퀄리저널 프로젝트 문제해결 보고서

1 개요

본 보고서는 **1005_1 퀄리저널 프로젝트 인수인계 보고서**, **1004_1B 퀄리저널 개발 계획서**, **1004_2A 고도화 report**, **1004_3A 작업계획 report** 를 분석하여 현행 시스템의 미흡한 점과 잠재적 리스크를 찾아내고, 이를 해결하기 위한 실행 가이드를 제시한다. 퀄리저널은 하루 한 키워드를 중심으로 공식 뉴스, 학술 논문, 표준 문서, 커뮤니티 글 등 다양한 출처의 콘텐츠를 수집·큐레이션하여 최소 15 개의 고품질 기사 카드로 구성된 특별 호를 발행하는 시스템이다【642733083376290†L0-L14】. 선정된 기사와 메타데이터는 `selected_keyword_articles.json` 에 저장되며 편집자가 승인한 기사만 `selected_articles.json` 에 모아 발행본을 구성한다【685450869347888†L18-L27】.

2 현황 및 문제점

시스템은 모듈화된 파이프라인과 품질게이트 등 강점을 갖추었지만 실제 운영 과정에서 여러 한계와 리스크가 발견된다. 아래 표는 주요 문제를 요약한다. 각 문제는 해당 문서의 근거를 함께 제시한다.

분류	주요 문제 (키워드)	근거
편집 과정	**편집자 의존성과 UI 부재** – 기사 승인과 코멘트 입력이 JSON 파일을 직접 편집하는 방식에 의존하여 사용자 경험이 낮고 오류 가능성이 높음	고도화 보고서는 기사 선택과 코멘트 입력을 위해 JSON 을 직접 수정해야 하므로 사용자 경험이 낮다고 지적한다【610174338146614†L519-L527】. 작업계획도 발행 전까지 편집자가 JSON 파일을 수동으로 살펴보며 승인 플래그와 `editor_note` 를 기록해야 한다고 언급한다【685450869347888†L29-L33】.
자동화	**수동 검수·15 개 승인 기준** – 최소 15 개 기사 승인이라는 품질 게이트 때문에 편집자의 수동 검수가 필수이다	고도화 보고서는 최소 15 개 승인이라는 기준 때문에 편집자의 수동 검수와 승인 작업이 필요하며, 긴급 모드는 임시 방편에 불과하다고 명시한다

분류	주요 문제 (키워드)	근거
		【610174338146614†L470-L483】.
필터링	**정적 임계값** – 커뮤니티 점수 임계값과 최소 기사수 등 필터링 기준이 정적으로 설정되어 키워드에 따라 지나치게 엄격하거나 느슨할 수 있다	고도화 보고서는 커뮤니티 점수 임계값과 최소 기사수가 정적으로 설정되어 상황에 따라 엄격하거나 느슨할 수 있으므로 동적 조정이나 모델이 필요하다고 지적한다 【610174338146614†L486-L495】.
데이터 소스	**RSS 소스 불안정** – 공식 소스의 RSS URL 변경 시 수집 실패 위험	고도화 보고서는 공식 소스 URL 이 변경될 때 수집이 실패할 수 있어 자동 검증과 알림 시스템이 필요하다고 강조한다 【610174338146614†L496-L506】.
운영 도구	**PowerShell 기반 자동화** – Windows 환경에 종속된 PowerShell 스크립트는 f-string 오류와 경로/인코딩 문제로 유지보수 부담이 크다	고도화 보고서는 Windows 환경에 종속된 PowerShell 스크립트에서 f-string 오류와 인코딩 문제 등 유지보수 부담을 지적한다 【610174338146614†L508-L516】.
검색·분석	**트렌드 분석 및 추천 기능 부족** – 키워드를 운영자가 수동으로 지정해야 하므로 트렌드 변화에 대한 대응이 늦다	고도화 보고서는 키워드를 운영자가 수동으로 지정해야 해 트렌드 변화에 늦다고 언급한다【610174338146614†L532-L537】.
검색·저장	**검색 및 데이터 탐색 기능 미흡** – JSON 파일 기반 저장 방식은 빠른 검색이나 연관 기사 탐색에 한계가 있다	고도화 보고서는 현재 JSON 파일 기반으로 목록을 불러오기 때문에 빠른 검색이나 연관 기사 탐색이 어렵다고 지적한다【610174338146614†L540-L546】.
품질·테스트	**코드 품질·테스트 미흡** – 코드	고도화 보고서는 코드 리팩터

분류	주요 문제 (키워드)	근거
	리팩터링과 에러 로그 관리, 단위 테스트가 부족하여 장기적 확장성에 문제가 있다	링, 에러 로그 관리, 단위 테스트 추가가 필요하다고 명시한다【610174338146614†L548-L555】.
협업	**변경 이력 관리 미흡** – 여러 편집자가 동시에 작업할 경우 충돌과 이력 추적 문제가 발생할 수 있음	작업계획은 여러 편집자가 동시에 작업할 수 있도록 변경 이력 표시와 잠금/경고 메시지 등 협업 기능을 추가해야 한다고 제안한다【685450869347888†L998-L1009】.
백업	**아카이브 중복 및 백업 관리** – 동일 키워드로 여러 번 발행하는 경우 날짜만으로 구분하면 파일이 덮어써질 위험이 있으며, 아카이브 폴더의 용량 관리도 필요	고도화 보고서는 동일 키워드로 여러 번 발행할 경우 시간까지 포함한 파일명을 사용하고 아카이브 폴더는 백업·압축을 통해 용량을 관리해야 한다고 강조한다【610174338146614†L376-L381】.

3 개선 방안

문제점을 해결하기 위해 아래와 같은 개선 방안을 제안한다. 각 대안은 실행 가능성과 확장성을 고려하여 구체적인 실행 단계로 이어질 수 있도록 설명하였다.

3.1 편집 프로세스 개선

1. **관리자 UI 구축과 협업 기능** – JSON 을 수동 편집하는 대신 FastAPI 기반의 REST 백엔드를 구축하고 React 나 Vue.js 로 관리자 대시보드를 개발한다. 프론트엔드에서 기사 목록을 테이블로 표시하고 승인 체크박스, 코멘트 입력란, 검색·정렬·필터 기능을 제공하면 편집자가 직관적으로 작업할 수 있다【685450869347888†L966-L995】. 여러 편집자가 동시에 작업할 수 있도록 변경 이력 기록과 잠금/경고 메시지를 추가하여 협업 충돌을 방지한다【685450869347888†L998-L1009】.
2. **프리뷰 및 발행 버튼** – UI 에서 미리보기 기능을 제공하여 승인된 기사로 생성될 HTML 페이지를 검토한 뒤 발행하도록 하고, 발행 성공·오류 메시지를 명확히 표시한다【849264231269906†L518-L527】.

3.2 자동화·필터링 고도화

1. **LLM 을 활용한 기사 평가와 요약** – 전통적 키워드 매칭만으로는 뉴스 가치 판단이 어렵다. 대형 언어 모델(LLM)을 활용해 기사 내용의 시의성, 산업·표준에 미치는 영향, 논쟁성, 일반 독자의 관심도를 평가하도록 프롬프트를 설계한다【610174338146614†L560-L623】. 모델이 생성한 요약과 키포인트를 편집자가 검토하는 human-in-the-loop 구조를 유지하여 자동화된 선별 정확도를 높이고 편집자의 부담을 줄인다【610174338146614†L656-L667】.
2. **동적 임계값 알고리즘** – 커뮤니티 글의 점수 임계값과 최소 기사 수를 과거 수집 결과에 기반하여 자동 조정하는 알고리즘을 구현한다. 예를 들어 최근 일주일 동안 수집한 글의 점수 분포를 분석해 현재 키워드에 적합한 threshold 를 추천하고, 운영자가 UI 에서 조정할 수 있게 한다【610174338146614†L486-L495】.
3. **공식 소스 검증 스크립트** – RSS URL 변경이나 404 오류를 자동으로 감지하는 관리 스크립트를 작성하여 `official_sources.json` 을 주기적으로 검사하고 문제가 있을 경우 경고 메시지를 출력한다【610174338146614†L496-L506】.
4. **키워드 동의어 및 로테이션** – `keyword_synonyms.json` 을 활용해 키워드 변형과 동의어를 자동으로 검색하고, 운영자가 정의한 키워드 풀을 주간 또는 월간 주기로 자동 로테이션하는 기능을 구현한다【685450869347888†L945-L951】.

3.3 인프라 및 배포

1. **PowerShell 종속성 제거** – `orchestrator.py` 가 모든 단계를 제어할 수 있으므로 Windows 환경에 종속된 PowerShell 스크립트를 단계적으로 폐기한다. 대신 Python CLI 와 Unix cron/Windows 작업 스케줄러를 공통 인터페이스로 제공하여 플랫폼 의존성을 줄인다【610174338146614†L508-L516】.
2. **컨테이너화 및 CI/CD** – Dockerfile 을 작성하여 FastAPI 서버와 크롤러·요약 모듈을 하나의 이미지로 패키징하고, GitHub Actions / GitLab CI 를 이용해 빌드·테스트 후 자동 배포를 수행한다【849264231269906†L640-L645】. 컨테이너 환경에서는 OS 의존성 문제가 줄어들고 배포가 용이해진다【849264231269906†L640-L646】.
3. **로그·모니터링** – 수집부터 발행까지 단계별로 구조화된 로그를 남기고 Elastic Stack 또는 Grafana/Prometheus 와 연동하여 대시보드를 구성한다. Slack 또는 이메일 알림을 연동해 수집 실패나 발행 오류를 즉시 통지할 수 있도록 한다【849264231269906†L647-L655】.
4. **백업 및 파일 명명 규칙** – 동일 키워드로 여러 번 발행할 때는 시간까지 포함한 파일명을 사용하여 아카이브 충돌을 방지하고, 아카이브 폴더는 주기적으로 압축·백업하여 용량을 관리한다【610174338146614†L376-L381】.

3.4 검색·분석 기능 향상

1. **Elasticsearch 연동** – 발행된 기사와 원본 데이터를 Elasticsearch/OpenSearch 인덱스로 저장하여 빠른 검색과 연관 기사 탐색을 지원한다. 인덱스에는 키워드, 요약, 출처, 날짜, 점수 등을 저장하고, 독자용 검색 API 를 제공하여 과거 기사나 특정 주제의 아카이브를 손쉽게 찾을 수 있게 한다【610174338146614†L735-L744】【685450869347888†L851-L864】.
2. **트렌드 분석 및 키워드 추천** – 전일 또는 최근 일주일 동안 수집한 데이터에서 많이 언급된 표준명이나 키워드를 추출하고 Google Trends API 등 외부 트렌드 데이터와 결합해 자동 추천을 제공한다【610174338146614†L687-L724】. 또한 LLM 에게 전일 기사에서 공통 주제와 떠오르는 이슈를 요약하고 다음 발행에 적합한 키워드를 추천하도록 프롬프트를 설계한다【610174338146614†L703-L714】.

3.5 코드 품질 및 테스트

1. **모듈화와 리팩터링** – `orchestrator.py` 를 수집, 파싱, 필터링, 평가, 발행 등의 하위 모듈로 분리하고 각 모듈에 대해 함수형 설계와 단위 테스트를 작성한다. 비동기 처리와 병렬 요청을 도입하여 RSS 크롤링 속도를 향상시키고 코드 가독성을 높인다【849264231269906†L579-L585】.
2. **단위 테스트와 CI 통합** – pytest 등 테스트 프레임워크를 도입하여 크롤러, 평가기, 요약기, 발행기 등 핵심 모듈의 단위 테스트를 작성하고, CI 파이프라인에서 테스트를 자동 실행하여 코드 품질을 유지한다【610174338146614†L548-L555】.
3. **오류 처리 강화** – JSON 형식 오류와 데이터 무결성을 검증하기 위해 스키마 검증 도구를 도입하고, `repair_selection_files.py` 와 같은 교정 기능을 리팩터링하여 선택 파일 구조를 자동 검사하도록 한다【849264231269906†L115-L116】.

4 실행 가이드

아래 단계는 시스템을 개선하기 위한 실행 계획을 제시한다. 하루 단위로 진행해도 무리가 없는 범위로 분리하였다.

단계 1 – 환경 준비와 코드 분석

1. **가상환경 설정** – 프로젝트 루트에서 Python 3.x 가상환경을 만들고 `requirements.txt` 를 기반으로 패키지를 설치한다. 필요 시 `fastapi`, `uvicorn`, `praw` 등 추가 의존성을 확인한다【849264231269906†L693-L699】.
2. **기존 코드 리뷰** – `orchestrator.py`, `tools/*`, PowerShell 스크립트를 검토하여 현재 파이프라인의 동작을 파악한다. `selected_keyword_articles.json`, `selected_articles.json`, `official_sources.json`, `community_sources.json`, `keyword_synonyms.json` 구조를 분석하여 데이터 모델을 설계한다

【685450869347888†L885-L914】.

3. **컨테이너 환경 준비** – 기본 Dockerfile 을 작성해 FastAPI 앱과 스크립트를 패키징하고 컨테이너에서 실행을 테스트한다. 이 단계에서 OS 의존성 문제를 조기에 발견할 수 있다【685450869347888†L890-L906】.

단계 2 – 수집·필터링 고도화 및 키워드 자동화

1. **소스 검증 스크립트** – RSS 링크를 주기적으로 검사하는 스크립트를 작성하여 404 오류나 구조 변경을 탐지하고 경고를 기록한다【610174338146614†L496-L506】.
2. **동적 임계값 로직** – 최근 수집된 커뮤니티 글의 점수 분포를 분석해 적절한 threshold 를 계산하고 `config.json` 에서 동적으로 업데이트하는 기능을 구현한다【610174338146614†L486-L495】.
3. **키워드 로테이션과 동의어 확장** – `keyword_synonyms.json` 을 업데이트하여 주요 키워드의 변형을 정의하고, 주간/월간 키워드 풀을 자동 회전하는 로직을 추가한다【685450869347888†L945-L951】.
4. **스케줄러 설정** – 수집·발행 작업을 cron 또는 Windows 작업 스케줄러에 등록하고, 성공·실패 결과를 Slack 이나 이메일로 통지하는 기능을 구현한다【685450869347888†L953-L963】.

단계 3 – 관리자 UI 구축 및 편집 프로세스 개선

1. **REST 백엔드 개발** – FastAPI 로 키워드 등록, 기사 목록 조회·승인·코멘트 입력, 발행 트리거 등의 API 를 구현한다【685450869347888†L966-L980】.
2. **프론트엔드 대시보드** – React/Vue.js 기반 대시보드를 개발하여 기사 목록을 표 형태로 보여주고 승인 체크박스, 코멘트 입력란, 검색·정렬·필터 기능을 제공한다【685450869347888†L986-L995】.
3. **미리보기·발행 기능** – 선택된 기사로 생성될 HTML 페이지를 미리보기하는 기능과 발행 버튼을 구현하고, 발행 성공/실패 메시지를 UI 에 표시한다【849264231269906†L518-L527】.
4. **협업 기능** – 변경 이력 기록과 잠금/경고 메시지를 추가하여 여러 편집자의 동시 작업을 지원하고, 승인/보류/편집자 코멘트의 변경 내역을 데이터베이스에 저장한다【685450869347888†L998-L1009】.

단계 4 – AI 요약·추천 및 데이터 분석

1. **요약/평가 모듈** – OpenAI API 또는 사내 LLM 을 이용해 기사 본문을 요약하고 시의성·영향력·논쟁성·관심도를 평가하는 프롬프트를 설계한다【610174338146614†L560-L623】. 요약문과 키포인트를 프론트엔드에 표시하여 편집자가 빠르게 검토할 수 있게 한다.
2. **트렌드 분석·키워드 추천** – Elasticsearch 나 자체 분석 모듈에서 지난 주 수집 데이터를

분석해 많이 언급된 키워드를 추출하고, Google Trends API 등 외부 데이터와 결합해 다음 키워드를 추천한다【610174338146614†L687-L724】.

3. **독자 행동 데이터베이스** – PostgreSQL 등 데이터베이스를 도입하여 독자의 클릭·조회 기록을 저장하고 개인화 추천 알고리즘을 구현한다. 추천 결과에 대해 편집자가 가중치 조정이나 제외 처리를 할 수 있는 human-in-the-loop 기능을 제공한다【685450869347888†L1036-L1055】.
4. **대시보드 통계 탭** – 키워드별 관심도, 기사별 조회수, 추천 클릭률 등 지표를 시각화하는 분석 탭을 대시보드에 추가한다【685450869347888†L1058-L1066】.

단계 5 – 모듈화·배포 자동화 및 최종 테스트

1. **모듈 분리와 테스트 작성** – `orchestrator.py`를 수집, 필터링, 발행, 추천 등의 독립 모듈로 분리하고 pytest 기반의 단위 테스트를 작성한다【685450869347888†L1070-L1077】.
2. **CI/CD 파이프라인 구축** – GitHub Actions / GitLab CI를 사용해 코드 푸시 시 자동으로 도커 이미지 빌드, 테스트 실행, 배포를 수행한다【685450869347888†L1078-L1089】.
3. **성능 최적화** – 비동기 처리와 병렬 수집 로직을 적용해 크롤링 속도를 향상시키고, 로그/모니터링 시스템을 통해 성능 및 오류 상황을 지속적으로 관찰한다【685450869347888†L1091-L1097】.
4. **최종 검증 및 문서화** – 데이터 수집, 필터링, 키워드 회전, 편집 UI, AI 요약·추천 기능이 기대한 대로 동작하는지 종합 테스트를 수행한다【685450869347888†L1101-L1109】. 마지막으로 인수인계를 위한 문서와 개발 가이드를 작성하여 프로젝트 팀에 전달한다【685450869347888†L1111-L1150】.

5 결론

퀄리저널 시스템은 일일 키워드 뉴스 발행이라는 명확한 목표를 가지고 있지만, 현재는 편집자의 수작업과 정적 설정에 많이 의존하고 있고, 검색·추천·UI 등에서 현대적 서비스 수준에 미치지 못한다. 본 보고서는 인수인계 문서와 개발 계획서를 기반으로 편집 프로세스, 자동화, 데이터 관리, 검색·분석, 코드 품질 측면에서 문제를 식별하고 해결 방안을 제시했다. 제안된 실행 가이드를 단계적으로 적용하면 운영자의 부담을 줄이고 독자에게 더 가치 있는 맞춤형 정보를 제공하는 현대적 키워드 뉴스 플랫폼으로 발전할 수 있을 것이다.