ChatGPT

퀄리저널 관리자 시스템 (Admin API) 개선 작업 인수인계 보고서

프로젝트 범위 및 진행 시점

퀄리저널(QualiJournal) 관리자 시스템은 하루 키워드 뉴스를 수집·편집·발행하는 플랫폼의 백엔드/관리자 도구로서, 최근 안정성과 기능 개선 작업이 진행되었습니다 [1]. 본 문서는 해당 **Admin 시스템 개선 작업의 현황과 향후 계획**을 정리한 인수인계 자료로, 새로운 개발 세션에서도 이전 맥락을 잃지 않고 작업을 이어나갈 수 있도록 작성되었습니다 [2]. 2025년 10월 초 수립된 개선 실행계획서에 따라 접근성, 비동기 처리, 보안, 국제화 등 다양한 분야의 향상이 시도되었으며 [3], 이 문서에서는 그 중 **Cloud Run 배포 자동화**, **게이트 임계값(KPI) 슬라이더 기능**, **FastAPI 서버 시크릿/DB 연동 설정**, **주요 관리자 기능 구현 및 마무리**에 관한 진행사항과 향후 할 일을 기술합니다 [4] [5]. 모든 섹션에는 근거 자료를 인용하고 자체 점검 체크리스트를 포함하여, **개발 인계 시 혼동 없이 동일한 상태로 작업을 재개**할 수 있도록 하였습니다.

Cloud Run 배포 자동화 (인프라 환경)

현재까지 진행: 관리자 서버는 Docker 컨테이너로 패키징되었으며, Google Cloud Run 환경에 배포되어 동작 중입니다 [6]. 기존에 수동으로 이루어지던 빌드/배포 과정을 자동화하기 위해 CI/CD 파이프라인 구성이 진행되고 있습니다. 프로젝트는 GitHub 저장소와 연동된 **GitHub Actions 워크플로우**를 통해 코드 푸시 시 자동으로 Docker 이미지를 빌드하고 테스트한 후 Cloud Run에 배포하는 구성을 목표로 합니다 [7]. 이로써 코드 변경이 발생하면 수작업 개입 없이도 최신 버전이 Cloud Run 서비스에 자동 반영되도록 할 계획입니다.

향후 해야 할 일: Cloud Run 배포 자동화를 완성하려면 GitHub Actions에서 Google Cloud에 인증하고 컨테이너 이미지를 **Artifact Registry**에 푸시한 뒤, 해당 이미지를 Cloud Run에 배포하는 workflow를 구축해야 합니다 [8]. 구글에서 제공하는 `deploy-cloudrun` 액션을 활용하면, 컨테이너 이미지 경로 등 몇 가지 인자만으로 손쉽게 Cloud Run 서비스에 새로운 리비전을 배포할 수 있으며, 배포 완료 후 **서비스 URL**을 출력으로 받아 후속 단계에서 활용할 수 있습니다 [9]. 현재 CI/CD 설정의 인증 정보와 권한(Role) 세팅을 마무리하고, `main` 브랜치에 푸시될 때 위 작업이 트리거되도록 workflow YAML을 작성해야 합니다. 마지막으로 배포 자동화가 정상 동작하는지 검증하기 위해 워크플로우 로그에서 **이미지 빌드/배포 성공 메시지**를 확인하고, 실제 Cloud Run 서비스 URL에 접속하여 응답을 점검합니다 [10].

Self-Checkpoints:
- Cloud Run에 **배포된 서비스 URL**에 접속하여 API 응답이 오는지 확인합니다 (예: Actions 워크플로우 로그의 출력 URL을 `curl`로 호출하여 200 응답 확인) [10]. 특히 `/api/status` 엔드포인트를 호출해 서버의 상태와 KPI 지표가 정상적으로 반환되는지 확인합니다 [11].
- Cloud Run **환경 변수 설정**을 검사합니다. GCP 콘솔의 Cloud Run **Variables & Secrets** 탭에서 .env 파일의 내용이 올바르게 반영되어 있는지 확인하고, 필요한 시크릿값(DB 연결 정보, API 키 등)이 모두 설정되었는지 점검합니다 [12] [13].
- Cloud Run **로그 모니터링**을 통해 컨테이너 시작 시 오류가 없는지 확인합니다. 새 리비전 배포 후 Cloud Run 로그에 **dependency 오류** 또는 **마이그레이션 오류**가 없는지 살피고, 오류 발생 시 해당 리비전을 롤백하거나 패치합니다 [7].
- (필요시) `gcloud run services describe <서비스명>` 명령어로 서비스 설정을 확인하여, 이미지 태그와 환경변수가 최신 배포에 반영되었는지, 트래픽이 올바른 리비전에 할당되었는지 등을 검증합니다 [8].

게이트 임계값 슬라이더 및 KPI 자동 갱신

현재까지 진행: 편집자가 **품질게이트 임계값**(최소 승인 기사 수 등)을 UI에서 조절할 수 있도록 대시보드 상단에 슬라이더가 추가되었습니다 [14]. 슬라이더 값을 변경하면 백엔드의 `/api/config/gate_required` 엔드포인트에 PATCH 요청을 보내 `config.json`의 해당 값을 실시간으로 업데이트하며, 이를 통해 **승인 기준**(예: 15건)을 동적으로 변경할 수 있습니다 [15]. 또한 KPI 지표(수집된 기사 수, 승인된 기사 수 등)가 이전에는 수동으로 "새로고침"해야 갱신되던 것을 개선하여, 이제 프론트엔드에서 **30초 간격으로** `/api/status`를 호출해 자동 갱신하도록 구현했습니다 [16] [11]. 이로써 별도 버튼 없이도 일정 주기마다 대시보드의 KPI 수치가 최신 상태로 갱신되어, 편집자는 **실시간으로 현황 파악**이 가능해졌습니다 [17]. 이러한 게이트 슬라이더와 자동 새로고침 기능은 패치 설계 문서의 요구사항을 충족하는 것이며, 편집 업무의 편의성과 즉시성을 크게 높여주었습니다 [18].

유지보수 및 확인 사항: 슬라이더로 변경된 임계값은 **서버 메모리에 즉시 반영**되지만, Cloud Run 인스턴스 재시작 시 기본값이 적용될 수 있으므로 `config.json`의 기본 설정도 함께 조정해야 합니다. 해당 기능에 대한 통합 테스트를 작성하여, 슬라이더 조작 -> 임계값 변경 -> 발행 조건 체크까지 일련의 흐름이 기대대로 동작함을 검증할 예정입니다 [19] [20]. 또한 README 문서에 **게이트 임계값 조정 방법**(예: config.json 설정 위치 및 UI 사용법)과 **KPI 자동 갱신 동작**에 대한 설명을 추가하여 운영자가 기능을 이해하고 활용하도록 해야 합니다 [20]. 향후 필요시 슬라이더 UI의 민감도나 업데이트 주기 등을 조정할 수 있으며, 현재 구조가 단일 편집 세션에서만 임계값을 유지하므로 **다중 사용자 환경**에서의 동작(예: 다른 편집자가 볼 때 즉시 반영 여부)도 추후 고려 대상입니다.

Self-Checkpoints:
- **슬라이더 조작 검증:** 관리자 UI에서 슬라이더를 움직였을 때 개발자 도구(Network 탭)을 통해 `/api/config/gate_required`에 PATCH 요청이 정상적으로 전송되고 200 OK 응답을 받는지 확인합니다 [14]. 그런 다음 서버 측 `config.json` 또는 `/api/config/gate_required` GET (엔드포인트가 있다면) 호출을 통해 값이 실제로 변경되었는지 검증합니다.
- **발행 게이트 적용 확인:** 슬라이더로 임계값을 변경한 후, 승인된 기사 수에 따라 발행 가능/불가 상태가 올바르게 반영되는지 테스트합니다. 예를 들어 임계값을 10으로 낮춘 뒤 10개 기사 승인 시 발행이 가능해지는지, 다시 15로 올리면 동일 조건에서 발행이 막히는지 확인합니다 [15]. 이러한 로직은 통합 테스트 케이스로도 작성하여 자동화된 형태로 검증합니다 [19].
- **KPI 자동 새로고침 확인:** 대시보드 화면을 열어두고 새로고침 버튼을 누르지 않아도 **30초 내에 KPI 수치가 변화**하는지 관찰합니다 [21]. 특히, 별도의 수집/승인 동작을 한 뒤 잠시 기다렸을 때 수치가 갱신되어 표시되는지 확인합니다. 서버 로그에도 주기적인 `/api/status` 호출 기록이 남는지 점검하여, 클라이언트 타이머가 정상 동작하고 있음을 검증합니다 [11].

FastAPI 서버 설정 고도화 (시크릿 및 DB 연동)

환경변수 및 시크릿 관리: 개발/운영 환경 분리를 위해 설정값을 `.env` 파일 및 환경변수로 관리하도록 서버 설정을 개선했습니다 [22]. API 키, DB 비밀번호 등의 민감한 값은 코드에 하드코딩하지 않고 GCP **Secret Manager**와 연계된 Cloud Run **환경 변수**로 주입됩니다 [13]. 예를 들어 번역기 API 키나 Admin 토큰 값은 배포 시 환경변수로 설정되며, FastAPI에서는 `os.getenv()` 등을 통해 이를 불러와 사용하도록 구현되어 있습니다. 이러한 접근 방식은 시크릿 로테이션이나 운영 환경 변경 시 유연하게 대응할 수 있고, 코드 유출 시에도 주요 credentials가 드러나지 않는 이점이 있습니다 [23]. 현재 `.env` 예시 파일과 운영 환경 변수 값들이 정리되어 있으며, 새로운 서버 인스턴스가 시작될 때 해당 값들이 제대로 로드되는지 Cloud Run에서 검증하고 있습니다.

DB 연동 및 데이터 지속성: 기존 쿼리저널 시스템은 기사 데이터와 메타정보를 로컬 JSON 파일(`selected_keyword_articles.json`, `selected_articles.json` 등)에 저장해 사용하고 있었습니다 [24]. 이번 개선 작업에서는 **데이터베이스 도입 준비**를 병행하여, 향후 사용자 행동 로그나 다중 키워드 아카이브 등의 데이터를 영구 보존하고 질의할 수 있도록 설계 중입니다. 백엔드 코드 구조를 키워드 중심으로 개편하면서 **DB 스키마도 이에 맞게 재설계**하고자 하며 [25], ORM(Entity) 정의나 마이그레이션 설정이 필요한 경우 이를 추가할 계획입니다. 아직까지 기사 선정/발행 관련 데이터는 JSON 기반으로 동작하지만, 필요 시 PostgreSQL 등의 관계형 DB로 전환할

수 있도록 모듈화되어 있습니다 [24]. 예를 들어, 차후 **사용자 선호도 분석**이나 **개인화 추천** 기능을 위해 MongoDB나 PostgreSQL을 도입하고, **FastAPI의 SQLAlchemy** 연동을 설정하는 방안을 고려 중입니다 [26]. 현재 단계에서는 DB 연결이 필수적이지는 않지만, **ENV 파일에 DB 연결 문자열(DB_URL)**을 설정하고 해당 값으로 세션을 초기화하는 코드가 준비되어 있습니다.

보안 (인증/권한) 강화: 외부 배포를 앞두고 **토큰 기반 인증**의 기초를 도입했습니다. FastAPI의 Dependency 기능으로 **요청 헤더에 사전 공유된 토큰**이 포함되어야만 민감 엔드포인트를 호출할 수 있도록 하는 미들웨어를 구현하였습니다 [27]. 현재는 간단히 환경변수 `ADMIN_TOKEN`에 설정된 정적 토큰값을 검사하는 수준이며, 이를 통해 **임의의 비인가 요청**이 주요 관리자 API를 호출하는 것을 막고 있습니다 [27]. 차후 OAuth2/JWT 기반의 정교한 인증 체계로 확장하기 전까지 임시 방편이지만, 패치 설계서에서도 권장된 조치이므로 이번 배포에 포함했습니다 [28]. 또한 **관리자용 페이지와 일반 사용자 페이지의 권한 분리**를 염두에 두고 구조를 마련 중입니다 [29]. 향후 모든 관리자 API (예: `/api/config/*`, `/api/report` 등)는 관리자 롤(role)만 접근 가능하도록 하고, JWT 토큰 발급 시 해당 role을 명시하거나 별도의 관리자 UI 경로를 분리하여 보호할 계획입니다 [30]. 이와 함께 Cloud Run 배포 환경에서는 HTTPS를 강제 적용하여 통신 암호화를 보장하고, 개발 모드에서도 필요한 경우 이 원칙을 지킬 수 있도록 CORS 도메인 설정 등을 점검했습니다 [31]. 현재까지는 기본 토큰 체크만 활성화되어 있으며, 금일 내로 이를 배포 환경에 반영해 **기본 보안 조치**를 완료하는 것을 목표로 합니다.

Self-Checkpoints:
- **환경 변수 로드 확인:** 서버 시작 로그나 Cloud Run 환경 설정을 통해 필요한 환경변수가 모두 세팅되었는지 확인합니다. 예를 들어 `.env`에 정의한 `DB_URL`, `API_KEY`, `ADMIN_TOKEN` 등이 실제 컨테이너 인스턴스에 주입되었는지 Cloud Run 콘솔의 변수 목록에서 검증합니다 [12]. 잘못된 값이 없는지, 불필요한 민감 정보가 출력되지 않는지도 함께 점검합니다.
- **DB 연결 상태 테스트:** 만약 데이터베이스를 설정한 경우, 서버 시작 시 **DB 연결 성공 로그**가 출력되는지 확인하거나, 간단한 쿼리를 실행해보는 테스트 엔드포인트로 연결을 검증합니다. 아직 JSON 파일 기반이라면, 기존 흐름대로 JSON 파일 읽기/쓰기가 정상 작동하는지 (`selected_articles.json` 업데이트 등을 시나리오 테스트) 확인합니다 [32]. 데이터 정합성을 위해 필요한 경우 `repair_selection_files.py` 등의 도구로 JSON 데이터 무결성도 확인합니다.
- **토큰 인증 동작 검증:** 토큰이 필요한 엔드포인트(예: `/api/report` 등)에 **인증 헤더 없이 호출**을 시도해 403 또는 401 에러가 발생하는지 확인하고, 올바른 토큰을 포함했을 때 요청이 성공하는지 시험합니다 [27]. 현재 적용된 정적 토큰 값은 `.env`에 지정되어 있으므로, 해당 값과 일치하지 않는 토큰으로 요청시 접근 거부되는지를 로그와 응답으로 확인합니다. 아울러 CORS 정책이 의도대로 적용되어, 허용된 도메인에서의 요청은 통과되고 그렇지 않은 경우 차단되는지도 브라우저 개발자 도구를 통해 점검합니다.

관리자 기능 추가 구현 및 마무리

주요 기능 구현 현황 및 계획

퀄리저널 **관리자 기능 중 미구현된 핵심 항목**으로 **보고서 생성, 뉴스 카드 요약·번역, 결과 내보내기** 등이 식별되었으며, 이는 중기 개선계획의 요구사항에 포함되어 있었습니다 [5]. 현재까지 서버에는 `/api/report` 및 일부 `/api/export/{fmt}` 엔드포인트만 기본 형태로 존재할 뿐, **POST 기반의 보고서 생성**이나 **카드 요약/번역** API는 구현되어 있지 않은 상태입니다 [33]. 프론트엔드 관리자 UI에도 해당 기능을 트리거하는 버튼이나 메뉴가 없어서 현 시점에서는 이 기능들을 사용할 수 없습니다 [34]. **다음 단계**로 백엔드 API와 프론트엔드 UI 양측에 이 기능들을 추가 구현하여, 편집자가 **일간 보고서를 생성**하고 **뉴스 카드를 자동 요약·번역**하며 **최종 선정 기사 목록을 파일로 추출**할 수 있도록 하는 작업이 진행됩니다 [35]. 아래는 각 기능별 구현 계획과 현재 준비 상황입니다.

- **일일 보고서 생성 (POST `/api/report`):** FastAPI 서버에 **새 보고서 생성 엔드포인트**를 추가해야 합니다. 이 요청을 받으면 내부적으로 `tools/make_daily_report.py` 스크립트를 실행하여 현재까지 편집자가 승인한 기사 목록(`selected_articles.json`)을 취합, Markdown 형식의 **데일리 보고서** 파일을 생성합니다 [36]. 생성된 보고서는 `archive/reports/{날짜}_{키워드}.md` 경로로 저장되고, API 응답으로 해

당 **파일 경로 또는 다운로드 URL**을 JSON 형태로 반환하게 됩니다 [37] . 현재 `/api/report` GET 엔드포인트(과거 테스트용)가 일부 존재하지만, 이를 대체/확장하여 **POST로 호출 시 보고서 생성이 실행**되도록 구현할 계획입니다 [33] . 이 기능을 통해 편집자는 하루 키워드 뉴스를 요약한 **Markdown 보고서**를 누적 저장하고 필요 시 열람하거나 공유할 수 있게 됩니다.

- **뉴스 카드 요약·번역 (POST `/api/enrich/keyword`, `/api/enrich/selection`):** FastAPI에 두 가지 **요약/번역 엔드포인트**를 추가합니다. `/api/enrich/keyword`는 수집된 전체 기사(특정 키워드 기준)에 대한 요약/번역을, `/api/enrich/selection`은 **편집자가 선정한 기사들에 한정**하여 요약/번역을 수행합니다 [38] . 두 엔드포인트 모두 내부적으로 `tools/enrich_cards.py`를 호출하여, 해당 기사들의 제목, 본문 등을 요약하고 필요한 경우 기계 번역(API 기반)을 수행합니다 [38] . 결과물은 기사별 요약/번역된 텍스트를 포함한 파일로 저장되며 (예: `archive/enriched/{키워드}_{날짜}.json` 또는 `.md` 등 형식 고려), 마찬가지로 **결과 파일 경로**를 응답으로 반환합니다 [39] . 이를 통해 편집자는 원본 기사를 일일이 읽지 않고도 요약본을 참고하거나, 다국어 번역된 콘텐츠를 확인하여 기사 선정에 활용할 수 있습니다. 현재 이 기능도 서버/클라이언트에 구현이 전무한 상태이므로, **백엔드 스크립트 호출 로직부터 응답 처리까지 신규 개발**이 필요합니다 [4] . 요약/번역 과정은 시간이 다소 걸릴 수 있으므로, 초기 버전에서는 동기 방식으로 호출하되 **타임아웃** 설정 등을 유의해야 합니다 (추후 비동기 작업 큐로 개선 가능 [40]).

- **발행 결과 내보내기 (GET `/api/export/md`, `/api/export/csv` 완성):** 서버 측에 이미 `/api/export/md`와 `/api/export/csv` 엔드포인트가 존재하지만 현재는 일부 포맷만 지원하거나 출력 형식에 개선이 필요한 상태입니다 [33] . 이를 **완전 구현**하여, 최종 발행된 기사 리스트(`final selection`)를 Markdown 또는 CSV 파일로 다운로드할 수 있도록 합니다. Markdown 내보내기에서는 편집된 기사 제목, 요약, 링크, 코멘트 등을 포함한 템플릿을 작성하고, CSV의 경우 각 기사의 주요 필드를 쉼표로 구분하여 출력합니다 [41] . 특히 CSV의 경우 **Excel에서 열때 한글 인코딩 문제가 없도록 BOM(Byte Order Mark)**을 파일 맨 앞에 추가합니다 [41] . (현재 `/api/export/csv` 응답에 BOM이 없을 경우 Excel에서 깨지는 이슈가 있어, 이를 반영함.) 이 개선이 완료되면 UI에서 버튼 클릭 한 번으로 **현재까지 발행된 뉴스 목록을 파일로 저장**할 수 있어, 백업을 받거나 통계 작업을 위한 데이터를 손쉽게 확보할 수 있습니다 [42] .

프론트엔드 UI 연동 및 사용성 개선

위 새로운 백엔드 기능들을 **관리자 대시보드 UI**에서 사용할 수 있도록, 관련 버튼과 연동 로직을 추가합니다. 우선 보고서 생성을 위해 대시보드에 `"일일 보고서 생성"` 버튼을 배치하고, 클릭 시 프론트엔드에서 `/api/report` POST 요청을 보내도록 구현합니다 [43] . 응답으로 보고서 파일 경로(URL)가 오면 UI 상에 **다운로드 링크** 또는 “보고서 열기” 버튼을 제공하여 사용자가 결과물을 바로 확인할 수 있게 합니다 [44] . 필요에 따라 보고서 내용을 즉시 볼 수 있도록 **모달 창**이나 새 탭으로 열리는 기능도 고려합니다 [43] .

뉴스 카드 요약/번역의 경우에도 `"카드 요약/번역"` 버튼을 각 섹션에 추가하여, 누르면 `/api/enrich/...` 엔드포인트를 호출하고 완료 시 결과 파일 링크를 표시합니다 [45] . 요약/번역 결과는 번역된 요약문 등 **텍스트 파일**일 가능성이 높으므로, UI에서 바로 열람하거나 다운로드할 수 있게 할 것입니다 [45] . 마지막으로 내보내기(export) 기능에 대해서는 대시보드에 `"Export MD"`, `"Export CSV"` 버튼 (또는 하나의 드롭다운 메뉴) 등을 추가합니다 [46] . 해당 버튼 클릭 시 `/api/export/md` 혹은 `/api/export/csv` GET 요청을 보내 브라우저 다운로드를 바로 실행합니다 (예: `<a href="/api/export/csv" download>Export CSV</a>` 형태로 링크를 구성하거나, Fetch API로 응답을 받아 파일을 생성하는 방식) [47] . 이렇게 함으로써 편집자는 GUI상에서 **별도 툴 없이도** 최종 결과물을 원하는 포맷으로 저장할 수 있게 됩니다 [48] .

UI/UX 측면에서 **진행 상태 표시**와 **오류 처리**도 고려합니다. 예를 들어 보고서 생성이나 요약 작업은 수초 이상 걸릴 수 있으므로, 사용자가 버튼 클릭 후 기다리는 동안 **로딩 스피너**나 진행 메시지를 표시합니다 [49] . 작업 완료 시 자동으로 스피너를 숨기고 결과 링크/버튼을 활성화하여 다음 액션을 취할 수 있게 합니다. 만약 작업 중 에러가 발생하면 (예: 스크립트 예외 등) 사용자에게 **오류 경고창**이나 메시지를 표시하고, 필요하다면 해당 오류를 서버 로그에도 남겨 개발자가 원인을 파악할 수 있게 합니다 [50] . (추후 고도화 단계에서 `/api/tasks/...` 비동기 엔드포인트로 전환되면, 진행

률 표시 등 실시간 피드백이 가능해지겠지만 현 단계에서는 동기 처리 후 완료 통보만 하는 것으로 충분합니다 [51].) 이러한 UI상의 세부 개선을 통해 새로 추가되는 기능들이 **사용성**을 갖추고, 편집 workflow에 자연스럽게 녹아들도록 할 계획입니다 [52].

Self-Checkpoints:
- **보고서 생성 기능 테스트:** 관리자 화면에서 "일일 보고서 생성" 버튼을 클릭하고, 서버 응답으로 **보고서 파일 경로(JSON)**를 받으면 UI에 표시되는 링크를 눌러 실제 Markdown 보고서가 열리는지 확인합니다 [53] [37]. 생성된 `.md` 파일 내용이 예상대로 (제목, 선정 기사 리스트, 요약, 코멘트 등) 포함되어 있는지 검토하고, `archive/reports/` 디렉토리에 올바른 이름으로 저장되었는지도 서버 파일시스템에서 확인합니다.
- **요약/번역 기능 테스트:** "카드 요약/번역" 버튼 클릭 시 일정 시간 후 결과 링크가 나타나는지 확인하고, 해당 링크를 통해 **요약된 카드 내용**을 열람합니다 [45]. 여러 개의 기사가 요약된 경우 포맷이 보기 좋게 정리되어 있는지, 번역 결과가 존재하는지 등을 체크합니다. 기사 수가 많은 경우에도 UI가 멈추지 않고 (스피너 표시) 잘 동작하는지 확인하고, 비정상 입력 또는 오류 상황(예: 스크립트 오류 유도)에 대한 메시지 처리가 되는지도 확인합니다 [49].
- **Export 기능 테스트:** "Export MD" 및 "Export CSV" 버튼을 각각 눌러 **파일 다운로드**가 즉시 시작되는지 확인합니다 [46]. 저장된 Markdown 파일을 열어 내용이 Markdown 형식으로 잘 구성되어 있는지 (헤더, 리스트 등), CSV 파일을 엑셀로 열었을 때 한글이 깨지지 않고 올바르게 표시되는지 (BOM 추가 여부) 확인합니다 [41]. 만약 CSV 필드 구분이나 인코딩에 문제가 있다면 로그나 예외로 표시되는지 확인하고, 필요시 해당 부분을 수정합니다.
- **예외 상황 및 UI 피드백:** 의도적으로 스크립트를 실패시키는 등 예외 상황을 만들어 본 뒤, 사용자가 이해할 수 있는 오류 팝업이 뜨는지 확인합니다 [50]. 예를 들어 보고서 생성 중 스크립트에서 Exception 발생 시 UI에 "보고서 생성 실패: 오류 메시지…" 알림이 나타나는지, 그리고 서버 로그에도 stacktrace가 찍혀 개발자가 원인을 추적할 수 있는지 확인합니다 [40].

통합 테스트 및 문서화 (최종 마무리)

새로운 기능 구현이 완료되면 **백엔드 통합 테스트**와 **문서 업데이트**를 통해 마무리합니다. 우선 Pytest 기반으로 `/api/report`, `/api/enrich`, `/api/export` 등의 엔드포인트에 대한 **테스트 스크립트**를 작성합니다 [19]. 예를 들어 `/api/report` 호출 시 실제로 보고서 파일이 생성되고 응답 JSON에 해당 경로가 포함되는지, `/api/enrich/selection` 호출 시 예상된 요약 결과 파일이 나오는지, `/api/export/csv` 호출 결과 다운로드된 CSV의 내용과 형식이 올바른지 등을 자동으로 검증합니다 [19]. 이러한 테스트는 로컬에서 수동 실행은 물론, CI 파이프라인에도 통합하여 **코드 변경 시 자동 검증**되도록 합니다 [54]. (GitHub Actions 워크플로우에 Pytest 단계를 추가하여, 실패 시 배포가 진행되지 않도록 하는 방안을 적용합니다.) 프론트엔드에 대해서는 Selenium 등의 UI 테스트까지는 아니더라도, 백엔드 API 단위테스트와 함께 **시나리오 테스트** 형태로 주요 흐름을 점검할 예정입니다 [55]. 예컨대 가상으로 여러 기사를 수집->승인한 뒤 보고서 생성 및 CSV 내보내기까지 호출해 보는 통합 테스트를 구현하여, **전체 작업 플로우의 회귀 테스트**를 진행합니다 [55].

병행하여 **프로젝트 문서화** 작업을 수행합니다. 주요 README.md를 업데이트하여 새로 추가된 API 엔드포인트 목록과 사용 방법을 상세히 기술합니다 [56]. 예를 들어 `/api/report`, `/api/enrich/...`, `/api/export/...` 각각의 **용도와 요청/응답 형식**을 표로 정리하고, 프론트엔드에서 해당 기능을 어떻게 사용할 수 있는지 설명을 추가합니다 [56]. 또한 **운영 매뉴얼** 측면에서, “승인 임계값 변경 방법”, “키워드 입력 및 발행 절차”, “주간 리그레션 테스트 수행 방법” 등을 정리하여 관리자용 안내 자료로 남깁니다 [56]. 이 과정에서 발견된 미비사항 (예: 환경 설정 항목 누락, 종속 패키지(requirements.txt) 변경 등)도 문서에 반영하고, 특히 **비동기 엔드포인트 미구현에 따른 폴백 동작**에 대한 설명을 추가합니다 [57]. (현재 UI가 `/api/tasks/...`가 없으면 자동으로 동기 엔드포인트로 폴백함) 이러한 문서화는 차후 새로운 개발자가 투입되거나 운영팀이 시스템을 사용할 때 **원활한 이해를 돕기 위한 필수 단계**입니다 [58]. 최종적으로 문서와 테스트까지 완료함으로써, 이번 중기 개선 범위의 과제를 성공적으로 마무리하고 안정된 상태로 운영에 인계할 수 있게 됩니다 [58].

Self-Checkpoints:
- **테스트 통과 확인:** 로컬 환경 및 CI 환경에서 `pytest`를 실행하여 **모든 테스트 케이스가 성공**하는지 확인합니다 [19]. 새로운 기능과 관련된 테스트 (보고서 생성, 요약, 내보내기 등)뿐 아니라 기존 기능 테스트도 회귀 검증하여, 최근 변경으로 인한 사이드 이펙트가 없음을 보장합니다. 필요하면 커버리지 리포트를 확인하여 주요 로직이 충분히 테스트

되었는지 검토합니다.
- **문서 최신화 확인:** 업데이트된 README.md와 기타 문서를 훑어보며, 새로 추가된 엔드포인트나 UI 기능에 대한 언급이 빠진 곳은 없는지 확인합니다 [57]. 예를 들어 **게이트 슬라이더 사용법**, **KPI 자동갱신 소개**, **비동기 대체 동작 설명** 등이 README에 포함되어 있는지 점검합니다 [20]. 또 환경 설정 섹션에 새로 필요한 환경변수가 모두 기록되었는지, 설치 및 실행 절차가 현재 코드 기준으로 올바른지 확인합니다. 운영 매뉴얼 문서(있다면)에도 위 Self-Check 리스트에 준하는 검수 방법과 정기 점검 사항 등이 포함되었는지 살펴봅니다.
- **기능 최종 점검:** 개선된 관리자 시스템을 실제 시나리오로 한 번 실행해 봅니다. 하나의 키워드를 선택해 **수집→편집→발행→보고서 생성→CSV 추출**까지 전체 흐름을 수행하여, **모든 단계가 문제없이 이어지는지** 확인합니다 [52]. 특히 여러 신규 기능이 추가됨에 따라 앱 전체적인 안정성을 관찰해야 합니다. Cloud Run 로그에 새로운 오류는 발생하지 않는지, 응답 시간은 적절한지 모니터링합니다. 이상이 없다면 릴리스 노트를 작성하고 버전을 태깅하여 배포를 완료합니다 [59]. 이후 장기적으로 계획된 추가 과제들(예: AI 요약 고도화, 키워드 역사 큐레이션, 트렌드 분석 등)은 별도 로드맵으로 관리하며, 현재 구축된 **Admin 시스템**을 기반으로 차근차근 확장해나갈 예정입니다 [60].

---

[1] [3] [4] 1005_A쿼리계획execution_plan.pdf
file://file-1W7XdBQtN4y9u8KV41BXun

[2] [5] [11] [14] [15] [16] [17] [18] [19] [20] [21] [27] [28] [29] [30] [31] [33] [34] [35] [36] [37] [38] [39] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [55] [56] [57] [58] [60] 1010_1_나머지 계획_쿼리저널 관리자 시스템 개선 작업 인수인계 보고서.pdf
file://file_0000000070e46230a8784b27c8c24d4d

[6] 1005_1A_개선방안report.pdf
file://file-HzdZ5Qgkfa9y6poNMdPG28

[7] [24] [26] [32] [59] 1004_3A작업계획report.pdf
file://file-S3G4TQFPei3GeAJcXjN6B3

[8] Deploy to Cloud Run with GitHub Actions | Google Cloud Blog
https://cloud.google.com/blog/products/devops-sre/deploy-to-cloud-run-with-github-actions/

[9] [10] GitHub - google-github-actions/deploy-cloudrun: A GitHub Action for deploying services to Google Cloud Run.
https://github.com/google-github-actions/deploy-cloudrun

[12] Configure environment variables for services | Cloud Run | Google Cloud
https://cloud.google.com/run/docs/configuring/services/environment-variables

[13] [23] Configure secrets for services | Cloud Run | Google Cloud
https://cloud.google.com/run/docs/configuring/services/secrets

[22] [25] 1003_4UI.pdf
file://file-Gcxm7FdLSYTHtsLZnDwtiJ

[40] [54] 1006_A 최신 쿼리저널 개발 방향.pdf
file://file_0000000009346208b3d423d008c9620e