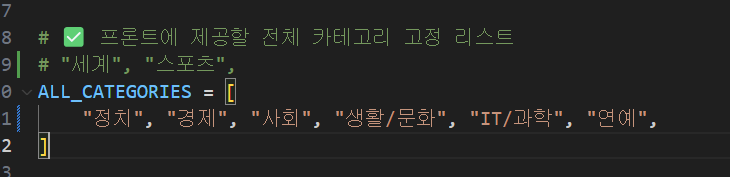
1. **app/constants/category\_map.py**
   1. "세계" 라인 주석 처리



1. **app/routes/category.py**
   1. 스포츠, 세계 제외



**1. 문제점 요약**

* 뉴스 기사 중복 저장 이슈
* DeepSearch API의 id 기준 중복 방지 한계
* 본문 추출 실패/누락/짧음으로 인한 요약 데이터 품질 저하
* 요약문이 너무 짧게 생성됨(900~1200자 문제 등)

**2. 해결 방향/핵심 변경점**

* **중복 필터 강화**  
  → news\_id, content\_url 모두 DB·메모리에서 체크  
  → 실제 저장 시 URL이 겹치면 저장하지 않음
* **본문 품질 필터**  
  → 본문 300자, 한글 비율 70% 기준 통과만 사용  
  → 누락시 selector/Trafilatura 기반 재추출
* **유사 뉴스 군집화 및 대표 요약**  
  → 임베딩 유사도 기반 뉴스 본문 군집화(0.8 이상)  
  → 비슷한 뉴스끼리 묶어서 대표 요약 1건 생성
* **요약 프롬프트 및 max\_tokens 상향**  
  → 1800~2000자 분량 반드시 생성하도록 프롬프트 명시  
  → max\_tokens=2200 등으로 설정

**3. 폴더별/파일별 변경 요약**

**app/utils/dynamo.py**

* get\_news\_card\_by\_content\_url() 함수 신규 추가 → DB content\_url 기준 중복 확인
* save\_news\_card()에서 content\_url 중복 체크 보강

**app/services/deepsearch\_service.py**

* fetch\_valid\_articles\_by\_category()에서
  + id, content\_url, title 중복 in-memory 체크
  + 필요시 DB 중복(주석처리/실전 사용 가능)
  + 본문 300자 미만/한글비율 미만이면 스킵

**app/tasks/collect\_news.py**

* get\_news\_card\_by\_id, get\_news\_card\_by\_content\_url 모두로 중복 필터
* 본문 필터/저장 로직 보강

**app/services/openai\_service.py**

* 임베딩 기반 유사 본문 군집화 함수(cluster\_similar\_texts) 추가
* summarize\_group(군집 대표 요약), summarize\_articles(전체 대본) 함수 분리
* max\_tokens, 프롬프트 등 개선

**app/tasks/generate\_frequency.py**

* 뉴스 본문 리스트 → 임베딩 군집화 → 각 군집 대표 요약 → 그 요약들로 전체 대본 생성
* 부족한 본문은 재추출/업데이트
* script(대본) 길이 체크 및 부족시 스킵

**코드 수정**

**1. app/utils/dynamo.py (변경!)**

|  |
| --- |
| import os  import boto3  from botocore.exceptions import ClientError  from datetime import datetime  from decimal import Decimal  from typing import Any  from app.constants.category\_map import CATEGORY\_MAP, REVERSE\_CATEGORY\_MAP  # ================================  # Decimal 변환 유틸 함수  # ================================  def deep\_convert(obj: Any) -> Any:      """      float → Decimal 변환 (DynamoDB 저장 시 오류 방지용)      """      if isinstance(obj, float):          return Decimal(str(obj))      elif isinstance(obj, list):          return [deep\_convert(i) for i in obj]      elif isinstance(obj, dict):          return {k: deep\_convert(v) for k, v in obj.items()}      return obj  # ================================  # DynamoDB 연결 및 테이블 객체  # ================================  dynamodb = boto3.resource("dynamodb")  news\_table = dynamodb.Table(os.getenv("DDB\_NEWS\_TABLE", "NewsCards"))  freq\_table = dynamodb.Table(os.getenv("DDB\_FREQ\_TABLE", "Frequencies"))  users\_table = dynamodb.Table(os.getenv("DDB\_USERS\_TABLE", "Users"))  bookmark\_table = dynamodb.Table(os.getenv("DDB\_BOOKMARKS\_TABLE", "Bookmarks"))  # ============================================  # 1. NewsCards 관련 함수  # ============================================  def save\_news\_card(category: str, article: dict, date\_str: str):      """      뉴스 기사 1건을 NewsCards 테이블에 저장      """      item = {          "news\_id": article["id"],          "category\_date": f"{category}#{date\_str}",  # GSI용 복합 키          "category": category,          "section": article.get("sections", [])[0] if article.get("sections") else "domestic",          "rank": article.get("rank"),          "title": article.get("title"),          "title\_ko": article.get("title\_ko"),          "summary": article.get("summary"),          "summary\_ko": article.get("summary\_ko"),          "image\_url": article.get("image\_url"),          "thumbnail\_url": article.get("thumbnail\_url") or article.get("thumbnail"),          "content\_url": article.get("content\_url"),          "publisher": article.get("publisher"),          "author": article.get("author"),          "published\_at": article.get("published\_at"),          "collected\_at": datetime.utcnow().isoformat(),          "companies": article.get("companies", []),          "esg": article.get("esg", []),          "content": article.get("content", "")      }      try:          news\_table.put\_item(Item=deep\_convert(item))      except ClientError as e:          raise Exception(f"[NewsCards 저장 실패] {e.response['Error']['Message']}")  def get\_news\_by\_category\_and\_date(category: str, date: str):      """      category와 date 기준으로 뉴스 목록 조회 (최신 30~60건)      """      key = f"{category}#{date}"      try:          response = news\_table.query(              IndexName="category\_date-index",              KeyConditionExpression="category\_date = :key",              ExpressionAttributeValues={":key": key}          )          return response.get("Items", [])      except ClientError as e:          raise Exception(f"[NewsCards 조회 실패] {e.response['Error']['Message']}")  def get\_news\_card\_by\_id(news\_id: str):      """      news\_id 기준으로 뉴스 상세 조회      """      try:          response = news\_table.get\_item(Key={"news\_id": news\_id})          return response.get("Item")      except ClientError as e:          raise Exception(f"[뉴스 상세 조회 실패] {e.response['Error']['Message']}")  def get\_today\_news\_grouped():      """      오늘 날짜 기준으로 카테고리별 뉴스 6건씩 묶어서 반환      """      today = datetime.now().strftime("%Y-%m-%d")      result = {}      for ko\_category, en\_category in CATEGORY\_MAP.items():          items = get\_news\_by\_category\_and\_date(en\_category["api\_name"], today)          result[ko\_category] = items[:6]      return result  def update\_news\_card\_content(news\_id: str, content: str):      """      news\_id 기준으로 본문(content) 필드 업데이트      """      try:          news\_table.update\_item(              Key={"news\_id": news\_id},              UpdateExpression="SET content = :c",              ExpressionAttributeValues={":c": content}          )      except ClientError as e:          raise Exception(f"[본문 업데이트 실패] {e.response['Error']['Message']}")  # ... 기존 dynamo.py 코드 끝에 붙이세요  def get\_news\_card\_by\_content\_url(content\_url: str):      """      content\_url 기준 뉴스 중복 여부 조회 (DB에서 스캔)      """      try:          response = news\_table.scan(              FilterExpression="content\_url = :url",              ExpressionAttributeValues={":url": content\_url}          )          items = response.get("Items", [])          if items:              return items[0]  # 이미 존재          return None      except ClientError as e:          raise Exception(f"[content\_url 중복 체크 실패] {e.response['Error']['Message']}")  # ============================================  # 2. Frequencies 관련 함수  # ============================================  def save\_frequency\_summary(item: dict):      """      공유 요약 스크립트 및 음성 정보 저장      """      try:          freq\_table.put\_item(Item=deep\_convert(item))      except ClientError as e:          raise Exception(f"[Frequencies 저장 실패] {e.response['Error']['Message']}")  def get\_frequency\_by\_category\_and\_date(category: str, date: str):      """      카테고리/날짜 기준 공유 요약 스크립트 조회      """      frequency\_id = f"{category}#{date}"      try:          response = freq\_table.get\_item(Key={"frequency\_id": frequency\_id})          return response.get("Item")      except ClientError as e:          raise Exception(f"[Frequencies 조회 실패] {e.response['Error']['Message']}")  # ============================================  # 3. Users 관련 함수  # ============================================  def save\_user(user: dict):      """      사용자 정보 저장 (신규 또는 업데이트)      """      if "created\_at" not in user:          user["created\_at"] = datetime.utcnow().isoformat()      if "profile\_image" not in user:          user["profile\_image"] = ""      try:          users\_table.put\_item(Item=user)      except ClientError as e:          raise Exception(f"[Users 저장 실패] {e.response['Error']['Message']}")  def get\_user(user\_id: str):      """      user\_id 기준 사용자 정보 조회      """      try:          response = users\_table.get\_item(Key={"user\_id": user\_id})          item = response.get("Item", {})          item.setdefault("profile\_image", "")          item.setdefault("created\_at", "")          return item      except ClientError as e:          raise Exception(f"[Users 조회 실패] {e.response['Error']['Message']}")  # ============================================  # 4. Bookmarks 관련 함수  # ============================================  def add\_bookmark(user\_id: str, news\_id: str):      """      북마크 추가 (user\_id + news\_id 조합)      """      item = {          "user\_id": user\_id,          "news\_id": news\_id,          "bookmarked\_at": datetime.utcnow().isoformat()      }      try:          bookmark\_table.put\_item(Item=item)      except ClientError as e:          raise Exception(f"[Bookmark 추가 실패] {e.response['Error']['Message']}")  def get\_user\_bookmarks(user\_id: str):      """      user\_id 기준으로 북마크 목록 조회      """      try:          response = bookmark\_table.query(              KeyConditionExpression="user\_id = :uid",              ExpressionAttributeValues={":uid": user\_id},              ScanIndexForward=False  # 최신 순 정렬          )          return response.get("Items", [])      except ClientError as e:          raise Exception(f"[Bookmark 조회 실패] {e.response['Error']['Message']}")  def remove\_bookmark(user\_id: str, news\_id: str):      """      북마크 삭제 (user\_id + news\_id 키)      """      try:          bookmark\_table.delete\_item(Key={"user\_id": user\_id, "news\_id": news\_id})      except ClientError as e:          raise Exception(f"[Bookmark 삭제 실패] {e.response['Error']['Message']}") |

**2. app/services/deepsearch\_service.py (변경!)**

|  |
| --- |
| import os  import httpx  from typing import List, Literal  from urllib.parse import urlparse  from bs4 import BeautifulSoup  import trafilatura  import re  from app.utils.dynamo import get\_news\_card\_by\_id, get\_news\_card\_by\_content\_url  from app.constants.category\_map import CATEGORY\_MAP  # ✅ 환경 변수에서 API 키 및 기본 설정 로드  DEEPSEARCH\_API\_KEY = os.getenv("DEEPSEARCH\_API\_KEY")  DEEPSEARCH\_BASE\_URL = "https://api-v2.deepsearch.com/v1"  USER\_AGENT = os.getenv(      "USER\_AGENT",      "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36"  )  HEADERS = {      "Authorization": f"Bearer {DEEPSEARCH\_API\_KEY}",      "Content-Type": "application/json",      "User-Agent": USER\_AGENT  }  # ✅ 주요 언론사별 도메인/본문 selector 지정  ARTICLE\_SELECTORS = {      "newsis.com": "div.view\_text",      "news1.kr": "div#articleBody",      "yna.co.kr": "div#articleBody, div#articleView",      "heraldcorp.com": "div.view\_con\_t",      "biz.heraldcorp.com": "div.view\_con\_t",      "kbs.co.kr": "div#cont\_newstext",      "sisajournal.com": "div.view\_con",      "asiatoday.co.kr": "div#articleBody",      "koreaherald.com": "div.article-text",      "sedaily.com": "div#v\_article",      "donga.com": "div.article\_txt",      "hankyung.com": "div#articletxt",      "joongang.co.kr": "div#article\_body",      "ohmynews.com": "div#article\_view",      "pressian.com": "div.view\_con\_tx",      "mt.co.kr": "div#textBody",      "edaily.co.kr": "div.news\_body",      "mk.co.kr": "div#article\_body",      "fnnews.com": "div.articleCont",      "busan.com": "div#news\_body\_area",  }  # ✅ 본문 내 "한글 비율" 체크 함수 (70% 이상만 유효 본문 인정)  def is\_korean\_text(text: str, threshold: float = 0.7) -> bool:      kor\_count = len(re.findall(r"[가-힣]", text))      total\_count = len(re.findall(r"[가-힣a-zA-Z]", text))      if total\_count == 0:          return False      return kor\_count / total\_count >= threshold  # ✅ 불필요한 안내/광고/추천/앱 영역 selector (BS4 제거용)  KNOWN\_TRASH\_SELECTORS = [      ".txt-copyright", ".adrs", ".sns-box", ".relate\_news", ".copy",      ".recommend", ".app-down", ".guide", ".comment", ".news\_app\_banner",      ".article-ad", "#recommend", "#comment", "#reply", ".promotion"  ]  UNWANTED\_KEYWORDS = [      "이 기사의 댓글 정책을 결정합니다",      "앱 다운", "빠르고 정확한 연합뉴스", "이 기사를 추천합니다",      "글자 크기 변경하기", "네이버 AI 뉴스 알고리즘",      "프리미엄콘텐츠", "본 기사와 무관한 광고", "해당 언론사에서 선정하며",      "사고로 해발", "기사 추천은", "모두에게 보여주고 싶은 기사라면",      "텍스트 음성 변환 서비스 사용하기", "글자로 지은 집7", "이 콘텐츠의 저작권은",      "섹션 정보는", "댓글 정책", "해당 언론사", "관련 업계에 따르면",      "인용된 모든 콘텐츠는", "당신의 의견을 남겨주세요", "클릭! 기사는 어떠셨나요?"  ]  # ✅ 본문 내 반복적으로 등장하는 안내/광고/제보/저작권 텍스트 정규식 패턴  REMOVE\_TEXT\_PATTERNS = [      # 기자 이름 + 이메일      r"^[가-힣]{2,4}\s?기자\s?[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$",      r"^[가-힣]{2,4}\s?기자$",      r"^[가-힣]{2,4}\s?\([^)]+@[^\s)]+\)$",      # 저작권 안내      r"^Copyright.\*", r"^무단[^\n]{0,20}전재.\*", r"^재배포 금지.\*",      # 제보 안내      r"^\[카카오톡\].\*", r"^\[메일\].\*", r"^\[전화\].\*",  ]  REMOVE\_TEXT\_PATTERNS\_COMPILED = [re.compile(pat) for pat in REMOVE\_TEXT\_PATTERNS]  EMAIL\_PATTERN = re.compile(r"[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}")  PHONE\_PATTERN = re.compile(r"\d{2,3}-\d{3,4}-\d{4}")  def clean\_text\_noise(text: str) -> str:      if not isinstance(text, str):          return ""      cleaned\_lines = []      for line in text.splitlines():          line = line.strip()          if not line:              continue          # 1. 줄 전체 제거 패턴 (기자, 저작권, 제보 등)          if any(pat.match(line) for pat in REMOVE\_TEXT\_PATTERNS\_COMPILED):              continue          # 2. 이메일, 전화번호 포함 여부 (줄 내부 검색)          if EMAIL\_PATTERN.search(line) or PHONE\_PATTERN.search(line):              continue          # 3. 일반 광고/안내 키워드 포함 여부 (in 연산자 기반)          if any(kw in line for kw in UNWANTED\_KEYWORDS):              continue          cleaned\_lines.append(line)      return "\n".join(cleaned\_lines)  # ✅ [방법1] selector + BS4 방식 본문 추출 (실패 시 전체 텍스트 fallback)  def extract\_content\_with\_bs4(url: str, timeout: float = 10.0) -> str:      """      [1] BeautifulSoup 기반 기사 본문 추출        - 기사 도메인별 selector 우선        - 없으면 전체 텍스트        - 광고/앱/댓글/추천 selector 영역 및 패턴 모두 제거        - 한글 인코딩 깨짐 방지(encoding 지정)        - 최종 clean\_text\_noise()로 정제      """      try:          response = httpx.get(              url,              timeout=timeout,              follow\_redirects=True,              headers={"User-Agent": USER\_AGENT}          )          # 한글 뉴스 인코딩 깨짐 방지(utf-8/CP949 등)          # response.encoding = response.apparent\_encoding or 'utf-8'          html = response.text          soup = BeautifulSoup(html, "html.parser")          # 광고/불필요 태그 삭제          for tag in soup(["script", "style", "noscript"]):              tag.decompose()          for selector in KNOWN\_TRASH\_SELECTORS:              for tag in soup.select(selector):                  tag.decompose()          # 도메인별 기사 selector → 없으면 전체 텍스트          domain = urlparse(url).netloc.replace("www.", "")          selector = ARTICLE\_SELECTORS.get(domain)          if selector:              article\_tag = soup.select\_one(selector)              if article\_tag:                  text = article\_tag.get\_text(separator="\n")              else:                  print(f"⚠️ selector 존재하나 해당 요소 없음: {domain} → {selector}")                  text = soup.get\_text(separator="\n")          else:              text = soup.get\_text(separator="\n")          # 본문 내 안내·광고·패턴·빈줄 제거          text = clean\_text\_noise(text)          return text.strip()      except Exception as e:          print(f"❌ [본문추출오류] {url} - {e}")          return ""  # ✅ [방법2] Trafilatura + BS4 혼합 방식 (최우선 방식)  def extract\_content\_flexibly(url: str, timeout: float = 10.0) -> str:      """      [2] Trafilatura 기반 본문 추출 우선        - 광고/앱/추천 selector/패턴 모두 제거        - 길이/문장수/한글비율 기준 통과 시만 반환(실패 시 BS4 fallback)        - \*실제 모든 뉴스 추출 함수에서 이 함수로 통일\*      """      try:          # Trafilatura로 원본 HTML fetch          html = trafilatura.fetch\_url(url)          if html:              soup = BeautifulSoup(html, "html.parser")              for selector in KNOWN\_TRASH\_SELECTORS:                  for tag in soup.select(selector):                      tag.decompose()              content = trafilatura.extract(str(soup))              # 길이/문장수/한글뉴스 기준 모두 만족해야 반환              if content and len(content) >= 300 and content.count('.') >= 5:                  content = clean\_text\_noise(content)                  if is\_korean\_text(content, threshold=0.7):  # 한글뉴스만 통과                      return content.strip()          # Trafilatura 실패/미달시 fallback: BS4 방식          text = extract\_content\_with\_bs4(url)          if text and len(text) >= 300 and text.count('.') >= 5:              text = clean\_text\_noise(text)              if is\_korean\_text(text, threshold=0.7):                  return text          return ""      except Exception as e:          print(f"❌ [혼합본문추출오류] {url} - {e}")          return ""  # ✅ [뉴스 수집용] fetch\_valid\_articles\_by\_category:  #    - 딥서치 API로 기사 리스트 조회 후 각 기사에 대해 extract\_content\_flexibly로 본문 추출 및 저장  def fetch\_valid\_articles\_by\_category( #여기를 수정정      category: str,      start\_time: str,      end\_time: str,      size: int = 60,      limit: int = 30,      sort: Literal["popular", "traffic"] = "popular",      section: Literal["domestic", "international"] = "domestic",      min\_content\_length: int = 300,      max\_try: int = 5  ) -> List[dict]:      seen\_ids = set()      seen\_urls = set()      seen\_titles = set()      results = []      page = 1      while len(results) < limit and page <= max\_try:          url = (              f"{DEEPSEARCH\_BASE\_URL}/articles/{category}"              if section == "domestic"              else f"{DEEPSEARCH\_BASE\_URL}/global-articles/{category}"          )          params = {              "date\_from": start\_time,              "date\_to": end\_time,              "sort": sort,              "page\_size": size,              "page": page          }          response = httpx.get(url, headers=HEADERS, params=params)          response.raise\_for\_status()          all\_articles = response.json().get("data", [])          if not all\_articles:              break          for article in all\_articles:              news\_id = article.get("id")              article\_url = article.get("content\_url")              title = article.get("title", "").strip()              # 1. 메모리 기준 중복 필터              if not article\_url or not news\_id:                  continue              if news\_id in seen\_ids or article\_url in seen\_urls:                  continue              if title and title in seen\_titles:                  continue              # 2. DB 기준 중복 필터 (아래 DB 중복 필터 코드는 실제로 운영 시 꼭 활성화  )              # if get\_news\_card\_by\_id(news\_id):              #     continue              # if get\_news\_card\_by\_content\_url(article\_url):              #     continue              seen\_ids.add(news\_id)              seen\_urls.add(article\_url)              if title:                  seen\_titles.add(title)              # 3. 본문 추출 및 유효성 필터              content = extract\_content\_flexibly(article\_url)              if not content or len(content) < min\_content\_length:                  continue              if not is\_korean\_text(content, threshold=0.7):                  continue              article["content"] = content              results.append(article)              if len(results) >= limit:                  break          page += 1      return results[:limit] |

**3. app/tasks/collect\_news.py (변경!)**

|  |
| --- |
| # app/tasks/collect\_news.py  import logging  from datetime import datetime  import pytz  from app.services.deepsearch\_service import fetch\_valid\_articles\_by\_category  from app.utils.dynamo import save\_news\_card, get\_news\_card\_by\_id  from app.constants.category\_map import CATEGORY\_MAP  from app.utils.dynamo import get\_news\_card\_by\_content\_url  # 로그 설정  logger = logging.getLogger()  logger.setLevel(logging.INFO)  def collect\_today\_news():      """      ✅ 매일 오전 6시 자동 실행: 자정~06시 사이의 뉴스 수집 및 저장      - CATEGORY\_MAP에 정의된 카테고리별로 인기 뉴스 호출      - 본문 길이 300자 이상인 기사만 저장      - 본문은 selector 기반 추출 (fallback 포함)      - 저장 시점에 본문도 함께 DynamoDB에 저장 (요약 및 TTS 생성을 위함)      """      # 한국시간 기준 오늘 날짜 범위 설정      kst = pytz.timezone("Asia/Seoul")      now = datetime.now(kst)      date\_str = now.strftime("%Y-%m-%d")      start\_time = f"{date\_str}T00:00:00"      end\_time = f"{date\_str}T06:00:00"      # 전체 카테고리 순회      for category\_ko, config in CATEGORY\_MAP.items():          category\_en = config["api\_name"]          section = config["section"]          logger.info(f"📰 {category\_ko} 뉴스 수집 시작 ({section})")          # 뉴스 API 호출          try:              articles = fetch\_valid\_articles\_by\_category(                  category=category\_en,                  start\_time=start\_time,                  end\_time=end\_time,                  size=80,                # 오버페치 후 필터링                  sort="popular",                  section=section,                  min\_content\_length=300,                  limit=30               # 최종 저장 수              )              logger.info(f"📥 {category\_ko} 유효 기사 수: {len(articles)}")          except Exception as e:              logger.error(f"[{category\_ko}] API 호출 실패: {e}")              continue          saved\_count = 0          # 기사 순회하며 저장          for rank, article in enumerate(articles, start=1):              news\_id = article.get("id")              if not news\_id:                  logger.warning(f"[{category\_ko}] ❌ ID 누락 → 스킵")                  continue              #여기 수정              # 중복 확인              if get\_news\_card\_by\_id(news\_id):                  logger.info(f"🚫 [ID중복] 뉴스 스킵: {news\_id}")                  continue              if get\_news\_card\_by\_content\_url(article.get("content\_url")):                  logger.info(f"🚫 [URL중복] 뉴스 스킵: {article.get('content\_url')}")                  continue              content = article.get("content", "")              if not content or len(content) < 300:                  logger.warning(f"[{category\_ko}] ⚠️ 본문 누락/부족 → 스킵: {news\_id}")                  continue              # 뉴스 저장 아이템 구성              news\_item = {                  "id": news\_id,                  "sections": article.get("sections", []),                  "rank": rank,                  "title": article.get("title"),                  "title\_ko": None,                  "summary": article.get("summary"),                  "summary\_ko": None,                  "image\_url": article.get("image\_url"),                  "thumbnail\_url": article.get("thumbnail\_url") or article.get("thumbnail"),                  "content\_url": article.get("content\_url"),                  "publisher": article.get("publisher"),                  "author": article.get("author"),                  "published\_at": article.get("published\_at"),                  "companies": article.get("companies", []),                  "esg": article.get("esg", []),                  "content": content  # ✅ 본문 포함 (selector 기반)              }              # DynamoDB 저장              try:                  save\_news\_card(category\_en, news\_item, date\_str)                  saved\_count += 1                  logger.info(f"✅ 저장 완료: {category\_ko} #{rank} - {news\_item['title']}")              except Exception as e:                  logger.error(f"[저장 실패] {category\_ko} #{rank}: {e}")          logger.info(f"📊 {category\_ko} 최종 저장 수: {saved\_count}") |

**4. app/services/openai\_service.py**

|  |
| --- |
| # app/services/openai\_service.py  import os  import openai  import numpy as np   # 추가!  openai.api\_key = os.getenv("OPENAI\_API\_KEY")  def get\_embedding(text: str) -> list:      res = openai.embeddings.create(          input=[text],          model="text-embedding-3-small"      )      return res.data[0].embedding  def cosine\_similarity(vec1, vec2):      return np.dot(vec1, vec2) / (np.linalg.norm(vec1) \* np.linalg.norm(vec2))  def cluster\_similar\_texts(texts, threshold=0.8):      embeddings = [get\_embedding(t) for t in texts]      clusters = []      for idx, emb in enumerate(embeddings):          added = False          for cluster in clusters:              if cosine\_similarity(emb, cluster['embedding']) > threshold:                  cluster['indices'].append(idx)                  added = True                  break          if not added:              clusters.append({'embedding': emb, 'indices': [idx]})      grouped = [[texts[i] for i in c['indices']] for c in clusters]      return grouped  def summarize\_group(texts: list, category: str) -> str:      prompt = (          f"다음은 '{category}' 분야에서 비슷한 뉴스 기사들입니다. "          f"중복 내용을 합치고, 하나의 자연스러운 요약(600자~800자)으로 써주세요. "          f"핵심 이슈와 내용만 간결하게 담아주세요.\n\n"          + "\n\n".join(texts)      )      response = openai.chat.completions.create(          model="gpt-4o-mini",          messages=[{"role": "user", "content": prompt}],          temperature=0.7,          max\_tokens=900      )      return response.choices[0].message.content.strip()  def summarize\_articles(texts: list[str], category: str) -> str:      prompt = (          f"당신은 지적이고 친절한 팟캐스트 진행자입니다. "          f"다음은 '{category}' 카테고리의 주요 뉴스 기사 요약입니다.\n\n"          "각 기사들의 내용을 기반으로, 청취자에게 쉽고 흥미롭게 전달할 수 있도록 "          "하나의 흐름을 가진 1800자 이상 2000자 이하 분량의 대본을 작성해주세요.\n\n"          "조건:\n"          "- 반드시 1800자 이상이 되어야 합니다.\n"          "- 너무 딱딱한 뉴스톤이 아니라 자연스럽고 전달력 있는 말투\n"          "- 중복 표현 없이 핵심 내용만 간결하게 구성\n"          "-'오늘 {category} 분야에서는 이런 이슈들이 있었습니다.'와 같은 도입부 및 마무리 멘트 포함\n"          "- 청취자가 실제로 듣는 팟캐스트처럼 읽기 좋게 구성"      )      context = prompt + "\n\n기사 요약 목록:\n" + "\n\n".join(texts)      response = openai.chat.completions.create(          model="gpt-4o-mini",          messages=[{"role": "user", "content": context}],          temperature=0.7,          max\_tokens=2200      )      return response.choices[0].message.content.strip() |

**# app/tasks/generate\_frequency.py**

|  |
| --- |
| # app/tasks/generate\_frequency.py  import logging  from datetime import datetime  from app.utils.date import get\_today\_kst  from app.utils.dynamo import (      save\_frequency\_summary,      get\_frequency\_by\_category\_and\_date,      get\_news\_by\_category\_and\_date,      update\_news\_card\_content,  )  from app.services.openai\_service import summarize\_articles  from app.services.tts\_service import text\_to\_speech  from app.utils.s3 import upload\_audio\_to\_s3\_presigned  from app.constants.category\_map import CATEGORY\_MAP  from app.services.deepsearch\_service import extract\_content\_with\_bs4  from app.services.openai\_service import summarize\_articles, cluster\_similar\_texts, summarize\_group  # 로그 설정  logger = logging.getLogger()  logger.setLevel(logging.INFO)  def generate\_all\_frequencies():      """      ✅ 매일 오전 6시 자동 실행: 카테고리별 뉴스 본문 기반 공유 대본/음성 생성      - 뉴스카드 DB에서 카테고리별 기사 최대 30개 사용      - 부족한 본문은 재추출      - 기사 본문을 GPT 요약하여 스크립트 생성      - ElevenLabs TTS로 변환 후 S3 업로드      - Frequencies 테이블에 저장 (스크립트 + Presigned MP3 링크)      """      date = get\_today\_kst()      all\_categories = CATEGORY\_MAP.keys()      logger.info(f"🌀 전체 카테고리 순회: {list(all\_categories)}")      for category\_ko in all\_categories:          category\_en = CATEGORY\_MAP[category\_ko]["api\_name"]          freq\_id = f"{category\_en}#{date}"          # 중복 방지: 이미 생성된 경우 스킵          if get\_frequency\_by\_category\_and\_date(category\_en, date):              logger.info(f"🚫 이미 생성됨 → 스킵: {freq\_id}")              continue          logger.info(f"📚 대본/음성 생성 시작: {category\_en} ({freq\_id})")          try:              # 해당 카테고리의 오늘 기사 불러오기              articles = get\_news\_by\_category\_and\_date(category\_en, date)              logger.info(f"📥 수집된 기사 수: {len(articles)}")              full\_contents = []              # 기사 본문 최대 30개까지 수집              for i, article in enumerate(articles[:40]):                  news\_id = article.get("news\_id")                  url = article.get("content\_url")                  content = article.get("content", "").strip()                  if not news\_id or not url:                      logger.warning(f"⚠️ #{i+1} URL 또는 ID 없음 → 스킵")                      continue                  # 본문이 짧거나 없으면 재추출 시도                  if not content or len(content) < 300:                      try:                          content = extract\_content\_with\_bs4(url)                          if content and len(content) >= 300:                              update\_news\_card\_content(news\_id, content)                              logger.info(f"♻️ #{i+1} 본문 보완 저장 완료 ({len(content)}자)")                          else:                              logger.warning(f"⚠️ #{i+1} 본문 추출 실패 또는 너무 짧음 → 스킵")                              continue                      except Exception as e:                          logger.warning(f"❌ #{i+1} 본문 재추출 중 오류: {e}")                          continue                  # 요약 전 텍스트 길이 제한 (토큰 초과 방지)                  trimmed = content[:3000]                  full\_contents.append(trimmed)                  logger.info(f"✅ #{i+1} 본문 사용 완료 ({len(trimmed)}자)")                  if len(full\_contents) >= 30:                      break              # 본문 수가 너무 적으면 스킵              if len(full\_contents) < 5:                  logger.warning(f"⚠️ 유효 본문 부족 → 스킵: {category\_en}")                  continue               # ⭐️ [신규] 중복/유사 기사 군집화              groups = cluster\_similar\_texts(full\_contents, threshold=0.80)              group\_summaries = []              for group in groups:                  if len(group) == 1:                      group\_summaries.append(group[0])                  else:                      # 여러 본문 묶음 → 대표 요약문 생성                      summary = summarize\_group(group, category\_en)                      group\_summaries.append(summary)              # GPT로 종합 스크립트 요약 생성              script = summarize\_articles(group\_summaries, category\_en)              if not script or len(script) < 500:                  logger.warning(f"⚠️ 요약 길이 부족 → 스킵: {category\_en}")                  continue              logger.info(f"📝 요약 완료: {len(script)}자")              # ElevenLabs로 TTS 변환 → S3 Presigned URL 생성              try:                  audio\_bytes = text\_to\_speech(script)                  audio\_url = upload\_audio\_to\_s3\_presigned(                      file\_bytes=audio\_bytes,                      user\_id="shared",                      category=category\_en,                      date=date,                      expires\_in\_seconds=86400  # Presigned URL 24시간 유효                  )                  logger.info(f"🔊 TTS Presigned URL 생성 완료")              except Exception as e:                  logger.warning(f"[TTS 업로드 실패] {category\_en}: {str(e)}")                  continue              # 결과 DynamoDB에 저장              item = {                  "frequency\_id": freq\_id,                  "category": category\_en,                  "date": date,                  "script": script,                  "audio\_url": audio\_url,                  "created\_at": datetime.utcnow().isoformat()              }              save\_frequency\_summary(item)              logger.info(f"✅ DynamoDB 저장 완료: {category\_en} ({freq\_id})")          except Exception as e:              logger.exception(f"[❌ {category\_en} 처리 실패] {str(e)}") |

requirements.txt

|  |
| --- |
| fastapi  uvicorn  python-dotenv  requests  boto3  httpx  openai  mangum  jwt  pytz  python-jose  bs4  trafilatura  numpy |