ChatGPT

QualiJournal 관리자 시스템 Cloud Run 배포 통합 가이드북

이 가이드북은 FastAPI 기반 **QualiJournal 관리자(Admin) 시스템**을 Google Cloud Run 환경에 배포하고 운영하기 위한 종합 안내서입니다. FastAPI 백엔드와 단일 HTML/JS 프론트엔드를 대상으로 하며, 개발자 및 운영자가 최종 배포를 완료하는 데 필요한 설정, 기능 구현, 테스트, 배포 절차를 순서대로 설명합니다. 각 섹션에는 실행 절차, 예시 코드, 예상 결과, 주의사항이 포함되어 있습니다.

1. 환경 구성 및 .env 설정 방법

환경 변수 설정은 배포 전 필수 작업입니다. 관리자 기능에 필요한 비밀 키나 설정값(예: ADMIN_TOKEN, DB_URL, API_KEY 등)을 `.env` 파일이나 환경 변수로 관리해야 합니다. 다음 지침을 따릅니다:

- **.env 파일 생성:** 로컬 개발환경에서는 프로젝트 루트에 `.env` 파일을 만들어 필요한 환경변수를 키=값 형태로 저장합니다. 예시:

```
ADMIN_TOKEN="초기_관리자토큰_예시값"
API_KEY="번역_API_사용시_키"
DB_URL="sqlite:///app.db"  # (필요 시 데이터베이스 사용 예시)
```

`.env` 파일에는 관리자 토큰과 API 키 등 **민감 정보**를 포함하므로 절대 Git에 커밋하지 않도록 합니다[1]. `.gitignore`에 추가하고, 협업 시 별도로 안전하게 공유하세요. FastAPI 앱 시작 시 `python-dotenv` 등을 사용해 `.env`를 로드하거나, 직접 `os.getenv`로 값을 읽어 환경설정을 적용합니다.

- **ADMIN_TOKEN 설정:** 모든 관리자 API 호출에 사용할 **관리자 인증 토큰**을 생성하여 `.env`에 `ADMIN_TOKEN`으로 저장합니다. 충분히 길고 예측 불가능한 문자열로 만드세요 (예: UUID나 32자리 이상의 난수). 서버 기동 시 이 값을 읽어 전역 변수로 보관합니다[2]. 클라이언트 요청의 토큰과 일치하지 않으면 접근을 거부하도록 구현할 것입니다 (자세한 방법은 후술).

- **API_KEY 설정(번역 기능):** 뉴스 카드 번역에 외부 API를 사용할 경우 해당 서비스의 API 키를 `.env`에 추가합니다. 예를 들어 Google Cloud Translate API를 사용한다면 `API_KEY="YOUR_GOOGLE_TRANSLATE_KEY"` 형식으로 넣습니다. `enrich_cards.py` 스크립트가 이 값을 읽어 번역을 수행하므로, Cloud Run 배포 시에도 동일한 환경변수를 설정해야 합니다[3]. API 키가 없으면 번역 기능은 생략되거나 오류를 낼 수 있으니 사전에 발급 및 설정을 완료합니다.

- **기타 환경 변수:** 데이터베이스나 파일 경로 등의 설정이 있다면 역시 `.env`로 관리합니다. 예를 들어 SQLite를 쓰거나 파일 경로를 지정하는 경우 `DB_URL`이나 `DATA_PATH` 등을 정의합니다. 프로젝트 구조상 `config.json`이나 기타 설정 파일이 있다면, 거기에 넣어도 되지만 **비밀 값**은 `.env`/환경변수로 관리하는 것을 권장합니다.

- **Cloud Run 환경 변수 주입:** Cloud Run에서는 컨테이너 이미지 내에 `.env`를 포함하지 않고, 배포시 환경변수를 주입하는 방식을 권장합니다. gcloud CLI로 배포할 때 `--update-env-vars` 옵션을 사용하거나, 배포 후 `gcloud run services update`로 설정할 수 있습니다. 예시:

```
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 명령은 컨테이너 배포와 동시에 ADMIN_TOKEN, API_KEY, DB_URL 값을 Cloud Run 서비스에 주입합니다 [4]. `<GCP_PROJECT_ID>`와 실제 토큰/키/URL 값들은 여러분의 환경에 맞게 대체하세요. 배포 후 `gcloud run services describe qualijournal-admin` 명령이나 Cloud Run 콘솔의 **Variables & Secrets**에서 설정된 환경변수를 확인할 수 있습니다 [5].

- **주의:** ADMIN_TOKEN과 API_KEY 같은 값은 배포 환경에서도 노출되지 않도록 유의하세요. Cloud Run에 설정한 환경변수는 기본적으로 서비스 내부에서만 접근 가능하지만, 응답이나 로그 등에 실수로 출력하지 않도록 코드에서 주의해야 합니다. 예를 들어, `print(ADMIN_TOKEN)` 같은 디버그 코드는 제거합니다. 또한, Cloud Run에 **Allow unauthenticated invocations**를 활성화하더라도(=별도의 Cloud IAM 인증 없이 공개 접근 허용), 우리 애플리케이션 레벨에서 ADMIN_TOKEN 검증을 수행하므로 외부에 토큰이 알려지지 않는 한 안전합니다 [6]. 반대로 Cloud Run의 **인증 필요** 설정을 켜면 (`--no-allow-unauthenticated`), ADMIN_TOKEN이 있어도 Google IAM 인증을 통과해야 해서 이중 인증 구조가 됩니다. 일반적으로는 Cloud Run 서비스는 공개 호출 가능하도록 해두고, **앱 레벨에서 토큰 인증으로 보호**하는 방식을 사용합니다 [6].

요약하면, 로컬에서는 `.env` 파일로 환경을 설정하고, Cloud Run 배포시에는 해당 값을 환경변수로 지정합니다. 이러한 환경 설정이 완료되면 다음 단계로 서버 기능 구현 및 연동을 진행합니다.

2. 일일 보고서 자동 생성 기능

일일 보고서는 하루 동안 선정된 기사들을 모아 Markdown 형태의 리포트를 만들어주는 관리자 기능입니다. 이를 위해 백엔드에 **POST** `/api/report` 엔드포인트를 구현하고, 프론트엔드에 "일일 보고서 생성" 버튼을 연동합니다. 또한 보고서 생성 결과(파일 경로 또는 내용)를 사용자에게 제공하여 확인할 수 있게 합니다.

2.1 백엔드 엔드포인트 구현: POST `/api/report`

- **엔드포인트 개요:** 편집자가 승인한 기사들로 구성된 데일리 리포트(마크다운 문서)를 생성하는 API입니다 [7]. 호출 시 내부에서 `tools/make_daily_report.py` 스크립트를 실행하거나 해당 함수를 호출하여 현재까지 **승인된 기사 목록**(예: `selected_articles.json`)을 취합합니다 [8]. 그런 다음 Markdown 형식의 일일 보고서 파일을 생성하며, 기본 파일명은 `archive/reports/YYYY-MM-DD_report.md` 형태를 따릅니다 [9]. (특정 날짜나 키워드별로 운용한다면 파일명에 키워드 슬러그를 포함할 수도 있습니다. 예: `YYYY-MM-DD_<키워드>_report.md`.)

- **구현 절차:** FastAPI의 APIRouter에 `/api/report` POST 라우트를 추가합니다. 관리자 전용이므로 **토큰 인증**을 적용합니다. 예를 들어, `Depends(verify_admin_token)`을 사용하여 잘못된 토큰에는 401 Unauthorized 응답을 하도록 합니다. 엔드포인트 함수 내부에서는 `make_daily_report()` 함수를 호출하여 Markdown 컨텐츠를 생성합니다. 이 함수는 내부적으로 `selected_articles.json` 파일 (최종 승인 기사 목록)을 읽어 당일 보고서를 Markdown 문자열 또는 파일로 만들어 줍니다 [8].

```
from fastapi import APIRouter, Depends, HTTPException
from app.auth import verify_admin_token
from tools import make_daily_report
```

```
router = APIRouter()

@router.post("/api/report")
def create_daily_report(token: str = Depends(verify_admin_token)):
    try:
        report_path = make_daily_report()  # 당일 보고서 생성 -> 파일 경로 반환
    except Exception as e:
        # 오류 발생 시에도 가능하면 보고서 생성 결과를 제공
        return {"ok": False, "error": f"Report generation failed: {e}"}
    return {"ok": True, "path": report_path}
```

위 코드는 개념적 예시입니다. 실제 `make_daily_report()` 구현은 **승인된 기사 목록을 모아 Markdown 텍스트**를 만들고, `archive/reports/` 디렉토리에 해당 날짜 파일로 저장하도록 작성합니다 [9]. 예외 상황에서도 파일이 어느 정도 생성되도록 설계하는 것이 좋습니다. 예를 들어 데이터가 일부 없어도 "데이터 없음" 등의 내용을 넣어 리포트를 만들고, `ok` 필드로 성공/실패를 표시할 수 있습니다. 위 예시에서는 오류 발생 시 HTTP 200과 `{"ok": False, "error": "..."}`를 반환하도록 했지만, 치명적 오류라 리포트 생성 자체가 불가능한 경우에는 HTTP 500 에러를 발생시켜도 됩니다. 중요한 것은 **ADMIN_TOKEN 인증을 거치고**, 보고서 생성 성공 시 경로 또는 내용을 응답하며, 실패 시 원인 정보를 전달하는 것입니다 [10] [11].

- **보고서 내용:** 생성되는 Markdown 리포트에는 보통 **당일 최종 발행 기사들의 목록과 요약, 기타 통계**가 포함됩니다. 예를 들어 “금일 승인된 기사 X건, 게이트 통과 O/X, …” 같은 요약 정보와 각 기사 제목/요약/링크/편집자 코멘트 등을 나열할 수 있습니다. 리포트 포맷은 필요에 맞게 `make_daily_report.py`에서 정의합니다. (참고: `/api/export/md`와 유사한 정보를 담을 수 있지만, 일일 보고서는 내부 검토용 통계 등을 추가로 포함할 수 있습니다 [12].)

- **응답 형태:** 위 예시에서는 JSON으로 `{ "ok": True, "path": "archive/reports/2025-10-13_report.md" }` 같은 형식을 반환합니다. `ok`는 보고서 생성 성공 여부, `path`는 생성된 Markdown 파일 경로를 나타냅니다. 경우에 따라 보고서 내용을 직접 반환할 수도 있습니다. 예를 들어 작은 크기의 리포트라면 `{ "content": "# 2025-10-13 Daily Report\n..." }`로 Markdown 원문을 응답할 수 있습니다 [13]. **파일 경로 vs 내용** 선택은 운영 방식에 따라 결정합니다:

- 경로 방식: 파일로 저장한 후 경로를 알려주면, 클라이언트(UI)에서 해당 경로를 열어보거나 다운받도록 할 수 있습니다. Cloud Run 컨테이너 파일시스템은 휘발성이라 장기 보존은 안 되지만, 컨테이너가 살아있는 동안은 `archive/reports` 디렉토리에 파일이 남아 있습니다. 이 방식의 장점은 서버에서 Markdown을 저장해 두어 추후 참조가 가능하고, 응답이 가벼워집니다. 단, 클라이언트가 그 파일에 접근할 수 있도록 정적 파일 제공을 설정해야 합니다 (아래 UI 연동에서 설명).

- 내용 직접 전달: 파일을 저장하지 않고 생성된 Markdown 텍스트를 바로 응답하면, 클라이언트는 이를 화면에 표시하거나 파일로 저장할 수 있습니다. 장점은 별도 정적 호스팅이 필요 없고 즉시 내용을 볼 수 있다는 것이지만, 아주 긴 리포트의 경우 응답 크기가 커질 수 있습니다.

보안상 민감한 내용이 아니라면 둘 중 편한 방식을 택해도 무방합니다. 여기서는 **파일 경로를 반환**하는 패턴으로 설명하며, UI에서 그 파일을 열어보는 방법을 다룹니다 (이 패턴은 요약/번역 결과에도 동일하게 적용됩니다).

- **엔드포인트 수동 테스트:** 구현 후 로컬 개발 모드에서 Uvicorn으로 서버를 실행하고, `curl` 등을 이용해 동작을 확인합니다. 예:

```
uvicorn app.main:app --reload
curl -X POST "http://localhost:8000/api/report" -H "Authorization: Bearer ADMIN_SECRET_TOKEN"
```

- 올바른 토큰을 넣으면 JSON 응답으로 `{"ok": true, "path": "archive/reports/2025-10-13_report.md"}` (또는 `content` 필드)을 받아야 합니다.
- 토큰 없이 호출하면 **401 Unauthorized**가 반환되는지 확인하세요 (인증 미들웨어가 잘 작동하는지 검증) [14]. 잘못된 토큰을 넣어도 동일하게 401 응답이어야 합니다.
- 생성된 `archive/reports/...report.md` 파일을 열어 내용이 제대로 들어갔는지 확인합니다. 예상했던 기사 수와 제목 등이 포함되어 있는지 점검합니다.

2.2 프론트엔드 UI 연동: 보고서 생성 버튼

- **대시보드 버튼 추가:** 관리자용 프론트엔드 (예: `index.html`)에 "일일 보고서 생성" 버튼을 추가합니다. 예시:

```
<button id="btnReport" onclick="genReport()">일일 보고서 생성</button>
<a id="linkReport" href="#" target="_blank" style="display:none;">보고서 열기</a>
```

여기서 `<button>`은 보고서 생성 액션을 트리거하고, `<a id="linkReport">`는 생성된 보고서를 열기 위한 링크입니다. 링크는 처음엔 숨김(`display:none`) 처리해두었다가, 보고서 생성 성공 시 나타나도록 할 것입니다 [15] [16].

- **JavaScript fetch 구현:** `genReport()` 함수를 정의하여 버튼 클릭 시 API를 호출하고 결과를 처리합니다:

```
async function genReport() {
 const btn = document.getElementById('btnReport');
 const link = document.getElementById('linkReport');
 btn.disabled = true;            // 1) 버튼 비활성화 (중복 클릭 방지)
 link.style.display = 'none';      // 링크 숨기기 (이전 결과 초기화)
 // TODO: 로딩 메시지나 스피너 표시 가능
 try {
  const res = await fetch('/api/report', {
   method: 'POST',
   headers: { 'Authorization': 'Bearer ' + ADMIN_TOKEN } // 토큰 헤더 추가
  });
  const data = await res.json();
  console.log(data); // 응답 결과를 콘솔에 출력 (로그용)
  if (res.ok && data.path) {
   // 2) 성공: 링크 href 설정 및 표시
   link.href = '/' + data.path;    // 정적 파일 경로로 접근 (예: /archive/reports/..)
   link.style.display = 'inline';
  } else {
   // 3) 실패: 오류 로그 및 사용자 알림
   console.error('Report generation failed:', data);
   alert('보고서 생성 실패: ' + (data.error || '원인 불명'));
  }
 } catch (err) {
  console.error('Error:', err);
```

```
    alert('보고서 생성 중 오류 발생');
  } finally {
    btn.disabled = false;          // 4) 항상 버튼 재활성화
  }
}
```

위 로직은 **(1)** 버튼을 누르면 일단 버튼을 비활성화하고 이전 결과 링크를 숨긴 후, **(2)** `/api/report`에 POST 요청을 보냅니다. 이때 **Authorization 헤더에 Bearer 토큰**을 넣어 인증을 수행해야 합니다 (UI 어플리케이션이 토큰을 알고 있다고 가정) [17]. 응답을 JSON으로 파싱하여 `res.ok`(HTTP 200계열)이고 `data.path`가 존재하면, 보고서 파일 경로가 정상 생성된 것으로 판단하여 `linkReport` 앵커의 `href`를 해당 경로로 설정하고 표시합니다 [18] [19]. **(3)** 만약 응답이 실패 상태이거나 `data.path`가 없으면, 에러를 콘솔에 출력하고 `alert`으로 실패 원인을 알려줍니다. (`data.error`가 백엔드에서 전달된 경우 그 내용을 표시하고, 없으면 "원인 불명"으로 표시) [20] [21]. **(4)** 마지막으로 `finally` 블록에서 버튼을 다시 활성화하여 다음 작업을 할 수 있게 합니다 [22] [23].

토큰 관리: 위 코드에서는 `ADMIN_TOKEN`을 직접 사용했지만, 실제 구현에서는 사용자의 로그인 흐름에 따라 토큰을 저장/관리합니다. 만약 별도의 로그인 없이 정적 페이지에서 바로 쓰는 경우, `<script>`에 토큰 값을 하드코딩하거나 (`ADMIN_TOKEN` 변수에 토큰 문자열 할당) 브라우저 prompt로 입력받는 등 구현할 수 있습니다. 보안상 클라이언트 소스에 토큰을 넣는 것은 노출 위험이 있으므로, 가능하다면 **세션 로그인**을 구현해 서버에서 쿠키 세션 검증을 하는 편이 좋습니다. 하지만 여기서는 관리자 시스템 특성상 내부자만 쓰는 도구이므로, 간단히 토큰을 구성원에게 배포하여 브라우저 콘솔에서 설정하거나, 페이지 로드 시 prompt로 입력받아 JS 변수에 저장하는 방법 등으로 처리할 수 있습니다. 핵심은 **API 호출 시 Authorization 헤더를 포함**하는 것입니다 [24].

- **정적 파일 제공 설정:** `genReport` 함수에서 `link.href = '/' + data.path;`로 경로를 잡았는데, 예를 들어 `data.path`가 `"archive/reports/2025-10-13_report.md"`라면 최종 href는 `/archive/reports/2025-10-13_report.md`가 됩니다. 이 경로에 사용자가 접근하면 Markdown 파일 내용을 볼 수 있어야 하므로, **백엔드 서버가 `/archive` 경로의 정적 파일을 서빙**하도록 설정해야 합니다. FastAPI에서는 `from starlette.staticfiles import StaticFiles`를 이용해 다음과 같이 설정할 수 있습니다:

```
from fastapi import FastAPI
from starlette.staticfiles import StaticFiles

app = FastAPI()
app.mount("/archive", StaticFiles(directory="archive"), name="archive")
```

이렇게 하면 `/archive/...` 경로로 들어오는 요청을 `archive/` 디렉토리의 파일로 매핑해줍니다. 따라서 Cloud Run에 배포하더라도 해당 파일을 브라우저에서 열람할 수 있습니다. 권한 제어를 위해 정적 파일에까지 토큰 검증을 적용하지는 않지만, URL을 아는 사람만 접근할 수 있고 민감한 정보는 아니므로 크게 문제되지는 않습니다. (필요시 정적파일도 보호하려면 `/api/download_report` 등 별도 엔드포인트로 파일을 읽어 `Response`로 내려주고, 여기에도 토큰을 요구하는 방법도 있습니다.)

- **UI 결과 확인:** 이제 사용자가 **"일일 보고서 생성"** 버튼을 클릭하면:
- API 요청이 보내지고 대시보드에서 로딩 중 상태로 대기합니다. (원한다면 "생성 중..."이라는 메시지를 버튼 옆에 표시할 수 있습니다.)
- 서버에서 Markdown 리포트 파일이 생성되고 JSON 응답이 옵니다.
- 응답이 성공적이면 "보고서 열기" 링크가 나타납니다. 관리자는 이 링크를 클릭해 새 탭으로 Markdown 파일을 열어볼 수 있습니다. 브라우저가 `.md` 파일을 바로 표시하지 못하면 다운로드될 수 있는데, Markdown은 텍스트 파일이므로 내려받은 뒤 메모장이나 VSCode 등으로 열어볼 수 있습니다.

- 응답에 `error`가 있는 등 실패한 경우 alert으로 이유를 봅니다. 예를 들어 `번역 API 키 없음`과 같은 메시지가 나오면 운영자가 환경변수 설정을 확인해야 합니다 [25].

- **참고:** UI에서 Markdown 내용을 직접 보여주고 싶다면, 링크를 노출하는 대신 `fetch`로 받아온 `data.content`를 사용하거나 파일 경로로 다시 AJAX GET 요청을 보내 Markdown 텍스트를 가져와 모달 창에 렌더링할 수도 있습니다. 다만 Markdown 렌더링을 위해 추가 JS 라이브러리가 필요하고 복잡도가 올라가므로, 초기 단계에서는 **파일 링크**로 여는 방식으로 충분합니다 [26]. 운영자는 새 탭이나 편한 편집기에서 Markdown을 검토한 후, 필요한 경우 내용을 복사해 발행 시스템에 붙여넣거나 활용하면 됩니다.

3. 뉴스 카드 요약/번역 기능

QualiJournal 시스템에서는 **수집된 뉴스 카드의 요약 및 번역** 기능을 제공합니다. 이는 하루 키워드에 대한 전체 기사 요약과, 최종 선정된 기사들에 대한 요약을 자동 생성하고 (선택적으로) 한국어로 번역해주는 작업입니다. 백엔드에는 두 개의 API 엔드포인트로 구현하며, 프론트엔드 UI에도 두 개의 버튼을 추가하여 관리자가 버튼 클릭만으로 요약 작업을 수행할 수 있게 합니다.

- **기능 개요:** 뉴스 요약/번역은 두 종류가 있습니다 [27]:
- **키워드 전체 뉴스 요약** – 해당 **키워드로 수집된 모든 기사**의 핵심 내용을 자동 요약하고, 번역 API 키가 설정된 경우 **영문 요약을 한국어로 번역**합니다. (예: 오늘의 키워드 관련 30개 기사 모두 요약 생성)
- **선정본 뉴스 요약** – 편집자가 **최종 발행 대상으로 선정한 기사들**만 모아 요약 및 번역본을 생성합니다. (즉, 데일리 리포트/발행본에 실릴 기사들의 요약문을 최종 정리)

이 두 기능 모두 `tools/enrich_cards.py` 스크립트를 기반으로 작동하며, YAML/JSON 등 데이터 파일을 갱신하고 결과 Markdown 파일을 생성하는 것으로 구현합니다 [28].

3.1 백엔드 엔드포인트 구현: POST `/api/enrich/keyword` 및 `/api/enrich/selection`

- **엔드포인트 설계:** 두 엔드포인트 모두 POST 메소드로 구현하고, ADMIN_TOKEN 인증을 요구합니다. `/api/enrich/keyword`는 "키워드 전체 요약"을 수행하고, `/api/enrich/selection`은 "선정본 요약"을 수행합니다. 내부 동작은 유사하며, 인자나 요청 본문은 필요 없습니다 (토큰만 있으면 됨).

- `enrich_cards.py` **활용:** 이 스크립트는 뉴스 기사 리스트를 입력으로 받아 각 기사에 **요약 필드(summary_en)**를 추가하고, 번역 API 키가 있을 경우 **한국어 요약(summary_ko)**도 생성하여 JSON 데이터에 반영하는 기능을 갖습니다 [29]. 또한 결과를 토대로 Markdown 문서를 작성하는 기능도 포함돼 있을 수 있습니다. 우리 API 구현에서는 이 스크립트를 **모듈로 import하여 함수 호출**을 할 수도 있고, 간단히 **서브프로세스로 실행**할 수도 있습니다. 설계를 단순화하기 위해, 여기서는 각각의 엔드포인트에서 `enrich_cards.py`를 호출하고, 해당 스크립트가 작업 완료 후 **Markdown 파일을 생성**한다고 가정하겠습니다.

- **전체 기사 요약** `/api/enrich/keyword`**:** 호출 시 `selected_keyword_articles.json` (키워드별 수집된 전체 기사 목록 파일)을 읽어 모든 기사에 대해 영문 요약을 생성합니다 [30]. 번역 API 키(`API_KEY`)가 설정되어 있으면 각 요약을 한국어로 번역하여 `summary_ko` 필드에 추가합니다 [3]. `enrich_cards.py`는 이 작업을 수행한 뒤, **각 기사 카드의 제목, 원문 링크, 영문 요약, 국문 요약, 편집자 코멘트** 등을 포함한 **Markdown 파일**을 생성합니다 [3]. 생성되는 파일명은 예를 들어 `archive/enriched/2025-10-13_<키워드>_ALL.md` 형식이며, 해당 키워드의 **전체 기사 요약본**을 담습니다 [3].

엔드포인트 구현은 `make_daily_report` 때와 비슷하게:

```
@router.post("/api/enrich/keyword")
def enrich_all(token: str = Depends(verify_admin_token)):
    try:
        result = run_enrich_all()  # 전체 기사 요약/번역 수행, Markdown 생성
    except Exception as e:
        return {"ok": False, "error": str(e)}
    return {"ok": True, "path": result.path}
```

여기서 `run_enrich_all()`은 `enrich_cards.py`의 기능을 호출하는 함수로, 성공 시 생성된 Markdown 파일의 경로 등을 포함한 결과를 반환한다고 가정했습니다. 실제로는 `subprocess.run(["python", "tools/enrich_cards.py", "--mode=all"])`처럼 호출할 수도 있습니다.

스크립트 동작: `enrich_cards.py`는 내부적으로 오류가 발생해도 가능한 한 Markdown 요약본 파일을 생성하도록 설계합니다 [31]. 예를 들어 번역 API 쿼터 초과 등으로 일부 기사 번역에 실패해도, 그 부분은 영문 요약만 넣고 나머지 기사들의 번역은 포함한 Markdown을 만들어낼 수 있습니다. 그러므로 API 엔드포인트는 스크립트 실행 후 **파일이 생성되었는지만 확인**하고 경로를 응답하면 됩니다. 혹여 키 파일 없음 등으로 정말 실패했다면, `error` 필드로 원인을 함께 전달합니다.

- **선정본 요약** `/api/enrich/selection`**:** 호출 시 `selected_articles.json` (최종 선정 기사 목록 파일)을 불러와 해당 기사들에 대해 요약/번역을 수행합니다. `enrich_cards.py`는 이 입력을 받아 각 선정된 기사에 summary_en/summary_ko를 채우고, 결과를 이용해 Markdown 파일을 작성합니다 [32]. **파일명**은 `archive/enriched/2025-10-13_<키워드>_SELECTED.md` 형식으로 저장되며, 금일 최종 선정본 기사들의 요약/번역 결과물을 담습니다 [33]. (키워드가 있다면 파일명에 포함되고, 없으면 날짜+SELECTED.md만 사용.) 이 역시 스크립트 내부에서 오류 발생 여부와 상관없이 Markdown이 생성되도록 구현되어 있습니다 [32].

엔드포인트 구현 구조도 위와 유사하게:

```
@router.post("/api/enrich/selection")
def enrich_selection(token: str = Depends(verify_admin_token)):
    try:
        result = run_enrich_selected()  # 선정본 기사 요약/번역 수행
    except Exception as e:
        return {"ok": False, "error": str(e)}
    return {"ok": True, "path": result.path}
```

로직은 내부적으로 `enrich_cards.py`를 선정모드로 실행하고 Markdown 결과 파일 경로를 받아 응답합니다.

- **응답 및 예외 처리:** `/api/enrich/*` API들도 `/api/report`처럼 `{ "ok": True/False, "path": "...", "error": "..." }` 형태의 JSON을 반환합니다. 성공 시 생성 파일의 경로를 포함하고, 실패 시 `ok: false`와 함께 오류 원인을 `error`에 담습니다. 예를 들어 번역 API 키가 설정되지 않아 작업이 제대로 수행되지 않은 경우 `{"ok": false, "error": "Translation API key not set"}`과 같이 응답할 수 있습니다 [34]. 클라이언트는 이 메시지를 받아 사용자에게 표시하여 원인을 파악할 수 있게 합니다. (현재 구현에서는 실패 시 서버가 HTTP 200을 주고 error 필드에 이유를 전달하는데, 상황에 따라 500을 리턴할 수도 있습니다. UI 단에서 에러 처리를 일관되게 하기 위해 200+error 패턴을 쓰는 사례도 있습니다.)

- **번역 API 설정:** 앞서 환경 변수에서 설정한 `API_KEY`가 바로 이 기능에 쓰입니다. `enrich_cards.py` 내부에서 `API_KEY = os.getenv('API_KEY')` 식으로 키를 받아 번역 요청을 수행할 것입니다 [3]. 만약 키가 없으면 번역을 건너뛰거나 Exception을 던질 수 있습니다. 그러므로 **배포 환경에 API_KEY 설정을 빠뜨리지 않았는지** 꼭 확인해야 합니다. 또한, 번역 API의 **요금제/쿼터 관리**도 중요합니다. 자동 번역은 비용이 발생하거나 사용량 제한이 있으므로, 운영 전에 키의 유효성과 쿼터를 확인하세요. 만약 쿼터 초과시 `"Translation API quota exceeded"` 같은 오류가 발생할 수 있으며, 이 경우에도 Markdown 결과 파일은 (번역 없이라도) 생성되도록 하고 UI에 오류 메시지를 표시하여 관리자에게 알려주는 것이 좋습니다 [31].

3.2 프론트엔드 UI 연동: 요약/번역 버튼 (전체 & 선정본)

- **버튼 추가:** 대시보드 HTML에 두 개의 버튼을 추가합니다. 하나는 "키워드 요약/번역" (전체 기사 요약용), 또 하나는 "선정본 요약/번역"입니다. 예시:

```
<button id="btnEnrichAll" onclick="enrichKeyword()">키워드 요약/번역</button>
<a id="linkEnrichAll" href="#" target="_blank" style="display:none;">ALL.md 열기</a>

<button id="btnEnrichSel" onclick="enrichSelection()">선정본 요약/번역</button>
<a id="linkEnrichSel" href="#" target="_blank" style="display:none;">SELECTED.md 열기</a>
```

위와 같이 각 기능별로 버튼과 결과 링크 쌍을 마련해둡니다 [35]. 예시는 간략화를 위해 버튼과 링크를 바로 나열했지만, 실제 UI에서는 섹션을 구분하거나 적절히 스타일링하면 좋습니다.

- **JavaScript 구현:** `enrichKeyword()`와 `enrichSelection()` 함수를 작성합니다. 로직은 앞서 `genReport()`와 거의 동일하며, 호출하는 URL만 다릅니다. 공통 패턴은:
- 버튼 비활성화 및 이전 링크 숨기기.
- `fetch`로 API POST 요청 보내기 (URL이 `/api/enrich/keyword` 또는 `/api/enrich/selection`).
- 응답 JSON 처리: 성공이면 링크 href 설정 후 표시, 실패면 오류 표시.
- 버튼 재활성화.

예시 (`enrichKeyword`의 경우):

```
async function enrichKeyword() {
  const btn = document.getElementById('btnEnrichAll');
  const link = document.getElementById('linkEnrichAll');
  btn.disabled = true;
  link.style.display = 'none';
  try {
    const res = await fetch('/api/enrich/keyword', {
      method: 'POST',
      headers: { 'Authorization': 'Bearer ' + ADMIN_TOKEN }
    });
    const data = await res.json();
    if (res.ok && data.path) {
      link.href = '/' + data.path;
      link.style.display = 'inline';
    } else {
      console.error('Enrich keyword failed:', data);
```

```
    alert('요약/번역 실패: ' + (data.error || '원인 불명'));
   }
  } catch (err) {
   console.error('Error:', err);
   alert('요약 처리 중 오류 발생');
  } finally {
   btn.disabled = false;
  }
}

async function enrichSelection() {
 const btn = document.getElementById('btnEnrichSel');
 const link = document.getElementById('linkEnrichSel');
 btn.disabled = true;
 link.style.display = 'none';
 try {
  const res = await fetch('/api/enrich/selection', {
   method: 'POST',
   headers: { 'Authorization': 'Bearer ' + ADMIN_TOKEN }
  });
  const data = await res.json();
  if (res.ok && data.path) {
   link.href = '/' + data.path;
   link.style.display = 'inline';
  } else {
   console.error('Enrich selection failed:', data);
   alert('요약/번역 실패: ' + (data.error || '원인 불명'));
  }
 } catch (err) {
  console.error('Error:', err);
  alert('요약 처리 중 오류 발생');
 } finally {
  btn.disabled = false;
 }
}
```

위 코드에서 **Authorization 헤더 추가** 부분과 응답 처리 방식은 보고서 생성과 동일합니다. 두 함수의 차이는 fetch 호출하는 URL과 사용하는 DOM element ID 정도입니다 [36] . 중복 코드를 줄이려면 공통 함수로 추상화할 수도 있으나, 이해를 돕기 위해 풀어서 작성했습니다.

- **UI 처리 흐름:** 사용자가 "키워드 요약/번역" 버튼을 누르면 `enrichKeyword()` 실행 → `/api/enrich/keyword` 요청 → 응답에 따라 `linkEnrichAll` 링크가 나타납니다. 이 링크는 예컨대 `2025-10-13_TOPIC_ALL.md 열기` 같은 이름으로 UI에 표시될 수 있습니다. 사용자가 클릭하면 새 탭에서 Markdown 요약본 파일이 열립니다. "선정본 요약/번역"도 동일하게 동작하며, 결과 링크(`linkEnrichSel`)를 통해 `..._SELECTED.md` 파일을 열어볼 수 있습니다 [37] [38] .

- **UX 고려:** 요약/번역 결과물은 최종 발행 콘텐츠에 가까우므로, **관리자가 바로 내용을 확인**할 수 있도록 하는 것이 중요합니다 [26] . 현재 설계는 Markdown 파일을 새 탭에서 여는 방식이지만, 추후 필요하다면 Markdown을 HTML로 변환해 대시보드 내에서 보여주거나, 편집 기능을 붙여 직접 수동 수정할 수 있게 할 수

도 있습니다. 초반에는 파일을 확인하고, 문제가 있으면 JSON 데이터 파일을 직접 편집하거나 재실행하여 수정하는 프로세스로 운영할 수 있습니다.

- **오류 처리:** 번역 API 키 누락 등으로 실패한 경우, 앞서 언급한 대로 `data.error`에 메시지가 담겨 옵니다. UI에서는 그 메시지를 alert으로 보여주므로 관리자가 즉시 원인을 파악할 수 있습니다[34]. 예를 들어 `"error": "Translation API key not set"`이 오면, 관리자는 .env에 API_KEY를 설정했는지 확인하고 Cloud Run 서비스 변수에 반영했는지 등을 점검하면 됩니다. 또는 `"error": "Translation API quota exceeded"`가 나오면 해당 API 키의 사용량을 확인해야 합니다[31]. 이렇듯 **실패 시에도 Markdown 결과 파일은 생성되었을 가능성**이 있으므로 (예: 영문 요약까지만 채워진 파일), 운영자는 `archive/enriched/` 폴더에 생성된 파일을 열어보고 일부 수동 보완할 수도 있습니다[31].

- **결과 확인:** UI에서 두 요약 기능을 실행한 뒤:

- `ALL.md 열기` 링크를 눌러 전체 기사 요약본을 확인합니다. 영문 요약과 번역 문단이 기사별로 잘 들어있는지, 모든 수집 기사에 대한 항목이 만들어졌는지 검토합니다.
- `SELECTED.md 열기` 링크를 눌러 선정본 기사 요약본을 확인합니다. 최종 발행할 기사들의 요약/번역문이 포함되어 있으며, 불필요한 기사가 들어가거나 누락된 기사가 없는지 확인합니다. 이 파일은 실제 발행물에 거의 그대로 활용될 가능성이 높으므로, 내용의 정확성을 특히 검토합니다 (번역 품질 등 포함).
- 필요 시 Markdown을 편집기에서 열어 간단히 수정(예: 번역 어투)한 후 발행용으로 사용할 수도 있습니다.

이로써 관리자는 하루 뉴스의 요약본(영문/한글)을 손쉽게 생성하고 검토할 수 있습니다. 다음은 생성된 결과물들을 Markdown/CSV로 **export**하여 별도 저장하거나 공유할 수 있는 기능을 구현합니다.

4. Export 기능 (Markdown/CSV 내보내기)

뉴스 편집 작업을 마친 후, **최종 선정된 기사 목록**을 파일로 추출하여 보관하거나 다른 시스템에 넘길 필요가 있습니다. Export 기능은 이러한 목적을 위해 **Markdown 파일** 또는 **CSV 파일** 형식으로 데이터를 제공하는 API입니다. 관리자 UI에서도 "Export MD"와 "Export CSV" 버튼으로 해당 기능을 노출합니다.

4.1 백엔드 엔드포인트 구현: GET `/api/export/md` 및 `/api/export/csv`

- **기능 설명:**
- `/api/export/md` (GET): **최종 발행본 Markdown 내보내기** 엔드포인트입니다[39]. 현재 `selected_articles.json`에 담긴 **최종 선정 기사 리스트**를 가져와, 마크다운 문서 형식으로 변환한 내용을 반환합니다[40]. 이 Markdown에는 선정된 기사들의 제목, 원문 URL, 요약(영문/한글), 편집자 코멘트 등이 모두 포함됩니다[40]. 사실상 이전에 생성한 일일 보고서와 비슷한 내용일 수 있지만, `/api/export/md`는 **파일 저장 없이 바로 다운로드용 콘텐츠를 응답**하는 데 초점을 둡니다[41].

- `/api/export/csv` (GET): **최종 발행본 CSV 내보내기** 엔드포인트입니다[42]. 선정 기사 목록을 불러와 각 기사의 주요 필드를 CSV (Comma-Separated Values) 형식으로 만들어 반환합니다[43]. 일반적으로 CSV 컬럼에는 기사 제목, 요약, 원문 URL, 편집자 코멘트, 날짜 등이 포함될 수 있습니다[44]. 이 CSV는 엑셀 등으로 열어볼 수 있게 인코딩에 신경 써야 합니다 (특히 한글 깨짐 방지).

- **구현:** 두 엔드포인트 모두 GET 요청에 대해 **파일 다운로드 응답**을 합니다. FastAPI에서는 `FileResponse`나 `Response` 객체를 사용해, 콘텐츠와 헤더를 수동으로 지정할 수 있습니다. 인증이 필요하므로 `Depends(verify_admin_token)`를 적용합니다.

예시 구현:

```
from fastapi import Response
import io, csv

@router.get("/api/export/md")
def export_markdown(token: str = Depends(verify_admin_token)):
    # 1. 최종 기사 데이터 로드
    data = load_selected_articles()  # selected_articles.json 로드 (파싱해서 리스트 반환)
    # 2. Markdown 문자열 생성
    md_lines = []
    for art in data:
        md_lines.append(f"- **{art['title']}** \n {art.get('summary_ko') or art.get('summary_en')}
\n *원문:* {art['url']} \n *코멘트:* {art.get('editor_note','없음')}\n")
    md_content = "# Daily News Summary (Final)\n\n" + "\n".join(md_lines)
    # 3. 응답 객체 생성 (Content-Disposition 헤더 설정하여 다운로드 트리거)
    headers = {"Content-Disposition": 'attachment; filename="final_selection.md"'}
    return Response(md_content, media_type="text/markdown", headers=headers)
```

```
@router.get("/api/export/csv")
def export_csv(token: str = Depends(verify_admin_token)):
    data = load_selected_articles()
    # 1. CSV 문자열 생성 (UTF-8 BOM 포함)
    output = io.StringIO()
    writer = csv.writer(output)
    # 헤더 행 쓰기 (원하는 컬럼명 지정)
    writer.writerow(["Title", "Summary_EN", "Summary_KO", "URL", "EditorNote"])
    for art in data:
        writer.writerow([
            art['title'],
            art.get('summary_en', ""),
            art.get('summary_ko', ""),
            art['url'],
            art.get('editor_note', "")
        ])
    csv_text = output.getvalue()
    # BOM 추가 (엑셀 한글 깨짐 방지)
    bom = "\ufeff"  # UTF-8 BOM 문자
    csv_content = bom + csv_text
    headers = {
        "Content-Type": "text/csv",
        "Content-Disposition": 'attachment; filename="final_selection.csv"'
    }
    return Response(csv_content, media_type="text/csv", headers=headers)
```

위 코드에서는 개념 전달을 위해 단순화했습니다. 실제 구현 시에는 `load_selected_articles()` 대신 직접 파일을 읽고 `json.loads`를 할 수도 있습니다. 중요한 부분은: - **Markdown 생성:** 리스트를 Markdown 리스트(`-`) 형태로 만들고, 제목은 **볼드**, 줄바꿈과 들여쓰기를 적절히 넣어 가독성을 높입니다. 필요에 따라 HTML `<br>` 태그나 표 형태로 구성해도 됩니다. `/api/report`에서 이미 Markdown 보고서를 만들었다면 그 코드를 재사용해도 됩니다. - **CSV 생성:** `csv` 모듈을 사용하여 문자열로 CSV를 만들고, **UTF-8 BOM** (`\ufeff`)을 맨 앞에 추가합니다 [45].

이 BOM은 Windows 엑셀에서 UTF-8 CSV를 열 때 한글이 깨지지 않도록 해줍니다 [45]. 또한 Content-Type을 `text/csv`로 지정하고 Content-Disposition에 `filename="...csv"`를 설정하여 브라우저가 다운로드를 인지하도록 합니다 [46]. - **공통:** 두 엔드포인트 모두 **파일을 서버에 저장하지 않고**, 요청에 대한 응답으로 바로 파일 내용을 보냅니다 [47] [46]. 즉, 일회성 다운로드 용도로만 사용되므로, 서버의 `archive/` 디렉토리에 따로 MD나 CSV를 남기지 않습니다. (원하면 남길 수도 있지만, 일일 보고서나 enriched MD 파일들이 이미 있으므로 이중으로 저장할 필요는 없어 보입니다.)

- **응답 및 브라우저 동작:** Content-Disposition 헤더에 `attachment`를 지정했으므로, 브라우저는 해당 요청 응답을 받으면 **파일 다운로드를 시작**합니다 [48] [46]. `filename`으로 제안된 이름(`final_selection.md/csv`)으로 사용자의 다운로드 디렉토리에 저장될 것입니다. Markdown은 일반 텍스트이므로 인코딩 문제는 없고, CSV는 BOM 덕분에 Excel에서 열 때도 정상적으로 한글이 표시됩니다 [49] [50].

- **엔드포인트 테스트:** 브라우저가 아닌 환경에서 테스트하려면, `curl -v -H "Authorization: Bearer ..."`로 해당 URL에 요청하여 응답 헤더와 내용을 확인하면 됩니다. 또는 Pytest로 `client.get("/api/export/md")`를 호출하고 `response.text`를 확인할 수 있습니다 [51] [52]. 실제 내용이 잘 오는지, CSV의 경우 `response.text.startswith("\ufeff")`로 BOM 존재 여부를 검증하는 테스트를 작성할 수도 있습니다 [53] [54].

4.2 프론트엔드 UI 연동: 내보내기 버튼 (MD, CSV)

- **버튼/링크 추가:** 대시보드에 Export용 버튼 (혹은 링크) UI를 추가합니다. 두 가지 구현 방법이 있습니다:
- **간단 링크 방식:** `<a href="/api/export/md" download="QualiNews_Today.md">Export MD</a>` 형태의 앵커를 만들어두고, 클릭하면 브라우저가 해당 GET 요청을 보내 바로 다운로드하도록 하는 방법 [55]. 이 경우 HTML 요소에 `download` 속성을 주어 브라우저가 자동 다운로드를 수행하게 할 수 있습니다 [56] [57].
- **JS fetch 방식:** `<button id="btnExportCsv">Export CSV</button>` 등의 버튼을 만들고, 클릭 시 JavaScript로 `fetch('/api/export/csv')`를 호출하여 응답 Blob을 받아 사용자에게 다운로드를 트리거하는 방법 [58].

첫 번째 방법은 코드가 간단하지만, **인증 헤더를 보낼 수 없다는 문제**가 있습니다. 우리의 `/api/export/*` 엔드포인트는 토큰 인증을 요구하므로, 단순 앵커 클릭으로는 Authorization 헤더가 포함되지 않아 **401 Unauthorized**가 뜰 것입니다. 이를 우회하려면 URL에 토큰을 쿼리스트링으로 넣는 방법도 있지만(`/api/export/md?token=...`), 토큰 노출 위험으로 권장되지 않습니다 [59].

그러므로 **JS fetch 방식**으로 구현하는 것을 권장합니다. 다만 fetch로 받은 데이터를 다운로드하는 추가 작업이 필요합니다.

- **JavaScript를 통한 다운로드:** Fetch API로 파일 응답을 받으면, `res.blob()` 혹은 `res.text()` 등을 이용해 내용을 추출할 수 있습니다. CSV는 바이너리가 아니므로 `res.text()`로 처리해도 되지만, 편의상 Blob으로 받아 처리하는 예를 들겠습니다:

```
async function exportCsv() {
  const res = await fetch('/api/export/csv', {
    headers: { 'Authorization': 'Bearer ' + ADMIN_TOKEN }
  });
  if (res.ok) {
    const blob = await res.blob();
    const url = URL.createObjectURL(blob);
```

```
    const a = document.createElement('a');
    a.href = url;
    a.download = "QualiNews_Today.csv";
    document.body.appendChild(a);
    a.click();
    a.remove();
    URL.revokeObjectURL(url);
  } else {
    const errData = await res.json().catch(() => ({}));
    alert('CSV export failed: ' + (errData.error || res.status));
  }
}
```

위 함수는 `/api/export/csv`를 호출하여 응답을 Blob (파일 데이터)으로 받고, 임시 `<a>` 엘리먼트를 만들어 `href`를 그 Blob 객체 URL로 지정한 뒤, `.click()`으로 다운로드를 트리거합니다. `download` 속성으로 파일명을 지정하고, 다운로드 후 `URL.revokeObjectURL`로 메모리를 정리합니다. Markdown의 경우도 동일한 방법으로 처리하면 됩니다 (`exportMd()` 함수를 만들어 blob을 받고 `QualiNews_Today.md`로 저장).

> **참고:** 위처럼 blob을 활용하지 않고 `window.open("/api/export/md")`를 쓰면 새 탭이 열리며 브라우저가 다운로드 처리할 수 있습니다. 하지만 이 역시 헤더를 붙일 수 없고, Cloud Run 서비스가 비인증 공개가 아닌 경우 401 페이지가 떠버립니다. 따라서 fetch + Blob + programmatic download 방식이 필요합니다.

- **UI 흐름:** Export 버튼을 누르면 즉시 파일 다운로드가 시작되므로, **특별한 UI 변화는 없습니다.** 다만 다운로드 중 표시를 주고 싶다면 버튼을 일시 비활성화하거나 "내보내는 중..." 메시지를 줄 수 있습니다. 일반적으로 CSV/MD 생성은 매우 빨리 끝나므로 로딩 표시가 없어도 무방합니다.

- **인증 처리 옵션:** 만약 이 관리자 UI가 완전히 내부에서만 쓰이고, 굳이 토큰까지 이중으로 확인할 필요가 없다고 판단되면, Export 엔드포인트에 한해 토큰 검증을 비활성화하는 방안도 고려할 수 있습니다 [60]. 그러나 원칙적으로 모든 API에 인증을 거는 것이 안전하므로, 여기서는 유지합니다. 대신 Token을 자동으로 붙이는 방법을 적용한 것입니다. (또 다른 방법으로, **HTTP Only 쿠키 세션**을 쓰면 브라우저가 자동으로 쿠키를 보내기 때문에 링크 클릭만으로 인증이 가능해지지만, 이 시스템은 토큰 기반이므로 해당되지 않습니다.)

- **사용자 가이드:** 운영자는 Export 기능을 이용해 **정기 백업 또는 통계 활용**을 할 수 있습니다 [61] [62]. 예를 들어, 매일 CSV 파일을 내려받아 사내 저장소에 보관하면 향후 누적 데이터를 분석할 수 있습니다. Markdown 내보내기는 당일 발행본을 마크다운으로 저장해두는 용도로 쓰일 수 있습니다 (예: 마케팅팀에 전달 등).

- **엑셀 인코딩 확인:** CSV를 다운로드한 후 실제로 Excel이나 한글 워드프로세서에서 열어보면서 한글이 깨지지 않는지 꼭 확인해야 합니다 [63] [57]. 앞서 BOM을 넣었으므로 일반적으로 Excel에서 잘 열리지만, 혹시라도 문제가 있으면 BOM 혹은 인코딩 방식을 재검토해야 합니다. (예: Excel이 CSV를 ANSI로 간주하면 깨질 수 있는데, BOM이 이를 UTF-8로 인식시켜 줄 것입니다.)

- **테스트:** Export 버튼을 눌렀을 때 파일이 잘 다운로드되는지, 파일명을 제대로 받는지 확인합니다. 브라우저에 따라 다운로드 동작이 다를 수 있으니 (예: Chrome, Edge, Firefox) 한두 가지로 시도해봅니다. 또한, 다운로드된 파일을 열어 내용을 검토합니다. Markdown 파일의 형식이 요구사항에 맞는지, CSV 컬럼 구성이 적절한지 확인합니다. 필요하면 컬럼 순서나 이름 등을 조정해 주세요.

이로써 핵심 기능 (보고서 생성, 요약/번역, 결과물 export)들의 UI 연동까지 모두 구현되었습니다. 다음으로, 이러한 관리자 API들이 **보안** 요구사항을 제대로 준수하는지 점검하고, 테스트를 통해 신뢰성을 확보하겠습니다.

5. 인증 및 보안 설정 점검

QualiJournal 관리자 시스템은 **관리자 전용 기능**을 다루므로, **인증 및 보안**이 매우 중요합니다. 여기서는 ADMIN_TOKEN 기반 인증의 적용 여부를 검토하고, 토큰 노출 방지 등의 주의사항을 다룹니다.

- **ADMIN_TOKEN 인증 적용 확인:** 앞서 구현한 모든 API 엔드포인트 (`/api/report`, `/api/enrich/*`, `/api/export/*` 등)는 `Depends(verify_admin_token)`을 통해 보호하고 있습니다. 이 `verify_admin_token` 함수(혹은 dependency)는 **클라이언트가 보낸 토큰이 올바른지 검증**하는 역할을 합니다. 일반적인 구현은:

```
# app/auth.py
from fastapi import HTTPException, Security
from fastapi.security import HTTPAuthorizationCredentials, HTTPBearer

ADMIN_TOKEN_VALUE = os.getenv("ADMIN_TOKEN")

def verify_admin_token(credentials: HTTPAuthorizationCredentials =
Security(HTTPBearer())):
    if credentials.scheme != "Bearer" or credentials.credentials !=
ADMIN_TOKEN_VALUE:
        raise HTTPException(status_code=401, detail="Unauthorized")
    return True
```

위와 같이 Authorization 헤더의 Bearer 토큰 값을 읽어 `.env` 또는 환경변수에 저장된 ADMIN_TOKEN과 비교하여 다르면 에러를 낼 수 있습니다. 또는 더 간단히, FastAPI의 `Header` dependency로 Authorization 헤더를 받아 문자열 파싱 후 비교해도 됩니다. 핵심은 **서버 기동 시 메모리에 보관한 토큰 값**과 **요청 헤더로 받은 토큰**을 정확히 비교하는 것입니다 [24]. 일치하지 않으면 401 Unauthorized(또는 403 Forbidden) 응답을 반환합니다 [24]. 이 검증이 누락된 엔드포인트가 없는지 확인하세요. (만약 `/archive` 정적 파일을 올려둔 것을 보호해야 한다면 별도 처리가 필요하지만, 일반적으로 보고서나 요약 결과 파일에는 민감한 개인정보가 없으므로 크게 문제되지 않습니다.)

- **환경변수 토큰 설정:** 운영 환경에서 `ADMIN_TOKEN` 값이 제대로 로드되었는지 확인합니다. Cloud Run에 환경변수로 넣었다면, 배포 후 `gcloud run services describe` 명령으로 해당 값이 설정되었는지 (값은 숨김 처리되지만 키 존재 여부로 확인) 보거나, 의도치 않게 빈 값으로 배포되진 않았는지 검증합니다 [64]. 만약 토큰이 비어있다면 모든 요청이 인증을 통과해버릴 수 있으니 위험합니다.

- **토큰 노출 방지:**

- **URL 전송 자제:** 인증 토큰을 쿼리 파라미터로 전달하는 것은 앞서 언급했듯 지양해야 합니다 [59]. URL에 포함된 토큰은 브라우저 기록이나 서버 로그 등에 남아 노출 위험이 있습니다. 항상 HTTP 헤더로 전달하고, SSL을 통해 암호화된 통신을 사용하세요 (Cloud Run의 HTTPS 엔드포인트 사용).
- **로그 주의:** 서버 사이드에서 Authorization 헤더 전체를 로그에 찍지 않도록 합니다. FastAPI 기본 로거는 요청 경로와 상태 정도만 찍으므로 토큰이 로그에 남진 않지만, 혹시 수동으로 `print(request.headers)` 같은 코드를 쓰지 않도록 합니다.
- **프론트엔드 코드:** 토큰을 하드코딩해서 배포하면 웹소스를 통해 노출될 위험이 있습니다. 가능하다면 **환경변수 주입 + 서버세션** 방식으로 개선을 고려합니다. 예컨대 simple auth라도 UI 로그인 폼을 만들어 입력한 비밀번호를 서버에서 검증하고 세션 쿠키를 발급, 이후 요청에 쿠키로 인증하는 방식을 쓰면 토큰이 노출되지 않습니다. 다만 현재 범위에서는 ADMIN_TOKEN 자체를 공유하여 쓰는 구조이므로, UI 소스 관리에 유의해야 합니다.

(만약 UI를 Netlify 같은 데 호스팅하면 API 호출 시 토큰 노출 문제가 있으니, Cloud Run에서 index.html도 같이 제공하는 방식이 오히려 안전합니다.)

- **Cloud Run IAM 인증:** 앞서 Cloud Run에 `--allow-unauthenticated`를 적용했는데, 만약 이것을 false로 하면 Cloud Run 자체 인증(OAuth나 IAM)이 필요해집니다 [65]. 이 경우 토큰이 있더라도 Cloud Run이 차단하므로 401이 나옵니다. Cloud Run 레벨의 인증을 쓰는 대신 우리는 **애플리케이션 레벨 토큰 인증**을 사용하므로, Cloud Run 서비스 자체는 공개되어 있어도 토큰 없는 접근은 우리 앱에서 차단하는 구조입니다. 이를 이중으로 설정하지 않았는지 확인합니다.

- **동일 출처 정책(CORS):** 만약 관리자 UI와 API가 다른 도메인(출처)에 있다면 CORS 설정도 필요합니다. 하지만 일반적으로 index.html을 백엔드가 서빙하거나 동일 domain에서 호스팅하면 CORS 이슈는 없습니다. 현재 구조에선 Cloud Run 서비스 하나에서 프론트와 API를 같이 다루므로 문제없습니다. 다만, 만약 로컬에서 `file://`로 HTML을 열어 테스트하면 CORS 오류가 날 수 있는데, 이 때는 FastAPI에 `app.add_middleware(CORSMiddleware, allow_origins=["*"], ...)` 등의 설정을 임시로 하거나, HTML을 Live Server로 열어서 동일 호스트로 맞춰주세요.

- **인증 테스트:** 최종적으로 배포 환경에서 **토큰 인증이 잘 적용되었는지** 테스트합니다:

- 토큰 없이 `/api/status` 혹은 `/api/report` 등을 호출해보고 401/403이 나오는지.
- 올바른 토큰으로 호출해서 200이 나오는지 [66].

- 잘못된 토큰으로 호출시 당연히 거부되는지. 이 테스트는 자동화(테스트 코드)와 수동(curl) 모두 해보는 게 좋습니다. 아래는 Pytest 예시:

```
def test_auth_token_required(client):
    res = client.post("/api/report")  # 토큰 없이 호출
    assert res.status_code in (401, 403)
    res2 = client.post("/api/report", headers={"Authorization": "Bearer " + TEST_TOKEN})
    assert res2.status_code == 200
```

[67] 실제 구현에 따라 401 또는 403을 반환할 수 있어, 위처럼 두 경우를 모두 허용하고 있습니다.

- **추가 보안:** 내부 관리자용이라 해도, 가능한 보안 강화 방안을 고려합니다. 예를 들어 **IP 제한**(Cloud Run은 VPC접근 설정이나 Cloud Armor 등을 통해 특정 IP만 허용 가능), **VPN 내부망 사용** 등도 검토할 수 있습니다. 또 ADMIN_TOKEN은 **정기적으로 변경**하고 공유된 인원도 최소화하는 것이 좋습니다. 노출 시 즉시 새 토큰으로 교체하세요.

이상으로 핵심 기능 구현 및 인증 설정을 모두 마쳤습니다. 이제 **테스트와 최종 배포** 단계만 남았습니다. 기능별로 단위/통합 테스트를 작성해보고, Cloud Run에 배포한 뒤 최종 체크리스트를 점검해보겠습니다.

6. 테스트 및 배포 체크리스트

모든 기능이 구현되었으면, 배포 전에 **테스트**를 통해 문제가 없는지 확인해야 합니다. 또한 코드와 문서를 최신 상태로 정리해야 합니다. 이번 섹션에서는 Pytest 기반의 엔드포인트 테스트 작성 예시와, README 등 **문서화 작업** 및 **운영 체크리스트**를 정리합니다.

6.1 엔드포인트 기능 테스트 (Pytest 예시)

FastAPI에서는 `starlette.testclient.TestClient`를 이용해 서버를 띄우지 않고도 요청을 보낼 수 있습니다. 프로젝트의 `tests/` 폴더에 통합 테스트 스크립트를 작성해 두면, 코드 수정 후 **회귀 테스트**를 자동화할 수 있어 안정성이 높아집니다 [68]. 아래는 주요 기능들에 대한 테스트 시나리오 예시입니다:

- **일일 보고서 생성 테스트:**
- 상황: `selected_articles.json`에 몇 개의 더미 기사를 넣어둔 상태에서 `/api/report` 호출.
- 방법: 테스트 코드에서 `selected_articles.json` 경로를 가리키는 변수를 monkeypatch로 대체하여 가상의 데이터를 사용하도록 합니다. 또는 make_daily_report 함수를 monkeypatch하여 실제 파일 접근 없이도 known output을 리턴하게 합니다.
- 기대: 응답 `status_code == 200`이고, `data["path"]`가 존재하며, 해당 경로에 Markdown 파일이 생성되어 있을 것.

- 구현 예시:

```
from pathlib import Path
def test_report_generation(monkeypatch, client, tmp_path):
    # 가상 데이터 설정
    sample_articles = [ { "title": "Test News", "url": "http://example.com",
"summary_en": "Hello world", "summary_ko": "안녕하세요" } ]
    # selected_articles.json 임시 경로에 sample_articles 저장
    fake_json = tmp_path / "selected_articles.json"
    fake_json.write_text(json.dumps(sample_articles))
    # 서버 모듈의 경로 변수를 임시로 바꾸기 (server_quali.SELECTED_ARTICLES_PATH 등)
    monkeypatch.setattr(server_quali, "SELECTED_ARTICLES_PATH", str(fake_json))
    # 요청 보내기
    res = client.post("/api/report", headers={"Authorization": f"Bearer {TEST_TOKEN}"})
    assert res.status_code == 200
    data = res.json()
    assert data.get("path"), "응답에 path 없음"
    # Markdown 파일 생성 확인
    md_path = Path(data["path"])
    assert md_path.exists()
    content = md_path.read_text()
    # Markdown 내용에 제목과 한글 번역이 포함되었는지 확인
    assert "Test News" in content and "안녕하세요" in content
```

위 테스트는 보고서 생성 API가 호출되면 `selected_articles.json`의 더미 데이터를 바탕으로 Markdown 파일을 만들어주는지 검증합니다 [69] [70]. `ok` 필드가 True일 것을 확인해도 되고, 여기서는 `path` 존재 여부와 생성된 파일 내용으로 검증했습니다. (monkeypatch를 활용하여 테스트 격리를 합니다.)

- **요약/번역 기능 테스트:**

- 이 역시 비슷한 방식으로 `/api/enrich/selection` 등을 테스트합니다. `selected_articles.json`에 샘플 데이터를 넣어두고, `enrich_cards.py`의 외부 API 호출 부분을 monkeypatch하여 실제 번역 API를 호출하지 않도록 할 수 있습니다 (예: `monkeypatch.setattr(enrich_cards, "translate_func", lambda text: "번역")` 식으로).

- 그런 다음 `/api/enrich/selection` 호출 → 응답 확인 (`path` 존재) → 파일 읽어 내용에 영문/한글 요약이 들어있는지 확인합니다.

- 스크립트 내부에서 subprocess를 썼다면, 테스트에서는 해당 subprocess 호출을 가로채 결과를 미리 써주는 식으로 할 수도 있습니다. (조금 더 복잡하니, 가능하면 enrich_cards를 함수로 만들어 import하여 호출하는 구조로 짜면 테스트가 용이합니다.)

- **Export 기능 테스트:**

- Export는 브라우저 다운로드를 위한 것이어서 테스트 코드에선 **응답 본문 내용**을 확인하면 됩니다 [51].
- `/api/export/md`: `res = client.get("/api/export/md")` 후 `res.text`를 가져와 Markdown 형식의 일부가 포함되어 있는지 본다. 예를 들어 샘플 기사 제목이나, Markdown 리스트 기호(`-`)가 존재하는지 등으로 검증합니다.
- `/api/export/csv`: `res = client.get("/api/export/csv")` 후 `res.text`를 검사합니다.
  - **BOM 확인:** `assert res.text.startswith("\ufeff")`로 BOM이 포함되었는지 체크 [53].
  - **내용 확인:** CSV의 header나 데이터가 포함되었는지 본다. 예:
    `assert "Title" in res.text or "Test News" in res.text` [54].

- 또한 status code 200인지, 헤더에 `text/csv` 등이 제대로 세팅됐는지도 `res.headers`를 통해 확인 가능합니다. 필요하다면 `res.headers["content-disposition"]`에 파일명이 잘 들어있는지도 테스트합니다.

- **토큰 인증 테스트:**

- 잘못된 또는 없는 토큰에 대해 각 API가 401/403을 주는지, 올바른 토큰에는 통과되는지 확인합니다. 이는 모든 엔드포인트에 공통이므로 한두 개만 대표로 테스트해도 좋지만, **실수로 특정 엔드포인트에 데코레이터를 안 붙였을 경우**도 대비해 전 엔드포인트를 점검하는 게 이상적입니다.

- 예시로 `/api/report`에 대해:

```
def test_auth_token_required(client):
    res = client.post("/api/report")  # 토큰 없이
    assert res.status_code in (401, 403)
    res = client.post("/api/report", headers={"Authorization": "Bearer " + TEST_TOKEN})
    assert res.status_code == 200
```

이 테스트를 `/api/enrich/keyword`, `/api/export/csv` 등에도 적용하면 좋습니다 [66]. 모두 통과하면 인증 데코레이터가 빠짐없이 적용된 것입니다.

- **기타 테스트:** 기존에 존재하는 기능들 (예: `/api/status` 건강 체크나, 임계값 설정 API 등)이 있다면, 새로운 기능 추가 후에도 이들이 정상 동작하는지 회귀 테스트를 돌립니다. 특히 뉴스 수집/승인 관련 기능이 이번 변경으로 영향받지 않았는지 확인해야 합니다. tests/ 폴더의 기존 테스트가 모두 통과하는지 확인하고, 필요시 fixture 데이터를 최신 상태에 맞게 업데이트합니다 [71].

- **pytest 실행:** `pytest` 명령을 통해 전체 테스트가 통과되는지 확인합니다. CI 파이프라인이 있다면 푸시 시 자동으로 테스트를 돌리도록 구성하세요. 테스트가 모두 **green**이어야만 배포를 진행합니다. 하나라도 실패하면 그 원인을 파악하여 코드 수정 후 다시 테스트합니다.

6.2 문서 (README 등) 최신화 및 운영 가이드 정리

코드와 기능이 최종 완성되었으면, **프로젝트 문서**를 업데이트해야 합니다. 이는 협업자나 향후 관리자를 위해 중요합니다. 특히 QualiJournal 관리자 시스템은 여러 신규 기능이 추가되었으므로 README와 운영 매뉴얼에 아래 사항들을 반영합니다 [72] [73]:

- **신규 기능 사용법:** 이번에 추가된 **일일 보고서 생성**, **뉴스 요약/번역**, **Export (Markdown/CSV)** 기능의 용도를 소개하고, 사용 방법을 단계별로 설명합니다 [72]. 예를 들어 README에 "**일일 보고서 생성:** 관리자 대시보드에서 '일일 보고서 생성' 버튼을 누르면 YYYY-MM-DD_report.md 파일이 생성됩니다..." 등의 안내를 적습니다. UI 스크린샷이나 실행 전후 비교를 첨부하면 이해가 쉬울 것입니다.
- **환경 변수 및 설정:** README에 **환경변수(.env) 설정 예시**를 추가합니다 [73]. ADMIN_TOKEN과 API_KEY, DB_URL 등의 항목을 나열하고 각각 무엇을 의미하는지, 어떻게 설정하는지 안내합니다. 예:
- `ADMIN_TOKEN` : 관리자 인증 토큰 (임의의 긴 문자열, 실행 전 서버/클라이언트에 동일하게 설정)
- `API_KEY` : 뉴스 요약 번역에 사용되는 API 키 (예: Google Translate API 키). 없으면 번역기능 비활성.
- `DB_URL` : (선택) 데이터베이스 연결 문자열. SQLite 사용 시 `sqlite:///app.db` 형식. 이와 함께 **Cloud Run 배포 시 환경변수 지정 방법**도 간략히 요약합니다 (gcloud CLI 예시 등) [73].
- **게이트 임계값 및 자동화(KPI) 설명:** (프로젝트에 해당 기능이 있다면) README에 **게이트 통과 기준 설정법**, **KPI 지표 자동 갱신** 등의 최근 변경사항도 포함합니다 [74]. 예를 들어, 승인된 기사 수가 일정 기준 넘으면 발행 가능 상태가 되는 로직이 있다면, 그 임계값 조정 API (`/api/config/gate_required`) 사용법을 적습니다.
- **배포 방법 요약:** README에 **Cloud Run 배포 절차**를 요약합니다 [73]. 예를 들어 "Docker 빌드 -> GCR 푸시 -> gcloud run deploy 명령" 순서를 간단히 소개하거나, CI/CD를 사용하는 경우 해당 워크플로우를 언급합니다. 이 가이드북의 핵심 내용을 추려, 빠르게 배포할 사람을 위한 TL;DR을 제공하는 것입니다. (예: "GitHub Actions를 통해 main 브랜치 푸시 시 자동 배포되도록 설정되어 있습니다. 수동 배포 시 아래 명령 참조: ...")
- **운영 체크리스트/매뉴얼:** 이 가이드북에서 다룬 **운영 체크포인트**들을 README나 별도 운영 매뉴얼 문서에 요약해둡니다 [75]. 예를 들어:
- 일일 점검: 서비스 상태 (/api/status), 로그 모니터링, etc.
- 새 기능 관련 점검: 번역 API 키 유효성, 토큰 주기적 변경 일정, etc.
- 장애 대응: 보고서 생성 실패 시 어떻게 재시도/보완하는지, 번역 API 한도 초과 시 어떻게 할지 등 Q&A 형태 정리.
- Export 데이터 백업: Export 기능으로 주기적으로 데이터를 내려받아 보관할 것 등을 명시 [62].
- **API 명세 정리:** 관리자용 API들의 명세 표를 README에 추가하면 유용합니다. 각 엔드포인트별 메소드, 파라미터, 응답 형식을 표로 정리하고, 예시 요청/응답도 첨부합니다. 이는 개발자뿐 아니라 운영자가 API를 직접 호출해보고 싶을 때 참고가 됩니다.
- **기술 스택 및 패키지:** 이번 기능 추가로 새로운 라이브러리를 도입했다면 (예: `python-dotenv`, 혹은 번역 API 클라이언트 등), README의 **Requirements** 섹션에 반영합니다. 또한 Dockerfile 변경사항도 README에 언급해두면 좋습니다.
- **유지보수 방안:** 앞으로 기능 추가나 변경 시 **문서도 함께 업데이트하는 프로세스**를 갖추는 것이 좋습니다 [76]. README 최하단에 "문서 최종 업데이트 날짜"를 적고, 변경 로그를 간략히 추가하는 것도 고려합니다 (예: v1.1 - 2025.10.13: 보고서/요약/내보내기 기능 추가).

마지막으로, 이 모든 변경사항을 커밋하고 푸시하여 저장소를 최신화합니다. 이제 **Cloud Run에 실제 배포**하여 운영 환경에서 최종 점검을 진행합니다.

7. 배포 절차 (Cloud Run 기준)

이 섹션에서는 완성된 QualiJournal 관리자 시스템을 Cloud Run에 배포하고, 실제 운영 단계에서 확인해야 할 사항들을 정리합니다.

7.1 Docker 이미지 빌드 및 푸시

Cloud Run에 배포하려면 컨테이너 이미지가 필요합니다. 프로젝트 루트의 `Dockerfile`을 최신 코드와 종속성에 맞게 업데이트했는지 확인하세요 [77]. 새로운 Python 패키지를 썼다면 `Dockerfile`의 `pip install -r requirements.txt`에 해당 패키지가 포함되어야 합니다. 문제가 없다면, 로컬에서 우선 이미지를 빌드해봅니다:

```
# 1. 도커 이미지 빌드 (태그는 GCP Artifact Registry 경로 등으로 지정)
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
```

위에서 `<GCP_PROJECT_ID>`는 GCP 프로젝트 ID로, Artifact Registry를 사용한다면 도메인을 포함한 URL로 적어야 합니다 (예: `asia-northeast3-docker.pkg.dev/my-project/qualijournal-admin:latest`). 해당 프로젝트에 Artifact Registry 권한이 설정돼 있어야 하며, `gcloud auth configure-docker`를 한번 실행하여 도커가 푸시 인증을 하도록 준비합니다.

> **CI/CD 참고:** 이미 GitHub Actions 또는 Cloud Build CI 파이프라인이 설정되어 있다면, 수동으로 도커 빌드/푸시하지 않아도 됩니다. 예를 들어 main 브랜치에 푸시하면 CI가 자동으로 빌드/배포하도록 구성되었을 수 있습니다 [78] [79]. 다만, CI 설정이 오래되었다면 새 환경변수를 추가해줘야 할 수 있으니 (예: API_KEY, DB_URL), CI 설정 파일(.github/workflows/ci.yml 등)을 열어 `--set-env-vars` 부분을 최신화합니다 [80]. 또한 CI에서 배포 전 `pytest`를 실행하도록 해두면 안전합니다 [81].

7.2 Cloud Run 서비스 배포

이미지가 준비되었으면 Cloud Run에 배포합니다. gcloud CLI를 사용하는 방법은 다음과 같습니다:

```
# 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} \
  --allow-unauthenticated
```

위 명령에서 `<GCP_PROJECT_ID>`와 리전(예시는 서울 리전 asia-northeast3)을 실제 값으로 대체하십시오 [4]. `--update-env-vars`를 통해 환경변수도 배포 시점에 함께 설정하였습니다. (`ADMIN_TOKEN`, `API_KEY`, `DB_URL`의 값은 해당 터미널 세션에 환경변수로 설정되어 있다고 가정하거나 직접 `=값`으로 써넣으세요.)

배포가 성공하면 Cloud Run 서비스의 URL(예: `https://qualijournal-admin-<hash>-<region>.a.run.app`)과 리비전 정보가 출력됩니다 [82]. 해당 URL이 우리 관리자 대시보드에 접근하는 주소가 됩니다. (기억해두거나 북마크해두세요.)

Allow unauthenticated: 위 명령에서는 `--allow-unauthenticated`를 명시하여 Cloud Run이 **누구나 접근 가능**하도록 설정했습니다. 이는 앞서 설명한 대로, **앱 자체에서 토큰 인증을 할 것이므로 Cloud Run 레벨에서 막지 않기 위함**입니다 [6]. 만약 내부망에서만 쓰고 싶고 Cloud IAM 인증을 겹쳐 쓰고 싶다면 이 플래그를 제거하거나

`--no-allow-unauthenticated`로 설정할 수 있습니다. 그러나 그럴 경우 일반 브라우저에서 바로 접근이 불가하고, OAuth2 또는 JWT로 Google 인증을 통과해야 하므로 번거롭습니다 [6]. 대부분의 경우 `allow-unauthenticated: true`로 두는 것이 편리합니다. (현재 CI 설정에도 false로 해둔 예가 있을 수 있는데, 실제 운용을 고려해 결정하라고 언급하고 있습니다 [6].)

7.3 환경 변수 수정 (배포 후)

배포 커맨드에서 이미 env를 넣었다면 이 단계는 건너뛰어도 됩니다. 만약 **배포 후 추가/수정**할 환경변수가 생겼다면 `gcloud run services update` 명령으로 주입할 수 있습니다:

```
gcloud run services update qualijournal-admin \
  --update-env-vars API_KEY=new-api-key-value
```

이런 식으로 실행하면 새로운 리비전을 만들지 않고 환경변수만 업데이트할 수도 있습니다 (단, 서비스 재시작이 발생합니다). 또는 Cloud Run 콘솔 UI에서 해당 서비스를 편집하고 변수 값을 수정할 수 있습니다. 일반적으로 토큰이나 API 키 교체 시 위 방법으로 변경합니다.

7.4 배포 후 확인해야 할 체크리스트

배포를 완료했으면 실제 운영환경에서 모든 것이 잘 동작하는지 꼼꼼히 확인합니다. 다음 체크리스트를 따라 점검하세요:

1. **서비스 기본 동작 확인:** Cloud Run 서비스 URL (`qualijournal-admin` 서비스 URL)을 브라우저에서 열어봅니다. `index.html` (대시보드 화면)이 제대로 표시되는지 확인합니다. 만약 "Not Found" 또는 유사 에러가 나온다면, 아마 **정적 파일 경로 설정** 문제일 수 있습니다. FastAPI 앱이 `/` 경로 요청에 index.html을 반환하도록 구현되어야 합니다. (만약 구현 안했다면, `app.mount("/", StaticFiles(directory="static", html=True), name="frontend")` 등의 설정이 필요했을 수 있습니다. 또는 `/`에 대해 FileResponse로 index.html을 주도록 처리.) 어쨌든 **관리자 UI 화면**이 떠야 합니다. 화면이 보이면 HTML/JS/CSS 리소스가 모두 로드되었는지 개발자 도구(콘솔/네트워크 탭)로 확인합니다.
2. **상태 체크 API 테스트:** 토큰 없이 Cloud Run URL의 `/api/status`에 접근 시도해보고, **HTTP 401 Unauthorized**를 받는지 확인합니다 [83]. 그런 다음 올바른 토큰을 가지고 `/api/status`를 호출하여 **HTTP 200 OK**와 함께 JSON 데이터(예: 게이트 통과 여부 등)이 나오는지 봅니다 [84]. 이 `/api/status`는 예시이며, 혹시 존재하지 않는다면 `/api/report`를 바로 테스트해도 됩니다. 아무튼 하나의 보호된 API에 대해 토큰 없을 때 거부, 있을 때 통과를 확인함으로써 **환경변수로 설정한 ADMIN_TOKEN이 제대로 작동**하는지 검증합니다 [85].
3. 커맨드 예시:

   ```
   curl -i "https://<CloudRunURL>/api/status"
   curl -i -H "Authorization: Bearer YOUR_ADMIN_TOKEN" "https://<CloudRunURL>/api/status"
   ```

   첫 번째는 401, 두 번째는 200이 나와야 합니다.
4. **기능 동작 확인 (UI 통해):** Cloud Run 환경에서 관리자 UI에 접속하여 실제 기능 버튼들을 눌러봅니다. 로컬에서 테스트한 것과 동일하게 작동하는지 확인해야 합니다.
5. "일일 보고서 생성" 클릭 -> 약간의 대기 후 "보고서 열기" 링크 등장 -> 클릭하여 내용 확인. Cloud Run에서는 **파일시스템이 인스턴스 내에 한정**되어 있으므로, 링크가 열리는 주소가 `https://<CloudRunURL>/archive/reports/....md` 형태일 것입니다. 이 파일이 잘 열리면 OK입니다. 만약 링크 클릭 시 404 에

러가 난다면, 아마 **정적 파일 제공이 설정 안되었거나 경로 문제가 발생**한 것입니다. Cloud Run의 컨테이너 로그를 보고 파일이 생성되었는지, 경로가 맞는지 확인해야 합니다. (또는 `/api/report` 응답으로 content를 주도록 변경하는 임시 대응도 가능하지만, 원인 파악이 우선입니다.)

6. "키워드 요약/번역" 클릭 -> 정상적으로 완료되고 링크가 나타나는지 확인. 실행 전에 Cloud Run 환경변수에 번역 API_KEY를 설정했는지 다시 확인합니다. 키가 없으면 당연히 `error`가 나올 텐데, UI alert에서 그 메세지를 받는지도 확인하세요 (예: "번역 API 키 설정 필요"). 키가 있었는데도 에러가 난다면, 로그를 보아야 합니다. Cloud Run 로그에서 enrich_cards.py 실행 관련 traceback이나 메시지를 찾습니다. 또한, 번역 API 호출이 인터넷에 나가는 것이므로, Cloud Run 서비스에 필요한 **VPC egress 설정이나 API Endpoint 접근 권한** 등이 필요한지 점검합니다. (일반 웹 API라면 바로 나갈 수 있지만, 만약 Cloud Run이 VPC 내부에 있고 나가기 막혀있다면 설정 변경 필요.)
7. "선정본 요약/번역" 클릭 -> 위와 동일하게 확인.
8. "Export MD/CSV" 클릭 -> 누르는 즉시 파일 다운로드가 시작되는지, 다운로드가 안되면 브라우저 콘솔에 에러는 없는지 확인합니다. 만약 401이 떴다면 Authorization 헤더 전달이 잘 안된 것입니다. fetch 함수에서 header 설정을 재확인하세요. 또는 Cloud Run allow-unauthenticated 설정이 false여서 그런 것일 수도 있으니, 그 경우 Cloud Run 서비스 IAM에 프로젝트의 모든사용자(allUsers)에게 Invoker 권한을 부여하거나 (콘솔 IAM 설정), allow-unauth를 true로 바꾸고 redeploy 해야 합니다.
9. 다운로드된 CSV 파일을 엑셀로 열어 한글 깨짐 없는지, 컬럼이 의도한 대로 나오는지 확인합니다. Markdown 파일은 메모장 등으로 열어서 내용이 맞는지 봅니다.
10. **로그 모니터링:** Cloud Run의 Logs에서 **ERROR 레벨 메시지**나 **예상치 못한 경고**가 있는지 확인합니다. 특히 `make_daily_report`나 `enrich_cards` 실행 시 예외 trace가 찍히진 않았는지 봅니다. 만약 partial fail이 있었지만 처리한 경우, 그 메시지도 로그에 남겨 디버깅에 도움이 되도록 했을 수 있습니다 [34]. 예를 들어 "Translation API not set"을 로그에 남겼다면, 그런 부분을 확인하고 조치하세요.
11. **성능 및 리소스 확인:** 초기 Cloud Run 인스턴스 기동 시 메모리/CPU 사용량을 모니터링합니다 (Cloud Run Metrics 활용). `enrich_cards.py` 실행 시 메모리를 많이 잡아먹거나 CPU를 많이 쓸 수 있으니, 디폴트 256Mi, 1 vCPU로 부족하면 Cloud Run 설정에서 인스턴스 크기를 높여야 할 수도 있습니다. 또한 동시 요청에 대한 설정 (concurrency)을 고민해볼 수 있는데, 관리자 시스템은 사용자가 1~2명 정도일 것이므로 동시성 1로 두어도 무방합니다.
12. **CI/CD 동작 확인:** 만약 GitHub Actions를 설정해두었다면, main에 머지했을 때 CI가 빌드/배포를 수행하는지 살펴봅니다 [86]. 로그를 통해 이미지 푸시와 배포가 성공했는지 확인하고, CI 환경변수에 ADMIN_TOKEN 등이 설정되어있지 않아 실패하지는 않았는지 봅니다. 필요하다면 GitHub 저장소 Settings - Secrets에 ADMIN_TOKEN, API_KEY 등을 추가하고 워크플로우 yaml의 `--set-env-vars`에 반영합니다 [80].
13. **운영 습관:** 배포 후에는 **일일 운영 체크리스트**에 따라 시스템을 점검합니다. 예를 들면:
14. 아침에 접속해 `/api/status`로 수집된 기사 수와 승인된 수, gate_pass 여부를 확인 (이를 통해 오늘 발행 가능 여부 등 파악).
15. 뉴스 수집이 별도 프로세스로 돌아간다면 그 결과 (`selected_keyword_articles.json`이 갱신되었는지 등) 확인.
16. 필요한 경우 "키워드 요약/번역"을 눌러 새로 수집된 기사들의 요약본을 미리 살펴보고 품질 확인.
17. 편집 작업(기사 승인/코멘트 작성 등)은 기존 대로 JSON을 편집하거나 UI에서 한다면 그 부분 사용.
18. 발행 시점에 "일일 보고서 생성"을 눌러 최종 리포트 작성 -> 내용 확인 -> "Export"로 산출물 다운로드 및 보관.

19. 끝으로 당일 Cloud Run 로그에 이상 없었는지, 에러는 없었는지 확인. 이러한 루틴을 정하고 지키면 시스템 운영이 안정적입니다.

20. **문제 발생 대응:** 만약 어떤 기능이 오작동하면, 우선 긴급하게 수동으로 처리하고 (예: 번역 API 동작 안 하면 영문 요약만으로 발행, 보고서 생성 실패하면 로그 기반으로 수작업 작성 등) 사후에 문제를 해결합니다. 이 때 이 가이드북을 참고해, 관련된 부분 (환경 설정, 코드, CI 설정 등)을 점검합니다. 필요 시 **에러 메시지별 조치법**을 운영 매뉴얼에 추가해두세요 (예: "보고서 생성 실패: 데이터 없음 -> selected_articles.json에 기사가 하나도 없는지 확인" 같은 Q&A) [87].

21. **데이터 백업:** Cloud Run 컨테이너는 무상태이므로, `archive/` 디렉토리의 파일들은 영구 보존되지 않습니다 (배포 새 리비전 시 초기화됨). 따라서 보고서 Markdown이나 요약본 파일을 따로 아카이빙하려면 **Export CSV/MD 기능으로 내려받아 로컬 또는 별도 저장소에 보관**해야 합니다 [62]. 예를 들어 Git으로 매일 리포트를 커밋해두거나, Cloud Storage에 업로드하는 스크립트를 추가하는 것도 고려할 수 있습니다. 현재 버전에서는 수동 Export에 의존하지만, 향후 자동화할 수 있습니다.

모든 점검을 마치면 QualiJournal 관리자 시스템의 Cloud Run 배포가 성공적으로 완료된 것입니다. 이제 운영자는 본 가이드북과 README에 정리된 절차를 참고하여, 일일 뉴스를 원활하게 관리하고 발행할 수 있을 것입니다. 향후 기능 추가나 변경 시에는 이 문서를 업데이트하며, 테스트와 CI 파이프라인을 통해 안정성을 유지하시기 바랍니다. 이상으로 QualiJournal 관리자 시스템 최종 배포/운영 가이드 작업을 마칩니다.

---

[1] [2] [3] [7] [8] [9] [11] [12] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [66] [67] [69] [70] [71] 1011_3QualiJournal 관리자 시스템 최종 가이드북.pdf
file://file-6Hu18vVrjQweSb5SZz6a6M

[4] [5] [10] [13] [14] [27] [64] [77] [82] [83] [84] [85] [86] 1013_1QualiJournal 관리자 시스템 최종 통합 작업 가이드북.pdf
file://file-6DfuAdPY3q666RS3cV5ydM

[6] [65] [79] [80] [81] 1012_2QualiJournal 관리자 시스템 작업 가이드북.pdf
file://file-Ad1vex4HDdK79hKp9LXwzj

[68] [72] [73] [74] [75] [76] [87] 1011_4QualiJournal 관리자 시스템 운영 가이드북.pdf
file://file-CVSpPWa43WEvDV2r8Qoh3k

[78] 1012_1QualiJournal 관리자 시스템 최종 단계별 작업 가이드북.pdf
file://file-8wm4iV9GGuoV7FL4ABABAZ