ChatGPT

QualiJournal 시스템 진단 보고서 (2025-10-06 기준)

번호	점검 항목	진단
1	선정본 구조 및 승인 게이트 검증	**PASS**
2	config.json과 config_schema_definition.json 스키마 일치 여부	**FAIL**
3	community_sources.json 및 official_sources.json 설정 (구조, 임계값, 죽은 링크 등)	**WARN**
4	orchestrator.py와 run_quali_today.ps1 워크플로우 (단계 흐름 불일치, 인코딩 오류, PS 의존성)	**FAIL**
5	tools/ 이하 스크립트들의 기능 중복/충돌 및 역할 적정성	**WARN**
6	admin/index.html과 server_quali.py 연계 (경로 불일치, 폴백 부족, 인코딩 지정 누락 등)	**FAIL**
7	selected_keyword_articles.json 및 selected_articles.json 데이터 무결성 (구조, 중복, 누락, 승인 플래그)	**WARN**
8	archive/ 발행본 파일명 규칙 및 다중 발행 충돌 예방	**WARN**
9	requirements.txt 패키지 의존성 누락/충돌 가능성	**FAIL**
10	전반적인 경로 지정, 파일명 규칙 일관성, 모듈 호출 체계	**WARN**

1. 선정본 구조 및 승인 게이트 검증

진단 결과: PASS – 편집자 승인 품질게이트(QG)가 정상적으로 동작합니다. 발행 시 **최소 15개의 승인된 기사**가 요구되며, 해당 기준은 설정 옵션 `require_editor_approval`을 통해 강제됩니다 [1]. 최신 패치를 통해 config.json에 `"gate_required": 15` 필드가 추가되어 품질게이트 기준을 설정 파일로 관리할 수 있게 되었으며 [2], 응급 발행 시에도 이 값을 일시 조정했다가 복원하는 절차가 문서화되었습니다. (예: 승인 기사 부족 시 `require_editor_approval`을 False로 변경해 일시적으로 발행한 후 즉시 원복 [3].) 전반적으로 선정본(selected_articles.json) 구조도 표준 형식을 준수하여 최종 발행에 활용되고 있습니다 [4].

2. config.json ↔ config_schema_definition.json 스키마 일치 여부

진단 결과: FAIL

문제점 요약: 설정 파일인 `config.json`의 내용과 이를 검증하는 `config_schema_definition.json` 간에 불일치가 발생하고 있습니다. 최신 패치에서 config.json에 **새로운 필드**(예: 품질게이트 임계값 `gate_required`)가 추가되었지만 [2], 스키마 정의가 이에 맞게 업데이트되지 않아 구조 불일치 오류가 존재합니다. 그 결과, 스키마 검증을 통과하지 못하거나 설정값이 무시되는 문제가 발생할 수 있습니다.

상세 분석: QualiJournal 시스템의 config.json은 **품질 기준(QG)**, 팩트체크(FC) 기준, 스코어 가중치, 섹션 순서 등 여러 설정을 담고 있으며 [5], 각각의 키와 자료형이 스키마 파일에 정의되어 있습니다. 그러나 현재 스키마 정의가 오래되어 config.json의 실제 필드 구성을 반영하지 못하고 있습니다. 예를 들어, 패치로 도입된 `gate_required` 필드는 스키마에 정의되지 않아 검증 시 오류가 발생하거나 무시될 가능성이 있습니다. 또한 config.json의 일부 키 혹은 구

조(예: `features` 내 옵션들)도 스키마와 이름이나 계층이 어긋나 있을 수 있습니다. 이러한 스키마-설정 불일치는 **Pydantic** 기반 모델 검증이나 JSON 스키마 검증 과정에서 문제를 야기하고, 잘못된 설정이 runtime에 발견되는 원인이 됩니다. 즉, 개발자가 의도한 설정값이 적용되지 않거나, 초기 로드 시 예외가 발생할 수 있습니다.

개선 방안: `config_schema_definition.json`을 **최신 config.json에 맞게 갱신**해야 합니다. 추가된 `gate_required` 키를 스키마에 반영하고 자료형(정수)과 허용 범위를 명시합니다. 기존 필드들도 이름, 타입이 정확히 일치하도록 조정해야 합니다. 패치 적용 시 변경된 설정 구조 (예: 새로운 features 옵션이나 섹션 설정) 역시 모두 스키마에 포함시킵니다. 스키마 업데이트 후에는 config.json을 해당 스키마로 **검증하는 절차**를 CI나 서버 기동 시에 포함하여, 불일치가 재발하지 않도록 합니다. 또한 Pydantic 모델을 사용하는 경우, config를 매핑하는 BaseModel을 스키마와 함께 업데이트하여 **런타임 검증**을 강화합니다. 최종적으로, 스키마와 설정 파일 간 변경이 있을 때 **동시에 유지관리**하도록 개발 프로세스를 마련해야 합니다 (예: 설정 변경 시 스키마 검증 실패를 알려주는 테스트 추가).

3. community_sources.json 및 official_sources.json 설정 진단

진단 결과: WARN

문제점 요약: 커뮤니티 소스 필터 설정과 공식 소스 목록의 관리 측면에서 일부 개선이 필요합니다. 현재 community_sources.json에 정의된 **키워드 포함 여부, 제목 길이 제한, 차단 패턴, 최소 추천수/댓글수, 점수 임계값** 등이 지나치게 엄격하게 설정된 경우 커뮤니티 수집 결과가 "0건"이 되는 문제가 지적되었습니다 [6] [7]. 또한 official_sources.json에 등록된 **RSS 피드 URL 중 일부가 변경**되었으나 파일에 업데이트되지 않아 404 오류를 유발할 가능성이 있습니다 [6]. 이러한 요인으로 커뮤니티/공식 소스 데이터의 **품질과 완전성**에 위험이 존재합니다.

상세 분석: community_sources.json에는 Reddit 등 커뮤니티 수집을 위한 세부 조건이 포함되어 있는데, 현행 설정으로는 잡음 제거를 위해 **필터 기준이 매우 높게** 설정되어 있어 유의미한 글까지 걸러내는 경우가 있었습니다 [8]. 실제로 과거에는 Reddit/포럼 모듈이 제대로 작동하지 않아 커뮤니티 기사 수가 0건이 되는 문제가 보고되었고, 이는 필터 임계값의 과도함과 크롤러 미비가 원인입니다 [7]. 한편, official_sources.json의 RSS 링크들은 운영자가 수동으로 유지해야 하는데, **일부 공식 뉴스 소스의 URL 변경**(예: 사이트 개편으로 피드 주소 변화)으로 인해 더 이상 수집되지 않는 사례가 발생했습니다 [6]. 이러한 죽은 링크는 수집 단계에서 예외를 일으키거나 해당 출처 기사가 전무하게 만들며, 시스템의 **커버리지 누락**으로 이어집니다. 또한 임계값 설정 문제로 양질의 커뮤니티 글도 제외되면 전체 기사 풀이 감소하여 매일 15개 이상 선정이 어려워질 수 있습니다.

개선 방안: 필터 임계값을 재조정하여 너무 엄격하거나 느슨하지 않도록 튜닝해야 합니다 [8]. 실제 수집된 커뮤니티 글 통계를 기반으로 최소 추천/댓글 수 조건 등을 완화하고, 금지 키워드/도메인 목록도 주기적으로 검토해 유용한 글이 걸러지지 않도록 합니다. 커뮤니티 수집기가 현재 **0건 문제**를 완전히 해소하도록 Reddit API 인증과 rate limit 처리, 포럼 HTML 파싱 보완 등 기술적 개선도 병행해야 합니다 [7]. 공식 소스의 경우 **정기적인 링크 점검**을 수행해 404나 피드 구조 변경에 대응해야 합니다 [6]. 예를 들어, 일정 주기마다 각 RSS URL에 HTTP 요청을 보내 상태를 확인하고, 실패 시 해당 소스를 업데이트하거나 관리자에게 알리는 프로세스를 갖춥니다. 공식_sources.json 파일을 변경할 때는 기록을 남기고 필요한 경우 대체 소스를 추가해 콘텐츠 부족을 예방합니다. 마지막으로, community/official 소스 설정 변경 시 쉽게 반영될 수 있도록 **관리 UI**나 설정 파일 가이드라인을 제공하여, 운영자가 실시간으로 품질 기준을 조절하고 죽은 링크를 교체할 수 있게 합니다.

4. orchestrator.py ↔ run_quali_today.ps1 워크플로우

진단 결과: FAIL

문제점 요약: Python 오케스트레이션 모듈과 PowerShell 자동화 스크립트 간의 **단계 흐름 불일치**와 Windows 환경 종속성이 심각한 문제입니다. 현재 **일일 발행 프로세스**는 Windows PowerShell 스크립트(`run_quali_today.ps1`)를 통해 orchestrator.py 및 각종 도구 스크립트를 순차 실행하는 방식으로 구현되어 있는데 [9], 이로 인해 **플랫폼 의존성**이 높고 자동화 흐름이 이중으로 관리되고 있습니다 [10]. 또한 Flask/FastAPI 서버

에서 Python subprocess로 orchestrator를 호출할 때 발생하는 **인코딩 오류**(cp949 디코딩 문제)로 UnicodeDecodeError 예외가 발생, 작업이 중단되는 치명적인 버그가 존재했습니다 [11].

상세 분석: 개선된 웹 UI는 비동기 작업 실행을 기대하지만, 현재 백엔드는 여전히 PowerShell 스크립트 기반으로 **동기식 흐름**을 사용하고 있습니다. 이로 인해 UI에서는 /api/tasks/flow/daily 등의 경로를 호출해도 서버에 해당 **비동기 엔드포인트가 없어 404 오류**가 발생했습니다 [12]. 즉, orchestrator.py 자체는 개별 단계(`--collect`, `--publish` 등) 실행만 지원하고, 전체 흐름 제어는 PS 스크립트가 담당하는 구조입니다. 이러한 이원화로 **Orchestrator와 PS 간 단계 불일치**가 발생했는데, 예를 들어 orchestrator.py는 내장된 워크플로우 없이 일부 작업만 수행하고, 나머지 보충/동기화 단계는 별도 Python 스크립트로 분리되어 PS가 호출합니다. 이는 동일 로직이 분산되어 **관리 복잡성**을 높이고, Windows 이외 환경에서는 일정 관리가 불가능한 한계입니다 [10].

인코딩 오류의 원인은 서버에서 `subprocess.run(..., text=True)`로 orchestrator를 실행할 때, **Windows 기본 로캘(CP949)**을 사용해 서브프로세스 출력을 디코딩하면서 ANSI 색상코드나 UTF-8 문자가 제대로 해석되지 못한 데 있습니다 [11]. 그 결과 orchestrator 출력에 포함된 특수 ANSI 코드나 한글이 디코딩 오류를 일으켜 `UnicodeDecodeError`가 발생, 프로세스가 중단되었습니다. 이 문제는 PowerShell 콘솔에서 컬러 출력 등의 잔재를 파이프라인으로 받을 때 특히 촉발되었습니다. 요약하면, **비동기 API 미구현**, **PowerShell 종속**, **로캘 인코딩 버그**가 결합되어 일일 발행 워크플로우의 안정성을 저해하고 있었습니다.

개선 방안: 단기적으로 **인코딩 오류**는 이미 패치되어 모든 subprocess 호출에 `encoding="utf-8", errors="replace"` 옵션을 지정함으로써 해결되었습니다 [13]. 이는 Windows 로캘에 무관하게 UTF-8 기반으로 출력을 파싱하도록 하여 UnicodeDecodeError를 방지합니다 [14]. 또한 서버 환경 변수 `PYTHONIOENCODING`을 UTF-8로 강제 설정하여 하위 프로세스 출력이 항상 UTF-8로 전달되도록 조치하였습니다 [15] (패치 적용됨).

근본적으로는 **PowerShell 워크플로우를 탈피**해야 합니다. 중기 개선으로 제안된 대로, 백엔드에 **비동기 작업 처리 API**를 구현하여 /api/tasks/flow/daily 등의 요청이 오면 Python에서 asyncio를 통해 orchestrator를 백그라운드 실행하고 진행 상태를 추적하는 방식으로 전환합니다 [16] [17]. 이렇게 하면 더 이상 외부 PS 스크립트에 의존하지 않아도 되며, Linux 등 이식성도 확보됩니다. orchestrator.py도 현재 수집/발행 위주 기능을 보다 세분화하여 모듈화하고, PS가 수행하던 보조 단계(재구성, 보충, 동기화 등)를 내부 함수나 API 호출로 통합하는 것이 필요합니다. 궁극적으로 **하나의 오케스트레이션 파이프라인**으로 일원화하고, UI에서는 해당 작업을 **비동기 트리거**하여 진행률을 모니터링하도록 구조를 개선해야 합니다 [12] [17].

덧붙여, FastAPI 도입에 따라 RESTful한 설계 원칙을 준수하여 API를 재구성하고 (HTTP 메서드와 자원 경로의 명확화 [18]), 작업 취소나 최근 작업 조회 등의 관리 기능도 추가하면 사용자 경험이 향상될 것입니다. 이러한 개편으로 PS 스크립트는 궁극적으로 **퇴역**시키고, 크로스플랫폼 지원과 유지보수성 향상을 달성해야 합니다.

5. tools/ 스크립트들의 기능 중복 및 역할 적정성

진단 결과: WARN

문제점 요약: tools/ 디렉토리 아래 여러 보조 스크립트들이 존재하는데, 이들 간에 **기능 중복**이나 책임 경계가 불분명한 부분이 있습니다. 예를 들어, 선정 기사 JSON을 조정하는 `sync_selected_for_publish.py`와 포맷을 교정하는 `repair_selection_files.py`가 모두 **발행 JSON 수정**을 담당하여 연속 실행해야 하는 등 한 과정으로 묶을 수 있는 기능이 분리되어 있습니다 [19] [20]. 또한 승인 기사 부족 상황을 처리하는 방식도 `augment_from_official_pool.py` (공식기사 보충)와 `force_approve_top20.py` (일괄 승인) 두 가지가 있어 운용 로직이 중복되고 있습니다. 이런 **중복 및 분산 설계**는 유지보수 어려움과 잠재적 충돌의 원인입니다 [21].

상세 분석: 현재 tools 폴더에는 수집 이후 데이터를 가공/보완하기 위한 여러 스크립트가 나뉘어 있습니다 [22] [23]. 모듈화 자체는 **단계별로 기능을 분리**하여 유지보수성을 높이는 장점이 있지만 [24], 일부 스크립트는 역할 구분이 모호합니다. 예를 들어, `repair_selection_files.py`는 selected_articles.json의 구조를 표준 딕셔너리 형태

(`{"date":..., "articles": [...]}`)로 교정하는데 [20], 사실 `sync_selected_for_publish.py`에서 승인 기사 추출 후 저장 시 이 포맷을 바로 적용하도록 하면 별도 교정 단계가 불필요합니다. 현재는 두 스크립트를 순차 실행하여 포맷 오류를 예방하고 있는데, 이는 **기능 중복**의 한 사례입니다.

또 다른 중복은 **품질게이트 우회/충족 방법**입니다. 승인된 기사가 15개 미만일 경우 `augment_from_official_pool.py`로 공식 기사 풀에서 추가하여 기사 수를 보충하고 승인 기사도 최소 15개에 이르도록 돕는데 [25], 이와 별개로 편집자가 `force_approve_top20.py`를 실행해 상위 점수 20개의 기사를 강제로 approved 처리하는 방법도 있습니다 [26]. 전자는 기사 **수**를 늘리는 접근이고 후자는 승인 **비율**을 높이는 접근으로, 목적은 같지만 다른 방법이 공존합니다. 운용상 둘 중 하나만 필요하거나 상황에 따라 선택해야 하는데, 잘못 사용하면 예컨대 이미 자동 보충된 상태에서 다시 일괄승인을 실행하여 품질이 낮은 기사까지 승인되는 등 **책임 과잉**의 위험이 있습니다.

또한 orchestrator.py 자체도 일부 collect/publish 기능을 내장하면서, 다른 보조 스크립트 호출에 의존하는 구조여서 **오케스트레이션 로직이 분산**되어 있습니다. 이로 인해 동일한 데이터 구조 접근이나 파일 I/O 코드가 여러 스크립트에 중복 구현되어 있을 가능성이 높습니다. 중복 구현은 버그 발생 시 파급을 키우고 수정 누락을 초래할 수 있습니다 (예: JSON 파일 경로나 키 이름 변경 시 모든 스크립트를 다 고쳐야 함).

개선 방안: 우선 각 툴 스크립트의 **책임 범위**를 명확히 재정의하고, 겹치는 기능은 통합하는 것이 좋습니다. `sync_selected_for_publish.py` 실행 시 자동으로 출력 포맷을 표준화하도록 개선하면 `repair_selection_files.py`를 별도로 돌릴 필요가 없게 됩니다. 만약 포맷 이상을 교정하는 추가 로직이 있다면 sync 단계에 흡수하거나, 최소한 두 기능을 하나의 명령으로 묶어 **원클릭 실행**되도록 하는 것이 바람직합니다.

승인 기사 부족 상황 처리는 **단일 루틴으로 일원화**해야 합니다. 공식기사 자동보충과 점수 상위기사 자동승인 중 정책적으로 하나를 선택하고, 가능하면 편집자가 개입하여 승인할 시간을 주는 것이 바람직합니다. 예컨대 기본은 augment_from_official_pool로 기사 수만 늘리고, 그래도 기준 미달 시 발행을 지연시키거나 편집자에게 알림을 주는 방안으로 하고, 정말 긴급한 경우에만 편집자가 `force_approve_top20.py`를 수동 실행하도록 가이드하는 등 사용 지침을 명확히 합니다. 장기적으로는 **UI 상에서 부족한 기사를 표시하고 편집자가 승인 여부를 결정**하도록 하여, 스크립트로 강제 승인하는 절차를 없애는 것이 바람직합니다.

추가로, 오케스트레이션 로직을 리팩토링하여 **중복 코드를 모듈화**합니다. 여러 스크립트에서 공통으로 수행하는 JSON 파일 입출력, 필터링 등의 기능은 공용 함수로 빼내고, orchestrator.py에서 이들을 호출하도록 구조화하면 기능 중복과 충돌을 줄일 수 있습니다. 보고서에서도 제시된 바와 같이 orchestrator를 수집/파싱/평가/발행 등 여러 모듈로 분리하고 함수형으로 재작성하는 리팩토링이 권고됩니다 [27]. 이를 구현하면 각 단계별로 단위 테스트도 가능해져, 향후 기능 변경 시 일관성을 유지하고 중복에 의한 오류를 방지할 수 있습니다.

6. admin/index.html ↔ server_quali.py 연계 문제

진단 결과: FAIL

문제점 요약: 관리자용 프론트엔드 페이지(index.html)와 백엔드 서버(server_quali.py) 간에 **API 경로 불일치**와 폴백 처리 미흡으로 인한 기능 불능 문제가 있었습니다. 최신 웹 UI는 `/api/tasks/flow/daily`와 같은 **비동기 엔드포인트** 호출을 기대하지만, 백엔드는 아직 해당 경로를 지원하지 않아 **404 오류**가 발생했습니다 [12]. 초기에는 이러한 경우 자동 폴백이 구현되어 있지 않아 사용자가 버튼을 눌러도 동작하지 않았습니다. 또한 index.html (라이트/다크 모드) 파일과 서버 코드 간 **경로 설정이 일치하지 않아** 잘못된 위치의 파일을 참조하거나, 서버를 루트가 아닌 위치에서 실행할 경우 UI 로드에 실패하는 문제가 있었습니다 [28]. 아울러 HTML의 인코딩 메타 태그나 서버의 콘텐츠타입 헤더에 **UTF-8 지정이 누락**되어 한글 등이 깨질 가능성도 지적되었습니다.

상세 분석: 기존 QualiJournal UI는 라이트/다크 두 종류의 정적 HTML을 제공했고, 해당 UI에서는 동기식 API (`/api/flow/daily` 등)를 호출하도록 되어 있었습니다 [12]. 개선된 UI는 하나의 페이지에서 다크/라이트 테마를 전

환하도록 통합하는 한편, 비동기 작업 수행을 위해 `/api/tasks/flow/...` 경로를 사용하도록 스크립트가 수정되었습니다. 그러나 server_quali.py에는 여전히 동기식 엔드포인트(`/api/flow/*`)만 존재하여 새로운 UI와 맞물리지 않았습니다. 그 결과 **프론트-백엔드 간 API 불일치**로 사용자는 진행 상황을 볼 수 없었고, 여러 번 클릭을 유발하거나 아예 기능이 동작하지 않는 상황이 벌어졌습니다 [12].

특히 `/api/tasks/flow/*` 호출 시 404가 나는 문제에 대해, 초기에 UI 쪽에 **폴백 로직이 없어서** 사용자 경험이 크게 저해되었습니다. 즉, 서버가 해당 경로를 지원하지 않을 경우 자동으로 기존 동기 엔드포인트를 쓰도록 하는 처리가 필요했으나 빠져 있었던 것입니다.

경로 불일치 문제는 정적 파일 서빙 측면에서도 나타났는데, admin 폴더 내 index.html을 서버가 제대로 찾지 못하는 케이스가 있었습니다. 서버 코드가 상대경로로 admin 폴더를 가리키는데, **프로세스 실행 위치가 달라지거나 폴더명이 변경되면** 경로가 꼬여버립니다 [29]. 실제로 서버를 프로젝트 루트에서 실행하지 않으면 데이터 파일을 찾지 못하는 위험이 보고되었고, admin 폴더 구조에 대한 문서화가 필요했습니다 [30] [28].

인코딩 지정 누락의 경우, HTML 파일에 `<meta charset="UTF-8">` 태그나 서버의 `Content-Type: text/html; charset=utf-8` 헤더 설정이 부족하면 브라우저가 한글을 잘못 해석할 수 있습니다. 다행히 현대 브라우저는 UTF-8을 기본 가정하지만, 명시하지 않으면 간헐적 이슈가 발생할 수 있습니다. 또한 JSON 응답 역시 `ensure_ascii=False` 등 UTF-8 설정이 필요합니다. 이러한 세부 사항이 초기 구현에서 간과되었을 가능성이 있습니다.

개선 방안: 최신 패치에서는 **UI 폴백 기능**이 추가되어, 서버에 비동기 `/api/tasks/*` 엔드포인트가 없을 경우 **자동으로 기존 동기** `/api/flow/*` **엔드포인트로 대체 호출**하도록 개선되었습니다 [31]. 이로써 현재 서버 버전에서도 사용자가 단일 버튼으로 작업을 실행할 수 있게 되었고 404 오류가 발생하지 않습니다 (이미 적용됨). 그러나 이는 임시 방편이므로, **중기적으로는 백엔드에 /api/tasks 엔드포인트 자체를 구현**하여 UI의 기대와 완전히 호환되도록 해야 합니다 [17]. 계획된 바와 같이 asyncio 기반으로 작업을 비동기로 실행하고 진행 상태를 제공하는 API를 추가하면 폴백 없이도 정상 동작할 것입니다.

프론트엔드 측면에서는 **라이트/다크 테마를 하나의 index.html로 통합**하여 이중 관리 문제를 해결했습니다 [32]. 이제 하나의 페이지에서 CSS 변수와 토글 버튼으로 테마를 전환하도록 되어 유지보수성이 개선되었습니다. 이 통합된 페이지는 WCAG 2.1 명도대비 준수를 위해 색상값도 조정되었습니다 [32].

경로 문제를 줄이기 위해, 서버 실행 관련 **문서 업데이트**와 코드 개선이 이뤄졌습니다. README에 항상 프로젝트 루트에서 서버를 실행하도록 명시하였고 [33], 코드 상에서도 `Path(__file__).resolve().parent` 등을 활용해 **루트 경로를 자동으로 탐지**하도록 패치가 제안되었습니다 [34]. 실제 패치에서 server_quali.py가 이러한 방식을 채택했다면, admin 폴더의 위치나 실행 디렉터리에 상관없이 올바른 경로로 정적 파일과 데이터 파일을 찾을 수 있게 됩니다. 운영 시에도 admin 폴더명을 변경하지 말 것과 같은 가이드가 추가되었습니다 [28].

추가로, index.html 파일의 `<head>`에 `<meta charset="UTF-8">`를 명시하고 서버에서도 모든 응답에 UTF-8 헤더를 설정하도록 하여 **인코딩 문제 예방**을 권장합니다. JSON 출력 시에도 `ensure_ascii=False`로 한글이 유니코드 이스케이프되지 않도록 하고, 필요 시 프론트엔드에서 decode 처리 없이 바로 볼 수 있게 해야 합니다.

요약하면, 현재는 **폴백 기능 구현**과 **테마 페이지 통합**으로 단기 이슈를 해결했고 [31] [32], 앞으로 **비동기 API 구현**과 **경로 참조 코드 개선**을 통해 프론트-백엔드 연계를 근본적으로 정상화할 것입니다.

7. selected_keyword_articles.json 및 selected_articles.json 데이터 무결성

진단 결과: WARN

문제점 요약: 일일 키워드 기사 모음(`selected_keyword_articles.json`)과 최종 발행 대상 기사 목록(`selected_articles.json`)의 **구조와 무결성**은 대체로 양호하나, 데이터 운영상 **주의점**이 있습니다. 중복 기사 제거와 승인 플래그 동기화가 올바르게 이루어져야 하며, 두 파일 간 내용 불일치나 일부 기사 누락이 발생하지 않도록 관리가 필요합니다. 또한 승인 플래그(`approved`)의 잘못된 설정(예: 승인되지 않은 기사에 True 표기)이 없는지 검증이 요구됩니다. 현재까지 치명적 버그는 없지만, **중복 및 누락 방지 장치**가 사람에 의해 좌우된다는 점에서 개선 여지가 있습니다.

상세 분석: `selected_keyword_articles.json`에는 해당 날짜 키워드로 **수집된 모든 기사**와 메타데이터(제목, 요약, 출처, 날짜, 점수 등) 및 편집자 승인 여부(`approved`), 코멘트(`editor_note`) 필드가 저장됩니다 [35]. 편집자는 이 작업 파일에서 각 기사별로 approved 플래그를 True로 설정하고 코멘트를 작성하게 됩니다. 이어 `selected_articles.json`에는 **승인된 기사만 추려서** 최종 발행 리스트를 구성하며, 발행 스크립트는 이 파일만을 참고하여 뉴스 페이지를 생성합니다 [4]. 이러한 구조 자체는 정상이며, 두 JSON 파일의 필드 구성도 표준적인 딕셔너리 형태를 따릅니다 (예: {"date": "...", "articles": [...]} 형태로 날짜와 기사 배열 포함) [20].

다만, **데이터 무결성** 측면에서 몇 가지 유의할 사항이 있습니다. 첫째, 중복 제거: 여러 소스로부터 수집된 기사 중 **동일 기사**(같은 URL이나 제목)인 경우 한 번만 저장되어야 합니다. 현재 Collector 단계에서 기존 자료와 비교하여 **중복 기사 제거**를 수행하도록 설계되어 있습니다 [36]. 이 로직이 제대로 작동한다면 selected_keyword_articles에 중복 항목은 나타나지 않습니다. 둘째, 승인 동기화: 편집자가 approved 표시를 한 후에는 반드시 `sync_selected_for_publish.py`를 실행하거나 해당 기능을 호출하여 **selected_articles.json을 갱신**해야 합니다 [37]. 만약 이 단계가 누락되면 selected_articles.json이 이전 상태를 유지하여 발행 시 승인된 일부 기사가 반영되지 않는 누락이 발생할 수 있습니다. 다행히 일일 자동화에서는 보정 스크립트 실행 후 동기화까지 순서대로 수행하도록 되어 있어 이러한 누락을 방지하고 있습니다 [38].

또한 승인 플래그의 부적절한 설정 가능성: 일반적으로 수집된 기사는 기본값으로 approved=False이며, 사람이 수동으로 True로 변경합니다. 그러나 응급 상황에서 `force_approve_top20.py`를 사용하면 **상위 20건을 일괄 승인(True)** 처리할 수 있는데 [26], 이 경우 편집자의 검토 없이 기사가 승인되므로 콘텐츠 품질에 영향이 있을 수 있습니다. 이는 도구 사용상의 이슈지 시스템 버그는 아니지만, 결과적으로 selected_articles에 검증되지 않은 기사가 포함될 수 있는 리스크입니다. 또한 앞서 언급한 require_editor_approval을 False로 변경하는 응급 발행 모드에서는 approved=False인 기사도 발행에 포함될 수 있으므로, **발행 후 해당 플래그들을 정리**하지 않으면 혼란이 생길 수 있습니다.

개선 방안: 중복 방지는 현재 구현된 수집 단계의 로직을 주기적으로 확인하고, 새로운 소스 추가 시에도 동일 URL 필터가 적용되도록 유지해야 합니다. 필요하면 기사 식별에 URL 외에 제목 유사도 등을 추가해 중복 제거 정확도를 높입니다. 또한 수집이 끝난 후 selected_keyword_articles를 생성할 때도 중복 체크를 한 번 더 수행해 이중 안전망을 갖추는 것이 좋습니다.

승인 동기화는 가급적 **자동화**하여 편집자 실수를 줄여야 합니다. 예를 들어, 편집 UI에서 "발행" 버튼을 누르면 백엔드에서 자동으로 sync 및 포맷 교정까지 수행하도록 하여, 사람이 따로 스크립트를 돌리지 않아도 두 JSON이 일치하도록 합니다. 현재도 PowerShell 스크립트 내에 해당 단계가 있지만 [38], 추후 웹 UI화될 경우 이를 서버측에서 책임지도록 합니다. 또한 selected_articles.json 생성 시 이전 데이터와 **비교 검증**하여, 혹시 있을지 모르는 중복이나 불일치가 발견되면 경고를 로깅하는 방법도 고려할 수 있습니다.

승인 플래그와 긴급 발행 관련해서는, **편집 Workflow 개선**으로 해결해야 합니다. 궁극적으로는 JSON 파일을 수동 편집하지 않도록 하고 UI상에서 체크박스로 승인/취소를 관리하게 하면 휴먼에러를 줄일 수 있습니다 [39]. 이미 제안된 대로 Flask/FastAPI 백엔드 + Vue/React 프론트엔드 기반의 **관리자 UI**를 구축하여 JSON 수정을 대체하면, 승인 플래그 오류나 코멘트 누락 등의 실수를 방지할 수 있습니다 [40]. 또한 응급 상황시에도 UI에서 "강제 발행" 옵션을 제공하고, 이는 내부적으로 require_editor_approval을 일시 조정하거나 자동승인 스크립트를 호출한 뒤 원복하는 절차를 거치게 하여, 사후에 플래그 상태가 일관되도록 하는 것이 좋습니다.

현재 selected_keyword_articles.json과 selected_articles.json 자체는 **무결성에 큰 문제 없이 정상 활용**되고 있으므로 [4], 상기한 관리 절차와 도구 사용 가이드만 철저히 지켜도 PASS에 가깝습니다. 다만 장기적으로는 데이터베이스 도입 등을 통해 이러한 JSON 기반 관리의 한계를 극복하는 방향도 고려될 수 있습니다 (이력 관리, 질의 효율성 향상 등).

8. archive/ 발행본 파일명 규칙 및 다중 발행 충돌 예방

진단 결과: WARN

문제점 요약: 발행 완료된 뉴스는 `archive/` 디렉토리에 `YYYY-MM-DD_KEYWORD.md/html/json` 형식으로 저장되고 있습니다 [41]. 이 방식은 **하루에 한 번** 발행을 전제로 하므로 동일 날짜에 **복수의 발행본**을 생성하려 할 경우 파일명 충돌이나 덮어쓰기가 발생할 수 있는 잠재적 문제가 있습니다. 현재 시스템에서는 하루에 한 키워드만 발행하기 때문에 크게 드러나지 않았지만, 만약 동일 날짜에 재발행하거나 다중 키워드를 발행한다면 이름 규칙이 겹쳐 관리에 혼동이 생길 수 있습니다. 또한 archive 폴더에 산출물이 누적됨에 따라 **용량 관리**와 백업 전략도 고려해야 합니다 [42].

상세 분석: 현행 아카이브 저장 규칙에서는 날짜(Date)와 키워드만으로 파일명을 구성하므로, **동일한 날짜+키워드 조합은 유일**해야 합니다. 시스템이 하루 한 번 수동 발행되는 구조라면 문제가 없으나, 실무에서 편집 실수로 같은 키워드로 두 번 발행을 시도하거나, 긴급 수정으로 재발행하는 경우 **이전 파일이 덮어써져** 최초 발행본 기록이 손실될 수 있습니다. 심지어 archive 디렉토리가 버전 관리되지 않는 한 이러한 덮어쓰기는 추적도 어렵습니다.

또 다른 시나리오는 **동일 날짜에 서로 다른 키워드**로 두 회 분량을 낼 경우입니다. 이때는 파일명에 키워드가 포함되므로 충돌은 없겠지만, 운영 정책상 하루에 여러 키워드를 발행할지 여부가 분명치 않습니다. 만약 향후 키워드 자동 추천 등의 기능으로 하루에 다수 키워드를 발행하게 된다면, 파일명 규칙부터 재고해야 할 것입니다.

보고서에서도 "중복 발행 시에는 파일명을 `키워드_YYYYMMDD_HHMM` 형태로 조정하는 등 규칙을 마련해야 한다"고 권고하고 있습니다 [43]. 이는 동일 날짜에 여러 번 발행할 경우 **시간이나 회차를 추가로 명시**하여 각각을 보존하라는 제안입니다. 현재 구현에는 이러한 룰이 없으므로, 만약 운영자가 수동으로라도 파일명을 변경해 저장하지 않는 한 충돌을 피할 수 없습니다.

개선 방안: 파일명 규칙을 개선하여 잠재 충돌을 예방해야 합니다. 가장 간단한 방법은 제안된 대로, **타임스탬프를 추가**하는 것입니다. 예를 들어 두 번째 발행부터는 `YYYY-MM-DD_KEYWORD_2` 혹은 시분(HHMM) 정보를 덧붙여 파일명을 생성합니다 [43]. 구현상, 발행 스크립트가 archive에 동일 이름 파일이 존재하는지 체크하고 존재하면 인덱스를 붙이거나 현재 시간을 추가하도록 하면 자동화할 수 있습니다. 이렇게 하면 의도치 않은 덮어쓰기를 막고 모든 발행본을 보존할 수 있습니다.

또 다른 방안은 **archive 폴더 구조를 계층화**하는 것입니다. 예컨대 날짜별 폴더를 만들고 그 아래 키워드별 파일을 두는 형태(archive/YYYY-MM-DD/KEYWORD.json 등)로 관리하면 동일 날짜에 여러 키워드도 체계적으로 구분되고, 같은 폴더 내에서는 한 번만 발행된다는 전제하에 충돌이 없도록 할 수 있습니다. 그러나 동일 키워드의 재발행 문제는 여전히 남으므로, 결국 파일명에 시간이나 버전 정보를 넣는 방법이 필요합니다.

추가로, **백업 및 용량 관리**도 함께 고려합니다. archive 폴더가 장기간 누적되면 용량이 커질 수 있으므로 일정 주기마다 압축 백업하거나, 또는 Git 등 버전 관리 저장소에 올려두고 로컬에서는 최근 몇 개만 유지하는 식의 정책도 가능할 것입니다 [42]. 운영자는 정기적으로 archive를 백업해 두고, 혹시 있을지 모르는 동일 파일명의 덮어쓰기를 수동으로라도 감지할 수 있도록 **발행 로그에 아카이브 경로를 기록**해 두는 것도 도움이 됩니다.

결론적으로, 현재까지는 하루 한 번 발행 원칙으로 큰 문제는 없었지만, **잠재적 충돌 방지책**을 마련하여 향후 시스템 확장이나 예외 상황에 대비해야 합니다.

9. requirements.txt 의존성 누락/충돌 가능성

진단 결과: FAIL

문제점 요약: 패키지 종속성을 명시한 `requirements.txt` 파일에 **일부 필수 라이브러리 누락**이 발견되었습니다. 특히 최근 서버 개편에 도입된 FastAPI, Uvicorn, Pydantic 등과, Reddit 크롤링에 필요한 PRAW 등의 라이브러리가 리스트에 포함되어 있지 않았습니다. 그 결과 새로운 환경에서 애플리케이션을 배포하거나 세팅할 때 필요한 패키지가 설치되지 않아 **ImportError**가 발생할 우려가 있습니다. 또한 패키지 버전 충돌 가능성도 관리되고 있지 않아, 향후 업데이트 시 호환성 문제가 생길 여지가 있습니다.

상세 분석: QualiJournal 시스템은 기존에 Flask 기반이었거나 간단한 서버로 동작했을 가능성이 있는데, 최신 패치에서는 **FastAPI 프레임워크**로 전환하고 Uvicorn 서버를 사용하는 방향으로 나아가고 있습니다 [44]. 그러나 이러한 변경에 따른 의존 패키지(FastAPI, Uvicorn, Pydantic 등)가 requirements.txt에 추가되지 않은 채 수동으로 설치하여 사용하는 흔적이 있습니다. 실제 단기 개선 과제로 해당 패키지들을 pip install 하도록 지침이 있었는데 [44], 이는 곧 requirements.txt에 반영이 누락되었음을 시사합니다.

또한 **Reddit API 사용**을 위해 일반적으로 Python Reddit API Wrapper (PRAW) 라이브러리를 사용하는데, 이 역시 파일에 누락된 것으로 보입니다. 커뮤니티 수집 모듈이 완전치 않다는 언급이 있었고 [7], 만약 향후 Reddit 수집을 정상 구현한다면 PRAW 또는 유사한 패키지가 필요할 것입니다. 이 밖에도 requests, feedparser 같은 웹 수집 관련 패키지가 있다면 모두 명시되어야 하나, 현재 requirements.txt가 간소화되어 있을 가능성이 있습니다.

의존성 충돌 측면에서는, FastAPI와 Pydantic, Uvicorn의 버전 호환, 또는 PRAW와 다른 패키지 간 버전 상충 등을 들 수 있습니다. 예를 들어 FastAPI 최신 버전은 Pydantic v2를 사용하는데, 코드가 v1 기준이라면 Pydantic 버전 명시가 필요합니다. 이런 부분이 관리되지 않으면 개발 환경과 배포 환경의 동작이 달라질 수 있습니다.

개선 방안: requirements.txt를 최신화하여 모든 필요한 패키지를 빠짐없이 포함시켜야 합니다. FastAPI, Uvicorn, Pydantic을 해당 버전과 함께 추가하고, PRAW도 커뮤니티 수집 기능에 맞는 버전으로 명시합니다 (예: `praw>=7.5`). 또한 bs4(BeautifulSoup), feedparser, requests 등 사용 중인 모든 라이브러리를 재점검해 누락된 항목이 없도록 합니다.

의존성 추가와 함께, 가능하면 각 패키지의 **버전 범위**도 지정하여 향후 업데이트로 인한 호환성 문제를 방지합니다. 예를 들어 FastAPI의 경우 `fastapi==0.95.1` 식으로 현재 안정 버전을 고정하거나, `fastapi>=0.95,<0.100` 식으로 범위를 줘서 과도한 메이저 업그레이드를 피합니다. Pydantic도 FastAPI와 맞는 버전을 기재하고, PRAW 역시 주요 버전 호환을 고려합니다.

나아가, **테스트 환경에서 새로 환경을 구성**해 보는 것이 좋습니다. 빈 가상환경에서 requirements.txt로 설치를 해보고, 어플리케이션이 구동되는지 확인하면 누락된 의존성을 쉽게 발견할 수 있습니다. CI 파이프라인에 이러한 검증을 추가하면 추후 의존성 누락을 자동으로 감지할 수 있습니다.

패치 과정에서 지침으로 언급된 패키지 (FastAPI, Uvicorn, Pydantic 등)는 모두 현재 requirements.txt에 반영하여, 다른 개발자나 운영자가 **문서만으로 세팅**할 수 있도록 해야 합니다 [44]. 이 작업이 완료되면 의존성으로 인한 ImportError나 ModuleNotFoundError 문제는 해결될 것입니다.

10. 전반적인 경로 지정 방식, 파일명 규칙 일관성, 모듈 호출 체계

진단 결과: WARN

문제점 요약: 프로젝트 전반에 걸쳐 **파일 경로 사용 방식**과 **파일/폴더 명명 규칙**이 일관되지 않아, 환경 변화에 취약한 부분이 있습니다. 예를 들어, 코드에서 경로를 다룰 때 하드코딩된 상대경로에 의존하는 경우가 있으며, 대소문자나 언

더스코어 사용에서도 약간의 불균형이 관찰됩니다. 또한 하위 모듈(스크립트) 호출 체계가 통합되지 않고 산발적으로 이루어져, 구조를 파악하기 어렵게 합니다. 이러한 요인들은 경로 또는 파일명 변경 시 **예기치 않은 오류**를 유발하거나, 새 개발자가 진입할 때 혼선을 줄 수 있습니다.

상세 분석: 경로 지정 방식의 문제는 앞서도 일부 언급되었듯이, **프로젝트 루트 기준의 상대경로**를 코드에서 직접 사용하는 사례로 나타납니다. server_quali.py나 orchestrator.py 등에서 `'config.json'`, `'admin/index.html'` 등을 열 때 경로를 문자열로 지정했다면, 실행 디렉토리에 의존하여 파일을 찾게 됩니다. 실제로 "서버를 루트에서 실행하지 않으면 데이터 파일을 찾지 못함"이라는 위험이 지적되었고 [28], 이는 경로 처리 미흡을 잘 보여줍니다. 개선 계획에서는 이 문제를 해결하기 위해 `Path(__file__).resolve().parent`를 사용해 **코드에서 루트 경로를 자동 탐지**하도록 제안하고 있습니다 [34]. 이는 올바른 방향으로, 현재 패치에 반영되었다면 경로 안정성이 높아졌겠지만, 과거 버전까지 포함해 점검할 필요가 있습니다.

파일명 규칙은 대체로 snake_case로 통일되어 있으나, "QualiJournal"라는 명칭이 코드 내에서는 `quali` 등으로 축약되어 사용되며, 일부 혼용이 있습니다. 예를 들어 run_quali_today.ps1, server_quali.py에서는 "quali"를 쓰지만, README나 문서상 제목은 QualiNews/QualiJournal 등으로 혼재되어 있어 초기에는 파일을 찾는 데 혼란이 있을 수 있습니다. 폴더 구조 면에서도 admin 폴더 도입으로 경로가 변경되었는데, 문서와 코드 양쪽에서 이 명칭이 일치해야 합니다. 만약 누군가 admin 폴더 이름을 바꾸면 경로가 깨지는 문제도 제기되었고 [29], 따라서 **폴더/파일 이름은 변경을 가정하지 않고 코드에 고정**되어 있다고 볼 수 있습니다.

하위 모듈 호출 체계의 불명확성은, orchestrator와 개별 스크립트들의 관계에서 드러납니다. 이상적으로는 orchestrator(혹은 메인 제어 스크립트)가 모든 모듈을 import하여 함수 단위로 호출하는 구조가 일관적이지만, 현재는 PowerShell 스크립트가 각각의 Python 파일을 **개별 프로세스로 실행**하는 식이라 한눈에 흐름을 파악하기 어렵습니다. 또한 일부 기능은 CLI 인자(`--collect` 등)로 orchestrator를 부르고, 다른 기능은 별도 .py 스크립트를 실행하는 방식이 혼재되어, **모듈 호출 방식에 일관성**이 없습니다. 이는 곧 유지보수자가 각 기능이 어디서 시작되고 어떻게 연결되는지 이해하기 어렵게 합니다.

개선 방안: 경로 처리 개선: 모든 파일 경로 접근에는 `os.path` 또는 `pathlib.Path` 기반으로 현재 파일 위치나 설정값을 기준으로 계산하도록 리팩토링해야 합니다. 패치에서 제안된 기법처럼 `Path(__file__).parent` 등을 활용하면, 예를 들어 server_quali.py가 config.json을 열 때 `PROJECT_ROOT = Path(__file__).resolve().parents[0]` (혹은 적절한 상위) 기준으로 경로를 만들 수 있습니다 [34]. 이렇게 하면 실행 위치와 무관하게 항상 올바른 경로를 참조하게 됩니다. 또한 경로 문자열 사이에 하드코딩된 구분자나 상수 대신, 설정 파일에 주요 경로를 모아두고 참조하거나, `import importlib.resources` 등으로 패키지 리소스를 불러오는 방법도 고려할 수 있습니다.

파일명 규칙 일관성: 코드와 문서 전반에 사용하는 명칭을 통일해야 합니다. QualiJournal vs QualiNews 등의 용어 혼용을 피하고, 소스 폴더와 파일 이름도 이에 맞게 정리합니다. 이미 주요 코드 파일은 snake_case로 잘 되어 있으므로 유지하되, README나 주석 등에서도 동일한 표기를 써서 혼란을 줄입니다. admin 폴더명은 코드에 고정되어 있으므로 변경하지 말고, 필요한 경우 해당 폴더 경로를 설정으로 외부화하여 유연성을 주도록 고려합니다. 하지만 작은 프로젝트에서는 폴더 구조를 고정하는 편이 오히려 명확하므로, **문서에 폴더 구조를 명시**하여 개발자가 이를 준수하도록 하면 됩니다 [30].

모듈 호출 체계 정비: 장기적으로 orchestrator.py를 **여러 모듈로 분리**하더라도, 호출 진입점은 일관되게 관리해야 합니다. 예를 들어 `main.py` 하나를 두고, 그 안에서 subparser 혹은 함수 호출로 모든 기능(수집, 재구성, 보충, 발행 등)을 실행하게 만들면, PowerShell 없이도 동일한 흐름을 Python만으로 구동할 수 있습니다. 또, FastAPI 서버에서도 이 `main.py`의 함수를 호출하도록 연계하면 UI/API/CLI가 모두 단일 진입점을 공유하게 됩니다. 현재 계획된 개선 방향인 **마이크로서비스 분리** 및 **Docker화**도 이러한 모듈 경계를 명확히 하는 데 도움이 될 것입니다. 각 마이크로서비스가 자기 역할에 맞는 경로와 모듈만 참조하도록 하면, 경로 혼동이나 호출 뒤엉킴이 줄어듭니다.

마지막으로, **위험 대응**으로 제시된 방안을 지속 적용합니다. 경로/파일명 혼동 위험에 대해서는 이미 "README와 인수인계서에 루트에서 실행해야 함을 명시하고, 코드에서 경로 자동 탐지"가 대응전략으로 제시되었습니다 [28] [45]. 실제

로 문서를 통해 이러한 사항을 공지했고, 코드에도 반영되었다면 현재 WARN 수준으로 낮아졌다고 평가합니다. 앞으로도 새로운 모듈이나 경로를 추가할 때 이 가이드라인을 준수하여, **일관되고 안전한 경로 관리 체계**를 유지해야 할 것입니다.

---

1 4 9 19 20 22 23 25 26 35 37 38 1004_2A고도화report.pdf
file://file-Ko3WmavWUUxUmLMC1s6fso

2 11 12 13 14 15 16 17 18 21 28 29 30 31 32 33 34 44 45 1005_A퀄리계획execution_plan.pdf
file://file-1W7XdBQtN4y9u8KV41BXun

3 6 8 10 24 27 39 40 41 42 43 1004_3A작업계획report.pdf
file://file-S3G4TQFPei3GeAJcXjN6B3

5 1003_1A퀄리 뉴스 제안.pdf
file://file-9ckyN6UsWeG3crtkDBreya

7 36 1003_5A퀄리뉴스를 전면적으로 리팩터링하여 안정적이고 확장 가능한 시스템으로 구축하려면.pdf
file://file-C1X8pMSvAVd6Xs392FEnwq