ChatGPT

QualiJournal 관리자 시스템 최종 단계별 작업 가이드북

QualiJournal 관리자 시스템의 주요 개선 작업 중 일부는 이미 구현되었지만, **일부 핵심 기능들은 아직 미완료 상태**입니다. 지금까지 **게이트 임계값 조절 슬라이더**, **KPI 지표 자동 새로고침**, **ADMIN 토큰 기반 인증 보안 적용** 등의 개선이 이루어졌고 [1], **다크/라이트 모드 테마 통합**, **비동기 API 폴백 처리**, **CSV 인코딩 문제 수정**, **일별 아카이브 관리 로직 구축** 등의 보완 작업도 완료되었습니다 [2]. 이제 남은 작업들(일일 보고서 생성, 뉴스 요약·번역, 결과물 Export UI 연동, 테스트/문서화, 보안 설정, 환경변수 구성, DB 연동 등)을 수행하여 **최종 안정성과 기능 완성**을 달성해야 합니다 [3] [4]. 이 가이드북은 **백엔드 개발자와 운영자**를 대상으로, 이러한 **남은 과제들을 단계별로 실행**할 수 있도록 안내합니다. 각 단계에는 **목적**, **필요 파일**, **CLI/API 예시**, **예상 결과**, **주의사항**이 포함되어 있으므로 순서대로 따라하며 최종 작업을 완수하시기 바랍니다.

단계 1: 환경 설정 및 준비

목적: 백엔드/프론트엔드 실행에 필요한 **환경변수**를 설정하고 운영 준비를 합니다. 특히 ADMIN 토큰과 번역 API 키, (필요한 경우 DB 연결정보) 등을 `.env` 파일과 클라우드 환경에 올바르게 구성하여 **보안 설정**을 완료합니다.

필요 파일: 프로젝트 루트의 `.env` **파일** (또는 해당 환경변수 설정을 대체하는 구성).
예시 `.env` 내용:

```
ADMIN_TOKEN="your-admin-token-12345"
API_KEY="your-translation-api-key"      # Papago 또는 Google Translate 등
DB_URL="postgresql://user:pass@host:5432/qualidb"  # (선택) DB 사용 시
```

CLI/API 예시:
- **로컬 환경에서 .env 적용 및 서버 실행:**

```
# .env 파일 로드 (필요 시 python-dotenv 패키지 활용 가능)
source .env
# 환경변수가 로드된 상태에서 서버 실행 (예: uvicorn)
uvicorn app.main:app --host 0.0.0.0 --port 8000
```

- **Cloud Run 환경변수 설정 (CLI 예시):**

```
# Cloud Run 서비스 업데이트 시 환경변수 지정
gcloud run services update qualijournal-admin \
 --update-env-vars ADMIN_TOKEN=<발급한 토큰>,API_KEY=<번역API키>,DB_URL=<DB연결문자열
>
```

(또는 Cloud Run 콘솔의 **Variables** 섹션에서 `ADMIN_TOKEN`, `API_KEY`, `DB_URL` 등을 수동으로 등록할 수 있습니다 [5].)

예상 결과:
- 로컬에서는 `.env`에 설정한 환경변수가 서버 구동 시 자동으로 반영되고, Cloud Run 등 배포 환경에서는 해당 변수

가 **주입**됩니다 [6] . 서버 시작 로그에 ADMIN_TOKEN, DB_URL 등의 설정 로드 여부가 표시될 수 있습니다.
- Admin API 호출 시 **인증 토큰**을 요구하도록 백엔드 미들웨어가 동작하며, 프론트엔드도 이 토큰을 알고 있어야 정상 호출됩니다 [7] .
- (DB 연동 시) 서버 구동 시 **DB 연결 시도** 로그가 나타나거나 `/api/status` 호출 시 DB 기반 통계를 반환하는 등 동작 변화가 있을 수 있습니다 [8] .

주의사항:
- **절대로** `.env` **파일을 원격 저장소에 커밋하지 말고**, 프로젝트 `.gitignore` 에 포함시켜야 합니다 [9] . 민감한 키(ADMIN_TOKEN, API_KEY, DB_URL 등)는 코드에 하드코딩하지 않고 **환경변수**로 관리합니다.
- **ADMIN_TOKEN** 값은 추측이 어려운 **충분히 복잡한 랜덤 문자열**로 생성합니다 [7] . 토큰을 변경한 경우 **프론트엔드에서 사용하는 토큰도 반드시 동일하게 업데이트**해야 합니다 [10] .
- **API_KEY**는 Papago, Google Translate 등의 **번역 API 키**를 사용하며, 설정하지 않을 경우 요약문은 영문으로만 출력되고 번역이 생략됩니다 [11] . 번역 기능을 활용하려면 해당 API의 사용 한도 및 과금 정책을 확인하십시오.
- **DB_URL**은 향후 JSON 저장 방식에서 데이터베이스로 전환할 때 사용합니다 [12] . 현재 시스템은 기본적으로 파일(JSON)에 데이터를 저장하지만, DB 사용을 계획한다면 **DB 인스턴스 설정**(예: PostgreSQL 생성, 외부 접속 허용, 방화벽 규칙 등)을 선행해야 합니다 [13] . 배포 환경에서 DB를 사용할 경우 VPC 및 인증 정보가 올바르게 설정되었는지 확인하고, **초기 마이그레이션** 시 충분한 테스트와 백업 후 진행하십시오 [14] .

단계 2: 일일 보고서 생성 기능 구현

목적: 하루 한 번 발행되는 **일일 뉴스 보고서**(특별호)의 자동 생성 기능을 완성합니다. 기존에 `tools/make_daily_report.py` 스크립트를 수동 실행하여 Markdown 리포트를 생성하던 것을, **백엔드 API**와 **관리자 UI 버튼**으로 트리거할 수 있도록 개선합니다. 이를 통해 운영자가 **클릭 한 번으로** 일일 보고서를 생성하고 열람할 수 있게 됩니다.

필요 파일:
- 백엔드: `tools/make_daily_report.py` (일일 보고서 생성 로직을 담은 스크립트)
- 백엔드: `app/main.py` 또는 `app/routes.py` 등 (새로운 `POST /api/report` 엔드포인트 추가)
- 프론트엔드: 관리자 대시보드 페이지 (예: `Report 생성` 버튼 UI 추가 및 API 호출 연동)

CLI/API 예시:
- **REST API 호출 (curl):** 일일 보고서 생성 트리거

```
curl -X POST "https://<배포도메인>/api/report" \
  -H "Authorization: Bearer ${ADMIN_TOKEN}"
```

※ 요청 본문은 필요 없으며, 헤더에 인증 토큰만 포함하면 현재 날짜의 보고서 생성을 트리거합니다.
- **REST API 호출 (fetch):** 일일 보고서 생성 트리거

```
fetch("/api/report", {
  method: "POST",
  headers: { "Authorization": "Bearer " + adminToken }
})
.then(res => res.json())
.then(data => console.log("Report Generation Result:", data));
```

※ `adminToken` 은 프론트엔드에 저장된 ADMIN 토큰 값입니다. 성공 시 `data` 에는 보고서 생성 결과나 경로 등이 반환될 수 있습니다.

예상 결과:
- **백엔드**: `POST /api/report` 엔드포인트가 호출되면 `make_daily_report.py`의 로직을 실행하여 금일 키워드에 대한 Markdown 보고서를 생성합니다. 실행 결과로 생성된 Markdown 내용(또는 파일 경로)이 API 응답으로 반환되거나, 서버 `archive/` 디렉토리에 날짜별 파일로 저장됩니다. 예를 들어 `archive/2025-10-12_report.md`와 같은 파일이 생성될 수 있습니다.
- **프론트엔드**: 운영자는 대시보드에서 **"일일 보고서 생성" 버튼**을 클릭해 이 API를 호출할 수 있으며, 성공 시 화면에 "보고서 생성 완료" 등의 메시지를 보여주거나 곧바로 생성된 Markdown을 표시하는 UI를 구현할 수 있습니다. (예: 새 탭으로 Markdown HTML 프리뷰 페이지를 열거나, 다운로드 링크 제공 등)
- **응답 예시**: 보고서 생성이 성공하면 API 응답으로 `{ "success": true, "report_file": "2025-10-12_report.md" }` 또는 `{ "content": "<Markdown text>..." }` 등의 JSON을 반환할 수 있고, 프론트엔드는 이를 받아 사용자에게 결과를 알립니다.

주의사항:
- **API 엔드포인트 구현:** 기존에 `GET /api/report` 엔드포인트가 일부 존재했으나 보고서 생성 트리거 역할을 하지 못했으므로, **새로운 POST 메서드**를 구현해야 합니다 [15]. 이때 해당 함수에서는 `make_daily_report.py`의 핵심 함수를 호출하고, 예외 처리(보고서 생성에 필요한 데이터 부족 등)를 포함해야 합니다.
- **보고서 중복 생성 방지:** 같은 날에 여러 번 보고서 생성 요청이 오면 **중복 파일 생성**이나 **데이터 충돌**이 발생할 수 있습니다. 개선안 보고서에서는 동일 키워드로 여러 번 발행 시 **시간까지 포함한 파일명**을 사용하고, **archive 폴더 용량 관리를 위해 백업/압축**을 권장합니다 [16]. 따라서 파일명에 타임스탬프를 추가하거나 기존 파일을 덮어쓰지 않도록 처리하고, 장기적으로 archive 폴더 정리 정책을 수립하십시오.
- **UI 피드백:** 보고서 생성에는 여러 단계의 데이터 수집/가공이 포함되어 **수 초 이상의 시간이 소요**될 수 있습니다. 따라서 UI 버튼 클릭 시 로딩 스피너 또는 진행 상황을 표시하고, **생성 완료 후 성공/실패 여부**를 명확히 알려주는 것이 좋습니다. 예를 들어 "생성 중..." 안내와 함께, 완료 시 "보고서가 생성되었습니다" 또는 오류 발생 시 원인을 알리는 메시지를 출력합니다.
- **권한 및 인증:** 이 API 역시 ADMIN 권한이 필요한 관리자 전용 기능이므로, **호출 시 인증 헤더가 없으면 401 Unauthorized를 반환**해야 합니다 [17]. 테스트 환경에서 토큰 없이 호출하여 401이 나오는지 확인하고, 올바른 토큰으로 호출 시 200 OK와 결과가 나오는지 반드시 검증합니다.
- **데이터 준비 확인:** 보고서 생성을 하기 전에, **당일 키워드의 기사 수집 및 편집 작업**이 완료되었는지 확인해야 합니다. 승인된 기사 수가 기준치에 못 미르면 보고서가 불완전할 수 있으므로, **보고서 생성 전에 /api/status**로 현재 `selection_approved`(승인 기사 수)와 `gate_required`(최소 필요 수)를 비교하여 부족할 경우 생성 요청을 **보류**하는 것이 바람직합니다 [18].

단계 3: 뉴스 카드 요약·번역 기능 구현

목적: 수집된 뉴스 **기사 카드들의 요약 및 번역** 기능을 백엔드 API와 UI에 추가합니다. 이 기능은 두 가지로 구분됩니다: (1) **키워드 전체 기사 요약** – 해당 키워드로 수집된 **모든 기사**에 대한 요약문 생성 및 (선택적으로) 번역, (2) **최종 선정본 기사 요약** – 편집자가 승인하여 발행 대상이 된 **선정본 기사들**에 대한 요약/번역. 이를 통해 운영자는 **버튼 클릭만으로** 대량의 기사에 대한 핵심 요약문과 번역본을 얻을 수 있어 편집 효율을 높입니다.

필요 파일:
- 백엔드: `tools/enrich_cards.py` (뉴스 카드 요약/번역 스크립트; 이미 존재하지만 수동 실행됨)
- 백엔드: `app/main.py` 또는 API 라우터 파일 (새로운 `POST /api/enrich/keyword` 및 `POST /api/enrich/selection` 엔드포인트 추가)
- 프론트엔드: 관리자 대시보드에서 **"요약/번역"** 관련 UI (예: `"키워드 전체 요약"` 버튼, `"선정본 요약"` 버튼 등)

CLI/API 예시:
- **REST API 호출 (curl)** – 키워드 전체 기사 요약/번역:

```
curl -X POST "https://<배포도메인>/api/enrich/keyword" \
  -H "Authorization: Bearer ${ADMIN_TOKEN}"
```

(모든 수집 기사를 대상으로 요약/번역 수행)
- **REST API 호출 (curl)** – 선정본 기사 요약/번역:

```
curl -X POST "https://<배포도메인>/api/enrich/selection" \
  -H "Authorization: Bearer ${ADMIN_TOKEN}"
```

(편집자가 **승인한 기사들만** 대상으로 요약/번역 수행)
- **REST API 호출 (fetch)** – 예시

```
// 키워드 전체 기사 요약/번역 요청
fetch("/api/enrich/keyword", {
  method: "POST",
  headers: { "Authorization": "Bearer " + adminToken }
}).then(res => res.json())
  .then(data => console.log("Keyword enrich result:", data));

// 선정본 기사 요약/번역 요청
fetch("/api/enrich/selection", {
  method: "POST",
  headers: { "Authorization": "Bearer " + adminToken }
}).then(res => res.json())
  .then(data => console.log("Selection enrich result:", data));
```

※ 요약/번역 API의 경우 요청 본문없이 POST만으로 동작하되, 필요에 따라 쿼리스트링이나 본문으로 **목표 언어** 등을 지정할 수 있습니다. (예: `POST /api/enrich/keyword?lang=ko`와 같이 구현 가능)

예상 결과:
- **백엔드**: 새로운 `/api/enrich/keyword` 호출 시 `enrich_cards.py`를 통해 **모든 수집 기사**의 본문 요약을 생성하고, `/api/enrich/selection` 호출 시 **선정된 기사만** 요약하도록 구현합니다[19]. 요약 결과는 각 기사 카드 데이터에 **요약문 필드**(예: `summary`)와 **번역문 필드**(예: `translation`)를 추가하거나, 별도의 결과 파일(`enriched_<date>.json` 또는 Markdown 등)로 출력됩니다.
- 만약 번역 API 키(`API_KEY`)가 설정되어 있다면, **영문 요약문을 한국어 등 목표 언어로 번역**하여 결과에 포함합니다. 키가 없거나 설정되지 않은 경우에는 **요약은 영문으로만** 수행되고 번역 단계는 생략됩니다[11].
- **프론트엔드**: UI에 "요약/번역" 기능 버튼이 노출됩니다. 예를 들어 키워드별 뉴스 목록 상단에 `"전체 기사 요약"` 버튼을 추가하고, 클릭 시 `/api/enrich/keyword`를 호출하여 프로세스를 시작합니다. 성공 시 결과 요약문을 별도의 창이나 **미리보기 모달**에 표시하거나, 기사 카드 목록에 요약문을 함께 띄워 편집 참고자료로 활용할 수 있습니다. 선정본 요약의 경우 최종 발행본에 포함될 기사들에 대한 **국문 번역본 생성**이 주된 목적이므로, 번역 완료 후 바로 **Markdown 보고서에 반영**하거나 다운로드 가능하게 처리할 수 있습니다.

주의사항:
- **엔드포인트 구현:** 현재 서버에 `/api/enrich/keyword` 및 `/api/enrich/selection` 엔드포인트가 **구현되어 있지 않으므로**, 해당 POST API를 직접 추가해야 합니다[19]. 두 API 모두 장시간 실행될 가능성이 있으므로 **비동기 처리**나 **백그라운드 작업 큐** 적용을 고려할 수 있습니다. 단, 현재 시스템 구조상 간단히 구현하려면 요청을 받아 즉시 스크립트를 실행하고, 완료 후 응답을 주는 **동기 방식**으로 시작할 수 있습니다.

- **번역 API 키 및 호출 한도:** Papago나 Google 번역 API를 사용할 경우 1일 호출 한도와 속도 제한이 있습니다. 뉴스 기사 15개에 대해 한번에 번역 시 **요금**이 발생하거나 **지연**이 생길 수 있으므로, 필요한 경우 일부 기사만 번역하거나 캐싱 전략을 고려하십시오. 또한 API 키 노출에 유의하고, 키는 **환경변수로 관리**합니다 (Step 1 참고).
- **UI/UX:** 요약/번역 작업은 수십 개 기사에 대해 NLP를 수행하므로 **수십 초 정도의 시간**이 걸릴 수 있습니다. UI에서는 해당 버튼 클릭 시 **진행 상황을 표시**하고, 작업 완료 전까지 사용자가 다른 작업을 병행할 수 있도록 해야 합니다. 예를 들어 "요약 작업 진행중... (0/15 완료)"처럼 진행률을 표시하면 운영자가 대기 시간을 예상하기 용이합니다.
- **결과 활용:** 요약/번역 결과물은 최종 발행본 콘텐츠의 품질을 높이기 위해 사용됩니다. 따라서 **요약문의 정확성 검토**가 필요할 수 있으며, 생성된 번역문은 원문 뉘앙스와 맞는지 편집자가 확인해야 합니다. 편집자는 필요 시 번역문을 편집하거나 요약 결과를 일부 수정할 수 있는 UI(예: 요약문 편집 필드)를 활용하도록 계획합니다. 현재는 1차적으로 자동생성에 중점을 두고, **추후 "human-in-the-loop"** 방식으로 편집 수정 기능을 추가하는 것도 고려해 볼 수 있습니다 [20].
- **인증 및 로그:** 본 기능도 ADMIN 전용이므로 인증 미들웨어가 적용되고, **요약 진행 중 발생하는 예외**를 서버 로그에 남겨 추후 디버깅할 수 있도록 합니다. 예를 들어 외부 API 실패, 요약 모듈 에러 등의 상황을 로그로 기록하고, 프론트엔드에는 "요약 실패: 원문 길이 초과" 등 이해하기 쉬운 오류 메시지를 반환합니다.

단계 4: 결과물 Export UI 연동

목적: 최종 발행본 결과물(특별호 콘텐츠)을 **Markdown**과 **CSV** 파일로 쉽게 추출할 수 있도록, 기존에 구현된 Export API에 **프론트엔드 UI 연결**을 추가합니다. 운영자는 이를 통해 발행 결과를 다운로드 받아 아카이빙하거나, 외부 분석에 활용할 수 있습니다. (예: CSV로 다운로드하여 엑셀로 열거나, Markdown 파일을 이메일로 공유 등)

필요 파일:
- 백엔드: (구현 완료) `GET /api/export/md`, `GET /api/export/csv` 엔드포인트 – 최종 발행본 Markdown 및 CSV를 생성해 반환함 [4].
- 프론트엔드: 관리자 대시보드 UI에 **"Export Markdown"**, **"Export CSV" 버튼 또는 메뉴** 추가.

CLI/API 예시:
- **REST API 호출 (curl)** – Markdown Export:

```
curl -H "Authorization: Bearer ${ADMIN_TOKEN}" \
  -o "latest_report.md" "https://<배포도메인>/api/export/md"
```

(요청에 성공하면 `latest_report.md` 파일로 저장) [21]
- **REST API 호출 (curl)** – CSV Export:

```
curl -H "Authorization: Bearer ${ADMIN_TOKEN}" \
  -o "latest_report.csv" "https://<배포도메인>/api/export/csv"
```

(성공 시 `latest_report.csv` 다운로드)
- **REST API 호출 (fetch)** – CSV Export 예시:

```
fetch("/api/export/csv", {
 headers: { "Authorization": "Bearer " + adminToken }
})
.then(res => res.blob())
.then(blob => {
 // 다운로드를 위해 Blob을 파일로 변환
 const url = window.URL.createObjectURL(blob);
```

```
  const a = document.createElement("a");
  a.href = url;
  a.download = "latest_report.csv";
  document.body.appendChild(a);
  a.click();
  a.remove();
  window.URL.revokeObjectURL(url);
});
```

※ fetch로 파일 응답을 받을 때는 위와 같이 Blob으로 처리한 후 가상의 링크를 클릭하여 다운로드를 트리거합니다.

예상 결과:
- **백엔드**: `/api/export/md` 호출 시 최신 발행본을 Markdown 형식 문자열로 응답하고, `/api/export/csv`는 CSV 파일 내용을 응답으로 제공합니다. 응답 헤더에는 `Content-Type: text/markdown` 또는 `text/csv` 및 `Content-Disposition: attachment; filename=...` 등이 설정되어 **브라우저가 파일로 인식**하도록 합니다. (이 부분은 백엔드 구현에 따라 다르며, 필요시 FastAPI의 `FileResponse` 등을 사용하여 다운로드 처리합니다.)
- **프론트엔드**: UI에 "Export" 섹션이 나타나고, 운영자가 **버튼 클릭으로 파일 다운로드**를 수행할 수 있습니다. 예를 들어 "Export CSV" 버튼을 누르면 즉시 `latest_report.csv` 파일 다운로드가 시작되거나, 위 fetch 코드처럼 스크립트로 Blob을 생성해 저장할 수 있습니다. "Export Markdown" 버튼도 유사하게 동작하여 `.md` 파일을 내려받습니다.
- **CSV 파일 내용**: CSV의 헤더 행에는 기사 제목, URL, 출처, 요약문 등 필드가 포함되며, 본문 중 쉼표 등은 적절히 인용처리됩니다. **BOM(Byte Order Mark)**이 추가되어 있어 Excel로 열 때 한글이 깨지지 않습니다 [22].
- **Markdown 파일 내용**: Markdown에는 발행된 기사 카드들이 순서대로 나열되며, 마크다운 문법 (제목, 리스트, 링크 등)이 적용된 형태입니다. 이를 마크다운 뷰어나 에디터로 열면 기사 목록을 손쉽게 열람할 수 있습니다.

주의사항:
- **UI 연동 미구현 상태 확인:** 현재 Export API는 백엔드에 존재하지만 프론트엔드에 **버튼이 없어 사용 불가한 상태**입니다 [4]. UI에 반드시 해당 기능을 노출해야 하며, 위치는 대시보드 상단 또는 발행 관리 화면 등 적절한 곳에 배치합니다.
- **다운로드 처리:** 프론트엔드에서 파일 다운로드를 구현할 때, `fetch`의 응답을 바로 처리하지 않고 단순 `<a href="/api/export/csv" download>` 링크로 걸 경우 **인증 토큰 헤더가 누락**될 수 있으므로 주의합니다. 반드시 **XHR/fetch를 사용하여 헤더 포함 요청**을 보내거나, 백엔드에서 토큰을 쿼리파라미터 등으로 받을 수 있는 경우에만 링크를 직접 사용합니다. 보안을 위해 가급적 헤더 인증 방식을 유지하세요.
- **인코딩 검증:** CSV 한글 깨짐 문제가 해결되었는지 최종 확인합니다. 다운로드된 CSV를 메모장이나 Excel로 열어 **한글이 올바르게 표시**되는지 확인하세요 (만약 깨질 경우 백엔드에서 UTF-8 BOM이 누락된 것이므로, 파일 시작에 `\ufeff`를 추가해야 함). Markdown은 인코딩 문제가 적지만, Markdown 내 링크나 이미지 경로 등이 제대로 반영되는지 검사합니다.
- **Export 데이터 활용:** Export 받은 파일은 **운영상의 백업** 및 **추후 데이터 분석**에 요긴합니다. 가이드라인에 따라 운영자는 발행 후 정기적으로 CSV/Markdown을 내려받아 별도 저장소에 백업해두는 것이 좋습니다 [23]. 특히 Cloud Run이 스테이트리스여서 컨테이너가 재시작되면 내부 파일이 유실될 수 있으므로, 중요한 발행본은 수동 백업해두는 것을 권장합니다.

단계 5: 통합 테스트 및 문서화 작업

목적: 새로 추가된 기능들을 포함하여 전체 시스템의 **통합 테스트**를 자동화하고, 운영/개발 **문서**를 최종 갱신합니다. Pytest 등을 활용한 테스트 코드를 작성해 기능별 동작을 검증하고, README 및 관리자 매뉴얼을 업데이트하여 인수인계와 운영에 차질이 없도록 합니다. 이를 통해 **안정성 향상** 및 **회귀 오류 방지**를 목표로 합니다.

필요 파일:
- 테스트 코드: `tests/` 디렉토리에 각 기능별 테스트 스크립트 (`test_report.py`, `test_enrich.py`, `test_export.py` 등)
- 문서 파일: 프로젝트 `README.md`, 운영 가이드북, API 스펙 문서 등 (업데이트 대상)
- 기타: GitHub Actions 워크플로우 (CI 설정 파일), 요구 시 Pytest 환경설정 (`pytest.ini`)

CLI/API 예시:
- **테스트 실행 (로컬):**

```
pytest -v
```

(모든 테스트 케이스를 실행하여 통과 여부 확인. `-v` 옵션은 상세한 결과 출력)
- **테스트 실행 (특정 케이스):**

```
pytest tests/test_report.py::test_generate_report_without_token
```

(예: `test_generate_report_without_token` 이름의 케이스만 실행)
- **GitHub Actions CI 예시:** (workflow YAML 일부)

```
- name: Run tests
  run: pytest -q
- name: Build and Deploy to Cloud Run
  uses: google-github-actions/deploy-cloudrun@v0
  with:
    image: gcr.io/$PROJECT_ID/qualijournal-admin:${{ github.sha }}
    service: qualijournal-admin
    env_vars: ADMIN_TOKEN=${{ secrets.ADMIN_TOKEN }},API_KEY=${{ secrets.API_KEY }}
```

(코드 푸시 시 자동으로 테스트를 실행하고, 성공하면 Cloud Run에 배포하는 CI 파이프라인 예시)

예상 결과:
- `pytest`를 실행하면 **모든 테스트 케이스가 성공**해야 합니다. 예를 들어 `/api/report` 관련 테스트에서 토큰 없이 호출 시 401을 반환하고 [17], 올바른 토큰으로 호출 시 200과 보고서 내용이 생성됨을 확인합니다. `/api/export` 테스트에서는 CSV 응답에 BOM (`\ufeff`)이 포함되어 있고 Markdown/CSV에 예상 필드와 텍스트가 들어있는지 검증합니다 [22].
- 테스트 커버리지 측면에서는 **신규 기능 (보고서 생성, 요약, 내보내기)** 뿐만 아니라, 기존 기능(기사 수집, 키워드 변경 등)에 대한 **회귀 테스트**도 포함되어야 합니다 [24]. 이를 통해 최근 코드 변경이 다른 부분에 영향을 주지 않음을 보장합니다.
- 문서화 측면에서는 **README.md**에 환경변수 사용법(.env 예시, Cloud Run 설정법 등)과 **배포 절차** 요약, ADMIN_TOKEN 설정 방법, API_KEY 준비 방법, DB_URL 사용 시 주의사항 등이 추가됩니다 [25]. 또한 **최종 API 명세**, 프론트엔드 연동 방법 (어느 화면에서 어떤 버튼으로 해당 API 호출 가능 여부) 등을 최신 상태로 반영합니다.
- 최종 **운영 가이드북**에는 새로 구현된 기능(보고서 생성, 요약/번역, Export)에 대한 사용방법이 반영되고, **체크리스트** 항목들도 업데이트됩니다 (예: 매일 발행 전 요약 기능 사용 여부, Export 백업 등).

주의사항:
- **테스트 격리:** 테스트 코드 작성 시 실제 외부 API나 DB에 영향이 없도록 **모의 객체(Mock)**나 **monkeypatch**를 활용합니다. 예를 들어 번역 API 호출 부분은 별도의 함수를 분리하고 테스트에서 해당 함수를 가짜 구현으로 패치하여 실제

API 호출 없이도 동작을 검증합니다. ADMIN_TOKEN도 테스트 시작 전에 `os.environ['ADMIN_TOKEN'] = "test-token"` 등으로 설정하거나 테스트 클라이언트 생성 시 주입하여, **실제 토큰 값을 노출하거나 사용하지 않도록** 합니다 [26].
- **시드 데이터:** 기능 테스트를 위해 **예상되는 입력 데이터**를 준비해야 합니다. 예를 들어 보고서 생성 테스트 전에 `selected_articles.json`에 임시 기사 데이터를 넣거나, 관련 함수를 Monkeypatch하여 일정한 결과를 반환하게 할 수 있습니다. 이러한 방법으로 외부 의존성을 최소화하면서도 시나리오를 검증합니다.
- **CI 환경:** CI 파이프라인에서 테스트를 수행할 때 환경변수가 필요하면, CI 설정에 **시크릿**으로 ADMIN_TOKEN, API_KEY 등을 등록해 사용합니다 [27]. 테스트 단계에서만 사용하는 임시 값이어야 하며, 실제 운영 값과 다르게 세팅해도 무방합니다. 또한 CI에서 병렬로 여러 테스트를 돌릴 경우를 대비해, 테스트 간 **공유 상태(파일, DB 등)**가 없도록 주의합니다.
- **문서 최신화:** 문서 수정 시 **변경 내역**을 잘 남기고, 버전 관리를 통해 팀원들과 공유하십시오. 특히 운영 가이드, 인수인계 문서에는 새로운 **REST API Endpoint 목록**, **사용 방법**, **환경 설정법** 등이 빠짐없이 기재되어야 합니다. 문서화는 추후 유지보수나 신규 인력 투입 시 중요 참고자료가 되므로 상세하고 정확하게 작성합니다.

단계 6: 보안 설정 강화 및 최종 구성 점검

목적: 관리자 시스템의 **보안 설정**을 최종적으로 강화하고, 운영 관점에서 민감 정보 노출이 없도록 마무리합니다. 이미 ADMIN 토큰 기반의 인증 체계를 도입하여 **비인가 접근을 차단**하고 있으므로 [28], 여기서는 토큰/키 관리와 기타 보안 옵션들을 점검합니다. 또한 Cloud Run 등 배포 환경의 설정을 검토하여 **환경변수, 네트워크, IAM 권한** 등이 올바르게 구성되었는지 확인합니다.

필요 파일: (코드 변경보다는 설정 변경/확인 단계이므로 별도 파일은 없음)
- 환경 설정: `.env` 및 Cloud Run 환경변수 설정 화면
- GCP IAM & Cloud Run 설정: (필요 시) Cloud Run 서비스의 **인증 요구 설정**, VPC 연결 등

CLI/API 예시:
- **토큰 없이 API 호출 테스트:**

```
curl -i "https://<배포도메인>/api/status"
HTTP/1.1 401 Unauthorized
...
```

(인증 없는 요청에 401이 반환되는지 확인)
- **토큰 사용 API 호출 테스트:**

```
curl -i -H "Authorization: Bearer ${ADMIN_TOKEN}" "https://<배포도메인>/api/status"
HTTP/1.1 200 OK
{ "gate_required": 15, "selection_approved": 15, ... }
```

(정상 토큰으로 호출 시 200 OK와 응답 데이터 확인)
- **Cloud Run 서비스 설정 확인 (gcloud CLI):**

```
gcloud run services describe qualijournal-admin --
format="value(spec.template.spec.containers[0].env)"
```

(설정된 환경변수 목록을 확인하여 ADMIN_TOKEN, API_KEY 등이 포함되어 있는지 점검)

예상 결과:
- **ADMIN_TOKEN 보안성**: 모든 관리자용 API가 **Authorization 헤더**를 요구하며, 올바른 토큰을 제공하지 않으면 **401/403 오류**가 반환됨을 최종 확인합니다. (이미 기능 구현단계에서 테스트했지만, 운영 환경에서도 동일하게 적용됨을 재확인)
- **환경변수 노출 방지**: Cloud Run 환경변수 설정, GitHub Actions 시크릿 등으로 민감 정보가 안전하게 저장되고, **어느 곳에도 평문 노출되지 않음**을 보장합니다. 예를 들어 Cloud Run 로그에 환경변수가 출력되지 않고, `.env` 파일이 리포지토리에 존재하지 않는 상태여야 합니다.
- **클라이언트 보안**: 프론트엔드에서 ADMIN_TOKEN을 사용할 때, 브라우저 콘솔이나 네트워크 패널을 통해 토큰이 노출될 수 있습니다. 이는 피할 수 없지만, **클라이언트 코드를 배포할 때 소스 맵 제거** 등으로 쉽게 찾아볼 수 없게 하거나, IP 제한 등의 추가 조치를 검토합니다. (단, 근본적으로는 별도의 로그인 시스템을 도입하는 것이 안전하지만, 현 단계에서는 단일 토큰으로 간소화)
- **네트워크 및 IAM**: Cloud Run 서비스가 불필요한 경우 **공개로 설정되지 않도록** 검토합니다. 현재 ADMIN_TOKEN 인증이 있으므로 Cloud Run의 **--allow-unauthenticated** 설정은 크게 문제되지 않지만, 보다 안전하게는 Cloud Run 호출을 특정 IAM 사용자만 가능하게 설정한 뒤 프론트엔드에서 그 IAM로 호출하는 방법도 고려할 수 있습니다. 또한 DB를 사용한다면 Cloud Run의 VPC Access Connector가 제대로 붙어서 DB에 접속 가능한지, **방화벽 규칙**은 제한적인지 살펴봅니다.
- **의존성 업데이트 및 취약점 검사**: (선택) 프로젝트의 라이브러리 의존성을 최신으로 유지하고, `pip audit`이나 `safety` 같은 도구로 알려진 보안 취약점이 있는 패키지가 없는지 검사합니다. 필요 시 신속히 업데이트하거나 패치 버전을 적용합니다.

주의사항:
- **토큰 교체 주기:** 운영 정책상 정기적으로 ADMIN 토큰을 변경해야 할 수 있습니다. 토큰을 교체할 때는 **서버 환경변수와 프론트엔드 설정을 동시에 업데이트**하고, 배포를 통해 적용합니다. 이전 토큰은 즉시 폐기하고 로그에 노출되지 않도록 주의합니다.
- **로그 관리:** 서버 로그에 민감정보가 찍히지 않도록 합니다. 예를 들어 요청 로그에 전체 헤더를 남기지 않거나, DEBUG 모드 로그를 끈 채 운영합니다. 또한 Cloud Run 로그 및 애플리케이션 로그를 **주기적으로 모니터링**하여, 보안 이벤트(예: 반복되는 401 Unauthorized 시도 등)를 확인합니다.
- **HTTPS 사용:** Cloud Run 도메인은 기본적으로 HTTPS를 사용하므로 별도의 SSL 설정이 필요 없지만, 만약 커스텀 도메인을 매핑했다면 SSL 인증서가 제대로 적용되었는지 확인합니다. 모든 API 통신은 HTTPS 상에서 이뤄져야 하며, http 연결이 필요하다면 리디렉션 등으로 강제적으로 https를 사용하게 구성합니다.
- **미구현 보안 항목:** 다중 관리자 계정, OAuth 로그인, 세션 관리 등은 향후 구현 가능한 기능이지만 현 시점에서는 토큰 하나로 관리합니다 [29]. 이로 인해 토큰이 유출되면 위험하므로, **토큰 취급에 각별히 주의**하고 필요하면 IP 제한, 접근 VPN 등의 추가 보안을 검토하십시오.

단계 7: Cloud Run 배포 및 CI/CD 최종 마무리

목적: 변경된 코드와 기능들을 **최신 버전으로 배포**하고, 지속적인 배포가 가능하도록 **CI/CD 파이프라인**을 점검/구축합니다. 이 단계에서는 Docker 이미지를 빌드하여 Cloud Run에 배포하고, GitHub Actions 등 CI 설정을 최종 확인하여 **코드 푸시 -> 테스트 -> 배포** 흐름이 자동화되어 있는지 확인합니다. 이를 통해 운영자는 향후 코드 수정 시 수동 개입 없이도 안정적으로 서비스 업데이트를 할 수 있습니다.

필요 파일:
- Docker 관련: `Dockerfile` (애플리케이션 도커 이미지 빌드 정의)
- CI 관련: `.github/workflows/...yaml` (GitHub Actions 워크플로우 파일) 또는 Cloud Build 설정 파일
- GCP 배포: Cloud Run 서비스 (이미 생성되었을 경우 업데이트만 진행)

CLI/API 예시:
- **Docker 이미지 수동 배포 예시:**

```
# 1. 도커 이미지 빌드 (태그는 GCP Artifact Registry 또는 Container Registry 경로)
docker build -t asia.gcr.io/<GCP_PROJECT_ID>/qualijournal-admin:latest .
# 2. 이미지 레지스트리에 푸시
docker push asia.gcr.io/<GCP_PROJECT_ID>/qualijournal-admin:latest
# 3. Cloud Run 서비스에 이미지 배포 (환경변수 설정 포함)
gcloud run deploy qualijournal-admin \
 --image asia.gcr.io/<GCP_PROJECT_ID>/qualijournal-admin:latest \
 --region=asia-northeast3 \
 --platform=managed \
 --update-env-vars ADMIN_TOKEN=${ADMIN_TOKEN},API_KEY=${API_KEY},DB_URL=$
{DB_URL}
```

(위 명령에서 `<GCP_PROJECT_ID>`와 지역, 환경변수 값 등을 실제 값으로 대체하십시오.)
- **GitHub Actions 배포 트리거:**

```
git push origin main
```

(main 브랜치에 푸시하면, CI가 자동으로 테스트 후 Cloud Run으로 배포 실행. CI가 올바르게 구성된 경우)

예상 결과:
- **Cloud Run 배포 성공:** 새로운 기능(API 및 UI)이 포함된 컨테이너 이미지가 Cloud Run에 배포되어, **URL을 통해 정상적으로 서비스 동작**함을 확인합니다. 배포 로그나 Cloud Run의 Revision History에 최근 배포 시각과 이미지 해시가 반영됩니다.
- Cloud Run **환경변수** 설정이 최신 버전으로 유지됩니다. (만약 CI에서 env_vars를 전달하지 않았다면, 이전 단계 Step 1에서 수동 설정한 값이 계속 유지됨)
- **CI/CD 파이프라인**: GitHub Actions 또는 Cloud Build 등에서 **모든 테스트가 통과하면 자동 배포**되도록 동작합니다. 이를 확인하기 위해 의도적으로 코드를 약간 수정하여 푸시한 뒤, Actions 로그에서 테스트 -> 빌드 -> 배포 단계가 순차적으로 진행되고 성공하는지 살펴봅니다 [30] . 만약 테스트 실패 시 배포가 중단되는지, 배포 성공 후 Cloud Run 서비스가 새로운 이미지로 업데이트되는지를 확인합니다.
- **모니터링**: 배포 후 **Cloud Run 로그와 모니터링** 페이지를 확인하여 오류가 없는지 검사합니다. 예를 들어 새 엔드포인트(`/api/report`, `/api/enrich/...` 등) 호출 로그에 200 OK로 처리되었는지, 또는 오류/예외가 기록되지 않았는지 살펴봅니다. 또한 Cloud Run의 CPU/메모리 사용량이 이전과 크게 달라지지 않았는지 모니터링하여, 필요하면 서비스를 리소스 스펙을 조정합니다 [31] .

주의사항:
- **Dockerfile 최신화:** 새로운 패키지 설치나 코드 구조 변화가 있었다면 Dockerfile을 업데이트하고 `docker build`가 정상 동작하는지 미리 테스트합니다. 필요 없는 파일을 복사하거나 누락된 의존성이 없도록 Dockerfile 단계를 점검하세요.
- **배포 변수**: Cloud Run 배포 시 환경변수를 명령줄에 직접 넣는 경우, 쉘 이스케이프 및 보안에 유의합니다. CI에서는 **시크릿**으로 관리되지만 로컬 터미널에서는 history에 남을 수 있으므로, 가능하면 `--update-env-vars`에 값을 직접 넣기보다는 이미 콘솔에 설정되어 있는 경우 이 옵션을 생략하거나, CI를 통한 배포를 사용합니다 [27] .
- **트래픽 라우팅:** 새 revision이 배포된 후 Cloud Run의 **트래픽 할당**이 100% 신버전으로 갱신되었는지 확인합니다. 간혹 이전 리비전이 일정 트래픽을 유지하는 설정이 있을 수 있으니, 필요시 수동 조정합니다.
- **롤백 계획:** 최종 배포 이후 혹시 문제가 발견될 경우를 대비해, **이전 버전으로 롤백**하는 절차를 마련합니다. Cloud Run의 Revision History에서 이전 리비전을 재할당하거나, 문제가 된 최신 코드를 빠르게 수정하여 재배포할 수 있도록 준비합니다. CI/CD 파이프라인에도 긴급 중지 또는 롤백 스크립트를 마련하면 좋습니다.
- **배포 후 검증:** 배포 후 관리자 UI에서 **모든 신규 기능 버튼이 정상 동작**하는지 테스트합니다. 예를 들어 실제 UI에서 "일일 보고서 생성" 버튼 클릭 -> 성공 메시지 확인, "요약/번역" 버튼 -> 요약 결과 확인, "Export" 버튼 -> 파일 다운

로드 확인 등을 **실제 시나리오로 점검**합니다. 또한, 배포 후 API 엔드포인트에 대한 **스모크 테스트**(간단 호출 테스트)를 수행해 500 에러 등이 없는지 확인하세요.

최종 점검 체크리스트

마지막으로, **시스템 운영을 위한 주요 체크포인트**를 정리합니다. 아래 체크리스트를 따라 일일 운영 및 배포 후 점검을 수행하면, 누락된 부분 없이 안정적으로 시스템을 유지할 수 있습니다.

- [ ] **키워드 발행 전 기사 수 확인:** 매일 발행 전에 **승인된 기사 수**(`selection_approved`)가 **최소 필요 개수**(`gate_required`)를 만족하는지 점검합니다 [18]. 부족할 경우 추가 승인하거나 임계값을 조정하고, 발행 후 원상복구합니다.
- [ ] **품질 게이트 통과 여부:** `/api/status`의 `gate_pass` 값이 `false`이면 발행을 보류하거나 임계값을 낮춰 true로 만든 후 진행합니다 [32]. (발행 강행 시 품질 저하 우려)
- [ ] **수집 작업 완료 확인:** 해당 키워드의 **뉴스 수집이 정상 완료**되었는지 확인합니다. 예를 들어 `selected_keyword_articles.json` 수정 시간, `/api/status`의 `keyword_total` 기사 수 등을 보고 이상치(특정 소스 0건 등)가 없는지 체크합니다 [33].
- [ ] **대시보드 동기화 확인:** 관리자 UI의 KPI 통계 (누적 수집, 승인 건수 등)가 백엔드 `/api/status`와 일치하는지 확인합니다 [34]. 자동 새로고침이 안 되면 수동 새로고침을 통해 최신 상태로 갱신합니다.
- [ ] **일일 보고서 생성 기능 점검:** "일일 보고서 생성" 버튼을 눌러 보고서가 정상 생성되는지, 생성된 Markdown에 당일 기사들이 모두 포함되는지 확인합니다. 오류 발생 시 터미널/로그의 예외 메시지를 확인하고 조치합니다.
- [ ] **요약/번역 기능 점검:** "요약/번역" 버튼들을 눌러 요약문과 번역이 제대로 생성되는지 확인합니다. 생성된 요약이 기사별로 추가되었는지, 번역문은 자연스럽게 잘 나오는지 검토합니다. 번역 API 한도 소진 여부도 함께 확인하세요.
- [ ] **Export 다운로드 및 백업:** 발행 직후 **Export CSV/Markdown**을 실행하여 파일을 다운로드 받고, 이를 안전한 공간(예: 운영자 PC, 구글 드라이브)에 저장합니다 [23]. 다운로드한 파일을 열어 한글 깨짐이나 내용 누락이 없는지 검증합니다.
- [ ] **아카이브 용량 관리:** 서버의 `archive/` 폴더에 저장되는 일별 보고서/요약 파일이 누적됨에 따라 용량을 모니터링합니다 [35]. Cloud Run은 스테이트리스이지만 외부 스토리지를 연결한 경우, 주기적으로 오래된 파일을 압축 보관하거나 삭제합니다.
- [ ] **로그 및 모니터링:** Cloud Run **로그**를 통해 오류나 경고가 발생하지 않았는지 확인하고, **모니터링 지표**(CPU, 메모리, 요청 latency 등)를 주시합니다 [31]. 자원 사용량이 한계를 넘어서면 Cloud Run 인스턴스의 메모리/CPU 할당을 늘리는 것을 검토합니다.
- [ ] **토큰 및 접근 보안:** 프론트엔드와 백엔드의 ADMIN_TOKEN 값이 일치하는지 항상 확인합니다 [36]. 401 에러가 반복되면 토큰 불일치나 누락을 의심하고 바로 수정합니다. 외부에서 토큰 없이 접근 시도 로그가 있다면 원인을 파악하고 필요시 토큰을 변경하는 것도 고려합니다 [37].
- [ ] **데이터베이스 연동 (옵션):** DB_URL을 설정해 DB를 사용 중이라면, 배포 후 DB 연결에 문제는 없는지 (Cloud Run에서 VPC 연결 성공, 마이그레이션 오류 여부 등) 확인합니다 [14]. 테이블에 데이터가 잘 들어가는지, 쿼리 성능에 문제는 없는지 살펴봅니다.
- [ ] **CI/CD 동작 확인:** 최신 코드를 push 했을 때 GitHub Actions 등의 CI가 트리거되고, 테스트 통과 후 배포까지 자동으로 이루어지는지 확인합니다 [30]. CI 로그를 검토하여 오류 단계는 없는지 확인하고, 배포 완료 알림이나 상태도 모니터링합니다.
- [ ] **문서 및 인수인계:** 모든 변경사항이 README와 운영 가이드에 반영되었는지 마지막으로 점검합니다. 스크린샷이나 예시 값들이 오래되진 않았는지, 새로운 API 엔드포인트와 UI 사용법이 충분히 설명되었는지 확인합니다. 필요하다면 **사용자 매뉴얼**도 업데이트하여, 운영자가 UI에서 어떤 버튼을 눌러 어떤 결과를 얻을 수 있는지 쉽게 이해할 수 있게 합니다.
- [ ] **향후 개선 로드맵 확인:** 남은 장기 과제(예: 다중 사용자 권한, AI 추천 기능, 검색 기능 등)가 별도 로드맵으로 관리되고 있다면, 현재 버전 릴리스 후 해당 항목들을 정리합니다. 이번 배포로 인해 생긴 **아이디어나 TODO**도 기록해 두고, 차기 버전에 반영할 계획을 수립합니다.

以上の 단계들을 순차적으로 완료하면 QualiJournal 관리자 시스템의 모든 **남은 기능 구현과 안정화 작업이 마무리**됩니다. 이제 운영자는 일일 키워드 뉴스 발행 작업을 **UI 기반으로 편리하게 수행**할 수 있고, 개발팀은 **자동화된 테스트와 CI/CD**로 시스템 품질을 지속적으로 담보할 수 있습니다. 남은 과제들이 추후 차차 구현되더라도, 본 가이드북을 토대로 설정과 운영을 진행하면 무리 없이 현대적인 키워드 뉴스 플랫폼을 유지해나갈 수 있을 것입니다. 3 29

---

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 18 19 21 23 25 26 27 28 29 30 31 32 33 34 35 36 37 1011_4QualiJournal 관리자 시스템 운영 가이드북.pdf
file://file-CVSpPWa43WEvDV2r8Qoh3k

16 1005_1A_개선방안report.pdf
file://file-HzdZ5Qgkfa9y6poNMdPG28

17 22 24 1011_3QualiJournal 관리자 시스템 최종 가이드북.pdf
file://file-6Hu18vVrjQweSb5SZz6a6M

20 1004_3A작업계획report.pdf
file://file-S3G4TQFPei3GeAJcXjN6B3