ChatGPT

퀄리저널 프로젝트 인수인계 보고서

1. 프로젝트 개요: 퀄리저널의 목표와 주요 구조

퀄리저널(QualiJournal)은 **"하루 하나의 키워드"**를 중심으로 여러 출처의 콘텐츠를 자동으로 수집·큐레이션하고, 편집자 검수를 거쳐 **뉴스 형식의 특별 호**로 발행하는 시스템입니다. 편집자가 매일 한 개의 키워드를 선정하면 시스템이 해당 키워드와 관련된 **공식 뉴스, 블로그, 학술 자료, 커뮤니티 글** 등을 폭넓게 수집한 뒤, 이 중 **15건 이상의 고품질 기사**를 엄선하여 한 데 묶은 뉴스를 발행하는 것이 목표입니다 [1]. 수집된 모든 기사와 메타데이터(제목, 요약, 출처, 날짜, 점수 등)는 우선 `selected_keyword_articles.json` 파일에 저장되고, 편집자가 승인 표시한 기사들만 별도로 `selected_articles.json` 파일에 모아 최종 발행용 콘텐츠로 활용됩니다 [2].

이 시스템은 키워드 중심으로 **자동 수집 → 품질 평가(QG/FC) → 편집자 승인 → 발행**의 워크플로우를 갖추고 있습니다. 수집 단계에서는 다양한 유형별 크롤러가 설정된 소스들로부터 기사를 모으고, 전처리 단계에서 **품질 게이트(QG)**와 **팩트체크(FC)**로 신뢰도가 낮은 자료를 걸러냅니다. 평가 단계에서는 `editor_rules.json`에 정의된 **도메인 신뢰도 가중치, 키워드 적합도, 최신성, 이슈 중요도** 등을 반영하여 각 기사의 **가치 점수**를 계산합니다 [3]. 중요한 영문 기사는 핵심 문장을 추출한 후 **한국어 요약과 번역**을 생성해 제공하며(Glossary 적용으로 용어 일관성 유지) [4], 편집자는 이를 참고해 기사 선정 및 한 줄 코멘트 작성 등의 편집 작업을 합니다. **관리자 UI**를 통해 편집자는 후보 기사들을 검토하면서 `approved` 여부와 `editor_note`(편집자 한마디)을 입력하고, 발행 명령을 내려 최종 **HTML/Markdown 뉴스 페이지**를 생성합니다 [5]. 발행된 뉴스는 날짜와 키워드별로 **아카이브(archive)** 폴더에 저장되어 기록되며, 독자에게는 웹으로 제공됩니다.

퀄리저널의 핵심 목표는 "전 세계 표준·전자제조·AI 정보를 매일 자동 수집하고, 한국어 요약과 편집자 코멘트를 붙여 안정적으로 발행하는 것"입니다 [6]. 이를 통해 특정 기술 키워드와 관련된 역동적인 정보들을 **한 눈에 조망**할 수 있는 전문 큐레이션 뉴스를 제공하고자 합니다. 또한 최소 15개의 기사가 승인되어야 발행하도록 하는 **품질 게이트** 정책으로 뉴스 품질을 유지하고 있으며, 이 기준은 설정 파일의 `require_editor_approval` 옵션으로 강제됩니다 [7] [8]. 만약 승인된 기사 수가 부족하면 응급 방편으로 일시적으로 해당 옵션을 꺼서 발행한 후 다시 켜는 방법도 있지만, 기본적으로는 15건 이상을 확보한 후 발행하는 것을 원칙으로 합니다 [9] [10].

2. 시스템 아키텍처 및 파일 구조: 핵심 모듈, 디렉터리 설명

퀄리저널 시스템은 **Python** 기반 백엔드와 JSON 데이터 파일을 중심으로 동작하며, 일련의 모듈화된 컴포넌트로 구성되어 있습니다. 프로젝트의 주요 폴더 및 파일 구조와 그 역할은 다음과 같습니다:

- **feeds/** 폴더 – 다양한 **데이터 소스 정의 JSON 파일**들이 위치합니다. 예를 들어 `official_sources.json`(공식 뉴스/블로그 소스 목록), `community_sources.json`(커뮤니티/포럼 소스 목록), `paper_sources.json`(학술 논문 API 소스 목록) 등이 있으며, 각 파일에 소스의 URL, API 엔드포인트, 유형 등이 정의되어 있습니다 [11] [12]. 소스 파일 형식 예시: `{ "official": { "ipc": "https://ipc.org/rss", "iconnect007": "https://iconnect007.com/smt/rss" }, "community": { "reddit": { "subs": [...], "api": "reddit" }, "forums": ["https://www.eevblog.com/forum/"] }, "papers": { "arxiv": "https://export.arxiv.org/api/query?search_query={keyword}" } }` 형태로, 도메인별 RSS 피드 URL이나 API 정보가 담겨 있습니다 [13] [14]. 새로운 소스를 추가하려면 이 JSON 파일들을 편집하여 URL이나 쿼리를 등록하면 됩니다.

- **data/** 또는 **work** 디렉터리 – (만약 존재한다면) 일일 수집 결과와 중간 산출물이 저장되는 경로입니다. 현재 구현에서는 수집 결과가 주로 프로젝트 루트의 JSON 파일들로 관리되지만, 경우에 따라 경로를 `config.json`에서 설정할 수도 있습니다 [15]. 핵심 작업 파일은 아래 두 개입니다:

- `selected_keyword_articles.json` – **작업본**: 해당일 키워드로 수집된 **모든 기사 후보 목록**을 담은 JSON 파일입니다. 각 기사마다 제목(title), 요약(summary), 출처(source), 날짜(date), 점수(score), 승인 여부(approved), 편집자 코멘트(editor_note) 등이 포함됩니다 [16]. 수집과 평가 단계를 거친 뒤 이 파일이 생성되며, 편집자가 이 파일을 보고 발행할 기사(`approved=True`)를 선택하게 됩니다.

- `selected_articles.json` – **발행본**: 작업본 중 `approved=True`**인 기사만 모은 최종 발행 리스트**입니다. 발행 엔진은 이 파일을 참조하여 Markdown/HTML 뉴스 페이지를 생성합니다 [17]. 이 파일에는 주로 선택된 기사들의 목록과 메타데이터가 포함되며, 발행 직전 단계에 동기화됩니다.

- **archive/** 폴더 – 발행 완료된 뉴스의 **아카이브 저장소**입니다. 발행 시 생성된 Markdown, HTML, JSON 산출물이 `archive/YYYY-MM-DD_KEYWORD.md`, `YYYY-MM-DD_KEYWORD.html`, `YYYY-MM-DD_KEYWORD.json` 형식으로 보존됩니다 [18]. 동일한 콘텐츠를 Markdown·HTML·JSON 세 가지 형태로 모두 저장하므로, 이후 과거 발행물을 열람하거나 복구할 수 있습니다 [19]. 날짜별 폴더 구조가 아니라 파일명에 날짜가 포함되는 방식이며(하루 한 번 발행을 가정), 만약 하루에 동일 키워드로 여러 번 발행하는 예외 상황이 생긴다면 타임스탬프를 추가하는 등의 규칙이 필요합니다 [20].

- **tools/** 폴더 – 파이프라인 지원을 위한 다양한 **보조 Python 스크립트**들이 모여있는 디렉터리입니다. 수집된 원본을 재구성하거나, 기사 부족 시 보충하고, 승인 처리 및 포맷 교정, 최종 발행본 동기화 등을 수행하는 스크립트들이 포함되어 있습니다 [21]. 주요 스크립트와 기능은 다음과 같습니다:

- `rebuild_kw_from_today.py` – 당일 수집된 여러 원본 JSON들을 하나의 표준 구조(딕셔너리 + articles 배열)로 병합하여 `selected_keyword_articles.json` **작업본을 생성**합니다 [22] [23].
- `augment_from_official_pool.py` – 작업본의 기사 수나 승인된 기사 수가 부족할 경우, 미발행 상태로 보관된 **공식 기사 풀**에서 추가 기사들을 채워 총 기사량을 약 **20개 내외**로 보정하고, 최소 15개 이상이 승인되도록 돕습니다 [24].
- `force_approve_top20.py` – 기사 점수순으로 상위 20건을 자동으로 `approved=True`로 설정하여 **일괄 승인처리**를 지원합니다 [25]. (긴급 시 사용; 현재는 편집자가 수동 승인하는 것이 원칙입니다.)
- `sync_selected_for_publish.py` – 작업본 (`selected_keyword_articles.json`)에서 `approved=True`인 기사만 추출하여 `selected_articles.json`을 갱신하며, 필요한 경우 기존 기사 메타데이터를 병합합니다 [26]. 즉 **최종 발행본을 동기화**하는 스크립트입니다.

- `repair_selection_files.py` – `selected_articles.json` 등의 구조가 혹시 어긋났을 경우 표준 딕셔너리 구조(`{"date": ..., "articles": [... ]}`)로 교정하여 **데이터 포맷 오류를 예방**합니다 [27].

- **주요 모듈 코드** – 퀄리저널 기능은 여러 Python 모듈로 나뉘어 구현되어 있습니다. 예를 들어:

- `source_loader.py` – 위 **소스 JSON 파일들(feeds/)**`을 로드하여 소스 URL과 메타정보를 제공하는 모듈입니다 [11].
- `collect_news.py` / `collect_community.py` / `collect_papers.py` 등 – **수집기(Collector)**: 각각 공식 뉴스/블로그, 커뮤니티, 학술 논문 등을 수집합니다. RSS 피드 파싱이나 HTML 크롤링, API 호출 등을 통해 해당 키워드 관련 최신 콘텐츠 목록을 가져옵니다 [28] [29]. (예: `collect_news.py`는 `feeds/official_sources.json`의 소스들을 훑어 각 기사 링크와 본문을 수집하고, 키워드가 포함된 기사만 후보로 저장합니다 [30].)

- `evaluate.py` – **평가기(Evaluator)**: 수집된 기사에 대해 `editor_rules.json`에 정의된 규칙을 적용해 **종합 점수(total_score)**를 계산하고, 섹션(공식뉴스, 커뮤니티 등)이나 순위를 결정합니다 [15] [31]. 예를 들어 도메인 신뢰도 가중치, 키워드 매칭 빈도, 최신성, 댓글/추천 수 등을 고려합니다.
- `translate_summarize.py` – **요약·번역 모듈**: 중요 문장을 추출하고 GPT-4 API나 번역 API(DeepL, Google 등)를 활용해 **한국어 요약**을 생성합니다 [32]. 예산 절약을 위해 수집 단계에서는 규칙 기반 요약으로 대체하고, 발행 직전에만 API 호출을 수행할 수 있습니다 [33].
- `publish_engine.py` – **발행 엔진(Publisher)**: 승인된 기사들만 모아 섹션별로 정렬하고 템플릿을 적용하여 최종 **HTML/Markdown/JSON 뉴스 페이지**를 생성합니다 [31]. 생성된 결과물은 앞서 언급한 archive 폴더에 저장되고, 필요시 독자 웹 페이지에 반영됩니다.

- `orchestrator.py` – **오케스트레이션 모듈**: 전체 수집~발행 파이프라인을 CLI 명령으로 제어할 수 있는 진입점입니다. 이 모듈은 여러 단계(collect, rebuild, publish 등)를 하나로 묶어 스크립트 실행을 돕고, FastAPI 기반 **관리자 API/웹서버(app)**도 함께 제공하는 역할을 합니다. `run_quali_today.ps1` 등의 자동화 스크립트가 내부적으로 `orchestrator.py`를 호출하여 수집이나 발행을 트리거합니다 [23] [34]. (예: `orchestrator.py --collect --collect-community --use-external-rss` 등 인자를 통해 단계별 실행, 또는 `--publish-keyword`로 발행 수행).

- **관리자 UI 파일** – 퀄리저널 관리자용 프론트엔드는 **Black UI Lite** 테마를 활용한 경량 웹 페이지로 구현되었습니다. 프로젝트 내에 `index.html` (혹은 `admin/index.html`) 정적 파일로 존재하며, HTML/JS/CSS로 구성된 **대시보드 인터페이스**입니다. 해당 UI는 FastAPI의 정적 파일 서비스나 템플릿 렌더링을 통해 제공되며, 브라우저에서 열었을 때 편집 인터페이스가 나타납니다. UI 자산(예: CSS, JS)은 `static/` 혹은 `assets/` 디렉터리에 함께 포함되어 있을 수 있습니다. (자세한 UI 내용은 아래 **섹션 5**에서 설명합니다.)

3. 일일 발행 파이프라인 요약: 수집→재구성→보충→승인→발행

퀄리저널의 **일일 발행 파이프라인**은 다단계로 구성되어 있으며, Windows 환경에서는 PowerShell 스크립트(`run_quali_today.ps1`)로 이 전 과정을 자동 실행할 수 있습니다 [35] [36]. 아래는 하루 한 키워드 뉴스 발행을 위한 **수집(Collect) → 재구성(Rebuild) → 보충(Augment) → 승인 → 동기화(Sync) → 발행(Publish)**까지의 단계를 요약한 것입니다:

1. **수집 (Collect)** – 선택된 키워드에 대해 **공식 소스 기사와 커뮤니티 글 등을 크롤링**합니다. 예를 들어, RSS 피드가 있는 공식 뉴스 소스는 RSS를 파싱하고, RSS 없는 경우 HTML 목록 페이지에서 링크를 추출합니다. 커뮤니티의 경우 Reddit API를 통해 서브레딧 글을 검색하거나 포럼 페이지를 파싱합니다. `orchestrator.py` 모듈이 이 작업을 조율하며, 내부적으로 `--collect`, `--collect-community`, `--collect-keyword`, `--use-external-rss` 옵션 등을 순차 호출하여 모든 유형의 출처를 수집합니다 [23]. 필요에 따라 config에서 켜둔 외부 RSS(`external_rss.enabled`)도 추가로 수집하며, **동의어(keyword_synonyms)**가 정의된 경우 해당 키워드 변형까지 검색해 놓치지 않도록 합니다 [37]. 수집 결과는 임시 JSON들로 저장되는데(예: `news_{date}.json`, `community_{date}.json` 등), 이는 다음 단계에서 병합됩니다.

2. **재구성 (Rebuild)** – 방금 수집된 여러 원본 JSON들을 **하나의 작업본**으로 합칩니다. `tools/rebuild_kw_from_today.py` 스크립트가 이 역할을 수행하며, 수집 결과들을 표준화된 구조로 병합하여 최종 `selected_keyword_articles.json` 파일을 생성합니다 [38]. 이 파일에는 해당 키워드의 모든 후보 기사 목록이 담겨 있으며, 이후 편집 및 발행 과정의 기반이 됩니다. 재구성 과정에서는 기사별로 일관된 필드 구조를 갖추도록 정리하고, 중복 기사 제거나 잘못된 형식 교정도 수행합니다.

3. **보충 (Augment)** – 자동 수집 결과만으로 **충분한 기사 수가 확보되지 않은 경우** 보조 단계를 진행합니다. `selected_keyword_articles.json` 내 **승인 예정 기사 수가 15개 미만**으로 예상되면, `tools/augment_from_official_pool.py` 스크립트를 실행해 미발행 상태로 누적해둔 **공식 기사 풀**에서 관련

기사들을 추가로 가져와 보충합니다 [39]. 이를 통해 기사 총량을 약 20개 수준으로 늘리고, 편집자가 최소 15개 이상을 선택할 수 있도록 후보를 넉넉히 확보합니다 [24]. 예를 들어 이전에 발행하지 않았던 고득점 기사나 백업용 뉴스를 풀에서 골라 채워넣는 방식입니다. (참고: 기사 풀은 과거에 수집되었으나 발행되지 않은 기사들을 모아둔 저장소로, 초기에는 수동 관리가 필요할 수 있습니다.)

4. **편집자 승인 및 코멘트 입력 (Approval)** – 재구성/보충 단계까지 완료된 후, 편집자가 **관리자 UI** 또는 JSON 파일을 통해 기사를 선별하는 단계입니다. 편집자는 `selected_keyword_articles.json`을 열어 각 기사별로 `approved: true`로 표시할지 여부를 결정하고, 간략한 **편집자 한마디(**`editor_note`**)**를 작성합니다 [40]. 관리자 UI 상에서는 기사 리스트를 보며 체크박스로 선택 및 한 줄 코멘트를 입력할 수 있고, JSON을 직접 편집할 때는 해당 필드를 수동으로 채웁니다. 이 과정을 통해 최종 발행될 기사들이 결정되며, **최소 15개의 기사에** `approved=true`가 되도록 선정합니다. (일괄 선택이 필요할 경우 앞서 언급한 `force_approve_top20.py`로 임시 처리 가능하지만, 최종 검수는 사람이 수행합니다.) 편집이 끝나면 UI에서 **저장**하여 JSON에 반영하거나, JSON 파일을 디스크에 저장합니다.

5. **승인 수 점검 (Check)** – 승인 단계 후, 발행 조건을 충족했는지 간단히 검증합니다. 예컨대 PowerShell 스크립트 내에서 Python 원라이너를 사용하여 `selected_keyword_articles.json`을 불러 `approved=True`인 기사 개수를 집계하고, **15개 이상인지 확인**합니다 [41]. 만약 기준에 못 미칠 경우 로그에 경고를 출력하거나 발행 절차를 중단합니다. (운영 중이라면 이 시점에 담당자에게 경고 알림을 보내 후속 조치를 취하도록 할 수 있습니다.)

6. **동기화 (Sync)** – 편집자 승인 결과를 **발행본 JSON에 반영**합니다. `tools/repair_selection_files.py`를 먼저 실행하여 혹시 모를 JSON 구조상의 오류를 바로잡은 뒤 [42], `tools/sync_selected_for_publish.py`를 실행합니다. 이 스크립트는 작업본에서 `approved=True` 기사만 추출해 `selected_articles.json`을 업데이트하고, 기사 메타데이터 구조를 발행용으로 최종 정돈합니다 [42]. 그 결과, `selected_articles.json`에는 선택된 기사들의 리스트만 남게 되며 이후 발행 엔진이 이 파일을 사용합니다. (참고: 이 단계는 UI를 통해 바로 발행하는 경우 백엔드에서 자동 처리되므로 사용자는 신경 쓰지 않아도 됩니다.)

7. **발행 (Publish)** – 최종 단계로, **뉴스 페이지 산출물을 생성 및 공개**합니다. PowerShell 스크립트에서는 `orchestrator.py --publish-keyword "<오늘의 키워드>"` 명령을 호출하여 발행을 수행하며, Python 모듈 내부에서 `publish_engine.py`가 `selected_articles.json`을 읽어들여 Markdown/HTML 페이지를 생성합니다 [43]. 이때 뉴스 제목, 섹션 별 기사 목록, 편집자 코멘트 등이 템플릿에 채워지며, 완성된 Markdown과 HTML 파일이 **archive/** 폴더에 저장되고, JSON 형태의 메타데이터도 함께 기록됩니다 [44] [45]. 발행 직후 결과물을 웹에 배포하거나 메일/SNS로 전송하는 후속 작업은 운영 정책에 따라 수행할 수 있습니다.

    - **품질 게이트 적용**: 발행 명령은 내부적으로 **품질게이트**를 확인하여, 승인된 기사가 15개 미만이면 오류를 발생시키거나 UI의 발행 버튼을 비활성화시켜 **발행을 차단**합니다 [46]. 그러나 긴급히 발행이 필요한 경우(`--force` 옵션에 준하는 상황), 설정 파일(`config.json`)의 `features.require_editor_approval` 값을 일시적으로 `False`로 바꾸어 강행 발행할 수 있습니다 [9] [10]. 이 경우 발행 직후 해당 설정을 즉시 True로 복구하여 다음 발행부터는 정상적으로 조건을 적용합니다. 기본적으로는 시스템이 품질 조건을 충족하지 않으면 발행을 완료하지 않도록 설계되어 있습니다.

모든 단계가 완료되면, 하나의 키워드에 대한 **뉴스 레터 형식의 콘텐츠**가 준비됩니다. 전체 프로세스는 PowerShell 자동화 스크립트(`run_quali_today.ps1`)로 순차 실행 가능하며, 이 스크립트를 통해 **"수집→재구성→보충→(편집)→동기화→발행"**의 일련의 단계가 한 번에 수행됩니다 [47] [48]. (자동화 스크립트는 Windows 작업 스케줄러에 등록해 매일 정해진 시간에 실행할 수 있습니다. 이 부분은 **섹션 7**의 실행 팁에서 추가로 설명합니다.)

4. 주요 파일 및 설정 설명: config.json, editor_rules.json, 소스 JSON, 키워드 JSON 등

퀄리저널 프로젝트에서 동작을 제어하는 중요한 **설정 파일**들과 **데이터 정의 파일**들을 정리합니다. 이러한 JSON 설정들은 시스템의 크롤링/평가 동작과 품질 관리 정책을 세부 조정하는 데 사용됩니다:

- `config.json` – 시스템 전반 설정을 담은 파일입니다. 품질 필터 기준, 기능 토글, 파라미터 등이 정의되어 있습니다. 예를 들어:
- `community.filters` 섹션: **커뮤니티 출처 필터링 규칙**을 정의합니다. 키워드 포함 필수 여부 (`require_keyword`: true), 키워드 최소 토큰 수(`kw_min_tokens`), 특정 키워드 정규식 패턴 (`kw_regex`), 허용 도메인 리스트(`allow_domains`), 최소 제목 길이(`min_title_len`), 차단할 제목 패턴(`block_title_patterns`: 예, 채용공고/광고성 키워드) 등을 지정해 Reddit나 포럼 등의 **노이즈를 걸러내는 기준**을 설정합니다 [49]. 이를 통해 불필요한 글(예: 스팸, 광고글, 짧은 질문글 등)을 걸러내고 의미 있는 커뮤니티 글만 남기도록 합니다.
- `external_rss` 섹션: 외부 RSS 피드 사용 여부 (`enabled`)와 최대 수집 개수(`max_total`) 등을 설정합니다. 이 옵션을 켜면 공식 소스 외에 추가로 등록된 외부 키워드 RSS (예: 구글뉴스 키워드 피드 등)를 가져와 보충합니다.
- `features` 섹션: 특수 기능 토글 및 상수들을 모아둔 부분입니다. 예를 들어 **신뢰할 도메인 목록** (`trusted_domains`)이나 **편집자 승인 필수 여부**(`require_editor_approval`) 등이 있습니다. `require_editor_approval` 값은 발행 시 편집자 승인을 강제할지를 제어하며, 기본 True로 설정되어 품질 게이트를 적용합니다 [50]. False로 두면 15개 미만이어도 발행되지만, 일반적으로는 True로 유지해야 합니다.

- 그 외에도 뉴스 섹션 순서, 로그/백업 경로, 요약/번역 옵션, 스코어 임계값 등의 설정이 들어갈 수 있습니다 [15]. **config.json은 전체 시스템의 동작 정책을 관리**하므로, 운영 중 필요에 따라 필터 임계값이나 소스 사용 여부 등을 이 파일에서 조정합니다.

- `editor_rules.json` – **평가/편집 규칙 설정 파일**입니다. 기사 **스코어링 규칙**과 **AI 편집 가이드** 등이 정의되어 있습니다. 주요 내용으로:

- 도메인별 기본 점수, 키워드 매칭 가중치, 이슈 중요도 등 **기사 가치 평가 기준**이 포함됩니다 [3]. 예를 들어 ipc.org와 같이 신뢰도 높은 도메인의 기사는 기본 점수를 높게 주고, 제목에 키워드가 직접 포함된 경우 추가 가점을 부여하며, 최신 기사일수록 가산점, 오래된 정보는 감점하는 식의 규칙이 JSON으로 표현되어 있습니다.
- 편집장 AI 지원 기능을 위한 **코멘트 프롬프트/스타일**도 정의될 수 있습니다. (예: "이 기사의 의의를 한 문장으로 논평해주세요" 같은 프롬프트 템플릿과 글 길이, 톤 등의 가이드) [51]. 향후 GPT를 활용한 자동 코멘트 생성 시 이 설정을 참고합니다.

- 종합하면, `editor_rules.json`은 각 기사 카드의 점수를 계산하고 등급을 결정하기 위한 **가중치와 규칙표** 역할을 하며, 편집 보조를 위한 몇몇 설정도 포함합니다. 이 파일의 값을 조정함으로써 특정 출처를 우선하거나, 특정 키워드 관련 기사를 더 높이 평가하는 등 **큐레이션 결과를 세밀하게 튜닝**할 수 있습니다 [15].

- **소스 정의 JSON** – 앞서 **feeds/** 폴더에서 언급한 각종 소스 리스트 파일들입니다:

- `official_sources.json` – **공식 뉴스/블로그 소스 목록**입니다. 산업 뉴스 사이트, 기술 블로그 등의 RSS 피드 URL 또는 크롤링 URL이 키-값 쌍으로 정의되어 있습니다 [52]. 예) `"ipc": "https://ipc.org/rss"`, `"iconnect007": "https://iconnect007.com/smt/rss"`. 필요에 따라 신뢰도(high/medium/low) 같은 부가 정보도 함께 적어둘 수 있습니다. 이 파일은 **정기적으로 관리**해야 하는데, 예를 들어 피드 URL이 변경되거나 404 오류가 나는지 점검하여 업데이트합니다 [53].

- `community_sources.json` – **커뮤니티/포럼 소스 목록**입니다. Reddit의 특정 서브레딧이나 기술 포럼 URL 등이 포함됩니다. 예) `"reddit": { "subs": ["SMT", "ElectronicsManufacturing"], "api": "reddit" }`, `"forums": ["https://www.eevblog.com/forum/"]` 형식 [54]. 또한 이 파일에는 **커뮤니티 전용 필터 설정**이 들어있을 수 있는데, 예를 들어 최소 추천수, 최소 댓글수 등의 임계값을 정의해 인기 없는 글은 제외하도록 합니다 [49]. 너무 엄격하면 커뮤니티 글이 0건이 될 수 있으므로 해당 임계값을 상황에 맞게 튜닝하는 것이 중요합니다 [55].
- `paper_sources.json` – **학술 논문 소스 목록**입니다. ArXiv, IEEE Xplore 등의 API URL이나 검색 쿼리가 들어갑니다 [56]. 예) `"arxiv": "https://export.arxiv.org/api/query?search_query={keyword}"`. 키워드 치환을 통해 관련 논문 메타데이터(제목, 저자, 초록)를 수집합니다.

- 이 밖에도 도메인별로 세분화된 소스 리스트가 있을 수 있습니다 (예: `supply_sources.json`, `smt_sources.json` 등). 실제 개발 초기에는 SMT 전자제조 분야, 공급망/부품 분야 등으로 구분해서 소스를 관리하려 한 흔적이 있으며 [57] [58], 현재는 대부분 `official_sources.json`에 통합하거나 활용도가 낮은 파일은 미사용 상태일 수 있습니다. 전반적으로 **source JSON 파일들은 새로운 출처 추가/제거 시 수정**하는 곳으로, 운영 중 소스 변경이 필요하면 이 파일들을 업데이트한 후 수집기를 재실행하면 됩니다.

- `keyword_synonyms.json` – **키워드 동의어 목록 파일**입니다. 동일한 개념이지만 다른 표기법을 갖는 키워드들을 미리 정의하여, 키워드 뉴스 수집 시 **검색 범위를 확장**하는 역할을 합니다. 예를 들어 `"IPC-A-610"` 표준은 "IPC 610"처럼 공백이나 하이픈 유무에 따라 다르게 쓰일 수 있는데, 이러한 변형을 모두 인지하도록 `{"IPC-A-610": ["IPC 610", "IPC610"]}`과 같이 동의어를 맵핑해 둡니다 [37]. 수집기는 키워드 입력 시 이 파일을 참고하여 **동의어까지 함께 검색**하므로, 관련 기사를 놓칠 확률을 줄여줍니다. 새로운 키워드 운영을 시작할 때 이 파일에 주요 변형/약어를 추가해두면 유용합니다.

- **그 외 파일들**:

- `requirements.txt` – 프로젝트의 Python 패키지 **의존성 목록**입니다. (예: `feedparser`, `requests`, `fastapi`, `uvicorn`, `beautifulsoup4`, `praw` (Reddit API) 등등 필요한 라이브러리 명시) 개발 환경을 세팅할 때 이 파일 기반으로 패키지를 설치합니다 [59].
- `run_quali_today.ps1` – Windows 환경에서 일일 발행 파이프라인을 자동 실행하는 **PowerShell 스크립트**입니다. 내부에서 Python 명령어를 순서대로 호출하며, `${키워드}` 변수나 인자를 받아 특정 키워드 수집을 트리거할 수 있게 작성되었을 것입니다 [35] [47]. 예시로, `--collect`, `--augment` 등의 플래그를 차례로 실행하고 마지막에 `--publish-keyword`를 호출하는 논리로 구성됩니다 [60] [61].
- 로그/백업 파일 – config에서 지정한 경로에 생성되는 **로그 파일**이나, 발행물 외에 별도로 백업된 원본 JSON 등이 있다면 그에 해당합니다. 예를 들어 오류 발생 시 `error.log`, 혹은 archive 외에 별도 `backup/` 폴더에 날짜별 원본 JSON 복사본을 저장하는 등의 정책이 고려될 수 있습니다 [20].

요약하면, **config.json**과 **editor_rules.json**는 시스템 동작의 **설정 중심**이고, **여러 소스 JSON** 파일은 크롤러의 **데이터 소스**를 정의하며, **selected*.json**과 **archive**는 **데이터 흐름의 산출물**입니다. 이들 파일을 이해하고 적절히 조정하면 수집 범위나 품질 기준을 원하는 대로 변경할 수 있고, 새로운 키워드나 출처 추가도 손쉽게 할 수 있습니다.

5. 관리자 UI (Black UI Lite) 설명: 구성 요소, API 경로, 사용 방법, 파일 경로

퀄리저널에는 편집 workflow를 돕기 위한 **관리자 웹 UI (Black UI Lite)**가 제공됩니다. 이 UI는 편집자(관리자)가 **브라우저를 통해 기사 목록을 검토하고 승인·발행 작업**을 직관적으로 수행할 수 있게 해주는 도구입니다. 주요 특징과 구성 요소는 다음과 같습니다:

- **전체 구조**: 관리 화면은 **단일 페이지 어플리케이션**에 가깝게 동작합니다. FastAPI 백엔드가 정적 경로로 서빙하는 `index.html` 파일을 로드하면, 브라우저 상에 관리자 대시보드가 나타납니다. UI는 **검은색 계열 테마**로 디자인되어 있고, 주요 섹션으로 상단 헤더/메뉴, 기사 리스트 영역, 조작 버튼 영역으로 구성됩니다. (Black UI

Lite는 Bootstrap 기반의 어드민 템플릿을 가볍게 활용한 것으로, 불필요한 요소를 줄이고 핵심 기능에 집중한 UI입니다.)

- **키워드 뉴스 관리 탭**: UI에는 "키워드 뉴스" 전용 화면이 존재합니다. 편집자는 좌측 메뉴나 상단 탭에서 해당 항목을 눌러 이 화면으로 전환할 수 있습니다. 이 탭을 열면 백엔드에서 최근 수집된 `selected_keyword_articles.json` 데이터를 불러와 **기사 후보 리스트**를 표 형태로 표시합니다 [62] [63]. 표에는 각 기사의 **제목**, **출처**, **날짜**, **점수**(또는 상태)가 열로 나열되며, 긴 경우 카드(card) 형태로 변형되어 요약문도 보여줄 수 있습니다. 화면 진입 시 서버의 `/admin/keyword` 경로(API 또는 템플릿 렌더)를 통해 이 JSON 데이터를 전달받습니다 [63].

- **기사 리스트 및 조작**: 기사 리스트의 각 행(row)에는 편집 작업을 위한 입력 컨트롤이 배치되어 있습니다. **체크박스(또는 토글)** – 해당 기사를 발행에 **선택할지 여부**를 표시합니다. 체크하면 그 기사의 `selected`(=approved) 플래그가 True로 설정됩니다 [64]. **한줄 코멘트 입력란** – 각 기사에 대한 편집자 코멘트를 작성할 수 있는 작은 텍스트 필드가 행당 하나씩 있습니다 [64]. 여기에 입력한 내용은 `editor_note` 필드로 JSON에 저장되며, 발행될 뉴스 페이지에서 해당 기사 카드 아래 편집자 한마디로 표기됩니다. UI는 입력과 동시에 (또는 저장 버튼 클릭 시) Ajax를 통해 백엔드 API에 변경 내용을 전송(PATCH)하거나, 임시 상태로 가지고 있다가 나중에 한꺼번에 저장할 수도 있습니다 [65]. 현재 구현에서는 편집자가 코멘트를 입력하면 바로 JSON 파일을 갱신하도록 하여 **실시간 저장**을 지원합니다 (이를 위해 FastAPI에 `/admin/save_note` 등의 엔드포인트가 있을 수 있습니다).

- **키워드 입력 및 수집 트리거**: 화면 상단에는 당일 키워드를 표시하며, 필요 시 새로운 키워드를 입력받아 **수집을 시작**할 수 있는 입력폼과 버튼이 제공됩니다. 예를 들어 키워드 입력란과 옆에 "**수집**" 버튼이 있어, 편집자가 키워드를 입력하고 버튼을 누르면 **AJAX로 백엔드 API** (`/admin/keyword_collect?kw=<키워드>`)를 호출합니다 [62]. 서버 측에서는 해당 키워드에 대해 수집기(collector)를 **서브프로세스 또는 비동기로 실행**하며, RSS/크롤링을 진행합니다 [62]. 이 동안 UI에서는 "수집 중..."과 같은 로딩 표시를 하고, 중복 클릭을 막기 위해 버튼을 일시 비활성화합니다 [66]. 수집 완료는 백엔드에서 JSON 생성 완료 신호를 보내주거나, UI가 주기적으로 폴링하여 `selected_keyword_articles.json`이 갱신되었는지 확인하는 방식으로 처리합니다 [62]. 수집이 완료되면 UI 리스트가 자동 새로고침되어 **새로운 기사 후보들이 나타나게** 됩니다. (초기 버전에서는 UI에서 수집 트리거를 생략하고, 편집자가 직접 수집 스크립트를 돌린 후 결과만 UI에서 보는 방식이었지만, 가능하면 UI에서 시작부터 발행까지 모두 진행할 수 있도록 확장되었습니다 [67].)

- **검색/필터 및 정렬**: (현재 기본 기능 혹은 향후 추가) 기사 리스트 상단에는 **필터와 정렬 기능**이 제공됩니다. 예를 들어 승인 상태별 필터 (승인됨/보류/제외) 토글, 점수 정렬 또는 키워드로 검색 등이 가능하도록 설계되었습니다 [68]. 이를 통해 편집자가 많은 기사 후보 중 필요한 것만 빠르게 찾아볼 수 있습니다. (현 시점에 일부 필터링은 수동으로 JSON에서 이루어지고, UI에는 최소한의 정렬만 구현되었을 수 있습니다. 향후 React 등을 도입하면 강화 가능.)

- **발행 미리보기**: 발행 전에 편집자가 최종 결과를 확인할 수 있도록, UI에 "**미리보기**" 기능이 고려되었습니다. 예컨대 선택된 기사들로 생성될 HTML 페이지를 미리 확인하는 버튼/모달을 제공하여, 오탈자나 구성 이상을 발견할 수 있게 합니다 [69]. (이 기능은 아직 구현 중이거나 수동으로 archive의 HTML을 여는 방식으로 대체될 수 있습니다.)

- **발행 실행 버튼**: 화면 하단 또는 상단에 "**발행**" 버튼이 배치되어 있습니다. 이 버튼은 현재 선택된 기사들(`selected: true`)과 코멘트들이 모두 준비되었을 때 활성화됩니다. 편집자가 발행을 눌러 확인하면, 프론트엔드는 `/admin/publish_keyword` API 엔드포인트를 호출합니다 [70]. 백엔드는 해당 요청을 받아 내부적으로 `--publish-keyword` 명령을 실행하거나 동일 로직을 호출하여 **Markdown/HTML 페이지를 생성하고, archive에 저장**합니다 [71] [66]. 발행 프로세스가 성공적으로 끝나면 API 응답을 통해 UI에 성공 상태를 전달하고, UI는 "발행 성공" 알림과 함께 해당 아카이브 페이지의 링크 또는 파일경로를 화면에 표시해 줍니다 (예: " 2025-10-03_리플로우.html 발행 완료"). 만약 발행 조건을 충족하지 못해 실패한다면 (예: 승인 기사 수 부족), **오류 메시지**를 띄워주고 발행이 되지 않았음을 알립니다 [72]. 또한 발행 중 중복 클릭을 방지하

기 위해 버튼을 비활성화하거나 로딩 인디케이터를 표시하고, 오류 발생 시 메시지를 표시하는 등의 UI/UX 처리를 합니다 [66].

- **사용 방법 요약**: 관리자는 **FastAPI 서버를 실행**한 뒤(예: uvicorn으로), 브라우저에서 관리자 UI 페이지(`http://<server-ip>:8000` 혹은 `http://<server-ip>:8000/index.html`)에 접속합니다. 그날의 키워드를 UI에 입력하여 수집을 실행하거나, 미리 스케줄된 수집 결과가 있다면 바로 **기사 리스트를 검토**합니다. 그런 다음 적합한 기사들을 체크하고 한 줄 코멘트를 작성합니다. 모든 편집 작업을 마쳤으면 **"발행" 버튼**을 눌러 해당 키워드 뉴스가 생성되도록 합니다. 발행 완료 후 아카이브 파일을 확인하거나, 독자용 프론트엔드에 반영하여 실서비스에 게시합니다. 이 모든 과정이 UI 상에서 이루어지므로, JSON 파일을 일일이 열어 수정하는 것보다 훨씬 편리하며 협업에도 용이합니다 [73] [74].

- **파일 경로 및 배포**: 관리자 UI는 `index.html` 한 파일로 이루어진 정적 페이지이므로, FastAPI 앱에서 `StaticFiles(directory="...")`로 서빙하거나, Flask라면 `templates/index.html`로 렌더링해 놓았을 것입니다. UI와 통신하는 API 경로는 `/admin/...`으로 네이밍되어 보호되거나 (간단히 Basic Auth나 IP 제한), 혹은 개발 용도로만 쓰이게 설정합니다. UI 자산 파일(css, js 등)이 별도로 있다면 프로젝트의 `static/` 혹은 `frontend/dist/` 등에 위치할 수 있습니다. 실제 서비스 시에는 관리자 UI를 로컬 머신에서만 띄우고, 최종 산출물(HTML 페이지)은 정적 호스팅하거나 메인 웹사이트에 업로드하는 형태로 운용할 수도 있습니다.

(참고: 현재 관리자 UI는 기능적으로 기본적인 승인/발행에 중점을 둔 Lite 버전입니다. 추후 React/Vue 기반으로 재구현하고 다중 사용자 권한, 실시간 동기화 등의 기능을 추가하는 방향이 계획되어 있습니다 [75] [76].)

6. 향후 작업 방향 및 개선 제안

퀄리저널 시스템은 초기 형태를 갖추고 일일 운영이 가능한 상태지만, 향후 발전시킬 수 있는 여러 개선점이 있습니다. 최신 콘텐츠 큐레이션 트렌드와 AI 기술을 반영하여 다음과 같은 **8가지 개선 방향**을 제안합니다:

1. **모듈 고도화 및 리팩터링** – 코드 구조를 보다 **모듈화하고 견고하게 개선**합니다. 현재 파이프라인을 담당하는 `orchestrator.py`를 수집, 파싱, 필터링, 정규화, 발행 등의 하위 모듈로 분리하고, 각 모듈에 대해 **함수형 리팩터링**과 단위 테스트 작성으로 안정성을 높입니다 [77]. 또한 **비동기 처리와 병렬성**을 도입하여 성능을 향상시킵니다. 예를 들어 asyncio/Aiohttp를 활용해 여러 RSS 피드를 동시에 요청하거나, 쓰레드 풀로 병렬 크롤링하여 전체 수집 시간을 단축할 수 있습니다 [78]. 이러한 리팩터링을 통해 코드 가독성과 유지보수성이 향상되고, 새로운 기능 추가 시 회귀 없이 빠르게 적용할 수 있습니다.

2. **트렌드 기반 키워드 추천** – 키워드 선정의 자동화를 고도화합니다. 현재는 사람이 키워드를 선택하지만, 향후에는 **전일 뉴스 트렌드**나 외부 데이터 분석을 통해 **다음 발행 키워드**를 자동으로 추천하거나 순환시키도록 합니다 [79] [80]. 예를 들어 전날 수집된 기사들의 빈도를 분석해 연관 키워드를 뽑아내거나, Google Trends, 소셜 미디어 언급량, 산업 동향 보고 등을 참고해 **인기 상승 중인 키워드**를 포착합니다 [80]. 이를 알고리즘화하여 매일 아침 시스템이 "오늘의 추천 키워드"를 제안하거나, 키워드 풀(pool)을 만들어 **자동 로테이션**할 수 있습니다 [79]. 이렇게 하면 큐레이션 주제를 전략적으로 운영하면서도 편집자 부담을 줄일 수 있습니다.

3. **요약/번역 자동화** – 모든 영문 기사에 대해 **한국어 요약과 번역을 완전 자동화**합니다. 현재 일부 핵심 문장 추출과 API 번역을 도입했으나, 이를 한층 발전시켜 OpenAI GPT-4 등 **최신 LLM**을 활용한 고품질 요약문 생성을 전면적으로 적용합니다 [32]. 기사 본문에서 가장 중요한 2~3문장을 뽑고, 그 내용을 기반으로 자연스러운 한국어 요약을 생성한 뒤, 전문 용어는 미리 정의한 용어집(Glossary)으로 치환하여 일관성을 유지합니다 [81]. 또한 필요한 경우 GPT를 활용해 **본문 번역**도 수행하여, 독자가 원문을 읽지 않더라도 내용을 이해할 수 있도록 돕습니다. 이 모든 과정을 발행 전에 자동 수행함으로써, 편집자는 요약 품질을 검토만 하고 코멘트 작성에 집중할 수 있습니다. (추가로, 요약문을 통해 편집자가 기사 승인 여부를 판단하기 쉽게 만드는 효과도 기대됩니다.)

4. **검색 기능 개선 (Elasticsearch 도입)** – 누적되는 키워드 뉴스 콘텐츠에 대한 **검색 및 색인 시스템**을 구축합니다. 현재 JSON/파일 기반 저장은 검색에 한계가 있으므로, **Elasticsearch/OpenSearch**와 같은 전문 검색엔진을 도입하여 키워드별 아카이브를 색인하고 시간순·관련도순 검색을 최적화합니다 [82]. 이를 통해 관리자도 과거 어떤 기사가 어떤 키워드로 나왔는지 빠르게 찾을 수 있고, 독자용 사이트에도 **검색 창**을 제공하여 과거 뉴스나 특정 주제 관련 뉴스를 손쉽게 찾아볼 수 있게 합니다. 또한 ES의 Aggregation 기능 등을 활용하면 **트렌드 분석**에도 활용 가능하므로, 장기적으로 가치 있는 데이터베이스를 구축하게 됩니다.

5. **AI 편집장 기능 강화** – 편집자의 작업을 돕는 **AI 보조 기능**을 확충합니다. 예를 들어 **기사 추천/필터링**: AI가 수집된 기사들을 미리 평판 분석하여 높은 가치 기사를 추천하거나, 반대로 신뢰 낮은 기사(루머성, 중복 정보)를 표시해줄 수 있습니다 [83]. **편집자 코멘트 초안 생성**: `editor_rules.json`에 정의한 프롬프트와 스타일 가이드를 활용해 GPT 모델이 각 기사에 대한 **한 줄 논평** 초안을 자동 생성하도록 합니다 [51]. 편집자는 AI가 제안한 문장을 약간 다듬는 정도로 효율적으로 코멘트를 완성할 수 있습니다. **트렌드 예측**: AI가 특정 키워드의 향후 중요도를 예측해 다음 키워드를 미리 제안하거나, 관련 기사를 추가 발굴해줄 수도 있습니다 [83]. 이러한 human-in-the-loop 형태의 AI 도구들은 편집 업무의 **속도와 품질을 동시에 향상**시켜줄 것으로 기대됩니다.

6. **CI/CD 자동화 및 컨테이너화** – **지속적 통합/배포 파이프라인(CI/CD)**을 구축하여 운영 효율을 높입니다. 프로젝트를 Docker 컨테이너로 만들어, 개발/운영 환경 간 차이를 제거하고 배포를 용이하게 합니다 [84]. 예를 들어 Dockerfile을 작성해 FastAPI 서버와 관련 스크립트, 크론잡 등을 하나의 이미지로 만들고, Github Actions 또는 GitLab CI를 사용해 코드를 push하면 자동으로 빌드/테스트 후 서버에 배포되도록 구성합니다 [84]. 이를 통해 **일관된 환경에서 실행**되므로 윈도우/리눅스 간 스크립트 문제나 패키지 충돌을 방지할 수 있습니다. 또한 CI 단계에서 단위/통합 테스트를 실행해 새로운 코드 변경이 기존 기능에 문제를 일으키지 않는지 확인하고, 코드 품질을 지속적으로 관리합니다.

7. **에러 모니터링 및 로깅 강화** – 시스템 운영 시 발생하는 **각종 오류와 성능 지표를 모니터링**하는 체계를 갖춥니다. 수집부터 발행까지 단계별로 **구조화된 로그**를 남겨 언제, 어떤 키워드에, 몇 개 기사를 수집했고, 몇 초 걸렸으며, 에러는 없었는지 등을 기록합니다 [85]. 이 로그를 Elastic Stack이나 Grafana/Prometheus와 연계하여 대시보드화하면, 운영 현황을 한눈에 볼 수 있습니다 [85]. 또한 Slack이나 이메일 알림을 연동해 수집 실패나 발행 오류 시 즉각 통지받도록 설정합니다 [85]. 이를 통해 문제 발생 시 신속히 대응하고, 장기적으로 **성공률, 소요 시간 등의 통계**를 분석해 병목 구간을 최적화할 수 있습니다.

8. **사용자 피드백 및 개인화 분석** – 독자 측면의 데이터를 수집·분석하여 **콘텐츠 품질 향상**에 반영합니다. 예를 들어 발행된 키워드 뉴스에 대해 **조회수, 클릭률, 평균 체류 시간** 등의 데이터를 누적하고, 가장 인기 있던 키워드나 기사를 파악합니다. 이를 기반으로 편집 전략을 조정하거나, 향후 AI 추천 알고리즘 학습 데이터로 활용합니다. 관리자 대시보드에 이러한 **독자 반응 지표**를 시각화하여 제공하면, 편집자가 어떤 주제가 호응이 높았는지 참고하여 다음 키워드를 선정하는 데 도움을 줍니다 [86]. 나아가 개별 사용자 피드백(댓글, 설문 등)을 수렴하고 NLP 기술로 의견을 분석해 개선점을 찾는 것도 고려할 만합니다. 2025년 미디어 트렌드는 **개인화된 콘텐츠와 AI 추천**이 핵심이므로 [87] [88], 퀄리저널도 장기적으로 독자별 맞춤 뉴스레터나 구독 기능 등을 도입해 **서비스화**를 추진할 수 있을 것입니다.

以上의 개선 사항들은 현재 시스템의 **한계점**을 보완하고, 차세대 큐레이션 플랫폼으로 발전시키기 위한 로드맵입니다. 순차적으로 우선순위를 정해 도입하면, 퀄리저널의 **운영 효율성과 콘텐츠 경쟁력**을 크게 높일 수 있을 것입니다.

7. 실행 팁: 개발/운영 환경 및 명령어 정리

마지막으로, 후속 작업자가 **퀄리저널 프로젝트를 직접 실행하고 재시작**할 수 있도록 몇 가지 팁과 명령 사용법을 정리합니다:

- **개발 환경 설정**: Python 3.x 가상환경을 만들고 `requirements.txt`를 사용하여 필요한 패키지를 설치합니다. 예를 들어:

```
$ python -m venv venv
$ source venv/bin/activate  # (Windows에서는 venv\Scripts\activate)
$ pip install -r requirements.txt
```

요구사항 파일에 명시되지 않은 경우, 추가로 `uvicorn[standard]`, `fastapi`, `pandas`(데이터 처리용), `praw`(Reddit API용) 등이 필요할 수 있으니 확인하십시오. (기존 코드와 스크립트(`orchestrator.py`, `tools/*`, `run_quali_today.ps1`)도 함께 검토하여 의존 패키지를 파악합니다 [59].)

- **FastAPI 서버 실행 (관리자 UI 접속)**: FastAPI 앱이 정의된 모듈을 찾아 **uvicorn으로 실행**합니다. 일반적으로 main 모듈에 `app = FastAPI()` 객체가 있을 것입니다. 예를 들어 main.py 파일 내에 app이 있다면 아래 명령으로 개발 서버를 띄웁니다:

```
$ uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

(혹은 `server.py`나 `orchestrator.py` 등에 app이 정의됐을 수 있으니 파일명을 확인 바랍니다.) 서버가 뜨면 터미널에 API 경로들이 로그로 표시되고, `http://localhost:8000`에서 접속 대기 중일 것입니다. 이때 브라우저를 열어 `http://localhost:8000/index.html`(또는 `/admin` 경로)로 접속하면 **관리자 UI** 화면이 나타납니다. 만약 UI 화면이 뜨지 않거나 API 통신 오류가 있다면, **CORS 설정**이나 정적 파일 경로 설정을 확인해야 합니다.

- **일일 파이프라인 실행 (수동)**: 자동 스케줄러 외에 수동으로 당일 키워드 뉴스를 발행하고 싶다면, **PowerShell 스크립트**를 직접 실행하거나 필요한 Python 명령어를 순차로 호출하면 됩니다.
  Windows PowerShell 사용 시: 프로젝트 루트에서 `run_quali_today.ps1`을 우클릭하여 PowerShell에서 실행하거나, PowerShell 프롬프트에서 경로 이동 후 `.\run_quali_today.ps1`을 입력합니다 [35]. 스크립트는 오늘 날짜와 키워드를 이용해 수집→발행까지 모두 처리해줄 것입니다. 스크립트 실행 중 콘솔에 각 단계 진행 상황과 로그가 출력되니 참고하십시오.
  개별 단계 수동 실행: PowerShell 사용이 어려운 환경에서는 Python 명령으로 각 단계를 차례로 실행할 수 있습니다. 예를 들어:

```
$ python orchestrator.py --collect --collect-community --use-external-rss
$ python tools/rebuild_kw_from_today.py
# (필요시 편집자 승인 -> selected_keyword_articles.json 편집)
$ python tools/sync_selected_for_publish.py
$ python orchestrator.py --publish-keyword "오늘의키워드"
```

또는 간략히, 특정 키워드에 대해 한 번에 수집부터 발행까지 수행하려면:

```
$ python orchestrator.py --collect-keyword "리플로우" --use-external-rss
$ python orchestrator.py --publish-keyword "리플로우"
```

처럼 `--collect-keyword` 및 `--publish-keyword` 옵션을 사용할 수도 있습니다 [89] [90]. (이 방식은 내부에서 앞선 개별 단계를 모두 호출해 줍니다.)

- **일일 자동 스케줄 설정**: 운영 환경에서 매일 특정 시간에 자동으로 뉴스를 발행하려면 **작업 스케줄러**(Windows)나 **cron 잡**(Linux)을 설정합니다. Windows에서는 작업 스케줄러에 새 작업을 만들어 `powershell.exe -File "C:\path\to\run_quali_today.ps1"`를 매일 아침 7시 등 원하는 시간에 실행하게 예약합니다 [91]. Linux 환경에서는 `cron`에 `/usr/bin/python /path/to/orchestrator.py --collect-keyword "..." && ...` 등의 명령을 등록할 수 있습니다. 또한 실행 후 결과를 Slack이나 이메일로 보내도록 스크립트를 수정하면, 발행 성공/실패 여부를 원격으로 확인할 수 있어 편리합니다 [91].

- **관리자 UI 사용**: 서버가 떠 있고 수집 단계까지 완료되었다면, UI에 접속하여 **편집 및 발행 작업**을 합니다. UI에서 키워드 입력→수집을 할 경우 백엔드에서 수집기가 돈 뒤 자동으로 리스트가 표시됩니다. 또는 스케줄러로 이미 수집되었다면 UI에 바로 목록이 나타납니다. 각 기사에 대해 체크박스를 클릭(승인)하고, 코멘트를 입력한 뒤, 모두 완료되면 **발행 버튼**을 누르세요. 발행 후 성공 메시지가 나오면, archive 폴더를 확인하여 새 파일들이 생성되었는지 검증합니다. (개발 중에는 발행 전에 미리보기 HTML을 열어볼 것을 권장합니다.)

- **기타 팁**: 처음 프로젝트를 재가동할 때 **editor_rules.json과 config.json을 꼭 확인**하세요. 특히 `require_editor_approval` 설정이 True로 되어 있는지, `community.filters` 임계값이 적절한지 점검하는 것이 좋습니다. 또한 공식 소스 RSS 링크들이 현재 유효한지 테스트해보고(dead link가 있다면 `official_sources.json` 갱신), Reddit API 크레덴셜이나 토큰이 필요한 경우 미리 설정해야 커뮤니티 수집이 원활합니다. 만약 커뮤니티 수집 결과가 계속 0건이라면, `community_sources.json`의 필터 조건을 완화하거나 `praw.ini` 등 Reddit API 설정을 추가해야 합니다 [92] [93].

- **독자용 UI 연동**: 발행된 Markdown/HTML은 독자들이 보는 웹사이트에 게시되어야 의미가 있습니다. 현재는 관리자 UI에서 발행까지만 다루지만, 실제 운영 시에는 메인 사이트에 **키워드 뉴스 섹션**을 만들고, archive의 HTML을 해당 경로에 두거나 내용을 CMS에 복사하는 절차가 필요합니다 [94] [95]. 추후 개선으로 **독자용 프론트엔드 페이지**를 자동 생성/배포하는 기능도 고려 중이니, 당장은 수동이더라도 꾸준히 archive를 업데이트해 주는 것을 권장합니다.

以上의 단계와 팁을 따르면, 새로운 개발자가 프로젝트를 빠르게 재시작하고 운영 사이클에 합류할 수 있을 것입니다. 프로젝트 구조와 흐름을 이해했으므로, 작은 변경이나 개선을 가하면서 쿼리저널을 발전시켜 나가길 기대합니다. **Happy Coding!**

---

1 2 7 8 9 10 16 17 18 19 21 22 23 24 25 26 27 34 38 39 41 42 43 46 47 48 50 60 61 71 72 90 1004_2A고도화report.pdf
file://file-Ko3WmavWUUxUmLMC1s6fso

3 4 5 32 81 82 83 1003_5A쿼리뉴스를 전면적으로 리팩터링하여 안정적이고 확장 가능한 시스템으로 구축하려면.pdf
file://file-C1X8pMSvAVd6Xs392FEnwq

6 15 30 31 33 51 57 58 92 93 1003_1A쿼리 뉴스 제안.pdf
file://file-9ckyN6UsWeG3crtkDBreya

11 12 13 14 28 29 52 54 56 1003_3A 하루 한 키워드” 역사 큐레이션형 쿼리뉴스를 구축하기 위한 구체적 개발 가이드입니다.pdf
file://file-92Q3asQNjsGDWWy73Fkbpt

20 35 36 37 40 44 45 49 53 55 59 68 73 74 75 76 77 78 79 80 84 85 86 87 88 91 1004_3A작업계획report.pdf
file://file-S3G4TQFPei3GeAJcXjN6B3

62 63 64 65 66 67 69 70 89 94 95 1003쿼리뉴스 _키워드 뉴스_ 기능 설계 및 구현 전략.pdf
file://file-XoVMv2VGBLyHnFVetrmoGx