ChatGPT

퀄리저널 개발 계획서 (실전 중심 가이드)

1. 프로젝트 개요 및 목표 요약

퀄리저널(QualiJournal)은 “하루 한 키워드”를 중심으로 전자생산 및 기술표준 분야 뉴스를 역사적으로 큐레이션하는 플랫폼입니다. 편집자가 하루에 하나의 키워드를 선택하면 그와 관련된 **공식 뉴스, 학술 논문, 표준/정책 문서, 기업 블로그, 커뮤니티 글** 등을 폭넓게 수집하여, **최소 15건 이상의 고품질 기사 카드로 구성된 특별 호(news issue)**를 발행하는 것이 목표입니다 [1]. 이러한 키워드 뉴스는 기존의 일일 뉴스 모아보기와 별도로 동작하는 새로운 발행 라인으로서, 수집부터 편집, 발행까지 전 과정을 키워드 중심으로 처리합니다 [1].

퀄리저널이 해결하려는 문제는 **단편적인 최신 뉴스로는 얻기 어려운 심층 맥락**을 제공하는 것입니다. 하나의 키워드에 대해 **과거 10년 이상**의 관련 이슈와 자료를 축적하여 독자가 해당 기술/표준의 역사와 흐름을 한눈에 파악할 수 있도록 돕습니다 [2]. 예를 들어 “리플로우(Reflow)” 같은 키워드에 대해 관련 기술 뉴스, 관련 산업표준 변경 내역, 학술 연구, 커뮤니티 토론까지 맥락을 엮어 제공함으로써, 특정 키워드에 대한 **깊은 이해와 통찰**을 얻도록 하는 것입니다 [3] [4]. 이를 위해 다양한 출처의 데이터를 키워드로 연결해 **키워드 단위 역사 데이터베이스**를 구축하고, 신뢰도 낮은 자료는 걸러내는 품질관리 과정을 거칩니다 [5] [4]. 결과적으로 **키워드 기반 뉴스의 가치**는, 독자가 흩어진 정보를 하나로 묶은 **맥락 있는 스토리텔링 뉴스**를 접하게 하고, 기술 및 표준의 역사적 발전을 이해하도록 한다는 데 있습니다.

2. 전체 시스템 아키텍처

퀄리저널 시스템은 **수집기(collector)**, **평가기(evaluator)**, **요약기/번역기(summarizer)**, **발행기(publisher)**, **관리자 UI** 및 **저장소(storage)**의 계층으로 구성된 **계층형 아키텍처**를 갖습니다 [6]. 각 계층의 역할은 다음과 같습니다:

- **소스 계층(Source Feeds)**: 키워드 뉴스를 위해 참조할 여러 **데이터 소스** 목록을 정의합니다. 예를 들어 산업계 공식 뉴스 사이트 RSS, 학술 논문 API, 표준 문서 사이트, Reddit 등의 커뮤니티 URL 등을 JSON으로 관리합니다 [7] [8]. 이 계층의 설정은 `feeds/official_sources.json`, `feeds/community_sources.json`, `feeds/paper_sources.json` 등 파일로 구성됩니다.

- **수집기(Collector)**: 입력된 키워드에 대해, 사전에 정의된 모든 출처들을 대상으로 해당 키워드 관련 **데이터 수집 작업**을 수행합니다. 수집기는 뉴스, 논문, 표준, 커뮤니티 등 **유형별 모듈**로 나눠 동작하며, Python 기반의 크롤러와 API 연동으로 구현됩니다 [6]. 각 수집 모듈은 RSS 피드 파싱, 웹 크롤링, API 호출 등을 통해 **원문 데이터**(제목, 본문 등)를 가져옵니다 (아래 3번에서 상세 설명). 수집 결과는 우선 **JSON 형태**로 저장되며, 초기 단계에서는 파일(JSON)이나 경량 DB(SQLite/PostgreSQL)를 사용하고 향후 필요 시 Elasticsearch 같은 **검색 최적화 저장소**로 확장할 수 있습니다 [6].

- **평가기(Evaluator)**: 수집된 원문 자료들에 대해 **품질 게이트(Quality Gate) 필터와 점수 산정**을 수행합니다. 각 자료의 본문 길이, 신뢰도, 키워드 언급 빈도 등을 기준으로 자동 평가하여 저품질/무관한 항목은 걸러냅니다 [9]. 또한 **도메인 신뢰도, 키워드 적합도, 시간적 중요도** 등의 요소로 가중치를 계산해 점수를 부여합니다 [10]. 이 단계 결과로 기사별 메타데이터와 점수가 산출되며, `{id, keyword, score, type, meta, status}` 형식의 구조로 임시 저장됩니다 [9].

- **요약기/번역기(Summarizer)**: 수집된 영어 자료나 긴 텍스트에 대해서는 핵심 내용을 발췌하여 **한국어 요약문**을 생성합니다. 예를 들어 영문 뉴스 기사나 논문의 초록을 입력으로 받아 **중요 문장 추출** 후 OpenAI API 등의 번역기를 활용해 2~3문장 분량의 한국어 요약을 만듭니다 [11]. 번역 API 키가 없거나 자동 번역이 어려운 경

우는 규칙 기반 요약으로 대체하고, 사전에 정의한 **용어 Glossary**를 적용하여 전문용어 번역의 일관성을 유지합니다 [11]. 이렇게 요약/번역된 결과는 이후 편집자 검수 및 최종 발행에 사용될 데이터에 포함됩니다.

- **저장소(Storage) 계층**: 수집된 원자료와 평가/요약 결과는 **JSON 파일** 또는 **데이터베이스**에 저장됩니다. 개발 초기에는 각 키워드별 후보 기사를 `selected_keyword_articles.json` 같은 JSON으로 저장하고, 필요 시 경량 DB로 마이그레이션합니다 [6]. 추후 데이터가 많아지면 Elasticsearch 등 **검색 최적화 DB** 도입을 검토하여 키워드별 빠른 검색/조회가 가능하도록 확장합니다 [6]. (섹션 7에서 데이터 구조 상세 설명)

- **관리자 UI(Admin Interface)**: 편집자가 수집된 후보 기사들을 검토하고 발행 여부를 결정하는 **대시보드**입니다. Flask 또는 FastAPI 기반 백엔드와 간단한 프론트엔드(Vue/React 등)를 사용하여 구현합니다 [12]. 이 UI에서는 해당 키워드로 수집된 기사 목록을 **유형별, 점수별, 날짜순**으로 필터링/정렬해서 볼 수 있고, 각 기사에 대해 승인 여부와 코멘트를 입력할 수 있습니다 [13]. 관리자는 UI에서 **"발행" 버튼**을 눌러 최종 결과물을 생성하도록 트리거하며, 이 때 승인된 기사들만 모아 정제된 뉴스 페이지가 만들어집니다 (자세한 기능은 섹션 6 참조).

- **발행기(Publisher)**: 발행 단계에서는 승인된 기사들을 취합하여 **HTML/Markdown 형식의 최종 뉴스 페이지**를 생성합니다. 자료를 콘텐츠 유형별로 구분하고 시간순으로 배열하여 하나의 **키워드 역사 페이지**를 구성하며, 이를 `archive/<날짜>_<키워드>.html` 등의 파일로 저장합니다 [14]. 생성된 페이지는 즉시 웹에 게시하거나 정적 페이지로 배포되며, 아카이브 폴더에 누적되어 이후에도 열람할 수 있도록 버전 관리됩니다 [15].

以上의 구성요소들은 독립적인 모듈로 개발되어 파이프라인 형태로 연동됩니다. 예를 들어 **[키워드 입력] -> [수집기 모듈들 병렬 실행] -> [품질평가/요약] -> [JSON/DB 저장] -> [관리자 승인] -> [발행기 출력]** 의 흐름으로 동작하며, 각 단계는 CLI 명령어나 스케줄러에 의해 순차적으로 실행됩니다. 전체 시스템은 cron 또는 APScheduler를 활용해 정기 작업을 예약하거나, 관리자가 수동으로 키워드 수집을 트리거할 수 있습니다 [12]. Python 기반 구현으로 빠른 개발이 가능하며, 추후 검색 기능 강화를 위해 ElasticSearch 연동도 고려한 **유연한 아키텍처**입니다 [6].

3. 주요 모듈별 기능 정의 및 우선 구현 순서

퀄리저널은 여러 기능 모듈로 구성되며, 특히 **수집 모듈들을 순차적으로 구현**한 후 평가, 요약, 발행 모듈을 개발하는 일정으로 진행합니다 [16]. 수집 모듈의 구현 순서는 **학술 논문 → 표준 문서 → 뉴스/블로그 → 커뮤니티** 순을 권장합니다. 비교적 **구조화된 데이터**부터 우선 처리하고, 점차 **비정형 데이터** 수집으로 확장하는 흐름으로 개발함으로써 난이도를 단계적으로 높일 수 있기 때문입니다 (예: 논문/표준은 공식 API나 검색이 비교적 규칙적이고, 뉴스/블로그는 RSS 등 활용, 커뮤니티는 비정형 텍스트와 품질편차가 크므로 마지막에 구현) [17] [18]. 아래 각 모듈의 기능과 역할을 정의하며, 구현 우선순위를 표시합니다:

3.1 수집 모듈 (Collector Modules)

1. **학술 논문 수집 모듈 (collect_papers.py) – [1순위 구현]**
   기능: 입력 키워드와 관련된 **학술 연구 논문**의 메타데이터를 수집합니다. ArXiv, IEEE Xplore, Semantic Scholar 등의 API나 RSS 피드를 활용하여 키워드 검색을 수행합니다 [19] [20]. 예를 들어 ArXiv의 API 엔드포인트(`search_query={keyword}`)로 해당 키워드가 포함된 논문을 질의합니다. 수집된 논문에 대해 **제목, 저자, 발표년도, 초록(abstract), DOI** 등의 정보를 저장하며, 특히 제목이나 초록에 **키워드가 실제로 언급**되어 있는지 확인하여 관련성이 충분한 논문만 선별합니다 [21] [22]. 논문 전문 PDF까지 크롤링하지는 않지만, 메타데이터와 초록을 통해 키워드와의 연관도를 판단합니다. (구현노트: ArXiv 외에 추가 소스가 있다면 modular하게 확장 가능하도록 인터페이스 설계)

2. **표준/정책 문서 수집 모듈 (collect_standards.py) – [2순위 구현]**
   기능: IPC, ISO, IEC, JEDEC 등 **표준화 기구**나 정부의 기술 정책 문서 중 키워드와 연관된 자료를 수집합니다. RSS 피드가 없는 경우가 많으므로, 해당 기관 사이트의 **검색 페이지를 직접 스크레이핑**하거나 공개된 문서 목록을 파싱하는 방식으로 구현합니다 [23] [24]. 예를 들어 키워드가 특정 표준 규격 코드(IPC-2581, J-STD-001

등)라면 해당 키워드로 사이트 내 검색을 수행하고 관련 PDF 링크와 문서 메타정보를 가져옵니다. 수집 데이터에는 **문서 제목, 출처(기관명), 발행일자, 문서 URL** 등을 저장합니다 [23]. 표준 문서는 일반 기사보다 신뢰도가 높으므로 본문 전문을 굳이 파싱하지 않고 링크와 간략 요약만 저장해도 충분합니다 [23]. (구현노트: PDF 파일을 직접 다운로드해 서버에 캐시할지 여부는 초기에는 링크만 저장하고 필요시 추후 캐싱을 고려합니다)

3. **공식 뉴스/블로그 수집 모듈 (collect_news.py) – [3순위 구현]**
   기능: 키워드와 관련된 **공식 뉴스 기사 및 기업 블로그 글**을 수집합니다. 미리 설정된 전문 매체의 RSS 피드를 순회하며, 각 피드에서 **최신 기사 목록**을 불러옵니다 [25]. 가져온 목록의 **제목(title)** 또는 **요약(description)**에 키워드가 포함된 기사만 1차 선별하여 상세 크롤링 대상으로 삼습니다 [26]. RSS 피드가 없는 출처의 경우 **목록 페이지 HTML**을 파싱하여 최신 글 링크를 추출합니다 [27]. 선별된 기사 링크에 대해서는 `_pick_first_article_url()` 등의 함수를 이용해 실제 본문 페이지를 찾아내고, **제목과 본문을 크롤링**하여 저장합니다 [28]. 이 때 기사 본문에 해당 키워드가 존재하는지 다시 한 번 확인하고, 제목/본문 어디에도 키워드가 없으면 해당 기사는 제외합니다 [29]. 키워드 일치 시 대소문자 무시, 영문-한글 동의어 매핑 등을 고려하여 포착률을 높이는 것이 중요합니다 [30]. 예를 들어 편집자가 한글 키워드 "인공지능"을 입력했다면 "AI", "Artificial Intelligence"와 같은 영문 동의어나 약어도 함께 검색하여 영문 기사도 찾아내도록 합니다 [30]. 또한 키워드 관련 **과거 기사 보강**도 고민하는데, 최근 뉴스에 키워드 언급이 적을 경우 Google News API나 Bing News 검색을 통해 **과거 기사**를 추가 수집하는 방법을 2차 단계로 고려할 수 있습니다 [31] (초기 구현에서는 복잡도를 낮추기 위해 생략 가능). 뉴스/블로그 수집 모듈은 결과적으로 키워드가 포함된 공식 기사들의 **제목, 본문 전문, 출처, 발행일, URL** 등을 수집 데이터로 반환합니다.

4. **커뮤니티 글 수집 모듈 (collect_community.py) – [4순위 구현]**
   기능: Reddit, 전문 포럼 등 **커뮤니티 사용자 글** 중 키워드 관련 글을 수집합니다. 우선 Reddit의 API를 이용하여 사전에 설정된 서브레딧 목록에서 키워드로 **전체 텍스트 검색**을 수행합니다 [32] [33]. 예를 들어 r/electronics, r/SMT 등의 서브레딧에서 해당 키워드가 언급된 포스트를 검색하여 인기순으로 결과를 가져올 수 있습니다 [34]. 또한 Reddit 전체를 대상으로 키워드 검색 API를 활용하여 최신 화제 글을 찾는 것도 고려합니다 [34]. Reddit 이외의 커뮤니티(예: EEVblog 포럼, HackerNews 등)의 경우, RSS 피드 제공 시 RSS를 활용하고 그렇지 않으면 웹페이지를 크롤링하여 게시글 **제목, 조회수, 댓글수** 등을 추출합니다 [32] [18]. 추출된 커뮤니티 글 중 키워드가 제목이나 본문에 언급된 경우만 수집하며, **추천수(upvote)**나 **댓글수**, **본문 길이** 등의 지표를 활용해 노이즈가 많은 글을 걸러냅니다 [32] [35]. 예를 들어 댓글이 극히 적거나 내용이 너무 짧은 질문글 등은 제외하고, 키워드가 자주 언급되고 호응이 있는 토론글을 우선 저장합니다. 커뮤니티 모듈은 수집 결과로 **글 제목, 작성자 혹은 출처, 게시일, 본문 요약, URL, 추천/댓글 수 등의 메타정보**를 제공하며, 이후 평가 단계에서 다른 유형과 함께 처리됩니다. (구현노트: 커뮤니티 데이터는 비정형적이고 양이 방대할 수 있으므로, 초기에는 제한적인 서브레딧/포럼만 대상으로 삼고 점진적으로 확대합니다.)

3.2 평가 및 점수화 모듈 (evaluate.py)

기능: 모든 수집 자료에 대해 **품질 평가(quality gate)와 스코어링**을 수행합니다. 먼저 뉴스/블로그/커뮤니티 글 등에 대해 **품질 게이트(QG)** 기준을 적용하여, 지나치게 짧거나 의미 없는 콘텐츠를 자동 탈락시킵니다 [9]. 예를 들어 본문 길이가 700자 미만이거나 문장 수 5개 이하로 너무 짧은 경우, 혹은 로그인 필요/구독 유도/오류 페이지 등이 감지되면 해당 기사를 REJECT 처리합니다 [36]. 반면 키워드 뉴스 특성상 **게시 날짜가 오래되었다고 배제하지는 않으며**, 과거 글도 포함 허용합니다 [37]. 품질 게이트를 통과한 후보들에 대해서는 **팩트체크(Fact-Check) 기반 신뢰도 평가**도 진행합니다. 예컨대 본문 내에 **객관적 근거 요소**(숫자 데이터, 날짜/연도, 표준 규격 번호, 외부 하이퍼링크, PDF 첨부 등)가 많은지 탐지하여 신뢰 점수를 계산합니다 [38]. 숫자나 통계치가 있거나, 권위 있는 기관의 외부 링크가 포함된 경우 가점을 주어, 근거가 부족한 기사는 자동 HOLD 또는 제외 처리하여 전반적 신뢰성을 담보합니다 [38].

다음으로 각 기사별로 **키워드 관련성 점수**와 **종합 점수**를 산정합니다. **키워드 매칭 강도**를 정량화하여, 제목에 키워드가 직접 포함된 경우 높은 가중치를 주고 본문에 여러 번 언급되면 횟수에 따라 추가 점수를 부여합니다 [39]. 반대로 키워드 언급이 단 한번이거나 문맥상 중요하지 않으면 낮은 점수를 책정합니다. 동시에 **도메인 신뢰도 점수**를 부여하여, 편집자가 설정한 신뢰할 만한 공식 출처(예: `ipc.org`, `iconnect007.com` 등)에 대해서는 기본점수를 +5~+10 부여하고, 개인 블로그 등 불분명한 출처는 0 또는 음수 가중치를 줄 수 있습니다 [40]. 이러한 **도메인별 가중치 설정**은

`editor_rules.json`과 같은 설정 파일에 저장하여 관리하며, 필요시 업데이트 가능합니다 [40]. 또한 여러 출처에 **동일 또는 유사한 뉴스가 반복 등장**하는 경우 해당 주제가 업계에서 중요한 이슈로 판단하여 추가 보너스 점수를 줄 수도 있습니다 [41]. 예를 들어 동일 키워드로 수집된 후보들 중 제목이나 내용이 비슷한 기사가 여러 건이면 그 키워드가 화제성이 높다고 보고 가점을 주는 방식입니다.

품질 게이트 통과 여부(QG 결과), 팩트체크 점수, 키워드 매칭 점수, 도메인 점수 등을 종합하여 **최종 종합 점수(total_score)**를 계산합니다 [42] [43]. 예시 공식: `total_score = kw_score + domain_score + fc_score + bonus` 형태로 합산할 수 있습니다 [42]. 이렇게 산출된 점수와 함께 각 기사들의 상태/메타정보를 취합하여 최종 **후보 리스트 JSON**에 저장합니다. 이 JSON (`selected_keyword_articles.json`)에는 기사별로 **ID, 키워드, 유형(type), 출처, 제목, 요약문, URL, 키워드 언급 횟수, 도메인 신뢰도 점수, 종합 점수, 품질상태(QG/FC 통과 여부)** 등이 기록됩니다 [44]. 이 단계에서는 아직 어떤 기사를 발행할지 확정하지 않은 상태이므로, 자동 필터링에서 **탈락하지 않은 모든 후보**를 리스트에 포함시킵니다 [44].

(참고: 키워드 기반 수집은 일반 일일 뉴스 수집보다 **노이즈**나 **과거 자료**가 많이 섞일 가능성이 높으므로, 자동화된 품질 필터와 편집자 확인 작업이 필수적입니다 [45]. 적절한 키워드 선정과 위 단계별 필터링을 통해 최소 15건 이상의 양질의 기사를 확보하는 것을 목표로 합니다 [46]. 만약 최초 수집 결과 후보가 15건에 못 미친다면, **동의어 키워드 추가 검색**이나 **외부 뉴스 검색**을 한 번 더 수행하여 결과를 보강해야 합니다 [46].)*

3.3 요약 및 번역 모듈 (translate_summarize.py)

기능: 다국어이거나 분량이 긴 원문에 대해 **한글 요약문**을 자동 생성합니다. 키워드 뉴스는 한국어 독자를 대상으로 하므로, 영어로 된 뉴스 기사나 논문 초록은 번역하여 요약해야 합니다. 이를 위해 각 기사에서 **핵심 문장** 몇 개를 추출하고, OpenAI GPT API 등의 **자연어 요약 모델**을 활용해 한국어로 요약합니다 [11]. 예를 들어 주요 문장 2~3개를 뽑아 GPT에게 요약/번역하도록 프롬프트를 구성할 수 있습니다. 요약 결과는 한 기사당 **2~3문장 길이로 압축**하며, 원문의 중요 정보와 맥락을 담도록 합니다 [47].

만약 외부 AI API를 사용할 수 없는 환경이라면, **룰기반 요약(규칙 기반)**을 적용합니다 [47]. 예컨대 문장의 핵심 부분(주어,동사,목적어 등)을 연결하거나, 미리 정의된 용어 사전을 참조해 번역하는 식입니다. 또한 용어의 **일관성**을 위해 별도의 **용어 Glossary**를 관리하여 번역 시 참조합니다 [47]. (예: "PCBA"는 항상 "PCB 어셈블리"로 표기 등).

이 모듈은 evaluate 단계 이후에 수행되어, **평가 통과된 기사들**에 대해만 요약을 시도합니다. 생성된 한국어 요약문은 해당 기사 메타데이터에 추가되며, 편집자가 검토할 때 원문 대신 이 요약 내용을 볼 수 있도록 지원합니다. (물론 원문 URL도 제공하므로 원문 조회 가능) 요약문이 부자연스러울 경우 편집자가 UI에서 직접 수정하거나 코멘트를 달 수 있도록 하는 것이 바람직합니다.

3.4 발행 엔진 모듈 (publish_engine.py)

기능: 편집자가 **최종 승인**한 기사들만 모아 하나의 **완결된 키워드 뉴스 페이지**로 구성합니다. 발행 엔진은 내부적으로 Markdown 템플릿이나 HTML 템플릿을 사용하여 결과물을 렌더링합니다. 구현은 기존 일일 뉴스 발행 로직을 참고하여, 키워드 뉴스 전용 템플릿을 사용하는 방향으로 설계합니다 [48]. 예컨대 기존에 `--publish` 모드로 Markdown을 HTML에 끼워넣는 방식이 있다면, 키워드 뉴스의 경우 별도 `--publish-keyword` 모드를 추가하여 별도 템플릿이나 분기 로직을 사용합니다 [49] [50]. 템플릿 상에서는 **페이지 제목에 키워드 명시**, 간략 소개문을 넣을 수 있고, 본문에 기사 카드를 **유형별 섹션**으로 나누어 나열합니다 (예: " 공식 뉴스", " 학술 논문", "⚙ 표준 문서", " 커뮤니티 토론" 등으로 구분) [15]. 각 섹션 내에서는 가능하면 **연대기순(과거→최근)**으로 정렬하여 키워드 관련 사건의 흐름을 보여줍니다 [14]. 기사 카드 하나에는 **제목, 요약문, 출처명 및 날짜, 원문 링크** 등을 포함하고, 편집자 코멘트가 있는 경우 함께 표기합니다.

발행 엔진은 Markdown을 HTML로 변환하는 과정에서 만약 **요약문에 영어 문장이 남아있다면 한국어로 번역**하여 자연스럽게 출력하도록 합니다 [49]. (사전에 translate_summarize 단계에서 번역했겠지만 혹시 남은 영문 텍스트 처리용) 최종 생성된 HTML은 프로젝트 내 `archive/` 디렉터리에 저장하고, 파일명은 예를 들어 `2025-10-03_리플로`

우.html 처럼 **발행일과 키워드**를 조합하여 유일하게 짓습니다 [15] . 이렇게 하면 날짜별/키워드별 아카이브 관리가 용이하며, 파일이 누적되어도 식별이 가능합니다. 발행 직후에는 해당 파일을 웹 서버 경로에 복사하거나 정적 페이지로 호스팅하여 **독자용 프론트엔드**에서 접근할 수 있게 합니다. (예: /keyword/리플로우 경로로 라우팅하여 해당 HTML을 제공하거나, 정적 사이트 생성 방식으로 배포).

또한 버전 관리 측면에서는, 아카이브에 쌓인 HTML/Markdown을 Git 등의 VCS로 관리하거나 주기적으로 백업하여 **콘텐츠 이력 관리**를 합니다. 향후 동일 키워드로 업데이트 발행을 하거나 수정이 필요한 경우를 대비해 이전 버전을 보존해두는 것이 좋습니다.

(참고: 키워드 뉴스는 발행 이후 **독자들이 접근하는 UI**도 고려해야 합니다. 예를 들어 메인 웹 페이지에 "오늘의 키워드" 배너를 추가하고, 별도의 **키워드 뉴스 목록 페이지**를 만들어 과거 발행된 키워드 뉴스를 모아서 보여줄 수 있습니다 [51] . 이 부분은 최종 독자 프론트엔드 개발 범위에 속하며, 본 계획서에서는 관리자용 발행까지를 중점적으로 다룹니다.)

4. CLI 기반 작동 방식

개발 및 운영 편의를 위해 **CLI(Command Line Interface)** 기반으로 주요 기능을 실행할 수 있습니다. 즉, 키워드 수집부터 발행까지의 과정을 커맨드라인에서 단계별로 트리거하도록 구현합니다. 다음은 예상되는 CLI 명령어 예시와 동작 설명입니다:

- **키워드 수집**: --collect-keyword "<키워드>"
  예를 들어 $ python main.py --collect-keyword "CFX" 와 같이 실행하면, 시스템은 해당 키워드에 대한 수집 파이프라인을 한 번 동작시킵니다 [52] . 이 명령은 내부적으로 앞서 정의한 **수집 모듈들**(뉴스, 논문, 표준, 커뮤니티)을 모두 실행하여 데이터를 모으고, 품질평가 및 요약까지 수행한 후 결과를 JSON 등에 저장합니다. --collect-keyword 인자를 별도로 둠으로써 일일 정기 크롤링과 구분되는 키워드 전용 수집 모드를 명시적으로 트리거할 수 있습니다 [52] [53] .

- **승인 및 발행 준비**: 수집 완료 후, 편집자는 관리자 UI에서 기사를 선별하고 승인합니다. (CLI 환경만 가정한다면, 수동으로 JSON 파일의 approved 값을 편집하여 승인 표시를 할 수도 있습니다.) 현재 CLI로 직접 승인 작업을 하는 전용 명령은 두지 않으며, UI를 통하는 것을 원칙으로 합니다. 다만 추후 필요하면 --approve-keyword 같은 명령을 만들어 JSON 내 모든 기사를 일괄 승인하거나, 특정 ID만 승인하는 스크립트를 추가 구현할 수 있습니다. 기본 흐름에서는 **수집→UI 승인→발행**으로 이어집니다.

- **키워드 발행**: --publish-keyword "<키워드>"
  예를 들어 $ python main.py --publish-keyword "CFX" 명령을 실행하면, 이전 단계에서 **승인된 기사들만 모아** 최종 HTML/Markdown 페이지를 생성합니다. 이때 내부 로직은 해당 키워드의 selected_keyword_articles.json (또는 DB 저장 데이터)을 불러와 approved == true 인 항목만 선별하여 템플릿 렌더링을 수행합니다 [54] . 결과물은 지정된 출력 디렉토리(예: archive/)에 저장되고, 표준 출력이나 로그를 통해 성공 여부와 출력 파일 경로를 알려줍니다. --publish-keyword 옵션을 별도로 둠으로써 일반 일일 뉴스 발행(--publish)과 로직을 분리하며, UI에서도 이를 별도로 호출하기 쉽게 설계합니다 [49] [50] .

(주의: CLI 명령 실행 시 필요한 설정(예: API 키, 경로 등)은 config.json 등을 통해 미리 설정되어야 합니다. 또한 CLI 모드에서는 백엔드 서버 없이도 동작하도록 구현하여, 개발자가 로컬에서 테스트하거나 크론잡으로 자동 실행할 수 있게 합니다.)*

- **사용 예시 (로컬 실행 시나리오)**: 아래는 로컬 환경에서 키워드 "CFX"에 대한 뉴스를 수집하고 발행하는 일련의 CLI 명령어 사용 예시입니다:

```
# 1) "CFX" 키워드 관련 자료 수집 (수집 + 평가 + 요약까지 수행)
$ python main.py --collect-keyword "CFX"
... (수집 진행 로그 출력) ...
--> 결과: data/selected_keyword_articles.json 생성

# 2) 편집자가 JSON 검토 후 승인 필드 수정 또는 UI 통해 승인

# 3) 승인된 기사들만 모아 발행
$ python main.py --publish-keyword "CFX"
... (발행 진행 로그 출력) ...
--> 결과: archive/2025-10-03_CFX.html 생성
```

5. 파일/폴더 구조 설계

프로젝트의 파일 및 폴더 구조는 모듈 기능과 데이터 흐름에 맞게 **논리적으로 구성**합니다. 아래는 권장되는 디렉토리 구조와 주요 파일의 용도를 설명합니다:

- `feeds/` : 데이터 소스 정의 폴더. 하위에 출처별 JSON 파일들을 둡니다.
- `official_sources.json` – 공식 뉴스 및 기업 블로그 RSS/URL 목록 [55]. (예: `"ipc": "https://ipc.org/rss"`, `"iconnect007": "https://iconnect007.com/smt/rss"` 형식으로 RSS 피드 URL 매핑)
- `community_sources.json` – 커뮤니티/포럼 소스 목록 [56]. (예: Reddit의 경우 서브레딧 리스트와 API 타입 지정, 기타 포럼 URL 배열 등)
- `paper_sources.json` – 학술 논문 소스 목록 [57]. (예: ArXiv API URL 템플릿 포함)
- (필요시) `blog_sources.json`, `standard_sources.json` 등으로 세분화 가능. 하나의 `sources.json`에 모두 넣어 관리할 수도 있습니다.

- `data/` : 수집된 데이터와 중간 산출물이 저장되는 폴더.

- `selected_keyword_articles.json` – 최근 키워드 수집 결과의 **후보 기사 리스트** JSON 파일 [44] [58]. 수집/평가/요약 완료 후 생성되며, 관리자 UI에서 이 파일을 불러와 목록을 표시합니다. 키워드별로 파일을 나누지 않고 하나만 사용할 경우 최근 실행 결과로 계속 덮어쓰며, 필요시 날짜별 파일명 (`selected_keyword_<date>.json`)으로 관리할 수도 있습니다 [58].

- 기타: `raw_articles.json` (모든 원문 데이터를 저장한 파일, 필요시), `glossary.json` (번역 용어 사전), 로그 파일 등이 이곳에 포함될 수 있습니다. SQLite 등을 쓰게 되면 `data/` 경로에 DB 파일 (`qualijournal.db` 등)을 둘 수 있습니다.

- `output/` : 발행 준비 단계의 산출물이 위치하는 폴더. 예를 들어 **Markdown 형식의 임시 파일**이나 생성된 HTML 파일이 여기에 만들어집니다. 개발 초기에는 `output/keyword_news.md`처럼 **키워드 뉴스 임시 마크다운**을 생성해둔 후, 이를 `archive/`에 최종 복사하는 방식으로 활용할 수 있습니다. (혹은 output을 생략하고 바로 archive에 생성해도 무방합니다.)

- `archive/` : 최종 발행된 키워드 뉴스 **아카이브 저장소** 폴더입니다. 각 발행물 HTML/Markdown을 보존하며, 파일명은 `YYYY-MM-DD_<키워드>.html` 형식으로 관리합니다 [15]. 예시: `archive/2025-10-03_CFX.html`. 날짜를 접두로 하여 파일을 정렬하면 발행 순서가 한눈에 보여 유지관리가 수월합니다. 이 폴더는 웹서버를 통해 정적 호스팅하거나, 필요시 별도 저장소로 백업합니다.

- `src/` (또는 프로젝트 루트): 파이썬 소스코드 디렉토리. 주요 모듈 파일들이 존재합니다. 예를 들어:

- `source_loader.py` – 소스 JSON을 로드하여 각 출처별 크롤러에 설정을 전달하는 유틸리티 [7].
- `collect_news.py`, `collect_papers.py`, `collect_standards.py`, `collect_community.py` – 앞서 설명한 수집 모듈 구현들.
- `evaluate.py` – 품질 게이트 및 스코어링 로직 구현 [9].
- `translate_summarize.py` – 요약/번역 구현 [11].
- `publish_engine.py` – 발행 템플릿 렌더링 및 파일 저장 로직 [54].
- `admin_ui.py` (또는 Flask 앱 파일) – 관리용 웹 서버 엔트리, REST API 엔드포인트 정의 (섹션 6 참조).
- `main.py` – CLI 진입점 (인자 파싱하여 위 모듈 함수들을 호출).
- `editor_rules.json` – 편집 관련 규칙 설정 파일 [40]. 도메인 신뢰도 점수(+/-)나 기타 필터 기준을 저장해둠 (예: `"trusted_domains": {"ipc.org": 10, "randomblog.com": -5}` 등).
- `config.json` – 시스템 전역 설정 파일. API 키, 경로, 기본 옵션 등을 저장합니다 [59]. 예: OpenAI API 키, 요약문 문장길이 제한, Glossary 위치, DB 연결정보 등이 포함될 수 있음.

- `requirements.txt` – 필요한 파이썬 패키지 목록 [59] (예: requests, beautifulsoup4, feedparser, Flask 등).

- `tests/`: (선택) 모듈 단위 테스트 코드 및 샘플 데이터를 둘 수 있습니다.

이상의 구조는 개발 편의와 유지보수를 고려한 것입니다. 폴더/파일 구조는 프로젝트 진행에 따라 조정될 수 있지만, **모듈별 분리와 데이터/출력 분리 원칙**을 지켜 일관성 있게 관리합니다. 특히 구성 파일(`config.json`, `editor_rules.json` 등)은 코드와 분리하여 운영 중에도 변경이 용이하게 하고, 수집된 데이터와 발행 결과는 `data/`, `archive/` 등 별도 디렉터리에 보존하여 코드 배포와 데이터 보관을 분리합니다 [59].

6. API 및 UI 설계 개요

관리자용 UI는 편집자가 키워드 뉴스를 효율적으로 **검수 및 발행**할 수 있게 해주는 중요한 도구입니다. UI는 가급적 단순하게 구현하되 필요한 핵심 기능을 갖추어야 합니다. **Flask**나 **FastAPI** 기반으로 백엔드를 구성하고, **Vue.js**나 **React**로 프론트엔드를 구현하는 방식을 고려합니다 [12]. 아래에 UI에서 제공해야 할 주요 기능과, 이를 지원하는 **REST API 설계**를 개요합니다.

- **키워드 입력 및 수집 트리거**: 관리자 대시보드 화면에 **키워드 입력 폼**과 `"수집"` 버튼을 제공합니다. 편집자가 키워드를 입력하고 수집을 요청하면, 백엔드에 해당 키워드로 수집 작업을 시작하도록 호출합니다 [60]. 구체적으로는 `POST /api/keyword/<키워드>/collect` 엔드포인트를 설계하여 이 요청을 받으면 내부적으로 `--collect-keyword` CLI 프로세스를 시작하거나, 해당 기능을 스레드/비동기로 실행합니다 [60]. 수집 진행에는 수십 초가 소요될 수 있으므로, UI에서는 요청 후 "수집 중..." 인디케이터를 표시하고 **폴링(polling)** 또는 **WebSocket** 등을 통해 완료 여부를 확인합니다 [60]. 수집 완료 시 백엔드에서 success 응답을 보내거나, UI가 일정 지연 후에 자동으로 **기사 목록을 갱신**하도록 합니다.

- **기사 목록 열람 및 필터링**: 수집이 완료되면 UI는 해당 키워드의 후보 기사 목록을 표 형태로 표시합니다 [61]. 이 목록은 백엔드에서 `GET /api/keyword/<키워드>` 등의 요청으로 JSON 데이터를 받아 채웁니다 (혹은 첫 페이지 진입 시 서버사이드 렌더링으로 embed). 각 행에는 **기사 제목, 출처, 발행일, 점수, 요약문** 등이 보여집니다 [61]. 상단에는 필터 옵션을 제공하여 **유형별(news/paper 등), 점수순, 날짜순** 등 정렬/필터링이 가능하게 합니다 [61]. 예를 들어 "점수 50 이상만 보기", "커뮤니티 글만 보기" 등을 체크박스로 구현합니다. 이를 위해 `GET /api/keyword/<키워드>?type=news` 등의 쿼리 필터를 사용할 수 있습니다.

- **승인/보류 여부 선택**: 각 기사 행에는 **체크박스**나 토글 스위치가 있어 편집자가 해당 기사를 발행 포함 여부(approved)로 표시할 수 있습니다 [13]. 기본값은 모두 미승인(false) 상태이며, 편집자가 읽어보고 가치 있다

고 판단한 항목만 체크합니다. 또한 기사마다 **"한줄 코멘트"**를 입력할 수 있는 작은 입력란을 제공하여, 해당 기사에 대한 편집자의 코멘트를 작성할 수 있게 합니다 [62]. 이 코멘트는 최종 발행물에 편집자 노트로 표시될 수 있습니다 (선택 사항). 승인/코멘트 정보는 즉시 백엔드에 반영되도록 Ajax로 `PATCH /api/keyword/<키워드>/articles` 요청을 보냅니다 [63]. (혹은 "임시 저장" 개념으로 UI에 보유하다가 발행 시 한꺼번에 전송할 수도 있습니다.) 백엔드에서는 해당 키워드 JSON 데이터의 `approved` 필드와 `editor_note` 필드를 업데이트하고 저장합니다 [63].

- **선정 가이드 및 경고**: UI는 편집자의 판단을 돕기 위해 **선정된 기사 수를 실시간 표시**하고, 15건 이상 선택하도록 권장합니다 [64]. 만약 선택수가 15건 미만이면 발행 버튼을 비활성화하거나 경고 메시지를 띄워줍니다 [64]. 반대로 너무 많은 기사를 선택하려는 경우 (예: 30건 이상)에는 "너무 많은 기사는 독자의 집중을 떨어뜨릴 수 있으니 15~20건 내외를 권장" 등의 안내를 제공합니다 [64]. 이러한 UI 규칙은 편집 지침일 뿐 강제는 아니므로, 상한을 두지 않고 경고만 하는 수준으로 구현합니다 [64].

- **발행 실행**: 기사 선별이 끝나면 편집자는 **"발행" 버튼**을 눌러 최종 페이지 생성을 요청합니다. 이때 백엔드의 `POST /api/keyword/<키워드>/publish` 엔드포인트를 호출합니다 [65]. 백엔드는 해당 키워드에 대한 발행 프로세스를 시작하며 (`--publish-keyword` CLI를 실행하거나 내부 함수를 호출), 성공 시 HTTP 200 응답과 함께 결과 파일 경로 또는 URL을 반환합니다 [65]. UI는 발행 완료 신호를 받으면 "성공적으로 발행되었습니다" 라는 메시지와 함께, 결과 페이지를 열람할 수 있는 링크를 제공합니다. 발행 중에는 중복 클릭을 막기 위해 버튼을 비활성화하고, 에러 발생 시 이를 사용자에게 알립니다 [66].

- **관리 및 기타**: 하나의 키워드 뉴스 발행이 끝나면, 편집자는 새로운 키워드를 입력하여 다시 수집을 시작할 수 있습니다. 이때 이전 키워드의 데이터는 JSON 파일로 저장되어 남아 있으므로, UI에서 새 키워드 수집을 실행하면 기존 목록을 비우거나 새로고침합니다. 추가로 고려할 기능으로, 만약 첫 수집 결과가 너무 부족할 경우(결과가 10건 이하 등) **재수집 기능**을 제공할 수 있습니다 [67]. 예를 들어 "결과 부족 – 키워드를 변경하거나 범위를 넓혀 재수집하십시오" 안내와 함께, 키워드 수정 후 재시도를 지원하면 좋습니다 [67]. 다만 초기 버전에서는 편집자가 수동으로 키워드를 바꿔 다시 수집하는 것으로 충분합니다.

요약하면, 관리자 UI는 **키워드 입력 → 수집 → 후보 확인/편집 → 발행**의 워크플로우를 지원하는 **대시보드**입니다. REST API 설계는 이를 뒷받침하기 위해 다음과 같이 정리됩니다:

- `POST /api/keyword/<키워드>/collect` – 수집 트리거 API. 입력된 `<키워드>`에 대해 수집 파이프라인을 비동기로 실행합니다 [60]. 성공 시 202 Accepted 등을 반환하고, 완료 후 결과 JSON을 준비합니다. (추가로 `GET /api/keyword/<키워드>/status` 등으로 진행 상태를 조회하게 할 수도 있음).

- `GET /api/keyword/<키워드>` – 수집 결과 조회 API. 해당 키워드에 대한 현재 후보 기사 JSON 데이터를 반환합니다. UI 최초 진입이나 수집 완료 후 목록 새로고침 시 사용합니다.

- `PATCH /api/keyword/<키워드>/articles` – 기사 승인/코멘트 업데이트 API. 요청 바디에 {id: {approved: bool, editor_note: "..."}, ...} 형태로 다수 기사에 대한 편집자 수정을 보내면, 서버가 내부 JSON/DB를 갱신합니다 [63]. (혹은 `POST /api/keyword/<키워드>/approve`로 설계하고, 내부에서 전체 리스트를 갱신하는 방식도 가능). 단일 기사 수정은 `PATCH /api/keyword/<키워드>/articles/<id>` 형태로 세분화해도 됩니다.

- `POST /api/keyword/<키워드>/publish` – 발행 실행 API. 백엔드에서 해당 키워드의 발행 프로세스를 시작합니다 [65]. 처리 완료 후 결과 페이지 경로 또는 URL을 응답합니다. UI는 응답을 받아 사용자에게 성공 알림 및 링크 제공을 합니다.

이외에 인증이 필요하다면 `/api/login` 등 로그인 API를 둘 수 있고, 위 관리자 API들은 **인증 토큰**을 요구하도록 해야 합니다. 기본적인 설계는 위와 같으며, 구현 시 Flask의 blueprint 혹은 FastAPI의 router 등을 활용해 `/api/keyword` 하위에 엔드포인트들을 구성하면 구조화에 도움이 됩니다.

7. 데이터 포맷 – selected_keyword_articles.json 구조

키워드 뉴스의 수집 및 평가 결과는 `selected_keyword_articles.json` 파일에 저장되며, 이는 핵심 **데이터 교환 포맷**입니다. 관리자 UI와 발행 모듈은 이 JSON을 기반으로 작동하므로, 그 구조를 명확히 정의해야 합니다. 해당 JSON의 구조는 크게 **키워드 정보**와 **기사 목록 배열**로 나눌 수 있습니다 [58] :

```
{
 "keyword": "<키워드 문자열>",
 "date": "YYYY-MM-DD",
 "articles": [
  {
   "id": "<문서 고유 ID>",
   "type": "<자료 유형>",     // e.g., "news", "paper", "standard", "community"
   "title": "<기사 제목>",
   "source": "<출처 사이트명>",
   "url": "<원문 URL>",
   "published_date": "YYYY-MM-DD",  // (선택) 기사 작성일 혹은 발표일
   "summary": "<한국어 요약문>",
   "score": <종합 점수>,      // 평가 모듈이 부여한 숫자 점수
   "kw_hits": <키워드 언급 횟수>, // 본문 내 키워드 등장 빈도
   "trust_score": <신뢰도 점수>, // 팩트체크/도메인 기반 점수
   "approved": false,       // 편집자 승인 여부 (초기 false)
   "editor_note": ""        // 편집자 코멘트 (초기 빈 문자열)
  },
  { ... }, ... (이하 반복)
 ]
}
```

위 구조에서 `keyword`와 `date` 필드는 이 JSON이 담고 있는 키워드 이슈의 식별자입니다 [58] . date는 수집 실행일(혹은 발행일)을 기록해두며, 나중에 아카이브 관리나 순서 정렬에 사용됩니다. `articles`는 기사 객체의 배열로, 각 객체는 개별 기사/문서를 나타냅니다 [58] . 주요 필드 의미는 다음과 같습니다:

- `id`: 각 기사에 부여된 고유 식별자입니다. `"news_001"`, `"paper_005"` 등으로 생성할 수 있으며, 출처와 번호 조합이나 해시값 등으로 만듭니다. 이 ID는 UI에서 승인/코멘트 업데이트시 참조 키로 사용됩니다.
- `type`: 자료 유형을 나타내는 문자열 [9] . 예: `"news"` (공식뉴스/블로그), `"paper"` (학술논문), `"standard"` (표준/정책 문서), `"community"` (커뮤니티 글). 유형별로 UI에서 아이콘 표시나 섹션 구분에 활용됩니다.
- `title`: 기사/문서의 제목. 가능한 한 원문 그대로 저장하되 필요한 경우 길이를 조절합니다.
- `source`: 출처 사이트 또는 발행처 명 [44] . 예: `IPC.org`, `IEEE Xplore`, `Reddit(r/SMT)` 등. 도메인 신뢰도 평가와 UI 표시를 위해 저장합니다.

- `url`: 원문을 볼 수 있는 웹 링크 [44]. 발행 페이지에서 제목과 함께 하이퍼링크로 연결됩니다.
- `published_date`: 원 기사가 발행된 날짜 (알 수 있을 경우). 뉴스는 기사 날짜, 논문은 출판년도, 표준은 제정일 등이 될 수 있습니다. UI에서 연대기 정렬이나 표시 용도로 사용합니다.
- `summary`: 한국어 요약문 [44]. 영어 기사 등은 번역 요약된 2~3문장, 한글 기사인 경우 중요한 부분 발췌 등으로 구성됩니다. 이 필드가 최종 발행 페이지에 그대로 출력되므로 가독성을 위해 문장 처리에 유의합니다.
- `score`: 평가 모듈이 계산한 **종합 점수** [44]. 정수 또는 소수로 저장할 수 있으며, 점수가 높을수록 해당 키워드와 밀접하고 신뢰도 있는 기사임을 의미합니다. UI에서 정렬이나 필터링에 사용됩니다.
- `kw_hits`: (선택) 키워드 등장 횟수. 키워드 매칭 강도를 나타내기 위해 저장할 수 있습니다. 예: 본문에 키워드 5회 등장 등.
- `trust_score`: (선택) 팩트체크 기반 신뢰 점수. 숫자 데이터/링크 존재 여부 등으로 계산된 값입니다.
- `approved`: 편집자 승인 여부 [44]. 기본 수집 후에는 `false` 또는 `null`로 두고, UI에서 편집자가 선택한 경우 `true`로 변경합니다. 발행 엔진은 이 값이 true인 항목만 골라냅니다.
- `editor_note`: 편집자 코멘트 [44]. 편집자가 해당 기사에 대해 남긴 한 줄 의견이나 메모를 저장합니다. 기본값은 빈 문자열이며, UI에서 입력받아 업데이트합니다.
- 그 외: `meta`라는 하위 객체를 두어 원문에서 추출한 추가 메타데이터(예: 논문의 DOI나 저자, 커뮤니티 글의 댓글 수 등)를 저장할 수도 있습니다 [68]. 이 부분은 유연하게 설계합니다.

실제 `selected_keyword_articles.json`의 예시 레코드:


```
{
  "id": "news_0003",
  "type": "news",
  "title": "New SMT Assembly Technique Revolutionizes PCB Manufacturing",
  "source": "SMT Today",
  "url": "https://smttoday.example.com/news/12345",
  "published_date": "2018-07-21",
  "summary": "PCB 제조에서 새로운 SMT 조립 기술이 도입되어 생산 효율이 향상되었다는 소식...",
  "score": 87.5,
  "kw_hits": 3,
  "trust_score": 60,
  "approved": true,
  "editor_note": "초기 SMT 혁신 사례로 참고."
}
```


이처럼 `selected_keyword_articles.json`은 **키워드별 수집된 기사 리스트와 상태 정보를 모두 담은 중심 데이터**입니다. 개발 단계에서는 매 수집 시 이 파일을 덮어쓰는 방식으로 구현하고, 나중에 여러 키워드 결과를 동시에 관리하려면 키워드별 JSON을 분리하거나, DB 테이블 구조로 전환하면 됩니다 [58]. (예: 관계형 DB의 `documents` 테이블로 관리 [69]). 초기에는 JSON 파일로 시작하여 구현 용이성과 결과 확인의 편의를 취하고, 운영 안정화 후 **SQLite/PostgreSQL**로 마이그레이션하는 로드맵을 갖습니다 [6].

8. 발행 포맷 및 아카이브 관리

퀄리저널 키워드 뉴스의 **발행 포맷**은 웹에서 쉽게 볼 수 있는 **HTML 페이지** 또는 Markdown으로 생성된 후 HTML로 변환된 페이지입니다. 발행 엔진은 주어진 승인된 기사 리스트를 템플릿에 넣어 **하나의 문서(document)**를 만들게 됩니다. 이 문서에는 보통 다음과 같은 구성 요소가 포함됩니다:

- **헤더**: 키워드 명과 발행일, 간략 소개 문구. (예: "`[2025-10-03] 키워드 뉴스: 리플로우(Reflow)` – PCB 리플로우 공정의 과거와 현재 동향"). 사이트 디자인에 맞게 H1 또는 타이틀 부분을 구성합니다.

- **콘텐츠 섹션**: 기사 타입별로 구분된 섹션을 가집니다. 예컨대 "🏷️ 관련 표준 문서", "📄 관련 논문", "📰 관련 뉴스", "💬 커뮤니티 의견" 등의 제목을 달고 해당 그룹의 기사 카드를 나열합니다 [15]. (섹션 구분은 디자인에 따라 생략할 수도 있지만, 다양한 콘텐츠를 명확히 보여주기 위해 구분을 권장).

- **기사 카드**: 각 기사는 블록quote나 카드 형태로 표시됩니다. **제목**은 원문 링크로 걸고, 그 아래 **요약문**(한국어)을 2~3문장 보여줍니다. 이어서 **출처명과 발행일**을 작게 표시하고, 필요시 **편집자 코멘트**를 이탤릭체 등으로 첨언합니다. 예시:

  New SMT Assembly Technique Revolutionizes PCB Manufacturing – SMT Today, 2018-07-21
  PCB 제조에서 새로운 SMT 조립 기술이 도입되어 생산 효율이 향상되었다는 소식... (초기 SMT 혁신 사례로 참고.)

위와 같이 제목 하이퍼링크 + 출처/날짜 + 요약 + (코멘트) 순으로 구성할 수 있습니다.

- **푸터**: 기사 출처들의 약어 설명이나 기타 참고 링크. (예: "출처: SMT Today - 전자제조 전문 매체"). 또는 키워드 뉴스 아카이브 페이지 링크.

발행물은 초안 단계에서 **Markdown**으로 생성한 후 HTML 템플릿에 삽입하는 방식으로 만들 수 있습니다 [48]. 기존에 일일 뉴스를 Markdown -> HTML 변환하는 흐름이 있다면 이를 재사용합니다. Jinja2 등의 템플릿 엔진을 이용해 HTML 조각을 채우거나, Markdown 파일을 Pandoc/Markdown 파서로 HTML 변환하는 방법도 가능합니다. OpenAI API 기반 번역은 이미 요약 단계에서 거쳤다면, 발행 시에는 추가 번역은 거의 필요 없지만, 혹시 남아있는 영문이 있다면 이 때 처리를 합니다 [49]. (예: 요약문 중 영어가 남은 경우 API 호출로 번역 대체).

아카이브 관리: 발행된 HTML 파일은 `archive/` 디렉토리에 **축적**됩니다 [15]. 파일명에 날짜와 키워드를 포함하여 중복 없이 저장되며, 이를 통해 이력 관리가 가능합니다. 만약 같은 키워드로 재발행하는 경우 날짜가 다르면 별도 파일로 남고, 날짜가 같다면 (동일 날에 재발행) 덮어쓸 수 있습니다. 버전 관리를 위해 아카이브 파일들을 Git 등에 커밋하거나, 주기적으로 별도 스토리지에 백업하는 절차를 둡니다.

또한 **키워드별 뉴스 페이지 목록**도 제공해야 합니다. 예를 들어 `/archive/` 페이지나 관리자 화면에서 지금까지 발행된 키워드 (날짜순 또는 키워드 이름순)를 볼 수 있는 인덱스를 제공합니다. 이 인덱스는 아카이브 폴더를 스캔하여 만들어지거나, 발행 시 별도의 목록(JSON/DB)에 누적시켜 관리할 수 있습니다.

정리하면, 키워드 뉴스 발행 포맷은 **한 페이지에 키워드 관련 역사를 망라**하는 형태로, **Markdown 템플릿 → HTML 변환** 과정을 거쳐 완성됩니다 [48]. 최종 산출물은 파일로 저장되고, 필요에 따라 정적 웹으로 배포되거나 본 시스템의 일부로 제공됩니다.

9. 초기 테스트 및 로컬 운영 방법

개발 완료 후, 실제 키워드 뉴스를 **로컬 환경에서 테스트**하며 운영 방법을 익힙니다. 아래에 초기 테스트 시나리오와 로컬 구동 절차를 단계별로 설명합니다:

1. **환경 설정**: 코드와 함께 `config.json`, `feeds/*.json` 등의 설정 파일을 준비합니다. API 키(예: OpenAI Key)는 `config.json`에 넣고, `feeds/`에는 신뢰할 만한 출처들의 RSS/API 주소를 작성합니다. 예를 들어 `feeds/official_sources.json`에 `{"ipc": "https://www.ipc.org/news/rss", ...}` 등을 입력합니다. 또한 Python 가상환경을 구성하고 `pip install -r requirements.txt`로 필요한 패키지를 설치합니다.

2. **키워드 수집 실행**: 테스트 용도로 **키워드 하나**를 선택합니다. 가급적 관련 자료가 풍부할 만한 키워드가 좋습니다 (예: "AI 칩", "리플로우", "CFX" 등). 커맨드라인에서 해당 키워드를 수집 모드로 실행합니다.

   ```
   $ python main.py --collect-keyword "CFX"
   ```

   그러면 터미널에 각 수집 모듈이 돌아가는 로그가 출력됩니다. (예: "Loading sources... Fetching RSS from ipc.org... 10 items found... Filtered 2 items with keyword..." 등). 성공적으로 완료되면 **결과 JSON 파일**이 생성됩니다 [70]. 기본 설정대로라면 `data/selected_keyword_articles.json`가 이 역할을 하며, 내부에 수집된 기사들의 목록과 점수 등이 채워져 있을 것입니다 [70].

3. **수집 결과 검토**: 편집자는 생성된 JSON 파일을 열어 내용을 확인합니다. 우선 기사 개수를 파악하여, 목표치(약 15건 이상)에 근접하는지 봅니다. 만약 너무 적게 수집되었다면 (예: 5건 이하), 키워드 선택을 바꾸거나 `feeds/`에 소스를 추가해야 할 수도 있습니다 [67]. 또는 동의어/약어가 커버 안 되어 누락된 것일 수 있으므로, `collect_*.py` 로직의 키워드 매칭 부분을 조정해 재시도합니다. 한편 충분한 후보가 수집되었다면 각 항목의 `score`, `summary` 등의 필드를 확인하여 **자동 요약/번역 품질**을 점검합니다. 이 단계에서 간단한 통계를 출력해볼 수도 있습니다 (예: 점수 상위 5개 기사 로그 출력 등).

4. **관리자 UI 테스트**: 로컬에서 Flask/FastAPI 앱을 실행합니다. (`$ python admin_ui.py` 혹은 `$ flask run` 등으로 서버 기동). 브라우저에서 `http://localhost:5000/admin/keyword` 페이지를 열면, 기본 인터페이스와 함께 앞서 수집한 키워드의 기사 목록이 보일 것입니다 [71]. (초기에는 하드코딩된 키워드나 가장 최근 JSON을 불러오도록 구현 가능). UI에서 **승인 체크박스**를 몇 개 선택하고 코멘트도 입력해봅니다. 저장/반영이 실시간으로 되는지 (`selected_keyword_articles.json` 수정 확인 또는 네트워크 탭 확인) 테스트합니다. 사용성 테스트로서, 정렬/필터 기능이 동작하는지도 확인합니다 (예: 점수순 정렬 버튼이 있으면 눌러서 목록이 재배열되는지).

5. **발행 실행 및 결과 검증**: UI에서 "발행" 버튼을 눌러보거나, UI가 아직 미완성이면 CLI로 바로 발행을 시도합니다.

   ```
   $ python main.py --publish-keyword "CFX"
   ```

   명령 실행 후 로그에 **발행 성공 메시지**와 출력 파일 경로가 표시됩니다. (UI에서 눌렀다면 브라우저 alert이나 페이지에 결과 링크가 나타났을 것입니다.) 이제 `archive/` 폴더를 열어 생성된 HTML 파일을 찾아봅니다. 파일명을 확인하고 브라우저로 열어 **레이아웃과 콘텐츠**를 검증합니다. 기사 섹션이 의도대로 구분되어 있는지, 깨진 이미지나 링크는 없는지, 한글 요약이 잘 읽히는지 등을 체크합니다. 특히 인코딩 문제나 특수문자 렌더링 등을 살핍니다.

6. **결과 조정 및 재발행**: 만약 발행물에서 어색한 부분이 발견되면 (예: 요약문 오탈자, 번역 투 문장 등), 이를 수정하기 위해 편집자 코멘트로 보완하거나, source JSON 자체를 편집해서 직접 문구를 손볼 수도 있습니다. 편집자 코멘트는 이미 입력 가능하므로, 예컨대 요약이 부족한 경우 코멘트에 보충설명을 넣어 **발행본에 나타나도록** 합니다. 그런 다음 다시 `--publish-keyword`를 실행하면 수정사항이 반영된 새 HTML이 생성됩니다. (동일 날짜에 여러 번 발행하면, 파일명을 덮어쓸지 버전을 다르게 할지 결정해야 하는데, 초기 테스트에선 그냥 덮어써도 무방합니다.)

7. **추가 테스트 시나리오**: 다른 키워드로도 반복 테스트를 합니다. 예: "리플로우" 키워드로 수집해보기 – 이 경우 영문 기사도 많이 나올 수 있으므로 번역 품질을 특히 확인합니다. 또 학술 키워드 (예: "IPC-2581" 같은 표준 번호)를 넣어 표준 문서 모듈이 잘 작동하는지도 검토합니다. 각 시나리오별로 JSON과 HTML 산출물을 살펴보며 모듈별 개선점을 기록합니다.

8. **로컬 운영 방법**: 개발 완료 후 초기에는 로컬 PC나 사설 서버에서 수동으로 운영을 시작할 수 있습니다. 편집자는 매일 오전 특정 시간에 새로운 키워드를 정해 위 CLI 명령들을 순서대로 실행하거나, 관리자 UI를 통해 수집→승인→발행을 수행합니다. 이렇게 발행된 결과 HTML을 회사 블로그나 사내 위키 등에 게시하는 형태로 시작해볼 수 있습니다. 운영 과정에서 발견되는 크롤링 누락이나 잘못된 요약 등은 그때그때 코드의 규칙 파일(`editor_rules.json` 등)이나 Glossary를 업데이트하고, 소스를 리스타트하여 개선합니다.

9. **성능 모니터링**: 키워드 수집은 출처가 많을 경우 수십 초 이상 걸릴 수 있으므로, 테스트 단계에서 **병목 요소**를 모니터링합니다. 예: 특정 출처 응답이 느려 전체가 지연된다면 멀티스레딩 도입을 고려하고, 요약 API 호출이 오래 걸리면 비동기 호출로 전환하는 등을 실험합니다.

10. **테스트 결과 공유**: 테스트 진행 후에는 샘플 키워드에 대해 생성된 뉴스 페이지(예: CFX 키워드 결과)를 팀 내 공유하여 피드백을 받습니다. 이를 통해 유용성 평가 및 추가 요구사항(예: "연표 형태로 보여주면 좋겠다" 등)을 수렴하고 다음 개발 사이클에 반영합니다.

이상의 과정을 거쳐 로컬 환경에서 기능이 안정되면, 스테이징/프로덕션 환경으로 옮겨 실제 운영을 시작합니다. 초기 수동 운영을 통해 시스템 튜닝을 하고, 충분한 신뢰가 생기면 자동화나 일정관리(Cron)를 도입하는 식으로 확장합니다 [72].

10. 공개 클라우드 배포 시 고려사항

퀄리저널을 퍼블릭 클라우드 환경(예: AWS, GCP, Azure 등)에 배포하여 운영하려면, **CI/CD 파이프라인**, **인프라 설정**, **보안/백업** 등 추가적인 고려사항이 필요합니다:

- **CI/CD 구성**: 코드를 저장소(git)에 올리고 GitHub Actions, Jenkins 등의 CI 도구를 활용해 **자동 빌드/테스트/배포 파이프라인**을 구축합니다. 예를 들어 `main` 브랜치에 푸시될 때 자동으로 Docker 이미지를 빌드하고, 스테이징 서버에 배포하여 테스트를 거친 후 프로덕션에 배포하는 흐름을 설정합니다. 이를 통해 빈번한 업데이트와 버그 수정을 신속히 반영할 수 있습니다.

- **Containerization & 환경 설정**: 운영 환경의 일관성을 위해 **Docker** 컨테이너로 패키징하여 배포하는 것을 권장합니다. Python 환경, 필요한 패키지, cronjob 설정 등을 Dockerfile에 명시하고, 컨테이너 이미지를 클라우드 레지스트리에 저장합니다. 이 이미지에는 `config.json` 등의 기본 설정은 포함하되, **API 키나 비밀번호는 환경변수**로 주입하도록 처리합니다 (예: `OPENAI_API_KEY`를 배포시 주입). CI 파이프라인에서 민감정보를 안전하게 다루도록 Secrets 관리에 유의합니다.

- **데이터베이스 및 스토리지**: 초기엔 JSON 파일과 SQLite로 구현했지만, 클라우드 환경에서는 **관리형 DB 서비스**(예: AWS RDS의 PostgreSQL)로 전환하는 것이 좋습니다. 관계형 DB를 사용하면 동시 접근도 안전해지고, 추후 검색/분석에 SQL을 활용할 수 있습니다 [69]. 또한 대량의 기사 데이터를 효율적으로 검색하기 위해

Elasticsearch/OpenSearch 같은 검색 서버를 도입하는 것도 고려합니다 [6]. 클라우드에서는 이러한 서비스를 매니지드로 제공하므로, 필요에 따라 연계합니다. 한편 아카이브 HTML 파일 등은 **Object Storage(S3)**나 정적 호스팅에 올려 두는 방식을 검토합니다. 정적 페이지는 클라우드 CDN을 통해 캐싱/배포하면 전세계 어디서나 빠르게 제공할 수 있습니다.

- **스케일 아웃**: 사용자가 늘어나거나 하루에 여러 키워드를 발행하게 될 경우, 수집 및 API 호출량이 증가합니다. 이때를 대비해 시스템을 **수평 확장**할 수 있도록 설계합니다. 예를 들어 수집 작업을 잡(queue)에 넣고 워커 인스턴스를 여러 대 띄워 병렬 처리하거나, Kubernetes를 이용해 컨테이너를 오토스케일링하는 방법이 있습니다. 초기에는 단일 VM/컨테이너로도 감당 가능하지만, 구조적으로 분리(크롤러, 웹서버, DB 등)해두면 추후 확장이 용이합니다.

- **작업 스케줄러**: 편집자가 매일 수동으로 입력하는 프로세스 외에, **자동 추천 키워드**로 매일 정해진 시간에 수집/발행하는 기능을 추가할 수도 있습니다. 이를 위해 클라우드 환경의 **스케줄러**(예: AWS EventBridge/CloudWatch Scheduler, Unix cron 등)를 설정하여 `$ python main.py --collect-keyword <today_keyword>`를 정기 실행하도록 해볼 수 있습니다. 다만 키워드 선정은 자동화하기 어려운 영역이라서, 최소한 **자동 미수행 방지**(방화벽 설정 등)만 해두고 기본은 수동 트리거로 운용합니다.

- **보안 및 인증**: 공개 웹에 배포할 경우 **관리자 UI에 인증**을 반드시 적용해야 합니다. 간단한 방법은 HTTP Basic Auth를 걸거나, OAuth 연동 또는 자체 로그인 시스템을 구축하는 것입니다. Flask의 세션 혹은 JWT 기반으로 `/admin` 및 `/api` 엔드포인트에 관리자만 접근 가능하게 만듭니다. 또한 API 엔드포인트는 CORS 설정을 조정하여 외부에서 함부로 호출 못 하도록 하고, 중요한 액션(발행 등)은 CSRF 토큰이나 추가 확인 절차를 둘 수 있습니다. 서버 사이드에서는 **인증 토큰/세션 검증** 로직을 넣어 권한 없는 접근을 차단합니다.

- **네트워크 및 비용**: 크롤러가 외부 웹사이트를 수집할 때 발생하는 **트래픽 비용**과 API 호출 비용(OpenAI 등)을 고려합니다. 클라우드 서버의 아웃바운드 트래픽 요금 정책을 확인하고, 과도한 트래픽이 나오지 않도록 크롤링 주기/대상을 튜닝합니다. OpenAI API는 사용량에 따라 과금되므로, 한 달 예상 호출량을 계산해 예산을 확보하거나 캐싱 전략을 세웁니다 (예: 동일 기사에 대해 재번역하지 않고 결과 저장 등).

- **로그 및 모니터링**: 클라우드 환경에서는 애플리케이션 로그를 중앙 모니터링하는 것이 중요합니다. 예를 들어 AWS CloudWatch Logs나 Elastic Stack 등을 활용해 수집/발행 과정에서의 로그를 수집하고, 오류 발생 시 알람을 받을 수 있게 합니다. 또한 크롤러가 어디까지 진행되었는지, 몇 건을 수집했는지 등의 **커스텀 지표**를 수집해 대시보드를 만들면 운영에 도움이 됩니다.

- **백업 및 복구**: JSON이나 DB의 **데이터 백업 전략**을 세웁니다. 발행된 아카이브 HTML은 그 자체로도 기록이지만, 원본 JSON/DB (특히 논문 메타정보 등 귀중한 데이터베이스)는 주기적으로 스냅샷을 떠서 안전한 스토리지에 저장합니다. 클라우드의 Managed DB면 자동 백업 설정을 활용하고, JSON 파일을 계속 쓴다면 해당 파일을 주기적으로 버전 백업(S3에 일자별 업로드 등)합니다. 최악의 경우 시스템 장애 시 백업에서 복구할 수 있도록 절차를 마련합니다.

- **테스트/스테이징 환경**: 프로덕션 배포 전에 기능을 검증할 **스테이징 환경**을 클라우드에 마련합니다. 실제 운영 환경과 유사하게 설정하되, 데이터는 일부만 사용하거나 키워드도 테스트용으로 돌려봅니다. CI/CD 파이프라인에서 스테이징에 자동 배포 → 테스트 통과 → 프로덕션에 배포 식으로 단계를 만들면 안정성이 높아집니다.

이와 같은 고려사항을 바탕으로, 퀄리저널을 클라우드에 배포하면 **안정적이고 확장 가능**한 서비스로 운영할 수 있을 것입니다 [73] [69]. 처음에는 소규모로 시작하더라도, 아키텍처를 유연하게 설계하고 DevOps 모범사례를 적용하면 향후 기능 확장과 트래픽 증가에도 대응하기 쉬워집니다. 마지막으로, 클라우드 상에 민감한 API 키나 크레딧 정보가 노출되지 않도록 .env 관리와 IAM 권한 설정 등 **보안**에 각별히 신경 쓰면서 배포 작업을 수행합니다.

以上, **퀄리저널 "하루 한 키워드" 역사 큐레이션 뉴스 플랫폼** 구축을 위한 실전 개발 계획을 상세히 정리하였습니다. 이 가이드를 따라 모듈별로 차근히 구현하고 운영 방안을 준비하면, 키워드 중심의 새로운 뉴스 포맷을 성공적으로 런칭할 수 있을 것입니다 [74]. 앞으로 데이터가 축적되고 UI가 개선됨에 따라, 독자들은 더욱 풍부한 인사이트와 역사의 맥락을 얻을 수 있게 될 것으로 기대합니다 [75].

참고로, 개발 진행 중에는 예상치 못한 크롤링 이슈나 데이터 품질 문제가 나타날 수 있으므로, **유연한 대응**과 **지속적인 규칙 튜닝**이 필요합니다. 팀 내 역할을 분담하여 (수집, 요약, UI, 운영 등) 협업하고, 문제 발생 시 원인을 공유하며 해결하면 프로젝트를 효율적으로 추진할 수 있습니다 [76]. 이제 이 계획서를 토대로 구현에 착수하시기 바랍니다. **성공적인 개발을 기원합니다!**

---

[1] [26] [30] [31] [34] [37] [46] [52] [64] [67] 1003퀄리뉴스 _키워드 뉴스_ 기능 설계 및 구현 전략.pdf
file://file_0000000052d061fda5bbd6714bbf0328

[2] [3] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [19] [21] [23] [25] [27] [28] [29] [32] [47] [54] [55] [56] [57] [59] [61] [62] [68] [70] [72] [73] [74] [75] [76] 1003_3A 하루 한 키워드” 역사 큐레이션형 퀄리뉴스를 구축하기 위한 구체적 개발 가이드입니다.pdf
file://file-92Q3asQNjsGDWWy73Fkbpt

[4] [17] [18] [20] [22] [24] [33] [35] [69] 1003_2A역사 큐레이션형 키워드 뉴스 포맷을 실현하려면.pdf
file://file-495Xc4vLRe2iNSrwexdNLY

[36] [38] [39] [40] [41] [42] [43] [44] [45] [48] [49] [50] [51] [53] [58] [60] [63] [65] [66] [71] 1003퀄리뉴스 _키워드 뉴스_ 기능 설계 및 구현 전략.pdf
file://file-XoVMv2VGBLyHnFVetrmoGx