ChatGPT

QualiJournal Workflow 개선 및 고도화 계획

이 문서는 퀄리저널 관리자 시스템의 현재 워크플로우를 분석한 후, 향후 개선과 고도화를 위한 방향성을 제시한다. 패치 작업으로 **통합 발행 플로우**가 추가되었지만, 아직 비동기 처리, UI/UX, 데이터 관리, 보안 측면에서 개선할 여지가 많다. 아래에는 각 영역별 개선점과 이를 구현하기 위한 구체적 실행 계획을 정리하였다.

1. 현재 워크플로우 평가

1.1 장점

- **통합 발행 플로우 도입** – 최근 패치에서 `/api/flow/merged` 엔드포인트와 "원클릭(통합 발행)" 버튼을 추가함으로써 공식 수집 → 커뮤니티 수집 → 커뮤니티 Top-20 승인 → 병합 발행까지 한 번에 처리할 수 있게 되었다. 이 덕분에 운영자는 단순화된 클릭 한 번으로 일간 뉴스 발행을 할 수 있게 됐다.
- **엔진 분리** – `engine_core.py`와 `orchestrator.py`를 독립된 수집기로 유지함으로써 각 기능을 명확히 분리했다. 이는 소스별 로직을 개별적으로 수정할 수 있게 해 유지보수성을 높였다.
- **관리 UI의 폴백 로직** – 관리 UI는 `/api/tasks/*` 비동기 API가 없을 경우 `/api/flow/*` 엔드포인트로 자동 폴백하는 구조다. 비동기 기능을 점진적으로 추가하기 용이하다.

1.2 한계

- **비동기 처리가 없음** – 현재는 모든 플로우가 동기 방식으로 실행된다. 이 때문에 긴 작업 동안 UI가 응답하지 않고, 사용자가 진행 상황을 실시간으로 파악할 수 없다. 중기계획서에서는 비동기 작업 API 및 SSE 스트리밍을 도입해 작업 상태와 로그를 실시간 전송하도록 제안하고 있다.
- **TaskManager 미구현** – 작업 상태를 추적하는 TaskManager 인터페이스가 없어 진행 상황, 오류, 취소 상태를 저장하지 못한다. 중기계획서에서는 메모리와 Redis를 지원하는 TaskManager 설계를 권장한다.
- **UI/UX 부족** – 진행률 모달이 단순한 텍스트 목록에 불과해 작업 단계별 성과를 직관적으로 전달하지 못한다. 문서에서는 아이콘, 실행시간 표시, 타임라인 차트, KPI 자동 새로고침, 게이트 임계값 슬라이더 및 접근성 개선을 포함한 UI 개선을 요구한다.
- **보고서·내보내기 API 미완성** – 특집과 일간 보고서를 생성·번역·내보내기하는 `/api/report`, `/api/enrich/*`, `/api/export/*` 엔드포인트가 아직 구현되지 않았다.
- **테스트 및 문서화 부족** – 현재 테스트 케이스가 거의 없으며, 문서도 비동기 기능과 새 API를 설명하지 않는다. 계획서는 Pytest 기반 통합·회귀 테스트와 README 업데이트를 요구한다.
- **보안·인증 미흡** – 모든 엔드포인트가 인증 없이 호출 가능하며, HTTP로 서비스하고 있다. 앞으로 토큰 기반 인증과 HTTPS 도입을 고려해야 한다.

2. 개선 방안

2.1 비동기 API 및 작업 관리

1. **비동기 엔드포인트 구현** – `/api/tasks/flow/{kind}`, `/api/tasks/{job_id}`, `/api/tasks/{job_id}/stream`, `/api/tasks/{job_id}/cancel`, `/api/tasks/recent`를 구현한다. `kind`는 `daily`, `community`, `keyword`이며 각 작업은 `asyncio.create_subprocess_exec`를 이용해 비동기 실행된다.
2. **TaskManager 인터페이스 설계** – 메모리, Redis 등을 선택적으로 사용할 수 있는 저장소 인터페이스를 제공한다. 작업 상태와 로그는 `job_id`를 키로 저장하며, `status`, `steps`, `started_at`, `ended_at` 등의 필드를 포함한다. 초기 구현은 메모리 저장을 사용하되, 옵션으로 Redis를 설정할 수 있게 한다.

3. **SSE 스트리밍** – `/api/tasks/{job_id}/stream` 엔드포인트는 FastAPI의 `StreamingResponse`를 사용하여 `text/event-stream` 형식으로 상태 업데이트를 지속적으로 전송한다. 프런트엔드는 `EventSource` 객체로 구독하며, SSE를 지원하지 않는 브라우저는 `/api/tasks/{job_id}`를 일정 간격으로 폴링하도록 한다.
4. **작업 취소 기능** – `/api/tasks/{job_id}/cancel` 엔드포인트를 구현하여 하위 프로세스를 종료하고 상태를 `canceled`로 업데이트한다. 작업 도중 예외가 발생한 경우 `status`를 `error`로 저장하고 SSE 스트림을 종료한다.
5. **최근 작업 목록** – `/api/tasks/recent`는 최근 N개의 작업 요약을 반환한다. UI에서는 "최근 작업" 버튼을 추가하여 모달에 목록을 표시한다.

2.2 UI/UX 및 관리 기능

1. **진행률 모달 강화** – 각 단계에 아이콘(완료, ⏳ 실행중, 실패)을 표시하고 실행 시간을 밀리초 단위로 보여준다. 타임라인을 Canvas 혹은 SVG로 그려 시각적 흐름을 제시한다.
2. **게이트 임계값 슬라이더** – 승인 기준(예: 승인 ≥ 15)을 UI 슬라이더로 조정할 수 있게 하고, 변경 시 `/api/config/gate_required`(PATCH)로 서버 설정을 업데이트한다.
3. **KPI 자동 새로고침** – 30초 간격으로 `/api/status`를 호출하여 선정본 총량, 승인 수, 커뮤니티 후보, 키워드 특집 여부 등을 갱신한다.
4. **접근성 및 테마 개선** – 라이트·다크 테마를 하나의 HTML로 통합하고 CSS 변수로 색상을 정의한다. WCAG 2.1 대비기준을 만족하는 색상 조합을 선택한다.
5. **보고서·내보내기 UI** – 보고서 생성(`/api/report`), 카드 요약·번역(`/api/enrich/keyword`, `/api/enrich/selection`), 마크다운/CSV 내보내기(`/api/export/md`, `/api/export/csv`) 버튼을 UI에 배치한다. 성공 시 다운로드 링크를 표시한다.
6. **다국어 지원** – UI 텍스트를 JSON 파일로 분리하고 Locale에 따른 날짜/숫자 포맷을 지원하는 구조를 설계한다.
7. **로그 뷰어 및 검색** – 작업 로그를 보기 위한 별도 뷰어를 제공하고, 키워드 검색을 지원하여 에러 분석과 디버깅을 용이하게 한다.

2.3 데이터 파이프라인 개선

1. **공식·커뮤니티 데이터 병합 향상** – 일간 발행 시 `selected_articles.json`(공식)과 `archive/selected_community.json`(커뮤니티)를 병합하는 로직을 개선하여 중복 기사나 형식 불일치를 제거한다. 중복 ID는 하나로 합치되, `approved` 플래그와 편집자 코멘트는 보존한다.
2. **키워드 특집과 일간 발행 분리** – 특집 수집·승인·발행은 별도의 플로우로 유지하되, 특집 콘텐츠를 일간 뉴스에 부록 형식으로 포함할 수 있는 옵션을 제공한다.
3. **자동 승인 규칙 설정** – 커뮤니티 자동 승인 Top-N 기준(현재 20)을 설정 파일로 관리할 수 있게 하여 상황에 따라 조정한다.
4. **원본 소스 추적성** – 기사별로 원본 소스 URL, 수집 시간, 점수(품질/팩트/도메인) 등을 메타데이터로 포함시켜 검증과 회귀 테스트 시 활용한다.

2.4 테스트, 모니터링 및 문서화

1. **통합 및 회귀 테스트** – Pytest를 사용하여 신규 API의 정상 동작과 오류 처리, 작업 취소, SSE 스트리밍을 검증한다. Starlette TestClient로 SSE 구독을 시뮬레이션한다.
2. **회귀 테스트 자동화** – `tests/d3_regression_check.py`를 CI 파이프라인에 통합하여 도메인·키워드 규칙이 변경되어도 결과가 기대 범위에 있는지 확인한다.
3. **문서 업데이트** – README와 관리 가이드를 업데이트하여 비동기 실행 방법, 폴백 로직, 새 API, UI 기능, 게이트 조정 방법을 설명한다.
4. **모니터링 도구 도입** – Prometheus/Grafana를 도입해 API 응답 시간, 작업 실패율, 승인률, 큐 길이 등을 시각화하고 알림을 설정한다.

5. **로그 구조화** – Python `logging` 모듈로 JSON 형식 로그를 작성하여 분석이 쉽도록 한다. 심각도별 로그 수준을 구분하고, 필요한 경우 파일 회전(log rotation)을 설정한다.

2.5 보안 및 배포

1. **인증과 권한 관리** – FastAPI의 `Depends`를 활용하여 모든 API에 토큰 기반 인증을 적용한다. 관리자와 일반 사용자/guest 권한을 구분하고, 승인·발행 권한은 관리자에게만 허용한다.
2. **HTTPS 적용** – 프런트엔드와 백엔드 사이를 HTTPS로 통신하도록 설정하고, 개발/스테이징/운영 환경별 인증서를 구성한다.
3. **CORS 정책 설정** – API 호출 도메인을 제한하고, 적절한 `Access-Control-Allow-Origin` 헤더를 설정해 악의적 호출을 방지한다.
4. **비밀정보 보호** – API 키, 토큰, DB 접속정보 등은 `.env` 파일 또는 안전한 비밀 저장소에 저장하고 코드 저장소에 노출하지 않는다.
5. **스케일링 전략** – 비동기 작업이 늘어나면 Celery나 RQ 같은 작업 큐를 도입하고, Redis나 PostgreSQL 등 외부 저장소를 연결해 확장성을 확보한다. 컨테이너 기반 배포(k8s)로 오케스트레이션을 고려한다.

3. 고도화 로드맵 (예상 4주 계획)

주차	주요 작업	산출물
1주차	• TaskManager 설계 및 `/api/tasks/flow/*`, `/api/tasks/{job_id}` 구현 • `config.json`에 게이트 설정 API 추가 • UI 슬라이더와 게이트 API 연동 시작	초기 비동기 엔드포인트, 업데이트된 config API, 기본 TaskManager 모듈
2주차	• SSE 스트리밍 구현 및 `/api/tasks/{job_id}/stream` 추가 • UI의 EventSource 통합, 진행률 모달 강화, KPI 자동 새로고침 • 커뮤니티 자동 승인 기준을 설정값으로 노출	SSE 스트림 동작 확인, 개선된 모달 UI, 자동 새로고침 기능
3주차	• 보고서·요약·내보내기 API 구현 및 UI 연동 • 최근 작업 목록 기능 추가 • 테마 통합 및 접근성 개선	새로운 API 엔드포인트와 UI 버튼, WCAG 준수 테마
4주차	• Pytest 기반 통합·회귀 테스트 작성 및 CI 통합 • README/운영 가이드 업데이트 • 버그 수정 및 코드 정리	테스트 스위트, 업데이트된 문서, 안정화된 코드

4. 마무리 및 장기적 비전

비동기 처리와 사용자 경험 개선은 퀄리저널 시스템의 핵심 성공 요인이다. 위 계획을 따라 순차적으로 기능을 구현하고 문서화와 테스트를 병행하면 향후 고도화 단계와 마이크로서비스 분리에 대비할 수 있다. 장기적으로는 다음을 검토한다:

- **모듈 통합 또는 미세 서비스화** – 현재 두 수집기(`engine_core.py`, `orchestrator.py`)를 하나의 모듈로 통합하거나, 반대로 각 수집기를 독립 서비스로 분리하여 서버를 경량화할 수 있다.
- **AI 기반 요약/번역 자동화** – ChatGPT API나 오픈소스 모델을 활용하여 기사 요약과 번역 작업을 자동화하고, 인간 검수 과정을 줄인다.
- **플러그인 아키텍처** – 수집기나 평가·요약 모듈을 플러그인 방식으로 개발하여 새로운 소스를 손쉽게 추가하고 제거할 수 있게 한다.
- **모바일 친화 UI** – 관리자용 UI를 반응형으로 개선하여 태블릿과 휴대폰에서도 운영할 수 있게 한다.

이 문서를 기반으로 팀과 협의하며 일정과 우선순위를 조정하길 권한다. 꾸준한 테스트와 문서화가 품질을 유지하는 데 중요하며, 보안과 확장성을 염두에 두고 개발을 진행해야 한다.
