ChatGPT

QualiJournal 주요 파일 개선 및 패치 설계

이 문서는 **"QualiJournal Workflow 개선 및 고도화 계획"**에 근거하여 현재 코드 베이스의 주요 파일(`server_quali.py`, `index.html`, `orchestrator.py`, `engine_core.py`)을 어떻게 개선하고 패치할지 설계한다. 실제 구현은 단계적으로 진행될 수 있으며, 여기서는 필요한 변경 사항을 모듈별로 요약한다.

1. `server_quali.py` – 관리자 API 개선

1.1 비동기 작업 API 구현

중기 계획서에서 제안한 **비동기 작업 시스템**을 지원하려면 `server_quali.py`에 다음과 같은 엔드포인트와 구조를 추가해야 한다.

1. **TaskManager 클래스** – 메모리 또는 Redis 등에 작업 상태를 저장하는 추상화 계층을 구현한다. 작업 실행 시 생성된 `job_id`와 함께 상태(`status`, `steps`, `started_at`, `ended_at`)를 저장하고 조회할 수 있어야 한다.
2. `/api/tasks/flow/{kind}` **(POST)** – `asyncio.create_subprocess_exec`를 사용하여 `orchestrator.py` 또는 `engine_core.py`의 플로우를 백그라운드에서 실행한다. 입력으로 `kind`(`daily`, `community`, `keyword`)와 파라미터를 받아 `job_id`를 발급하고 바로 반환한다.
3. `/api/tasks/{job_id}` **(GET)** – 지정된 작업의 현재 상태와 로그를 JSON으로 반환한다. TaskManager에서 상태 정보를 읽어와 응답한다.
4. `/api/tasks/{job_id}/stream` **(GET)** – SSE(Stream-Sent Events)를 사용해 작업 상태를 실시간으로 스트리밍한다. FastAPI의 `StreamingResponse`를 이용하여 상태 업데이트를 전송하고, 작업 완료나 오류 발생 시 스트림을 종료한다.
5. `/api/tasks/{job_id}/cancel` **(POST)** – 실행 중인 하위 프로세스를 종료하고 작업 상태를 `canceled`로 변경한다.
6. `/api/tasks/recent` **(GET)** – 최근 N개의 작업 요약을 반환하여 UI에서 최근 작업 목록을 표시할 수 있게 한다.

1.2 동기 플로우 개선

현재 `/api/flow/merged` 엔드포인트는 공식 수집 → 커뮤니티 수집 → 자동 승인 → 병합 발행을 동기적으로 수행한다. 개선 계획에 따라 다음 기능을 추가한다:

- **게이트 임계값 API** – 승인 기준(예: 승인 ≥ 15)을 조정할 수 있도록 `/api/config/gate_required` (GET/PATCH) 엔드포인트를 추가한다. PATCH 요청 시 `config.json`을 업데이트하고 UI에 새로운 값을 반환한다.
- **보고서·요약·내보내기 API** – `/api/report`, `/api/enrich/keyword`, `/api/enrich/selection`, `/api/export/md`, `/api/export/csv`를 구현하여 각각 `tools/make_daily_report.py`, `tools/enrich_cards.py`를 호출하거나 현재 발행본을 마크다운/CSV로 내보낸다.
- **에러 및 로그 처리 개선** – 모든 하위 프로세스 호출을 `try/except`로 감싸 예외 발생 시 상태를 `error`로 저장하고 로그를 TaskManager에 기록한다.

2. `index.html` – 관리 UI 개선

2.1 비동기 UI 지원

UI는 `/api/tasks/recent`를 호출해 비동기 API 존재 여부를 확인하고, 존재할 경우 비동기 엔드포인트로 요청해야 한다. 이를 위해 다음 사항을 반영한다:

1. **SSE 스트림 구독** – 작업 시작 시 `/api/tasks/flow/{kind}`로 요청하여 받은 `job_id`로 `EventSource`를 생성하고, `/api/tasks/{job_id}/stream`을 구독해 단계별 상태를 실시간으로 갱신한다. SSE를 지원하지 않는 브라우저에서는 0.5초 간격으로 `/api/tasks/{job_id}`를 폴링한다.
2. **모달 UI 향상** – 모달 내부에서 각 단계에 아이콘과 실행 시간(ms)을 표시하고 타임라인을 Canvas/SVG로 그린다. 또한 '취소' 버튼을 활성화해 `/api/tasks/{job_id}/cancel`을 호출하도록 한다.
3. **게이트 슬라이더 및 KPI 새로고침** – 승인 임계값을 조정할 수 있는 슬라이더를 제공하고, 변경 시 `/api/config/gate_required`(PATCH)를 호출한다. 또한 `/api/status`를 30초 간격으로 요청해 KPI를 자동 갱신한다.
4. **테마 및 접근성** – 라이트/다크 테마를 하나의 HTML로 통합하고 CSS 변수를 사용해 색상과 폰트를 정의한다. WCAG 2.1 대비기준을 만족하는 색상 조합을 적용한다.
5. **보고서·내보내기 UI** – 보고서 생성, 카드 요약·번역, 내보내기 버튼을 추가하고 결과 링크를 표시한다.

2.2 최근 작업 목록 및 기타 기능

UI에 최근 작업 목록 버튼을 추가하여 `/api/tasks/recent` 결과를 모달 또는 로그 영역에 표시한다. 또한 작업 로그 검색 기능과 키워드 필터링을 제공해 디버깅을 용이하게 한다.

3. `orchestrator.py` – 비동기 플로우 지원과 병합 개선

1. **비동기 호출 대응** – TaskManager와 연동되도록 `orchestrator.py`를 조정한다. `--collect`, `--collect-community`, `--approve-top`, `--publish` 명령이 실행될 때 표준 출력에 JSON 형태의 단계별 로그를 출력하거나 파일로 저장하도록 한다. 이 로그는 `/api/tasks/{job_id}`에서 노출될 수 있다.
2. **선택본 병합 로직 개선** – `--publish` 단계에서 `selected_articles.json`(공식)과 `archive/selected_community.json`(커뮤니티)를 병합할 때, 중복 기사 제거와 기존 승인 상태 보존을 강화한다. 현재 `server_quali._sync_after_save()`에서 구현한 병합 로직을 참조하여 orchestrator 내에도 일관된 병합 로직을 둘 수 있다.
3. **자동 승인 정책 파라미터화** – 커뮤니티 자동 승인 Top-N 값(기본 20)을 명령줄 인자로 받을 수 있도록 `--approve-top` 플래그를 확장한다. 이는 `/api/config`를 통해 UI에서 조정할 수 있다.
4. **테스트 후크** – 각 단계마다 콜백을 받아 상태를 외부로 알리거나 테스트에서 주입할 수 있게 설계한다. 이를 통해 Pytest에서 세부 단계를 검증하기 쉽다.

4. `engine_core.py` – 장기적 고도화 제안

`engine_core.py`는 공식 소스 수집과 품질 평가(QG/FC) 파이프라인을 수행하는 핵심 엔진이다. 현재 구조는 모놀리식이므로 향후 다음과 같은 개선을 고려한다:

1. **모듈화 및 플러그인 구조** – 소스 수집, 콘텐츠 정규화, 품질 평가, 요약/번역, 발행 모듈을 플러그인 형태로 분리한다. 이는 새 소스 추가나 알고리즘 교체를 용이하게 한다.
2. **비동기 지원** – 수집과 평가 과정에서 CPU 또는 I/O 바운드 작업이 많으므로, `async`/`await`와 `aiohttp`를 적용해 동시에 여러 기사를 처리할 수 있게 한다. 이는 비동기 엔드포인트와 자연스럽게 연동된다.

3. **AI 요약·번역 통합** – 향후 AI 기반 요약/번역 기능을 내장해 번역 서버 의존을 줄인다. OpenAI API나 국내 대체 모델을 사용하여 실험하고, 성능과 비용을 비교한 후 선택한다.
4. **테스트 커버리지 확대** – 기사 수집·품질 평가·발행 과정에 대한 단위 테스트와 회귀 테스트를 추가해 변경 시 안정성을 확보한다.

5. 보안 및 배포 고려 사항

1. **토큰 기반 인증** – FastAPI의 `Depends`를 사용하여 모든 관리 API에 토큰 기반 인증을 적용한다. 관리자/편집자 권한을 구분하고, 승인·발행 권한은 관리자에게만 부여한다.
2. **HTTPS 적용** – 서비스는 기본적으로 HTTPS로 제공하고, 개발 및 스테이징 환경에 인증서를 적용한다. CORS 정책을 필요 최소한으로 설정한다.
3. **작업 저장소 외부화** – 메모리 기반 TaskManager는 서버 재시작 시 기록이 사라지므로, Redis나 PostgreSQL 같은 외부 저장소에 작업 상태와 로그를 저장하도록 설계한다.
4. **모니터링 및 알림** – Prometheus/Grafana를 배포하여 API 응답 시간, 작업 실패율, 승인률 등을 모니터링하고 알림을 설정한다.

6. 결론

이 설계는 중기 실행계획서에서 제시한 비동기 워크플로우, SSE 스트리밍, UI 개선, 보고서·내보내기 기능, 테스트 및 보안 요구를 반영한다. 각 파일의 구조를 확장하는 방향으로 패치를 설계하여, 향후 기능 추가와 유지보수를 용이하게 한다. 구현 시에는 단계별로 진행하되, 테스트와 문서화를 병행하여 품질을 확보해야 한다.
