QualiJournal Admin 고도화 전략

개요

현재의 관리자 기능은 FastAPI 기반 백엔드(server_quali.py)와 라이트/다크 테마 토글을 포함한 단일 프런트엔드(admin/index_lite.html), 그리고 간단한 사용 지침(README.txt)으로 구성되어 있다. 이 구조는 기본적인 기능을 제공하지만, 코드의 유지보수성과 사용성 측면에서 여러 개선점을 발견할 수 있다. 여기서는 **접근성·표준 준수**, **API 설계**, **사용자 경험(UX)** 세 가지 관점에서 문제를 분석하고 단계별 고도화 방안을 제안한다. 각 제안은 공인 표준 문서에서 제시하는 근거를 바탕으로 한다.

1. 접근성과 UI 디자인 개선

1.1 명암 대비 및 다크 모드 설계

현재 UI 는 라이트·다크 테마를 토글하는 버튼을 제공하지만, 색상 대비 비율에 대한 명시적 검증이 없고 두 테마가 모두 접근성 표준을 충족하는지 보장하지 않는다. **W3C 웹 콘텐츠 접근성 가이드라인(WCAG)**에 따르면 일반 텍스트와 배경의 **명암 대비는 최소 4.5:1** 이어야 하며, 큰 글자(약 18pt 이상)에는 3:1 비율을 적용할 수 있다【945726454056881†L40-L46】. 또한 명암 대비는 색상 자체가 아닌 빛과 어둠의 차이에 기반해야 하며, 색각 이상 사용자도 읽을 수 있어야 한다【945726454056881†L61-L74】. UI 를 개선할 때 다음 사항을 고려한다:

- 테마 색상표를 WCAG 2.1 의 4.5:1 명암 대비 기준에 맞춰 조정한다. 예를 들어 라이트 모드의 글자 색은 더 짙은 회색, 다크 모드의 배경은 더 어두운 색으로 설정할 수 있다.
- 버튼과 입력창에도 동일한 대비 규칙을 적용한다. 비활성화된 요소나 참고 문구는 시각적 위계는 유지하되 최소 3:1 이상의 대비를 확보한다.
- 테마 토글 버튼에는 명확한 레이블(예: “라이트 모드”, “다크 모드”)과 aria-label 속성을 추가하여 스크린 리더 사용자도 기능을 이해할 수 있게 한다.
- 이용자의 운영체제 설정(prefers-color-scheme)을 감지해 기본 테마를 자동 선택하고, 사용자가 토글을 사용하면 로컬 저장소에 테마를 저장하여 재방문 시 적용한다.

1.2 양식 접근성과 국제화

- 날짜·키워드 입력 필드에 <label> 요소를 추가하고 for 속성으로 연결하여 스크린 리더가 필드의 목적을 읽을 수 있도록 한다.
- 모든 버튼에 type="button"을 명시해 의도하지 않은 폼 제출을 방지한다.
- HTML 의 lang 속성을 ko 로 유지하되 국제화(i18n)를 고려해 UI 문자열을 별도 JSON 또는 속성으로 분리한다. 추후 영어 인터페이스가 필요할 경우 번역 파일만 교체하면 된다.

2. API 설계와 백엔드 개선

2.1 HTTP 메서드 의미 준수

server_quali.py 는 커뮤니티 목록 조회를 GET 으로, 커뮤니티 저장을 POST 로 구현하여 기본적인 분리는 돼 있으나, 추가적인 엔드포인트 도입 시에도 **HTTP 메서드 의미를 지켜야 한다**. IETF RFC 7231 에서 정의한 것처럼 GET 등 **안전한(safe) 메서드**는 서버 상태를 변경하지 않아야 하며 【887855184796408†L1310-L1315】, 클라이언트가 상태 변화를 기대하지 않는다. 반대로 POST 는 리소스의 상태를 변경하는 작업에 사용해야 한다. 이를 반영해 다음을 개선한다:

- /api/status 와 같이 데이터를 조회하는 엔드포인트는 GET 으로 유지하고, date, keyword 등 필터링 파라미터를 쿼리 문자열로 전달한다.
- 데이터 수정이나 명령 실행(/api/flow/daily, /api/flow/keyword, /api/community/save)은 POST 로 명확히 구분한다. 앞으로 추가할 엔드포인트(리포트 생성, 요약·번역 등)도 동작 특성에 따라 POST 또는 PUT 을 사용한다.
- RFC 7231 는 **안전한 메서드와 위험한 메서드를 사용자에게 구분해 보여줄 것을 권고**한다 【887855184796408†L1346-L1348】. 프런트엔드는 버튼 그룹을 시각적으로 구분하거나 확인 대화상자를 추가해 사용자에게 변경 작업임을 명확히 알린다.

2.2 경로 네이밍과 버저닝

RESTful API 의 기본 원칙은 **엔드포인트를 리소스(명사) 중심으로 설계**하고, 해당 리소스의 상태를 조작하는 데 HTTP 메서드를 사용하는 것이다. 기존의 /api/flow/daily 등은 동사를 포함하므로 /api/flow/daily 대신 /api/runs/daily 처럼 **명사형 리소스**로 표현하거나, /api/tasks/daily 등으로 변경한다. 또한 장기적으로 API 를 확장할 때는 /api/v1/... 형태의 **버전 관리를 도입**해 호환성을 유지할 수 있다.

2.3 비동기 처리와 오류 관리

현재는 subprocess.run()을 통해 오케스트레이터 스크립트를 동기적으로 호출한다. API 요청 시 외부 프로세스가 완료될 때까지 기다리므로 프런트엔드가 응답 지연을 경험한다. FastAPI 의 async 기능과 asyncio.create_subprocess_exec 를 이용해 **비동기 실행**으로 전환하면 응답을 즉시 반환하고, 작업 완료 여부는 폴링 또는 WebSocket 으로 확인할 수 있다. 또한 오류 발생 시 HTTP 상태 코드와 메시지를 명확히 전달해 클라이언트가 적절한 피드백을 제공하도록 한다.

2.4 기능 확장 계획

README_D3 에서 소개된 점수 계산·섹션 분류·일일 리포트 기능을 백엔드에 통합하려면 아래와 같은 엔드포인트를 설계한다:

- /api/report: 날짜와 키워드를 받아 tools/make_daily_report.py 를 실행해 MD·HTML·JSON 리포트를 생성하고 결과 경로를 반환한다.
- /api/enrich/keyword 및 /api/enrich/selection: 번역·요약 모듈을 호출해 키워드 기사나 선택본에 부가 정보를 추가한다.
- /api/export/md, /api/export/csv: 선택본을 마크다운 또는 CSV 형식으로 변환해 사용자 다운로드를 돕는다.

각 엔드포인트는 적절한 HTTP 메서드(POST)와 구조화된 입력 스키마를 사용하며, 처리 중에는 비동기적으로 작업을 수행한다.

3. 사용성 향상과 도구 통합

3.1 단일 UI 파일과 테마 스위치

기존에는 index_lite.html 과 index_lite_black.html 두 파일이 있었지만, 유지보수를 위해 하나의 HTML 파일에 라이트/다크 모드를 토글하는 방식으로 통합하였다. 앞으로도 **UI 코드 중복을 최소화**하고 스타일 시트에서 변수를 통해 색상과 레이아웃을 제어하도록 한다. Tailwind CSS 같은 유틸리티 프레임워크나 CSS 변수를 활용하면 테마 교체가 간단해진다.

3.2 로그와 상태 모니터링

현재 UI 는 작업 실행 결과를 JSON 형식으로 그대로 보여준다. 사용자 친화성을 위해 다음을 개선한다:

- 실행 단계별 성공/실패 여부를 타임라인 형태로 표시하고, 실패한 단계는 적색으로 강조한다.
- 다운로드 가능한 로그 파일 링크를 제공하여 문제 분석을 돕는다.
- refresh() 호출을 일정 주기로 자동 실행하여 KPI 값을 실시간으로 업데이트한다.

3.3 권한과 인증

공개 환경에 배포할 경우, API 엔드포인트를 적절히 보호해야 한다. 최소한의 조치로는 **토큰 기반 인증**을 도입하여 승인된 사용자만 관리자 기능에 접근할 수 있도록 하고, HTTPS 를 사용해 통신을 암호화한다. 향후 확장을 고려해 OAuth2 또는 JWT 기반 인증 방식을 사용할 수 있다.

결론 및 향후 로드맵

위 전략은 관리자 도구를 **접근성 준수**, **표준 기반 API 설계**, **사용성 향상** 측면에서 고도화하는 방안이다. W3C 의 명암 대비 기준에 따라 테마 색상을 조정하고【945726454056881†L40-L46】, RFC 7231 에서 정의한 안전한 HTTP 메서드의 의미를 준수하여 엔드포인트를 설계하는 것이 핵심이다【887855184796408†L1310-L1315】. 또한 비동기 처리, 명사형 경로, 버전 관리를 도입해 백엔드의 안정성과 확장성을 높일 수 있다. UI 측면에서는 하나의 파일로 테마를 전환하고 작업 로그를 시각적으로 표현하는 등 사용자 경험을 개선할 수 있다. 이러한 단계적 개선을 통해 QualiJournal 관리자 시스템은 보다 견고하고 사용하기 쉬운 도구로 발전할 것이다.