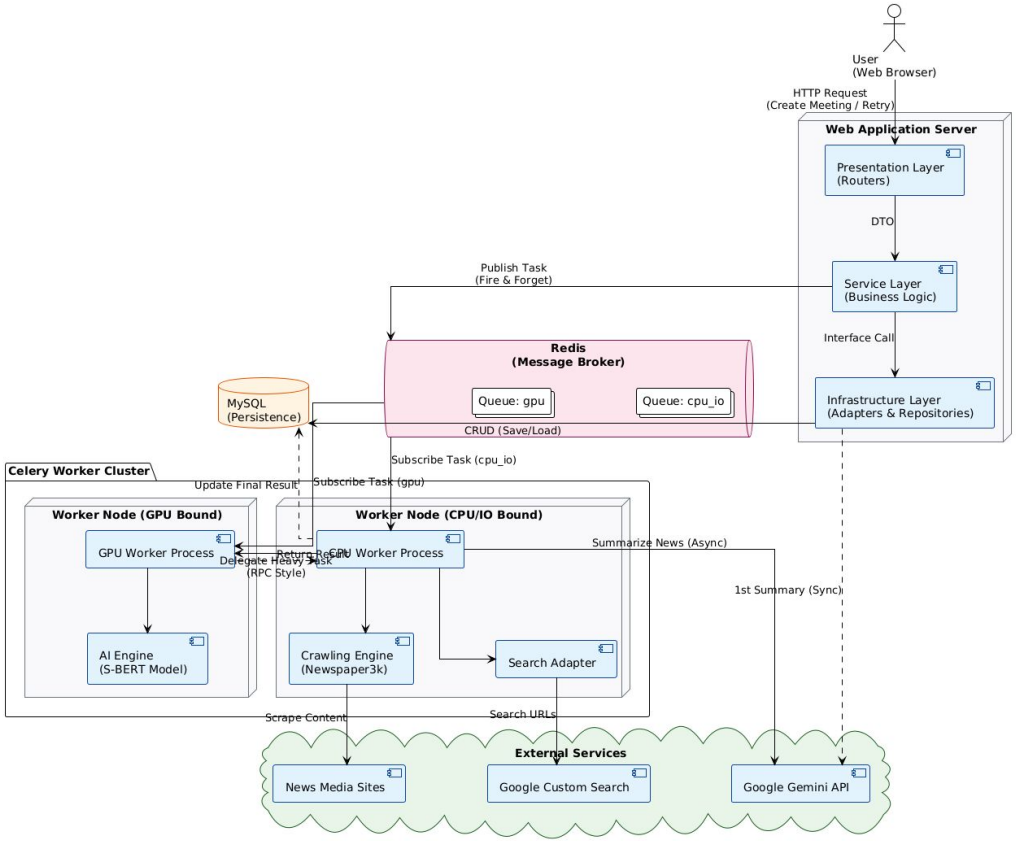
```python
celery_app.conf.update(
    timezone='Asia/Seoul',
    enable_utc=True,

    # 큐 정의: 일반 작업용(cpu_io)과 S
    task_queues=(
        Queue('cpu_io', default=Tru
        Queue('gpu'),
    ),

    # 기본 큐 설정
    task_default_queue='cpu_io',

    # 라우팅 설정: S-BERT 태스크는 무조건 'gpu' 큐로 보냄
    task_routes={
        'celery_worker.run_sbert_task': {'queue': 'gpu'}
    }
)
```

```python
# 3. S-BERT 분석 위임 (GPU 작업 호출 및 대기)
print("  [Step 3] S-BERT 분석 요청 (GPU 워커로 위임)...")
try:
    # allow_join_result()를 사용하여 하위 태스크가 끝날 때까지 대기
    with allow_join_result():
        selected_news = run_sbert_task.delay(summary_meeting, news_items).get(timeout=300)
    print(f"  [Step 3] S-BERT 분석 완료 (선별된 뉴스 {len(selected_news)}개)")
except Exception as e: ···
```

문제점: 뉴스 크롤링(단순 대기)과 S-BERT 연산(엄청난 계산)이 같은 작업 큐에 있을 경우, S-BERT 연산 시 자원을 다 써버려 뒤에 있는 크롤링 작업이 대기상태로 돌아가버림

해결: 작업의 성격에 따라 CPU / GPU 큐를 분리해서 설계

최종 산출물의 형태 및 기능



본 시스템의 최종 산출물

- **회의록의 요약문 및 키워드**
- **관련 뉴스의 요약 정보**
- **추출한 뉴스 URL**

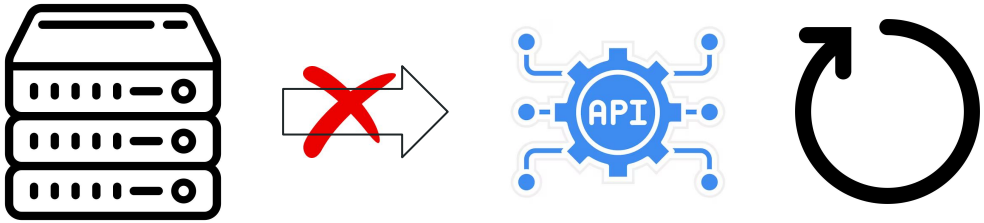
Risk Analysis & Reduction Plan

문제점 1: 외부 API (Google Gemini, Custom Search) 의존성에 따른 서비스 불안정성

- API 제공사의 서버 상태나 네트워크 지연, 또는 할당량 초과 발생 시 전체 서비스가 중단되거나 응답이 실패할 위험이 있다.

해결방안 1: 예외 처리 및 재시도 메커니즘 구현

- API 요청 실패 시 즉시 에러를 반환하지 않고 재시도를 수행할 수 있도록 구현한다. 또한 Fallback 프로세스를 마련한다.



Risk Analysis & Reduction Plan

문제점 2: 외부 API 접근 및 S-BERT 연산으로 인한 시스템 병목 현상

- . 외부 API 접근과 S-BERT 연산은 오래 걸리는 작업이므로, 요약부터 크롤링까지의 전 과정을 한 번에 처리한 후 사용자에게 보여주게 되면 병목 현상이 발생할 수 있음

해결방안 2: Celery & Redis 기반의 비동기 분산 처리

- 시간이 오래 걸리는 작업은 백그라운드로 처리하여 사용자의 웹 요청을 블로킹하지 않도록 설계
- S-BERT 모델을 서버 구동 시점이 아닌 실제 분석 요청이 들어왔을 때 로드하여 불필요한 메모리 점유를 최소화

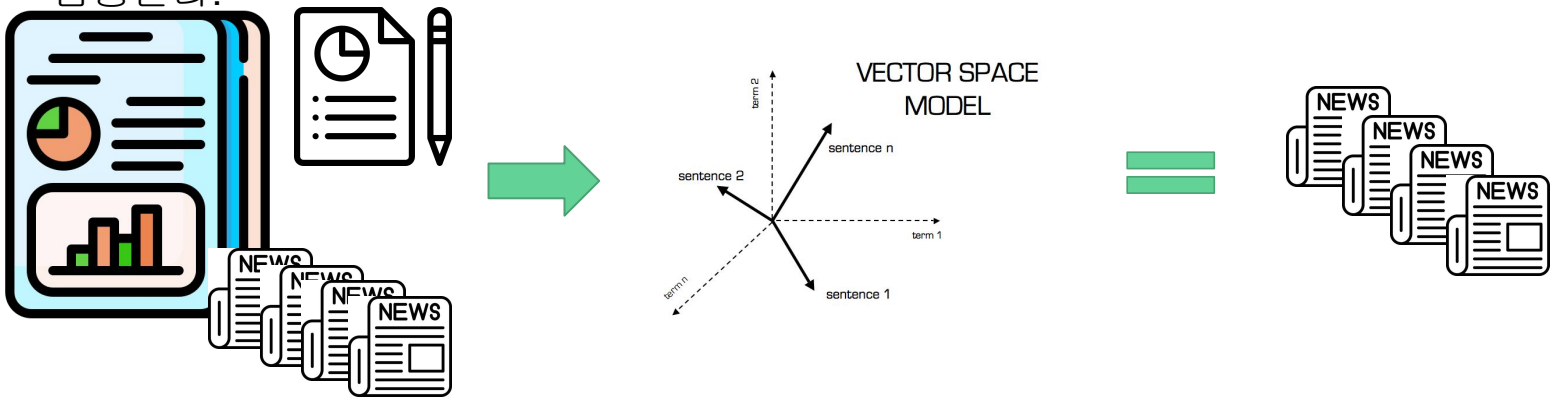
Risk Analysis & Reduction Plan

문제점 3: 뉴스 추천 정확도 저하

- 추출된 키워드가 단순 매칭되어 회의와 크게 상관없는 기사를 크롤링 할 수 있다.

해결방안 3: S-BERT 기반의 의미론적 필터링

- 단순히 키워드가 포함된 뉴스를 보여주는 것이 아니라, S-BERT를 통해 회의록 요약본과 뉴스 본문 전체의 벡터 유사도를 계산하여 상위 5개의 뉴스만을 엄선하여 문맥적 연관성을 검증한다.



시연 영상

https://youtu.be/FCrE6to9cIA

Success Criteria - 기능 요구사항

항목	설명	수행여부
사용자 관리	이메일 기반 회원가입, 로그인, 세션 유지, 로그아웃 기능	성공
회의록 생성 및 분석	사용자가 회의록 입력시 **LLM**을 통해 요약 및 키워드 추출	성공
뉴스 추천 (비동기)	추출된 키워드로 뉴스 검색 -> 클로링 -> 유사도 분석 ->요약	성공
뉴스 재분석	사용자가 원할 경우 특정 회의록의 뉴스 추천 로직만 재실행	성공
대시보드 (CRUD)	저장된 회의록 목록 조회, 상세 조회, 삭제 기능	성공

Success Criteria - 비기능 요구사항

항목	설명	수행여부
성능 및 안정성	회의록 작성 완료 버튼 클릭 후 **5**초 이내로 리다이렉트 **(**웹서버 병목 **X)** **Async/Await** 비동기 활용**, Queue Separation(CPU / GPU)** 백그라운드 작업이 실패하더라도 웹서버 전체가 죽지 않음	성공
유지보수성 확장성	**Layered Architecture**로 한 **Layer**의 수정이 다른 **Layer**에 영향을 주지 않음 **Interface, Adapter Pattern**을 활용하여 새로운 기능 추가 시 기존 코드 변경 불필요	성공
보안 안전성	로그인을 하지 않으면 **API** 접근 불가능	성공
정확성	사용자의 의도와 관련된 뉴스를 높은 정확도로 **(S-BERT** 기반**)** 추천	성공

Q&A

감사합니다