ChatGPT

QualiJournal 관리자 시스템 최종 통합 작업 가이드북

QualiJournal 관리자 시스템(백엔드 FastAPI 및 프론트엔드 HTML/JS)의 최신 개발 현황과 구현 방법을 정리한 가이드북입니다. 이 문서는 개발자와 운영 담당자가 **최종 통합 작업**을 순차적으로 따라할 수 있도록 작성되었습니다. 이미 **게이트 슬라이더 구현**, **KPI 자동 새로고침**, **ADMIN 토큰 인증 보안 적용**, **테마/인코딩 개선**, **코드 리팩터링** 등이 완료된 상태이며, 이 가이드북에서는 **남은 핵심 작업**들을 다룹니다 [1] [2]. 각 섹션은 **목적**, **준비 파일 또는 조건**, **실행 절차**, **CLI/API/JS 예시 코드**, **예상 결과**, **주의사항**으로 구성되며, Cloud Run 배포 환경을 기준으로 설명합니다. 순서대로 따라하면서 최종 기능 구현과 배포 및 운영 준비를 완료해 보세요 [3] [4].

1. 환경변수 설정 및 서버 실행 (.env / Cloud Run 환경)

목적

FastAPI 백엔드와 프론트엔드가 정상 동작하기 위해 필요한 **환경변수**를 설정하고 서버를 실행합니다. 특히 관리자 인증을 위한 `ADMIN_TOKEN`, 외부 API 사용을 위한 `API_KEY`, (필요 시) 데이터베이스 연결 `DB_URL` 등을 `.env` 파일과 Cloud Run 환경 변수에 올바르게 구성하여 **보안 설정**을 완료하는 것이 목표입니다 [5] [6]. 이를 통해 로컬 개발 환경과 클라우드 배포 환경 모두에서 동일한 설정으로 서버를 구동할 수 있습니다.

준비 파일 또는 조건

- 프로젝트 루트 경로에 `.env` **파일** (또는 동일한 내용의 환경 변수 설정)이 있어야 합니다. 이 파일에 아래와 같은 핵심 키들을 정의합니다 [7]:

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

  (`DB_URL`은 현재 JSON 파일을 DB로 대체하려는 경우에만 사용됩니다.)
- 위 `.env` 파일은 **Git 저장소에 커밋하지 않고** `.gitignore`에 포함시켜야 합니다 [8]. 운영 환경에서는 해당 값들을 Cloud Run 설정 또는 CI/CD 시크릿으로 관리합니다.
- (선택) **Cloud Run** 서비스가 미리 설정되어 있다면, 서비스의 환경 변수로 `ADMIN_TOKEN`, `API_KEY`, `DB_URL`을 추가할 수 있습니다.

실행 절차

1. **로컬 환경에서 서버 실행 전 `.env` 적용**: 로컬 개발 시 `.env` 파일의 변수를 시스템 환경에 로드한 후 서버를 실행합니다. 일반적으로 `python-dotenv`를 사용하거나, 아래와 같이 수동으로 `.env`를 source 한 뒤 uvicorn으로 서버를 구동할 수 있습니다:

```
# .env 파일 로드
source .env
# FastAPI 서버 실행 (uvicorn)
uvicorn app.main:app --host 0.0.0.0 --port 8000
```

이렇게 하면 `.env`에 지정된 `ADMIN_TOKEN`, `API_KEY` 등이 애플리케이션 실행 시 환경변수로 적용됩니다 [9] [10].

2. **Cloud Run 환경 변수 설정**: GCP Cloud Run에 배포할 때는 CLI나 콘솔을 통해 환경변수를 설정합니다. 예를 들어 Cloud Run을 업데이트할 경우:

```
gcloud run services update qualijournal-admin \
 --update-env-vars ADMIN_TOKEN=<발급한토큰>,API_KEY=<번역API키>,DB_URL=<DB연결URL>
```

또는 GCP 콘솔의 **Variables** 설정 화면에서 `ADMIN_TOKEN`, `API_KEY`, `DB_URL` 값을 수동으로 등록합니다 [11]. CI/CD를 사용한다면 GitHub Actions에서 배포 시 시크릿을 환경변수로 전달하도록 설정합니다.

3. **서버 실행 및 확인**: 로컬에서 `uvicorn`으로 서버를 실행하거나, Cloud Run 배포 후 서비스를 구동하여 **시작 로그**를 확인합니다. 서버 시작 시 로드된 환경변수들이 로그에 표시될 수 있으며 (`ADMIN_TOKEN`, `DB_URL` 등의 설정 로드 메시지) [10], 이후 관리자 API 호출 시 해당 토큰이 요구됩니다. 프론트엔드도 동일한 `ADMIN_TOKEN` 값을 알고 있어야 정상적으로 API를 호출할 수 있습니다 [12]. (DB를 사용하는 경우 서버 시작 시 DB 연결 시도 로그가 나타나고, `/api/status` 호출 시 DB 기반 통계를 반환하는 등 약간의 동작 변화가 있을 수 있습니다 [13].)

CLI/API/JS 예시 코드

- **Cloud Run 환경 변수 설정 (gcloud CLI)**:

```
gcloud run services update qualijournal-admin \
 --update-env-vars ADMIN_TOKEN=${ADMIN_TOKEN},API_KEY=${API_KEY},DB_URL=${DB_URL}
```

위 명령은 Cloud Run 서비스에 필요한 환경변수를 한꺼번에 설정하는 예시입니다. `<배포서비스명>`과 변수 값들은 실제 값으로 교체하세요.

- **서버 실행 (로컬)**:

```
source .env && uvicorn app.main:app --reload
```

`.env` 로드 후 uvicorn 개발 서버를 실행합니다. (또는 `Dockerfile` 실행 시에도 `ENV` 지시자나 CI에서 `.env`를 적용할 수 있습니다.)

예상 결과

- **로컬 환경**: `.env`에 설정한 환경변수가 FastAPI 애플리케이션에 적용되어, 예를 들어 `ADMIN_TOKEN` 값을 백엔드에서 `os.getenv("ADMIN_TOKEN")`으로 가져와 인증에 활용합니다 [14] [10]. 서버 콘솔 로그에 "Loading ADMIN_TOKEN..." 등의 메시지가 나타날 수 있습니다.
- **Cloud Run 환경**: Cloud Run 배포 시 지정한 환경변수가 컨테이너에 주입되어 동일하게 적용됩니다 [15] [10]. Cloud Run의 **Revision details**에서 해당 환경변수들이 설정되었는지 확인할 수 있습니다. 모든 관리자용 API는 이제 이 `ADMIN_TOKEN`을 요구하며, 올바른 토큰을 전달하지 않으면 **401 Unauthorized** 응답을 하게 됩니다 [16].
- **ADMIN_TOKEN 인증 동작**: 이후 ADMIN 전용 API 호출 시 요청 헤더에 `Authorization: Bearer <ADMIN_TOKEN>`을 포함해야 정상 응답을 받습니다. 잘못된 토큰이나 미제공 시 **401/403** 응답으로 거부됨을 확인하게 됩니다 [17]. 프론트엔드 측에서도 해당 토큰을 알고 있어야 하며(예: `localStorage` 등에 저장), API 호출 fetch 요청 시 헤더에 추가하도록 구현해야 합니다.

주의사항

- **민감정보 관리**: `.env` 파일은 **절대로 Git 등 원격 저장소에 커밋하지 않아야** 하며, 프로젝트 `.gitignore` 에 반드시 포함시킵니다 [8]. 모든 민감한 키(`ADMIN_TOKEN`, `API_KEY`, `DB_URL` 등)는 코드에 **하드코딩하지 말고 환경변수**로 관리하세요.
- **ADMIN_TOKEN 보안**: `ADMIN_TOKEN` 값은 **예측 불가능한 복잡한 랜덤 문자열**로 생성하는 것이 좋습니다 [18]. 운영 중에는 정기적으로 토큰 교체 주기를 가져가는 것도 고려합니다 [19]. 토큰을 변경하면 **프론트엔드에도 동일한 값으로 업데이트**해야 함을 잊지 마십시오 [18]. 토큰이 유출되지 않도록 콘솔 출력, 로그 등에 노출하지 않고, 토큰 없이 접근 시도에 대해서도 보안을 모니터링해야 합니다 [20] [21].
- **API_KEY 설정**: `API_KEY` 는 Papago, Google Translate 등 **외부 번역 API 키**를 가리킵니다. 해당 키를 설정하지 않으면 뉴스 요약문이 영어로만 출력되고 번역 단계는 생략됩니다 [22]. 번역 기능을 사용하려면 해당 API의 사용 한도 및 과금 정책도 함께 숙지하십시오 [22] [23].
- **DB_URL 및 DB 사용**: 현재 시스템은 기본적으로 JSON 파일로 데이터를 저장하고 있으나, 향후 데이터베이스(PostgreSQL 등)를 사용하도록 전환할 수 있습니다. `DB_URL` 은 그 준비를 위한 값입니다 [24]. 만약 배포 환경에서 DB를 사용한다면 **VPC 네트워크 연결**, **인증 정보**, **방화벽 규칙** 등이 올바르게 설정되어야 합니다 [24]. 초기 도입 시에는 마이그레이션을 신중히 진행하고 충분한 백업 후 실시하세요. Cloud Run에서 DB에 접속하려면 VPC Connector 설정이 필요하며, 배포 후 DB 연결 오류가 없는지 꼭 확인해야 합니다 [25] [26].

---

2. 일일 보고서 생성 (POST `/api/report`)

목적

일일 보고서는 하루 동안 수집된 뉴스 데이터의 주요 지표와 콘텐츠 요약을 한데 모은 특별 기사입니다. 이 섹션에서는 해당 **Daily Report를 자동으로 생성**하는 기능을 구현합니다. 기존에는 수동으로 `tools/make_daily_report.py` 스크립트를 실행하여 마크다운 리포트를 작성했지만, 이제 **백엔드 API 엔드포인트**와 **관리자 UI 버튼**을 통해 클릭 한 번으로 보고서를 생성하고 열람할 수 있도록 개선합니다 [27]. 이를 통해 운영자는 매일 손쉽게 당일 보고서를 생성하여 확인하거나, 기록으로 남길 수 있게 됩니다.

준비 파일 또는 조건

- **백엔드 스크립트**: `tools/make_daily_report.py` – 일일 보고서 생성 로직을 담은 기존 Python 스크립트가 있어야 합니다 [28]. 이 스크립트는 당일(또는 전일)의 키워드와 관련 뉴스 데이터를 모아 Markdown 포맷의 보고서를 생성하는 기능을 합니다.
- **백엔드 라우트 파일**: FastAPI에 새로운 엔드포인트를 추가해야 합니다. 예를 들어 `app/main.py` 또는 별도 `app/routes/report.py` 등에 **POST** `/api/report` 경로를 생성합니다 [28].
- **프론트엔드 파일**: 관리자 대시보드 HTML/JS 페이지에서 "일일 보고서 생성" 버튼 UI를 추가하고, 해당 버튼 클릭 시 API 호출(`POST /api/report`)을 수행하는 코드를 작성해야 합니다 [29]. (예: 관리도구 페이지에 버튼 `<button id="btn-generate-report">보고서 생성</button>` 추가)

실행 절차

1. **백엔드 엔드포인트 구현**: FastAPI 애플리케이션에 `POST /api/report` 라우트를 정의합니다. 이 엔드포인트는 `ADMIN_TOKEN` 인증이 필요하므로, 헤더에 인증 토큰을 검사하는 데코레이터나 `Depends` 미들웨어를 적용합니다 [30] [31]. 엔드포인트 내부 로직에서는 `make_daily_report.py` 의 핵심 함수를 호출하여 **당일 보고서 Markdown 생성**을 수행합니다. 구현 예시 (개략):

```
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:
        markdown_content = make_daily_report()  # 당일 보고서 Markdown 문자열 생성
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Report generation failed: {e}")
    # 생성된 Markdown 문자열이나 파일 경로를 반환
    return {"content": markdown_content}
```

위 예시는 개념적 설명을 위한 코드입니다. 실제 `make_daily_report()` 함수는 내부에서 당일 키워드에 대한 뉴스 통계를 모으고 Markdown 텍스트를 반환하거나 파일로 저장할 것입니다. 중요한 것은 **예외 처리**를 통해 보고서 생성에 필요한 데이터가 부족할 경우 500 에러나 적절한 에러 메시지를 주는 것입니다 [32].

2. **API 엔드포인트 테스트 (로컬)**: 구현한 `/api/report` 엔드포인트를 로컬 환경에서 서버 실행 후 `curl` 등을 이용해 테스트합니다. 이때 **인증 헤더**를 포함해야 하며, 잘못된 토큰으로 호출하면 401 응답이 오는지 확인합니다. 예시:

```
uvicorn app.main:app --reload  # FastAPI 서버 실행 (개발 모드)
curl -X POST "http://localhost:8000/api/report" \
  -H "Authorization: Bearer ADMIN_SECRET_TOKEN"
```

3. 잘못된 토큰이나 토큰 없이 호출한 경우 `401 Unauthorized` 응답이 반환되어야 합니다 [30] [33].
4. 올바른 토큰으로 호출하면 **JSON 형태의 보고서 데이터**가 응답으로 와야 합니다. 예를 들어:

```
{ "success": true, "report_file": "2025-10-12_report.md" }
```

또는

```
{ "content": "# 2025-10-12 Daily Report\n..." }
```

와 같은 형태로, 생성된 Markdown 내용을 직접 주거나 파일 경로를 알려줄 수 있습니다 [34].

5. **프론트엔드 연동**: 관리자 UI에 **"일일 보고서 생성"** 버튼을 추가하고, JavaScript로 해당 API를 호출하는 코드를 작성합니다. 예를 들어 jQuery 없이 순수 JS로 구현한다면:

```
document.getElementById('btn-generate-report').onclick = async function() {
  const token = localStorage.getItem('adminToken');  // 저장된 ADMIN_TOKEN
  try {
    const res = await fetch('/api/report', {
      method: 'POST',
      headers: { 'Authorization': 'Bearer ' + token }
    });
    if (!res.ok) {
      alert("보고서 생성 실패: " + res.status);
      return;
    }
```

```
      const data = await res.json();
      console.log("Report Generation Result:", data);
      alert("일일 보고서 생성 완료! 파일: " + (data.report_file || "내용 확인"));
      // TODO: data.content가 있다면 새 창에 Markdown 미리보기 띄우는 등 구현
    } catch (e) {
      console.error(e);
      alert("보고서 생성 중 오류 발생");
    }
  }
```

위 코드는 버튼 클릭 시 `/api/report` POST를 호출하고 결과를 alert으로 간단히 표시합니다 [35] [36]. 실제로는 `data.content`를 받아 화면에 표시하거나, 새로운 창/탭으로 Markdown을 보여주는 등 UX를 개선할 수 있습니다.

CLI/API/JS 예시 코드

- **REST API 호출 예시 (cURL)** – 일일 보고서 생성 트리거:

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

※ 요청 본문은 필요 없습니다. 인증 헤더만 포함하면, 서버에서 **현재 날짜의 보고서 생성 작업**을 수행합니다 [37]. 응답은 성공/실패 여부와 생성된 보고서 내용 또는 경로를 나타내는 JSON이 됩니다.

- **REST API 호출 예시 (Fetch)** – 일일 보고서 생성 트리거 (프론트엔드 JS):

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

위와 같이 JS 코드로도 호출할 수 있으며, 결과(JSON)에 대해 필요한 UI 처리를 수행하면 됩니다 [38].

예상 결과

- **백엔드 동작**: `POST /api/report` 호출 시 내부적으로 `make_daily_report.py`의 로직이 실행되어 **금일자 Markdown 보고서**를 생성합니다 [39]. 성공 시 API는 생성된 보고서의 내용을 직접 반환하거나, 서버 내에 저장된 파일 경로를 응답합니다 (예: `"2025-10-12_report.md"` 형태) [40] [41]. 또한 서버의 `archive/` 디렉토리에 날짜별 보고서 파일이 저장되도록 구현할 수 있습니다 (예: `archive/2025-10-12_report.md` 파일 저장) [42].
- **프론트엔드 동작**: 운영자가 관리자 대시보드에서 **"일일 보고서 생성"** 버튼을 클릭하면 해당 API가 호출되고, **성공 메시지**나 **생성된 보고서 표시**가 이루어집니다 [43] [44]. 예를 들어, "보고서 생성 완료" 알림을 띄우거나 새 창으로 Markdown 내용을 보여주는 UI를 구현할 수 있습니다. (향후 개선으로, 생성된 Markdown을 HTML로 렌더링하여 미리보기하거나, 바로 다운로드 링크를 제공하는 것도 고려할 수 있습니다.)
- **응답 예시**: 보고서 생성 성공 시 응답 예시는 다음과 같습니다:

```
{ "success": true, "report_file": "2025-10-12_report.md" }
```

또는

```
{ "content": "# 2025-10-12 Daily Report\n...\n" }
```

프론트엔드는 이 정보를 받아 사용자에게 **보고서가 생성되었음**을 알리고 필요한 후속 조치를 취합니다 [41]. (예: 파일 이름을 받아 "다운로드" 버튼을 노출하거나, content를 받아 화면에 표시)

주의사항

- **엔드포인트 추가 구현**: 기존에 `GET /api/report` 엔드포인트가 있었더라도, **보고서 생성을 트리거하는 기능은 새로운 POST로 구현**해야 합니다 [45]. 구현 시 `make_daily_report` 함수(또는 유사 함수)를 호출하고, 필요한 예외 처리를 할 것. 데이터 부족 등으로 보고서 생성이 불가능한 경우 적절한 오류 응답(예: 400 혹은 500 에러)을 반환합니다.
- **중복 생성 방지**: 같은 날짜에 여러 번 `/api/report`를 호출하면 **중복된 보고서 파일**이 생성되거나 내용 충돌이 발생할 수 있습니다 [46]. 이를 방지하기 위해:
- 이미 해당 날짜 보고서 파일이 있다면 새로운 요청을 무시하거나 덮어쓰도록 할지 결정합니다.
- 개선된 구현에서는 동일 날짜로 여러 번 생성 시 **타임스탬프를 파일명에 포함**하거나, 기존 파일을 백업/압축하는 방법을 권장하고 있습니다 [47]. 예를 들어 두 번째 생성부터는 `2025-10-12_report-2.md` 같이 이름을 바꾸거나, archive 폴더에 일정 기준 이상 파일이 쌓이면 **정리 정책**을 수립합니다.
- **UI 피드백 및 진행 표시**: 보고서 생성 작업은 **수 초 이상 시간이 소요**될 수 있습니다 (데이터 수집량에 따라). 따라서 사용자가 버튼 클릭 후 아무 반응이 없다고 느끼지 않도록 **로딩 스피너**나 "생성 중..."과 같은 진행 상태 표시를 UI에 추가하는 것이 좋습니다 [48]. 또한 **생성 완료 후 메시지**(또는 오류 발생 시 메시지)를 명확히 표시하여 사용자 경험을 높입니다. (예: "보고서가 생성되었습니다. (파일: 2025-10-12_report.md)" 또는 "보고서 생성 실패: 데이터 부족" 등의 경고를 alert이나 화면상에 표시)
- **인증 및 권한**: 이 API는 **관리자 전용 기능**이므로 반드시 ADMIN 토큰 인증을 요구해야 합니다. 토큰이 없거나 잘못된 경우 401 Unauthorized를 반환하는지 다시 한번 확인하세요 [33]. 배포 환경에서도 해당 동작을 동일하게 검증합니다. (예: Cloud Run 로그에서 인증 실패 시도를 모니터링하여 문제 발생 시 토큰 유출 여부 등을 점검합니다.)
- **데이터 준비 상태**: 보고서 생성을 하기 전에 당일 키워드에 대한 **기사 수집 및 승인 작업이 완료**되었는지 확인하는 것이 좋습니다 [49]. 만약 승인된 기사 수가 최소 필요치에 미달하는 등 데이터가 불충분하면, 생성된 보고서 품질이 떨어질 수 있습니다. 이런 경우 `/api/status` 등의 엔드포인트로 현재 `selection_approved`(승인된 기사 수)와 `gate_required`(최소 필요 기사 수)를 비교하여 부족할 경우 보고서 생성을 **보류**하거나 경고를 주는 기능을 고려하세요 [49].

---

3. 뉴스 카드 요약/번역 (POST `/api/enrich/keyword`, `/api/enrich/selection`)

목적

수집된 뉴스 기사들에 대해 **요약 및 번역** 기능을 제공하여, 운영자가 다수의 기사를 빠르게 요약하고 필요한 경우 다른 언어로 번역된 **뉴스 카드**를 얻을 수 있도록 합니다. 이 기능은 두 가지로 구분됩니다: 1. **키워드 전체 뉴스 요약** – 해당 키워드로 수집된 **모든 기사**의 핵심 내용을 자동 요약하고 (옵션으로) 번역까지 수행 [50]. 2. **선정본 뉴스 요약** – 편집자가 승인하여 발행 대상으로 선정된 **최종 기사들**의 요약 및 번역본 생성 (최종 발행본에 사용될 내용) [50].

예를 들어 영문 기사들이 수집되었다면, **키워드 전체 요약** 기능으로 모든 기사 본문을 한꺼번에 요약하고 한국어로 번역하여 요약문을 얻을 수 있습니다. 또한 **선정본 요약** 기능으로 최종 발행할 기사들의 요약문을 생성하고 국문 번역하여, 최종 발행본에 바로 활용할 수 있습니다 [51] [52]. 이를 통해 운영자는 버튼 클릭만으로 다수 기사에 대한 핵심 요약문과 번역문을 확보하여 편집 시간을 절약할 수 있습니다.

준비 파일 또는 조건

- **백엔드 스크립트**: `tools/enrich_cards.py` – 뉴스 카드 요약 및 번역을 수행하는 기존 스크립트가 있다고 가정합니다 [53]. 이 스크립트는 수집된 기사 목록(JSON 파일 등)에서 각 기사의 본문을 가져와 요약(AI API 사용)하고, 번역(API_KEY 사용)하는 기능을 포함하고 있을 것입니다.
- **백엔드 라우트 파일**: 새로운 API 엔드포인트 **POST** `/api/enrich/keyword`, **POST** `/api/enrich/selection`을 추가해야 합니다 [53]. FastAPI의 `APIRouter`에 두 경로를 정의하고, ADMIN 토큰 인증을 적용합니다. 구현은 `enrich_cards.py`의 함수를 호출하거나, 직접 해당 로직을 수행하도록 구성합니다.
- **외부 API 연동 모듈**: OpenAI, Naver Papago 등 **요약/번역을 위한 외부 API 호출 코드**가 필요합니다 [54]. 예를 들어 OpenAI GPT-3.5/4 API를 사용할 경우 `openai` Python 라이브러리를 이용하거나, Papago 번역 API를 사용할 경우 `requests`를 통해 HTTP 호출을 구현합니다. 이때 필요한 API 키는 환경변수 `API_KEY`를 통해 주입받도록 합니다 [55].
- **프론트엔드**: 관리자 대시보드 UI에 "요약/번역" 관련 버튼들을 추가해야 합니다 [56] [57]. 예를 들어:
- "키워드 전체 요약" 버튼 (모든 기사 요약/번역 실행)
- "선정본 요약" 버튼 (선정된 기사들만 요약/번역 실행)
  각 버튼에 대응하는 클릭 이벤트에서 `/api/enrich/keyword` 또는 `/api/enrich/selection` API를 호출하도록 JS를 작성합니다.

실행 절차

1. **환경 변수 설정 (API 키)**: 먼저 `.env` 또는 Cloud Run 환경에 `API_KEY` 값을 설정합니다. OpenAI API 키, Papago API 키 등 **요약/번역에 사용될 외부 서비스 키**를 확보하여 등록하세요 [58]. 예를 들어 OpenAI를 사용한다면 `.env`에 `API_KEY=<OpenAI-발급키>` 형식으로 넣고, 코드에서 `openai.api_key = os.getenv("API_KEY")`로 설정합니다 [59]. Papago와 같은 경우에는 요청 시 헤더에 `X-Naver-Client-Id`, `X-Naver-Client-Secret` 등을 함께 보내야 하므로, 해당 값들도 환경변수로 관리하세요.
2. **백엔드 엔드포인트 구현**: FastAPI에 `/api/enrich/keyword`와 `/api/enrich/selection` POST 라우트를 추가합니다. 이때 요청 본문은 필요 없거나, 있더라도 간단한 플래그/옵션 정도로 처리 가능합니다 (예: `lang=ko` 쿼리파라미터로 목표 언어 지정 등). 두 엔드포인트 모두 ADMIN 인증이 필요하므로 `Depends(verify_admin_token)` 적용은 필수입니다 [60] [61]. 구현 로직은 다음과 같이 구성됩니다:
3. `/api/enrich/keyword`: `enrich_cards.py` 내 함수를 호출하여 **모든 수집 기사**에 대한 요약문을 생성합니다. 각 기사별로 AI 요약 API를 호출하고, 결과 요약문을 생성합니다. 또한 `API_KEY`가 있다면 요약문을 한국어 등 **목표 언어로 번역**하여 저장합니다 [62] [63].
4. `/api/enrich/selection`: 위와 비슷하지만 대상 기사를 **승인된 기사들 (selection)**로 한정합니다. 최종 발행 대상 기사들의 요약 및 번역문을 생성합니다 [50]. 이 결과는 실제 발행 콘텐츠에 직접 활용될 수 있으므로 정확성이 특히 중요합니다.
5. 결과 처리: 요약/번역 결과를 각 기사 데이터 구조에 추가하거나(`summary` 필드, `translation` 필드 등) 별도의 결과물을 생성합니다 (예: `enriched_<date>.json` 파일 또는 Markdown 파일로 저장) [64]. 간단히는 성공 여부만 JSON으로 반환하거나, 처리한 기사 수 등을 응답할 수 있습니다.
6. **외부 API 호출 구현**: OpenAI GPT 예시로, 기사 요약은 다음과 같이 구현할 수 있습니다:

```
import openai
from pydantic import  BaseModel

class ArticleInput(BaseModel):
    text:  str

@router.post("/api/enrich/keyword")
def summarize_all_articles(token:  str  =  Depends(verify_admin_token)):
    # 1. 모든 기사 본문을 불러오기 (예: selected_keyword_articles.json)
```

```
    texts = load_all_articles()  # 사용자 정의: 기사 본문 리스트
    summaries = []
    for text in texts:
        prompt = f"다음 기사를 한국어로 요약해줘:\n\"\"\"\n{text}\n\"\"\""
        try:
            res = openai.Completion.create(engine="text-davinci-003", prompt=prompt,
max_tokens=1024)
        except Exception as e:
            logging.error(f"OpenAI API error: {e}")
            continue
        summary_text = res.choices[0].text.strip()
        summaries.append(summary_text)
    # (번역 API_KEY가 있으면 summary_text를 번역하여 translation 추가)
    save_summaries(summaries)  # 결과 저장 또는 전역 변수 업데이트
    return {"count": len(summaries), "message": "Summaries generated."}
```

위 코드는 이해를 돕기 위한 단순 예시입니다. 실제 구현에서는 비동기 호출 또는 Batch API 등을 검토하세요. OpenAI API 호출 시 **모델 토큰 제한과 비용**을 고려해 `max_tokens` 등을 조절합니다 [65] [66]. 한편, Papago API를 이용한다면 별도의 번역 함수 (`translate(text, target_lang)`)를 만들어 요약문을 번역하여 함께 저장합니다.

7. **프론트엔드 연동**: 관리자 UI에 **"전체 기사 요약"**, **"선정본 요약"** 버튼을 추가하고 API 호출 스크립트를 작성합니다. 예:

```
// 전체 기사 요약 요청 버튼
document.getElementById('btn-enrich-all').onclick = function() {
  fetch("/api/enrich/keyword", { method: "POST", headers: { "Authorization": "Bearer " +
adminToken }})
    .then(res => res.json())
    .then(data => console.log("Keyword enrich result:", data));
};
// 선정본 요약 요청 버튼
document.getElementById('btn-enrich-selected').onclick = function() {
  fetch("/api/enrich/selection", { method: "POST", headers: { "Authorization": "Bearer " +
adminToken }})
    .then(res => res.json())
    .then(data => console.log("Selection enrich result:", data));
};
```

실제 구현 시에는 응답 데이터에 따라 "**요약이 완료되었습니다**" 등의 메시지를 보여주거나, 요약문이 담긴 결과를 화면에 표시하는 처리가 필요합니다 [57] [67]. 예를 들어 모든 기사 요약이 끝나면 각 기사 카드에 요약문을 함께 출력하거나, 팝업 창에 요약 결과를 모아서 보여줄 수 있습니다.

CLI/API/JS 예시 코드

- **REST API 호출 (cURL)** – 키워드 전체 기사 요약/번역:

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

(모든 수집된 기사를 대상으로 요약/번역을 수행) [60]

- **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) [68]

- **REST API 호출 (Fetch)** – 프론트엔드 JS 예시:

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

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

※ 이 API는 **요청 본문 없이 POST만으로 작동**하도록 구현했습니다. 필요에 따라 요약 대상 언어 등을 쿼리스트링이나 본문으로 지정할 수도 있습니다 (예: `/api/enrich/keyword?lang=ko` 등) [69].

예상 결과

- **백엔드 동작**: 새로 구현된 `/api/enrich/keyword`를 호출하면 `enrich_cards.py`를 통해 **모든 수집 기사 본문에 대한 요약문**이 생성됩니다 [62]. `/api/enrich/selection` 호출 시 **선정된 기사들만** 추려서 요약/번역이 수행됩니다 [62] [70]. 처리 결과는:
- 각 기사 데이터에 요약문 필드(`summary`)와 번역문 필드(`translation`)가 추가되어 메모리에 유지되거나 파일에 저장됩니다 [71].
- 또는 별도로 생성된 결과 파일(e.g. `archive/enriched/2025-10-12.json` 혹은 Markdown)을 출력할 수 있습니다.
- **번역 적용 여부**: 환경변수 `API_KEY`(번역 API 키)가 설정되어 있다면, **영어 요약문을 한국어로 번역**하거나 그 반대 등 **목표 언어로 번역문**을 함께 생성합니다 [72]. 만약 키가 없거나 설정되지 않은 경우 **요약은 원문 언어로만 수행**되고 번역 단계는 생략됩니다 [72]. (예: Papago API 키를 넣었다면 영문 기사가 자동으로 한글 번역되고, 미설정 시 요약문은 영어로 남음)
- **프론트엔드 동작**: 관리자 UI에 **"요약/번역"** 기능 버튼들이 표시되고, 운영자가 클릭 시 해당 API가 호출되어 작업이 시작됩니다 [52] [73]. 성공적으로 완료되면:
- "전체 기사 요약"의 경우 요약 결과를 활용하여 **UI에 요약문을 표시**할 수 있습니다. 예를 들어 기사 리스트에서 각 기사 아래에 요약문을 작은 글씨로 보여주거나, 별도 모달 창에 모든 요약 결과를 나열할 수 있습니다.
- "선정본 요약"의 경우 **최종 발행본의 번역문 준비**가 주된 목적입니다. 번역이 완료되면 곧바로 그 내용을 Markdown 보고서에 반영하거나, 사용자가 확인 후 Export할 수 있게 처리합니다 [52] [74].
  운영자는 번역된 최종 요약문으로 발행본을 검토하고, 필요시 편집할 수 있습니다.
- **응답 및 로그**: API 응답은 처리 결과(예: 요약한 기사 수, 성공 여부 메시지 등)를 JSON으로 간략히 줄 수 있습니다. 대량의 텍스트(요약문 자체)는 파일에 저장하고 응답에는 요약 개수만 주는 방식을 택할 수도 있습니다. 또한 **서버 로그**에는 외부 API 호출 여부, 결과 요약/번역문 생성 여부 등이 기록되어 추후 디버깅에 도움이 됩니다 [75].

주의사항

- **엔드포인트 구현**: 현재 서버에는 `/api/enrich/keyword` 및 `/api/enrich/selection` 엔드포인트가 없으므로 **직접 추가 구현이 필요**합니다 [76]. 이 작업은 앞서 보고서 생성과 유사하게 FastAPI 라우트를 정의하는 수준이지만, **작업 시간이 오래 걸릴 수 있다는 점**을 유의하세요. 기사 10여 개 이상을 순차적으로 요약/번역하면 수십 초까지 소요될 수 있으므로, 필요하다면 **비동기 처리**(async 함수나 백그라운드 Task Queue)를 고려해야 합니다 [76]. 다만 현재는 간단히 구현하기 위해 요청을 받은 후 내부에서 동기적으로 스크립트를 실행하고 완료 시 응답하는 방식을 사용합니다.
- **외부 API 한계**: Papago나 Google Translate API를 사용할 경우 **일일 호출 한도** 및 **요금**에 유의해야 합니다 [23]. 예를 들어 한 번에 15개 기사 내용을 번역하면 API 호출이 15번 발생하여 비용이 누적될 수 있고, 속도 제한으로 지연이 생길 수 있습니다. 필요한 경우 일부 핵심 기사만 번역하거나, **결과 캐싱**을 도입하는 것도 방법입니다 [23]. 또한 API 키는 절대 노출되지 않도록 하고, 코드상에서 환경변수로 관리하세요 (Step 1에서 설정한 대로).
- **UI/UX 고려**: 요약/번역 작업은 여러 기사에 대한 NLP 처리를 수행하므로 **진행 상태 표시**가 중요합니다 [77]. UI에서는 버튼 클릭 시 "**요약 작업 진행중... (0/15)**"와 같이 **프로그레스 표시**를 해주면 운영자가 대기 시간을 예측하기 용이합니다 [77]. 또한 작업 완료 전까지 사용자가 다른 조작을 할 수 있도록, 비동기 호출로 처리하고 UI가 멈추지 않게 해야 합니다 (예: 버튼을 disabled 처리하되, 페이지 탐색 등은 가능하도록).
- **결과 검토 및 활용**: 자동 요약/번역은 편리하지만, **품질 검토가 필요**합니다 [78]. 생성된 요약문의 정확성, 번역문의 뉘앙스 등을 **편집자가 확인 및 수정**하는 프로세스를 고려하세요 [78]. 예를 들어 UI상에서 요약문을 편집할 수 있는 필드를 제공하여, 운영자가 AI 생성 결과를 약간 수정할 수 있게 하면 이상적입니다. 현재 버전에서는 일단 자동 생성에 집중하고, 향후 **human-in-the-loop** 방식의 보완을 검토합니다 [78].
- **인증 및 로깅**: 본 기능 역시 **ADMIN 전용**이므로 인증 미들웨어를 반드시 적용합니다. 토큰이 없으면 401을 반환하도록 테스트하세요. 또한 요약 진행 중 발생하는 예외(외부 API 에러, 타임아웃 등)를 서버 **로그에 남겨** 추후 디버깅할 수 있게 합니다 [75]. 프론트엔드에는 사용자 친화적인 오류 메시지를 전달하세요 (예: "요약 실패: 기사 본문 길이 초과" 등).

---

4. Export 기능 연동 (GET `/api/export/md`, `/api/export/csv`)

목적

최종 발행본 결과물(데일리 뉴스 특별호 콘텐츠)을 쉽게 추출하여 보관하거나 활용할 수 있도록, 이미 백엔드에 구현된 **Export API**를 프론트엔드 UI와 연동합니다. Markdown 포맷과 CSV 포맷 두 가지로 지원하며, 운영자는 버튼 클릭 한 번으로 **최신 발행 데이터를 다운로드**할 수 있습니다 [79]. 이를 통해 발행 결과를 로컬에 저장하거나, 외부 도구(예: Excel)로 분석하고, 아카이빙할 수 있습니다 [80]. (예: CSV를 다운로드하여 Excel로 열어 통계를 내거나, Markdown 파일을 이메일로 공유 등)

준비 파일 또는 조건

- **백엔드 구현**: `GET /api/export/md` 및 `GET /api/export/csv` 엔드포인트 – 이 두 API는 **최종 발행본**의 Markdown과 CSV 데이터를 생성하여 반환하는 기능을 이미 포함하고 있다고 가정합니다 [81]. (이 부분은 이전 개발 단계에서 완료된 상태여야 합니다. 만약 구현되지 않았다면, 현재 승인된 기사 리스트를 불러와 Markdown 텍스트와 CSV 텍스트를 각각 만들어 응답하도록 구현해야 합니다.)
- **프론트엔드**: 관리자 대시보드 UI에 **Export 관련 버튼 또는 메뉴**를 추가해야 합니다 [82]. 예를 들어 "Export Markdown", "Export CSV" 두 개의 버튼을 상단에 배치합니다. 이 버튼 클릭 시 각각 `/api/export/md`, `/api/export/csv` API를 호출하고, 반환된 파일을 사용자 브라우저에서 다운로드 처리하는 기능을 구현합니다.

실행 절차

1. **백엔드 Export API 확인**: `/api/export/md`와 `/api/export/csv`가 정상 동작하는지 테스트합니다. 이 엔드포인트들은 ADMIN_TOKEN 인증이 필요할 가능성이 높으므로, 헤더에 토큰을 포함하여 호출합니다. 예를 들어:

   ```
   curl -H "Authorization: Bearer ${ADMIN_TOKEN}" \
     "http://localhost:8000/api/export/md"
   ```

   로컬 테스트 시 Markdown 텍스트 응답이 오는지 확인하고, CSV도 마찬가지로 호출해봅니다. 이때 응답 헤더에 `Content-Type`과 `Content-Disposition`이 제대로 설정되어 있는지도 확인합니다 (파일 다운로드 처리를 위해) [83]. 만약 구현이 안 되어 있다면, FastAPI의 `FileResponse` 등을 이용하여 해당 포맷의 문자열을 파일로 반환하도록 작성합니다.
2. **프론트엔드 UI 구현**: "Export Markdown" 버튼 (`#btn-export-md`), "Export CSV" 버튼 (`#btn-export-csv`)을 추가하고, 클릭 이벤트 핸들러를 작성합니다. Markdown의 경우 바로 새 탭을 열어 보여주거나 다운로드하게 할 수 있고, CSV는 다운로드가 주 용도이므로 Blob 처리 예시를 살펴봅니다. CSV 다운로드 예시:

   ```
   document.getElementById('btn-export-csv').onclick = function() {
     fetch("/api/export/csv", { headers: { "Authorization": "Bearer " + adminToken } })
      .then(res => res.blob())
      .then(blob => {
        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);
      })
      .catch(err => alert("CSV Export 실패: " + err));
   };
   ```

   위 코드처럼 **Blob으로 응답을 받아** 클라이언트에서 가짜 링크 클릭으로 다운로드를 트리거합니다 [84] [85]. Markdown의 경우 text 형태이므로 Blob 처리 없이 `res.text()`로 받아 새 창에 열거나 파일로 저장할 수 있습니다. (CSV도 간단히 처리하려면 `<a href="/api/export/csv" download>`로 링크를 직접 제공할 수도 있으나, 이 경우 **인증 헤더가 포함되지 않으므로 위험**합니다 [86].)
3. **동작 테스트**: UI에서 Export 버튼들을 눌러보면서, 파일 다운로드 또는 표시가 제대로 이루어지는지 확인합니다. Cloud Run 배포 환경에서는 실제 인터넷 주소로 API를 호출하게 되므로 CORS 설정 등이 문제없는지도 체크하세요. 모든 Export API 호출에도 ADMIN 토큰 인증이 적용되어야 하므로, **로그인/토큰이 없는 상태에서는 다운로드가 진행되지 않아야** 합니다.

CLI/API/JS 예시 코드

- **REST API 호출 (cURL)** – Markdown Export:

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

(성공하면 `latest_report.md` 파일로 다운로드됨) [87]

- **REST API 호출 (cURL)** – CSV Export:

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

(성공하면 `latest_report.csv` 파일로 저장됨) [88]

- **REST API 호출 (Fetch)** – CSV Export (JS 예시):

```
fetch("/api/export/csv", { headers: { "Authorization": "Bearer " + adminToken } })
 .then(res => res.blob())
 .then(blob => {
   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);
 });
```

※ `fetch`로 이처럼 파일 응답(Blob)을 받아 처리해야 **인증 헤더를 포함**할 수 있습니다 [84]. 단순히 `<a href="/api/export/csv" download>` 링크로 제공하면 헤더를 넣을 수 없고, 토큰 인증을 우회하거나 토큰을 URL 파라미터로 노출해야 하므로 피해야 합니다 [86].

예상 결과

- **백엔드 응답**: `/api/export/md` 호출 시 **최신 발행본의 Markdown 문자열**을 응답하며, `/api/export/csv`는 CSV 파일 내용을 응답합니다 [83]. 백엔드 구현에 따라 `Content-Disposition` 헤더가 설정되어 **브라우저가 응답을 파일 다운로드로 인식**할 수도 있습니다 [83]. (예: `Content-Type: text/markdown; charset=utf-8`, `Content-Disposition: attachment; filename="2025-10-12_report.md"` 등)
- **프론트엔드 동작**: UI에서 "Export CSV" 버튼을 클릭하면 즉시 `latest_report.csv` 파일 다운로드가 시작되고, "Export Markdown" 버튼 클릭 시 `latest_report.md` 파일을 받거나, 새 창에 Markdown 텍스트를 표시합니다 [89] [90]. 사용자는 다운로드된 파일을 열어 **내용을 검증**할 수 있습니다.
- **CSV 파일 내용**: CSV 첫 행(header)에 기사 제목, URL, 출처, 요약문, 번역문 등의 필드명이 포함되고, 각 행에 해당 기사들의 정보가 들어갑니다 [91]. 한글이 깨지지 않도록 UTF-8 with BOM 형식으로 생성되었을 것입니다 (엑셀에서 CSV 열 때 한글 인코딩 깨짐 방지를 위해) [91]. 다운로드한 CSV를 메모장이나 Excel로 열어 모든 문자가 올바르게 표시되는지 확인하세요 [92].
- **Markdown 파일 내용**: Markdown 파일에는 발행된 기사 카드들이 순서대로 나열되어 있고, 제목/본문/출처 등이 마크다운 문법으로 서식화되어 있을 것입니다 [91]. 예를 들어 각 기사 제목이 `## 키워드: 기사제목` 형태로, 요약문은 일반 텍스트나 인용 블록으로, 기사 링크는 `[원문 보기](URL)` 형태로 포함되는 등, 마크다운을 마크다운 뷰어나 에디터로 열었을 때 가독성 있게 보이도록 구성합니다. 사용자는 이 Markdown을 통해 당일 발행본을 한눈에 확인할 수 있습니다.

주의사항

- **UI 연동 여부 확인**: 현재 Export API는 백엔드에 구현되어 있었으나 **프론트엔드에 버튼이 없어 사용 불가한 상태**였을 수 있습니다 [93] . 반드시 UI에 해당 기능 버튼을 노출하여 운영자가 찾기 쉽도록 합니다. 버튼 위치는 대시보드 상단 메뉴바나 발행 관리 화면 등 적절한 곳에 배치하세요 [94] .
- **파일 다운로드 처리**: 프론트엔드에서 파일 다운로드를 구현할 때 주의할 점은, `<a href="...">` 링크만으로 접근하면 **인증 토큰이 헤더에 포함되지 않는다**는 점입니다 [95] . 그러므로 반드시 `fetch` 등을 사용하여 Authorization 헤더와 함께 요청을 보내고, Blob 처리 후 `a.click()`으로 다운로드를 트리거하는 방식이 필요합니다 [96] . (만약 백엔드가 토큰을 URL 쿼리파라미터로 받도록 허용한다면 `<a href="/api/export/csv?token=...">`로 할 수도 있으나, 이는 토큰 노출 및 보안상 권장되지 않습니다. **헤더 인증 방식을 유지**하세요.)
- **인코딩 검증**: CSV의 경우 **한글 깨짐 문제가 있었는지 최종 확인**해야 합니다 [92] . 현재 구현은 UTF-8 BOM을 추가하여 Excel 등에서 깨지지 않을 것으로 예상되지만, 다운로드한 CSV를 실제로 열어 한글이 정상 표시되는지 테스트하세요. 만약 문제가 있다면 백엔드에서 CSV 출력 시 맨 앞에 `\ufeff`(BOM 문자)를 추가하도록 수정합니다. Markdown은 일반적으로 UTF-8로 문제 없으나, 내부에 이미지 경로나 특수 문자가 있다면 잘 표시되는지 확인합니다 [92] .
- **Export 데이터 활용 및 백업**: Export 받아둔 파일들은 **운영상의 백업**으로 활용할 것을 권장합니다 [97] . Cloud Run 환경은 **스테이트리스**라 컨테이너 재시작 시 로컬 파일(archive 폴더)이 유실될 수 있으므로, 중요한 발행본은 Export를 통해 정기적으로 외부에 저장해두는 것이 좋습니다 [98] . 예를 들어 매일 발행 후 Markdown과 CSV를 내려받아 구글 드라이브나 사내 파일서버에 업로드해두면 추후 데이터를 잃지 않고 관리할 수 있습니다.

---

5. 보안 및 인증 적용 (ADMIN_TOKEN 등)

목적

관리자 시스템에 대한 **보안 설정을 강화**하고 최종 배포 전에 민감 정보 노출이 없도록 점검합니다. 이미 ADMIN 토큰 기반 인증 체계를 도입하여 **비인가 접근을 차단**하고 있으므로, 여기서는 토큰/키 관리와 기타 보안 옵션들을 최종 검토합니다 [99] . 또한 Cloud Run 배포 환경에서 환경변수, 네트워크, IAM 권한 등이 올바르게 구성되었는지 확인하여, 운영 시에 잠재적인 보안 허점을 제거하는 것이 목표입니다 [99] .

준비 파일 또는 조건

- **환경 설정**: `.env` 파일 및 Cloud Run에 설정된 환경변수들을 확인합니다 [100] . 특히 `ADMIN_TOKEN`과 `API_KEY` 등이 예상대로 설정되었는지 재점검합니다.
- **GCP 설정**: 필요에 따라 GCP의 **IAM 및 Cloud Run 서비스 설정**을 검토합니다 [100] . Cloud Run이 클라우드 IAM 인증을 요구하도록 설정되었는지, VPC 연결을 사용 중이면 올바르게 구성되었는지 등을 확인합니다.
- (코드 변경보다는 설정 확인에 가깝기 때문에, 이 단계에서 수정할 코드 파일은 특별히 없습니다. 대신 인프라/설정에 집중합니다.)

실행 절차

1. **인증 미들웨어 동작 확인**: 토큰 없이 API를 호출할 경우 401 Unauthorized가 반환되는지 점검합니다. 예시:

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

이처럼 응답 헤더 상태 코드가 401임을 확인합니다 [101]. (`/api/status`는 간단히 인증 여부를 테스트하기 좋은 엔드포인트입니다.)

2. **정상 토큰 인증 확인**: 올바른 `ADMIN_TOKEN`을 넣어 호출하면 200 OK와 데이터가 반환되는지 확인합니다. 예:

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

응답 본문으로 서비스 상태 JSON이 정상 출력되는지 확인합니다 [102]. 이를 통해 **운영 환경에서도 토큰 인증이 제대로 적용**됨을 재확인합니다.

3. **Cloud Run 설정 검토**: Cloud Run 서비스에 설정된 환경변수를 조회하여 올바르게 들어가 있는지 확인합니다. CLI 예시:

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

이 명령은 현재 Cloud Run에 설정된 환경변수 목록을 보여줍니다. 출력에 `ADMIN_TOKEN`, `API_KEY`, `DB_URL` 등이 기대한 값(또는 마스킹된 상태)으로 존재하는지 확인합니다 [103]. 값이 없거나 잘못되어 있다면 `gcloud run services update ... --update-env-vars`로 재설정합니다 (Step 1 참고).

4. **Cloud Run IAM 및 트래픽 설정**: Cloud Run 서비스가 **--allow-unauthenticated**로 공개 설정되어 있는지, 아니면 IAM 인증을 요구하는지 확인합니다. 현 단계에서는 ADMIN 토큰 인증이 자체적으로 있으므로 공개 접근이어도 문제 없지만, 더욱 안전하게 하려면 Cloud Run 호출 자체를 특정 IAM 사용자만 가능하도록 설정할 수도 있습니다 [104] [105]. (이 경우 프론트엔드에서 Cloud Run Invoke 권한이 있는 서비스 계정 등을 통해 프록시 호출하거나 해야 하므로, 현재는 단순 토큰 인증으로 유지합니다.) 또한 **만약 DB를 사용 중이라면**, Cloud Run 서비스에 VPC Connector가 붙어있는지, DB와의 통신이 firewall에 허용되는지 등 네트워크 설정을 재점검합니다 [25].

CLI/API/JS 예시 코드

- **토큰 미제공 호출 테스트**:

```
curl -i "https://<배포도메인>/api/status"
```

기대 결과: `HTTP/1.1 401 Unauthorized` (헤더와 본문에 인증 실패 내용) [101].

- **토큰 제공 호출 테스트**:

```
curl -i -H "Authorization: Bearer ${ADMIN_TOKEN}" "https://<배포도메인>/api/status"
```

기대 결과: `HTTP/1.1 200 OK`와 함께 JSON 데이터 응답 [102].

- **환경변수 설정 확인 (gcloud)**:

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

(환경변수 배열 출력에서 각 ENV 이름과 값 확인) [103].

예상 결과

- **ADMIN_TOKEN 인증 작동**: 모든 관리자용 API가 `Authorization` 헤더에 올바른 토큰 없이는 접근 불가임이 최종 확인됩니다 [17] . (이 테스트는 개발 단계에서도 수행했지만, **운영 환경(Cloud Run)**에서도 동일하게 적용됨을 재확인합니다.)
- **환경변수 안전 관리**: Cloud Run의 설정이나 CI 설정 등에서 **민감 정보가 안전하게 저장**되고 있음을 확인합니다 [106] . 예를 들어 Cloud Run 로그에 환경변수가 평문으로 찍히지 않고, `.env` 파일이 저장소에 커밋되어있지 않아 공개되지 않았으며, CI/CD 설정에서 secrets로 관리되고 있는지 등입니다.
- **클라이언트 측 토큰 노출 주의**: 관리자 프론트엔드 앱이 단일 페이지로 동작하면서, `ADMIN_TOKEN`이 브라우저 **Network 패널이나 JS 소스에 노출**될 가능성이 있습니다 [20] . 이는 근본적으로 현재 구조(토큰 기반 인증)에서는 피하기 어렵지만, **소스 맵 제거** 등으로 코드에서 토큰 문자열을 쉽게 찾지 못하게 하는 조치를 취할 수 있습니다 [20] . 또한 가능하면 내부 운영망이나 VPN을 통해서만 접속하게 하거나, 추후 OAuth 로그인 도입을 고려하는 등 보안 강화를 검토합니다.
- **네트워크 및 IAM 설정**: Cloud Run 서비스가 **불필요하게 public**하지 않은지 검토합니다. 현재는 ADMIN_TOKEN으로 보호하고 있으므로 Cloud Run을 Public으로 두어도 외부 사용자는 기능을 못 쓰지만, 더 안전하게는 Cloud Run을 **인증된 호출만 허용**으로 바꾸고 Cloud Run Invoker 권한을 프론트엔드 앱에 부여하는 방법도 있습니다 [104] . (이 구현은 추가 개발이 필요하므로 현 단계에서는 권고 사항으로 둠)
  DB를 쓴다면 Cloud Run이 VPC에 연결되어 있고, DB 보안그룹/방화벽이 Cloud Run만 허용하는지 확인하여 **DB가 외부에 노출되지 않도록** 해야 합니다 [25] .
- **라이브러리 취약점 점검**: (선택사항) 프로젝트의 Python 패키지 등의 **의존성**을 최신 버전으로 유지하고, `pip audit`이나 `safety` 같은 도구로 **보안 취약점**을 점검했는지도 확인합니다 [107] . 알려진 보안 이슈가 있는 패키지는 업그레이드하거나 패치 버전을 적용하여 위험을 줄입니다.

주의사항

- **토큰 교체 정책**: 운영 정책상 필요하면 **정기적으로 ADMIN 토큰을 변경**하는 것을 고려하세요 [19] . 토큰 교체 시에는 서버의 환경변수와 프론트엔드의 토큰 값을 동시에 업데이트하고 배포해야 하며, 이전 토큰은 즉시 폐기(더 이상 허용 안 함)해야 합니다. 새로운 토큰은 안전한 채널을 통해 운영 담당자에게 공유하고, 변경 내역을 문서화해 두세요.
- **로그 민감정보 필터링**: 서버 로그에 **민감 정보(토큰, API 키)**가 기록되지 않도록 설정합니다 [108] . 예를 들어 FastAPI의 기본 Access Log 미들웨어가 Authorization 헤더 전체를 출력하지는 않지만, 혹시 디버그 로그에 환경변수 값이 찍히는 부분이 없는지 확인합니다. 운영 모드에서는 DEBUG 로그를 끄고 INFO 수준으로 동작시켜 과도한 정보 노출을 피하세요 [108] . 또한 Cloud Run 로그와 GCP Monitoring을 주기적으로 확인하여, **보안 이벤트**(예: 반복된 401 Unauthorized 시도 등)이 있으면 원인을 파악하고 대응합니다 [21] .
- **HTTPS 강제 사용**: Cloud Run에서 기본 제공되는 도메인은 HTTPS가 자동 적용됩니다만, 혹시 커스텀 도메인을 연결했다면 **SSL 인증서**가 제대로 설치되어 있는지 확인합니다 [109] . 모든 API 통신은 HTTPS를 통해 이뤄져야 하며, http로의 접근은 없도록 설정합니다 (필요시 http -> https 리디렉션 설정).
- **추가 보안 기능**: 현재는 단일 ADMIN 토큰으로 인증을 간소화했지만, **다중 관리자 계정**이나 **OAuth 기반 로그인**, **세션 관리** 등은 추후 고려할 수 있는 향상 기능입니다 [110] . 현 시점에서는 토큰 한 개로 관리하므로, **토큰 유출 시 위험**이 크다는 것을 항상 인지해야 합니다. 토큰은 최소한의 인원에게만 공유하고, 필요시 **IP 제한**이나 **VPN 접속** 등 네트워크 레벨 보안과 병행해서 운용하십시오 [111] .

---

6. UI 버튼 연동 (보고서 생성, 요약/번역, 내보내기 등)

목적

프론트엔드 **관리자 UI에 새로운 기능 버튼들을 연동**하여, 앞서 구현한 API 기능들을 운영자가 웹 인터페이스에서 직접 사용할 수 있도록 합니다. 여기서는 **"일일 보고서 생성"**, **"뉴스 요약/번역"**, **"Export (내보내기)"** 버튼들을 UI에 추

가/활성화하고, 각각 해당 API를 호출하는 동작을 구현합니다 [29] [112]. 또한 사용자 경험을 위해 버튼 클릭 시 **로딩 표시**, **완료/실패 알림**, **오류 처리 공통화** 등을 다룹니다.

준비 파일 또는 조건

- **관리자 대시보드 HTML/JS**: 관리자 웹 UI (`index.html` 혹은 `admin.html`) 파일을 열어서, 적절한 위치에 새로운 버튼 요소를 추가합니다. 또한 해당 파일이나 연결된 JS 스크립트에서 버튼 클릭 이벤트에 대응하는 함수를 구현할 준비가 되어야 합니다. (이미 API 호출을 위한 `adminToken` 관리나 fetch wrapper 등이 있다면 활용)
- **스타일/CSS (옵션)**: 버튼들의 스타일이 다른 UI와 어울리도록 CSS를 조정합니다. 또한 로딩 시 표시할 스피너나 메세지 영역이 필요하면 HTML/CSS로 미리 작성해둡니다.

실행 절차

1. **"일일 보고서 생성" 버튼 연동**:
2. **UI 추가**: 예를 들어 대시보드 상단에 다음과 같이 버튼을 추가합니다:
   `<button id="btn-generate-report" class="btn">일일 보고서 생성</button>`
   (class와 위치는 기존 UI 구조에 맞게 설정)
3. **JS 연동**: JS 코드에서 해당 버튼의 `onclick` 핸들러를 정의하여, **POST** `/api/report`를 호출하도록 합니다. 구체적인 코드는 앞서 **섹션 2**에서 작성한 fetch 예시를 참조하면 됩니다 [38].
   추가로, **로딩중 상태 표시**를 위해 버튼을 클릭하면 버튼을 비활성화하고 "생성 중..." 같은 문구를 나타낼 수도 있습니다. 응답이 오면 버튼을 다시 활성화하고 결과를 alert 또는 페이지에 표시합니다 [48].
4. **동작 확인**: UI에서 해당 버튼을 눌러보고, 콘솔 로그나 alert으로 성공/실패 메세지가 출력되는지 확인합니다. 정상 작동하면, 운영자는 이 버튼을 통해 매일 아침 손쉽게 Daily Report를 생성할 수 있게 됩니다.
5. **"뉴스 카드 요약/번역" 버튼 연동**:
6. **UI 추가**: 예를 들어 기사 목록 화면이나 키워드 상세 화면에 다음 버튼들을 추가합니다:
   `<button id="btn-enrich-all" class="btn">키워드 전체 요약</button>`
   `<button id="btn-enrich-selected" class="btn">선정본 요약</button>`
   (버튼 레이블은 이해하기 쉽게 붙입니다. 필요시 tooltip 등도 활용)
7. **JS 연동**: 각 버튼에 대해 **POST** `/api/enrich/keyword` 및 **POST** `/api/enrich/selection` 호출 코드를 작성합니다. 이는 **섹션 3**에서 설명한 fetch 예시와 동일합니다 [57] [113].
   또한 요약/번역 작업은 시간이 오래 걸릴 수 있으므로, 버튼 클릭 시 **진행률 표시**를 구현하는 것이 중요합니다. 예를 들어 전체 15개 기사 중 몇 개 완료되었는지 백엔드에서 알 수 있다면 주기적으로 업데이트하거나, 아니면 단순히 "요약 진행중... 잠시만 기다리세요" 메시지를 띄웁니다 [77].
8. **동작 확인**: UI에서 버튼을 누르면 콘솔에 결과 로그가 찍히거나, 완료 시 사용자에게 "요약 완료" 알림이 뜨는지 확인합니다. 그리고 실제로 요약문/번역문 데이터가 생성되어 Export 등에 반영되는지 점검합니다. 필요하면 UI에서 기사 리스트에 요약 결과를 표시하는 기능을 추가로 개발합니다.
9. **"Export (내보내기)" 버튼 연동**:
10. **UI 추가**: 대시보드나 발행 관리 화면에 다음 버튼들을 추가합니다:
    `<button id="btn-export-md" class="btn">Export Markdown</button>`
    `<button id="btn-export-csv" class="btn">Export CSV</button>`
11. **JS 연동**: 각 버튼에 대해 **GET** `/api/export/md`, **GET** `/api/export/csv` 호출을 처리하는 코드를 작성합니다. **섹션 4**의 fetch/Blob 예시를 참고하여, CSV 다운로드는 Blob으로 처리하고 Markdown은 `window.open`으로 새 탭을 열거나 Blob으로 처리합니다 [84] [83].
    이때 인증 헤더를 꼭 포함해야 하므로 `fetch(url, { headers: {Authorization: ...} })` 형태여야 합니다. UI/UX적으로 버튼을 누르면 즉시 파일 다운로드가 시작되므로, 별도의 로딩표시는 없어도 됩니다만, 네트워크가 느릴 경우 위해 간단한 "준비중..." 토스트를 띄웠다가 다운로드 시작 시 없앨 수도 있습니다.
12. **동작 확인**: UI에서 Export 버튼을 눌러 Markdown, CSV 파일이 제대로 브라우저를 통해 다운로드되는지 확인합니다. **파일명을 지정**했으면 해당 이름으로 저장되는지도 확인하고, 파일 내용을 열어 본문이 올바른지 (인코딩, 줄바꿈, 데이터 유실 없는지) 확인합니다 [114] [92].

CLI/API/JS 예시 코드

(본 섹션의 CLI/JS 코드는 이미 각 기능별 섹션에 제시되었으므로 생략합니다. 앞의 예시들을 참고하여 그대로 적용하면 됩니다.)

예상 결과

- 관리자 UI에 **신규 기능 버튼들이 모두 표시**됩니다 (보고서 생성, 전체 요약, 선정본 요약, Export MD, Export CSV). 이를 통해 사용자는 GUI 환경에서 모든 기능을 사용할 수 있습니다.
- 각 버튼 클릭 시 적절한 **API 호출이 이루어지고, 응답에 따른 피드백**이 제공됩니다. 예를 들어:
- "일일 보고서 생성" 버튼: 클릭하면 2~3초 후 "보고서 생성 완료" 알림이 뜨고, 실제 `archive/` 폴더에 Markdown 파일이 생성됨 115 40 .
- "전체 요약" 버튼: 클릭하면 10~20초 동안 진행 후 "요약 완료" 메시지 또는 요약 결과 표시가 나타남. (이 동안 UI가 freeze되지 않고 진행중 안내를 보여줘야 함) 77 .
- "Export CSV" 버튼: 클릭 즉시 `latest_report.csv` 파일 다운로드가 시작되고, 파일을 열면 최신 발행 기사들의 목록과 요약/번역 컬럼이 포함되어 있음 90 .
- **공통 UX 향상**: 모든 신규 기능 버튼에 대해 **오류 상황 처리**도 구현됩니다. 예를 들어 토큰 만료나 서버 에러 시 alert("실패: ...")를 보여주고, 콘솔에 상세 에러를 로그로 남기는 등 일관성 있는 에러 처리 패턴을 적용합니다. 또한 필요한 경우 각 버튼에 공통 스타일 (예: 아이콘이나 색상) 부여, disabled 상태 관리 등 UI 품질을 높입니다.
- **문서화**: UI 사용법이 README나 운영 매뉴얼에 반영되어, 새로 온 팀원이 UI에서 어떤 버튼이 무슨 기능을 하는지 쉽게 이해할 수 있게 됩니다 116 .

주의사항

- **버튼 노출 조건**: 만약 특정 기능이 아직 실험 단계이거나 운영상 선택적으로 쓰이길 원한다면, 해당 버튼을 **조건부로 노출**하는 것도 고려합니다 (예: `.env`에 FEATURE_EXPORT=true 설정 시에만 버튼 표시 등). 그러나 본 가이드에서는 기본적으로 모든 기능을 사용할 수 있게 하는 것으로 다룹니다.
- **로딩 및 상태 관리**: 한 버튼의 기능 실행 중에 다른 버튼을 눌러도 문제가 없도록 설계해야 합니다. 예를 들어 "전체 요약"이 진행 중일 때 "Export"를 눌러도 시스템에 문제는 없지만, 요약이 완료되지 않은 중간 데이터로 Export할 경우 일부 필드가 비어있을 수 있습니다. 따라서 **운영 절차상 요약 완료 후 Export**하도록 가이드하거나, 기술적으로 요약이 진행중이면 Export 버튼을 일시적으로 비활성화하는 등의 조치를 고려합니다.
- **공통 코드 구조**: 프론트엔드 JS에서 **중복 코드**(토큰 가져오기, fetch 에러 처리 등)는 함수로 빼서 관리하면 향후 유지보수에 좋습니다. 예를 들어 `apiPost(url)` 헬퍼 함수를 만들고 내부에서 `fetch(url, { method:'POST', headers:{Authorization:Bearer...}})`를 공통 처리하도록 하면, 코드량이 줄고 일관성도 높아집니다.
- **로그인 페이지 (추가 고려)**: 현재 ADMIN_TOKEN은 별도 로그인 UI 없이 상정되어 있는데, 추후 **로그인 페이지**를 만들어 initial token 입력을 받아 `localStorage`에 저장하게 하고, 이후부턴 토큰을 자동으로 붙여 요청하게 만드는 방식으로 개선할 수 있습니다. 이는 운영자가 일일이 토큰을 콘솔에 넣지 않게 해주므로 편의와 보안을 조금 높일 수 있습니다.

---

7. 통합 테스트 및 문서화 작업 (Pytest, README, UI 활용법 포함)

목적

새로 추가된 기능들을 포함하여 시스템 전반에 대한 **통합 테스트를 자동화**하고, 모든 관련 **문서**를 최신 상태로 업데이트합니다. 이를 통해 기능 동작을 사전에 검증하고 배포 전후 안정성을 확보하며, 인수인계나 운영 가이드에 누락이 없도록 합니다 117 . Pytest 등의 프레임워크로 자동화 테스트 코드를 작성하여 주요 시나리오를 검증하고, README 및 관리자 매뉴얼에 사용법과 설정법을 상세히 기록합니다 117 .

준비 파일 또는 조건

- **테스트 코드 디렉토리**: `tests/` 폴더를 준비하고, 각 기능별로 테스트 스크립트를 작성합니다 (ex: `tests/test_report.py`, `tests/test_enrich.py`, `tests/test_export.py` 등) [118]. Pytest를 사용하므로 파일명은 `test_*.py` 형태로, 각 함수에 `test_` 접두어를 붙입니다.
- **문서 파일**: 프로젝트의 `README.md`, 운영 가이드북(본 문서), API 명세 문서 등이 모두 업데이트 대상입니다 [119]. 새로운 환경변수, API Endpoint, 사용 방법 등을 반영해야 합니다.
- **CI 설정 파일**: GitHub Actions 등의 CI를 사용 중이라면 워크플로우 YAML 파일 (예: `.github/workflows/ci.yml`)을 열어 테스트 실행 단계와 배포 단계를 확인합니다 [120] [121]. 필요하면 pytest 실행이나 환경변수 세팅 등을 추가합니다 (CI 환경에서 ADMIN_TOKEN 등이 필요할 수 있으므로).

실행 절차

1. **테스트 케이스 작성**: 각 신규 기능 및 기존 주요 기능에 대한 테스트 함수를 작성합니다. 예를 들어:
2. `test_report.py`:

```
import pytest
from fastapi.testclient import TestClient
from app.main import app

client = TestClient(app)

def test_generate_report_without_token():
    response = client.post("/api/report")
    assert response.status_code == 401  # 인증 안 되면 401
def test_generate_report_success(tmp_path):
    # 토큰을 헤더에 넣어 성공 시도
    token = "TEST_ADMIN_TOKEN"
    app.dependency_overrides[verify_admin_token] = lambda: token  # 토큰 검증 우회
    response = client.post("/api/report", headers={"Authorization": f"Bearer {token}"})
    assert response.status_code == 200
    data = response.json()
    assert "content" in data or "report_file" in data
    # 추가로 content 문자열에 오늘 날짜 등이 포함되어 있는지 검증할 수도 있음.
```

   위는 간단히 설명용으로 작성한 예시이며, 실제로는 `verify_admin_token`을 override하거나 환경변수를 세팅해서 ADMIN_TOKEN을 테스트에 주입합니다.
3. `test_enrich.py`: `/api/enrich/keyword`와 `/api/enrich/selection`에 대해 토큰 없을 때 401, 있을 때 200을 확인하고, 결과 JSON에 예상 필드가 포함됐는지 검사합니다. 외부 API 호출은 모킹하여 항상 일정한 문자열을 반환하도록 처리합니다.
4. `test_export.py`: `/api/export/md`와 `/api/export/csv` 호출 결과를 받아, 상태 코드 200인지 확인하고, 응답 본문에 **예상되는 내용**이 포함됐는지 검증합니다. 예를 들어 CSV 응답 텍스트에 `\ufeff` (BOM) 문자가 있는지, 또는 Markdown 응답에 특정 키워드가 들어있는지 등을 테스트합니다 [122].
5. **모킹 및 테스트 격리**: 테스트는 실제 환경과 격리되어야 합니다. 예를 들어 번역 API나 DB에 영향을 주지 않도록, **Monkeypatch나 Mock 객체**를 사용합니다 [123] [124]. Pytest에서는 `monkeypatch` fixture로:

```
monkeypatch.setattr(external_module, "translate_text", lambda text: "번역된:"+text)
```

   이런 식으로 외부 API 호출 함수를 가짜로 대체할 수 있습니다. ADMIN_TOKEN도 테스트 시작 전에:

```
import os
os.environ['ADMIN_TOKEN'] = "test-token"
```

처럼 설정하여 실제 토큰 값을 쓰지 않고 테스트용 값으로 대체합니다 [124]. 이렇게 하면 보안상 민감한 실제 토큰을 노출하지 않고도 인증 로직을 검증할 수 있습니다.

6. **테스트 실행 (로컬)**:

```
pytest -v
```

명령으로 모든 테스트를 실행해봅니다 [125]. `-v` 옵션을 주면 세부 결과를 표시해주므로, 어떤 테스트가 실패했는지 쉽게 확인 가능합니다. 특정 테스트만 실행하려면:

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

과 같이 파일명::함수명으로 지정할 수 있습니다 [126].

7. **CI 파이프라인 통합**: GitHub Actions (또는 다른 CI) 설정에서 푸시 시 자동으로 `pytest`를 돌리는 단계가 있는지 확인합니다. 만약 없다면 워크플로우 YAML에 다음을 추가합니다:

```
- name: Run tests
  run: pytest -q
```

그리고 배포 단계 전에 tests가 실패하면 배포를 중단하도록 구성합니다. 이미 Action에서 Cloud Run 배포를 하고 있다면, 테스트 성공 후 `google-github-actions/deploy-cloudrun` 등으로 배포하도록 설정되어 있을 것입니다 [121]. 이때 **환경변수 시크릿**도 전달해야 하므로, workflow에서:

```
env_vars: ADMIN_TOKEN=${{ secrets.ADMIN_TOKEN }},API_KEY=${{ secrets.API_KEY }}
```

같은 식으로 Cloud Run에 시크릿을 주입하여 배포합니다 [127].

8. **문서화 업데이트**:
9. **README.md**: 개발 환경 세팅, `.env` 예시, Cloud Run 배포 방법, ADMIN_TOKEN/API_KEY 설정 방법 등을 추가합니다 [128]. 또한 새로운 API endpoint 목록과 간단한 사용법도 나열합니다.
10. **운영 가이드(본 가이드북)**: 앞선 내용처럼 모든 구현 및 사용 방법을 담고, 최종 체크리스트를 업데이트합니다 [129]. 특히 **API 명세, UI 사용법, 환경변수 설명** 등이 최신 상태인지 확인합니다.
11. **기타 문서**: API 명세서가 별도로 있다면 `/api/report`, `/api/enrich/*`, `/api/export/*` 엔드포인트를 추가하고 ADMIN 인증 필요 여부를 명기합니다. UI 매뉴얼이 있다면 스크린샷과 함께 새로운 버튼 사용법을 설명합니다.

CLI/API/JS 예시 코드

- **Pytest 모든 테스트 실행**:

```
pytest -v
```

- **Pytest 특정 테스트 실행**:

```
pytest tests/test_export.py::test_csv_has_bom
```

- **GitHub Actions workflow (발췌)**:

```
jobs:
  build_deploy:
    steps:
      - 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 }}
```

위 예시는 CI에서 테스트 후 Cloud Run에 자동 배포하는 단계의 설정 일부입니다 [121].

예상 결과

- **테스트 통과**: `pytest`를 실행한 결과 **모든 테스트 케이스가 성공**해야 합니다 [130]. 예를 들어 `/api/report` 관련 테스트에서 토큰 없이 호출하면 401이 나오는지, 올바른 토큰으로 호출 시 200과 보고서 내용이 생성되는지 확인하고, Export 테스트에서 CSV 응답에 BOM(`\ufeff`)이 포함되어 있는지, Markdown/CSV에 예상 필드와 텍스트가 있는지 검증합니다 [122] [131]. 신규 기능(보고서 생성, 요약, 내보내기)뿐만 아니라 기존 기능(기사 수집, 키워드 설정 등)에 대한 회귀 테스트도 포함되어야 합니다 [132]. 이를 통해 최근 코드 변경이 다른 부분에 영향을 주지 않았음을 보장합니다.
- **CI 자동화**: 코드를 Git에 push하면 CI 파이프라인이 트리거되어 **자동으로 테스트를 실행**하고, 모두 통과하면 Cloud Run에 배포까지 진행됨을 확인합니다 [133]. 테스트 실패 시 배포가 중단되어 잘못된 코드가 프로덕션에 올라가지 않게 하고, 테스트 성공 시에는 새로운 이미지가 배포되며, Cloud Run **Revision History**에 새로운 리비전이 생성됩니다. 이를 몇 번의 커밋으로 시도해 로그를 확인합니다.
- **문서 최신화**: `README.md`에는 이제 **환경변수 설정법 (.env 예시, ADMIN_TOKEN 생성법)**, **로컬 실행 및 배포 방법**, **주요 API 사용 예** 등이 모두 최신 상태로 반영됩니다 [134]. 관리자 운영 가이드(본 문서)에도 새 기능(보고서 생성, 요약/번역, Export)에 대한 사용 방법이 모두 포함되고, **체크리스트 항목들도 업데이트**됩니다 [129]. 예를 들어 "매일 발행 전 요약 기능 사용 여부"나 "Export 백업 실시 여부" 등이 체크리스트에 추가됩니다. 이러한 문서화는 추후 새로운 팀원이 프로젝트를 이어받을 때 중요한 참고자료가 되어, 안정적인 **인수인계**가 가능합니다 [135].

주의사항

- **테스트 격리 및 안정성**: 테스트 코드는 운영 DB나 외부 API를 건드리지 않도록 **격리**해야 합니다 [123]. 외부 API 호출은 Mock으로 대체하고, 필요 시 테스트용 임시 파일/DB를 사용하는 등의 조치를 취합니다 [123] [136]. 예를 들어 요약 API 호출 부분을 함수로 분리한 뒤 테스트에서 그 함수를 monkeypatch하여 실제 API를 부르지 않게 하거나, DB 연동을 쓰는 부분은 sqlite 메모리 DB 등으로 교체합니다.
- **시드 데이터 준비**: 통합 테스트 시에는 **예상 입력 데이터**를 미리 준비해야 합니다 [137]. 예를 들어 보고서 생성 테스트 전에 `selected_articles.json`에 몇 개의 임시 기사 데이터를 넣어두거나, 해당 함수(`make_daily_report`)의 내부에서 현재 날짜 기사 목록이 비어있을 경우를 가정하여 dummy 데이터를 넣어 테스트할 수 있습니다. 또는 해당 함수를 monkeypatch하여 항상 일정한 Markdown을 반환하도록 해 결과만 검증할 수도 있습니다 [136].

- **CI 시크릿 관리**: CI 환경에서 테스트를 수행할 때 필요한 환경변수 (ADMIN_TOKEN 등)를 CI 툴에 **시크릿**으로 등록해두고 사용하는지 확인합니다 [138]. 테스트용 값이어야 하며, 실제 운영 토큰을 쓰지 않도록 주의합니다. 또한 CI에서 여러 테스트를 병렬 실행할 경우를 고려해, **테스트 간 공유 자원**(파일, DB 등)이 충돌하지 않게 합니다 [138]. (예: 동일한 파일을 동시에 쓰지 않도록 tmp 디렉토리 활용 등)
- **문서 버전 관리**: 문서를 수정할 때는 변경 내역을 꼼꼼히 기록하고, 버전 관리를 통해 팀원들과 공유하세요 [135]. 특히 운영 가이드나 인수인계 문서에는 새로운 REST API 목록, 사용 방법, 환경 설정법 등이 **빠짐없이 기재**되어야 합니다 [139]. 문서화는 추후 유지보수나 신규 인력 투입 시 매우 중요하므로, 시간을 들여 상세하고 정확하게 작성합니다.

---

8. 배포/운영 시 체크리스트 (Cloud Run, CI/CD, .env 구성 등)

목적

최종 코드를 Cloud Run 등에 **배포**하고 실제 **운영 단계에서 점검해야 할 사항**들을 정리합니다. 이 단계에서는 변경된 모든 기능이 포함된 버전을 도커 이미지로 빌드하여 Cloud Run 서비스에 배포하고, CI/CD 파이프라인이 정상 동작하는지 확인합니다 [140]. 또한 운영 중에 주기적으로 확인해야 할 주요 체크포인트를 제공하여, 서비스 안정성을 유지하고 문제 발생 시 신속히 대응할 수 있도록 합니다 [133] [141].

준비 파일 또는 조건

- **Dockerfile**: 최신 코드와 종속성을 반영하도록 Dockerfile이 업데이트되어 있어야 합니다 [142]. (예: 새로운 Python 패키지를 설치했으면 Dockerfile에도 추가)
- **Cloud Run 서비스**: GCP 프로젝트에 Cloud Run 서비스(`qualijournal-admin` 등)가 생성되어 있어야 하며, 권한 및 네트워크 구성이 완료된 상태여야 합니다.
- **CI/CD 설정**: GitHub Actions 또는 Cloud Build 등의 CI/CD 파이프라인이 준비되어 있어야 하며, 실행 권한 (GCP 서비스 계정 키 또는 GitHub OIDC 설정 등)이 설정되어 있어야 합니다.

실행 절차

1. **Docker 이미지 빌드 (수동)**: 우선 로컬에서 Docker 이미지를 수동으로 빌드/푸시해봅니다 (CI 없이도 잘 동작하는지 확인 목적). 예:

```
# 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
# 3. Cloud Run 서비스에 이미지 배포 (gcloud CLI 사용)
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 리전 등은 실제 환경에 맞게 대체하십시오 [143] [144]. `--update-env-vars`를 통해 환경변수도 배포 시점에 넣어주었습니다. 배포가 성공하면 Cloud Run 서비스 URL과 리비전 정보가 출력됩니다.

2. **CI/CD 파이프라인 검증**: GitHub Actions 등에서 push 시 자동으로 Docker build & deploy가 되는지 체크합니다. 이미 앞서 Step 7에서 CI 세팅을 점검했으므로, 실제로 main 브랜치에 코드를 push하거나 재실행하여:
3. 테스트 -> 빌드 -> 배포 단계가 순차적으로 진행되는지,
4. 테스트 실패 시 배포를 건너뛰는지,
5. 배포 성공 시 Cloud Run 서비스에 새로운 revision이 뜨는지 등을 확인합니다 [133]. CI 로그를 보면서 any error가 없는지 확인하고, secrets가 제대로 적용되었는지도 점검합니다 (예: Actions 로그에 환경변수 노출 없는지).
6. **배포 후 기능 점검**: 새로운 버전이 배포된 후, Cloud Run 서비스 URL (또는 커스텀 도메인)로 접속하여 **관리자 UI 기능들을 실사용 시나리오로 테스트**합니다 [145] [146]:
7. 관리자 페이지 로드 확인
8. "일일 보고서 생성" 클릭 -> 성공 메시지 및 로그 확인
9. "전체 요약", "선정본 요약" 클릭 -> 요약 결과 반영 확인
10. "Export CSV/MD" 클릭 -> 파일 다운로드 및 내용 확인
    등을 실제 운영자 입장에서 실행해 봅니다 [147] [146]. 또한 Cloud Run의 **Logs** 탭에서 해당 요청들이 200 OK로 처리됐는지, 오류는 없는지 확인합니다 [148].
11. **모니터링 설정**: GCP Cloud Run의 Metrics나 Error Reporting 등을 활용해, CPU/메모리 사용량, 요청 실패율 등의 지표를 모니터링합니다. 특히 새 기능 추가로 자원 사용량이 늘었을 수 있으므로, 필요시 Cloud Run **메모리/CPU 할당을 증가**시키는 것도 고려합니다 [148]. 또한 Error Reporting에 새로운 예외가 기록되지는 않는지 관찰합니다.
12. **최종 운영 체크리스트 작성**: 아래에 제시된 체크리스트를 참고하여, 일일 운영 및 배포 후 점검해야 할 항목들을 한 번에 정리합니다. 이 체크리스트는 운영 매뉴얼의 일부로 제공하여, 운영자가 **매일 발행 작업 전후로 확인**할 수 있도록 합니다 [149] [150].

CLI/API/JS 예시 코드

- **Docker 이미지 배포 (수동)**:

```
docker build -t asia.gcr.io/my-gcp-project/qualijournal-admin:latest .
docker push asia.gcr.io/my-gcp-project/qualijournal-admin:latest
gcloud run deploy qualijournal-admin \
  --image asia.gcr.io/my-gcp-project/qualijournal-admin:latest \
  --region=asia-northeast3 --platform=managed \
  --update-env-vars ADMIN_TOKEN=xxx,API_KEY=yyy,DB_URL=zzz
```

- **CI/CD 트리거**:

```
git add .
git commit -m "Deploy new version"
git push origin main
```

(메인 브랜치에 푸시하면 CI가 빌드/배포 실행)

- **Cloud Run 서비스 설명 조회**:

```
gcloud run services describe qualijournal-admin --region=asia-northeast3
```

(서비스 설정 상세를 확인하여 환경변수, URL, 트래픽 분배 등 상태를 확인)

예상 결과

- **Cloud Run 배포 성공**: 새로운 기능(API 및 UI)이 포함된 컨테이너 이미지가 Cloud Run에 배포되어, 서비스 URL에 접속했을 때 정상적으로 동작함을 확인합니다 [133] [151]. Cloud Run의 **Revision History**에 최근 배포 시각과 이미지 SHA 태그가 기록되고, 트래픽이 100% 최신 리비전에 할당됩니다.
- **환경변수 유지**: Cloud Run 서비스의 환경변수 설정이 최신 버전으로 잘 적용되어 있습니다. 만약 CI에서 env_vars를 매번 덮어쓰지 않았다면, 이전에 설정된 값들이 계속 유지되며, 새 버전 코드에서도 동일하게 사용됩니다 [152]. (CI에서 env_vars를 지정했다면 그 값으로 업데이트되었을 것이고, 지정 안 했다면 수동 설정 값이 남음)
- **CI/CD 자동화**: GitHub Actions (또는 Cloud Build)가 모든 테스트를 통과시키고 자동으로 배포를 완료함으로써, **코드 푸시 -> 배포** 흐름이 사람 개입 없이 원활히 이뤄집니다 [153]. 테스트 실패시 배포가 스킵되는 것도 확인되므로, 향후 실수로 버그를 커밋해도 배포 전에 잡아낼 수 있습니다.
- **모니터링 정상**: 배포 후 Cloud Run 로그와 Monitoring 대시보드에 **오류 로그가 없고**, 리소스 사용량도 안정적임을 확인합니다 [148]. 새로운 엔드포인트들(`/api/report`, `/api/enrich/...`, `/api/export/...`)의 호출 로그가 200으로 기록되고 있고, 시간 지연도 허용 범위 내입니다. CPU/Memory 곡선도 이전과 큰 차이 없거나, 증가했더라도 허용 한계 내라면 괜찮습니다 (필요시 Cloud Run 서비스 설정에서 메모리 MB를 높일 수 있음).
- **운영 체크리스트 활용**: 최종적으로, 운영 담당자는 아래 **체크리스트**를 참고하여 일일 발행 작업과 배포 후 점검을 체계화할 수 있습니다. 이 체크리스트를 따라하면 누락된 부분 없이 안정적으로 시스템을 유지할 수 있을 것입니다 [149] [150].

주의사항

- **Dockerfile 최신화**: 새 패키지 추가나 코드 구조 변경이 있었다면 Dockerfile을 수정했는지 다시 확인합니다 [142]. 예를 들어 `requirements.txt`를 빌드 시 COPY하고 `pip install` 하는 부분이 최신인지, 불필요한 캐시 레이어가 없는지 점검하세요. Docker 이미지 빌드시 로컬에서는 되는데 Cloud Build/CI에서는 실패하는 경우, Dockerfile 경로나 context 문제일 수 있으니 확인합니다.
- **배포시 환경변수 전송**: Cloud Run 배포 명령어에 환경변수를 CLI에 직접 넣을 경우, Shell 히스토리에 남을 수 있으니 이후 지우거나 CI를 활용하는 것이 안전합니다 [154] [155]. 가능하면 환경변수는 **CI 시크릿**으로 관리하여 배포하도록 하고, 로컬에서 수동 배포 시에는 이미 콘솔에 설정되어 있다면 `--update-env-vars` 옵션을 생략하여 기존 값을 유지시키는 편이 낫습니다.
- **트래픽 분배 확인**: 새 Revision이 배포된 후 Cloud Run이 자동으로 **트래픽 100%**를 할당하지만, 혹시 수동으로 이전 버전을 일부 유지하도록 설정한 경우 남아있지 않은지 확인합니다 [156]. 필요시 Cloud Run 콘솔의 **Edit Traffic** 기능을 이용해 모든 트래픽을 신버전에 보내거나, 오래된 Revision을 **삭제**하여 오용을 방지합니다.
- **롤백 플랜**: 최종 배포 후 혹시 문제 발생 시를 대비해 **롤백 절차**를 마련합니다 [157]. Cloud Run의 Revision History에서 이전 리비전을 재할당하는 방법을 알아두고, 긴급시 코드 재배포 또는 토큰 교체 등을 할 수 있도록 준비합니다. 또한 CI/CD 파이프라인에도 비상시 배포 중단 또는 이전 버전으로 되돌리는 workflow를 수동 트리거로 만들어 두면 유용합니다.
- **배포 후 테스트**: 배포 후 반드시 **관리자 UI 전체 기능 테스트**를 수행합니다 [147]. 예를 들어 UI에서:
- "일일 보고서 생성" -> 성공 메시지 및 Markdown 확인
- "뉴스 요약/번역" -> 요약된 내용이 나온 것 확인
- "Export" -> 다운로드 파일 열어 내용 확인
  등의 **스모크 테스트**를 합니다 [146] [158]. 또한 API 엔드포인트를 `curl`이나 Postman으로 직접 호출해보는 것도 좋습니다 (500 에러 등이 없는지 확인). 모든 기능이 의도대로 작동하면 최종 배포를 성공으로 판단합니다.

최종 점검 체크리스트

마지막으로, 시스템 운영을 위한 주요 체크포인트들을 정리한 **체크리스트**입니다. 이 항목들을 **매일 발행 업무 전후**, **배포 직후** 등에 확인하면 누락 없이 안정적인 운영이 가능합니다:

- [ ] **키워드 발행 전 기사 수 확인**: 매일 발행 전에 **승인된 기사 수 (**`selection_approved`**)**가 **최소 필요 개수 (**`gate_required`**)**를 충족하는지 `/api/status` 등으로 점검합니다 [150] . 부족할 경우 추가 승인하거나 임계값을 조정하고, 발행 후에는 원래 임계값으로 복구합니다.
- [ ] **품질 게이트 통과 여부**: `/api/status`의 `gate_pass` 값이 `false`이면 발행을 보류하거나 임계치를 낮춰 `true`로 만든 후 진행합니다 [159] . (품질 게이트 미통과 시 발행을 강행하면 콘텐츠 품질 저하 우려가 있습니다.)
- [ ] **뉴스 수집 작업 완료 확인**: 해당 키워드의 **뉴스 수집 크롤러 작업이 정상 완료**되었는지 확인합니다. 예를 들어 `data/selected_keyword_articles.json` 파일의 최종 수정시각이나, `/api/status`의 `keyword_total` 기사 수 등을 보고 이상 여부(특정 소스 기사 0건 수집 등)를 점검합니다 [160] .
- [ ] **대시보드 통계 동기화**: 관리자 UI의 **KPI 통계 수치**(누적 수집 기사 수, 승인 건수 등)가 백엔드 `/api/status` 값과 일치하는지 확인합니다 [161] . 자동 새로고침이 동작하지 않으면 수동 새로고침 버튼을 눌러 최신 상태로 갱신하세요.
- [ ] **일일 보고서 생성 기능 점검**: 발행 직전에 **"일일 보고서 생성"** 버튼을 눌러 보고서가 정상 생성되는지 확인합니다 [162] . 생성된 Markdown에 당일 기사들이 모두 포함되어 있는지, 누락이나 오타는 없는지 검토합니다. 오류 발생 시 Cloud Run 로그나 터미널 출력의 예외 메시지를 확인해 조치합니다 [162] .
- [ ] **요약/번역 기능 점검**: **"요약/번역"** 버튼들을 눌러 요약문과 번역문이 제대로 생성되는지 확인합니다 [163] . 생성된 요약이 기사별로 추가되었는지, 번역문이 자연스러운지 검토합니다. 번역 API 무료 할당량이 소진되진 않았는지도 함께 확인하여, 필요 시 키를 교체하거나 다음날을 대비합니다 [163] .
- [ ] **Export 다운로드 및 백업**: 발행 직후 **Export CSV/Markdown**을 실행하여 파일을 다운로드 받고, 이를 안전한 저장소(운영자 PC, 구글 드라이브 등)에 업로드/보관합니다 [164] . 다운로드한 파일을 직접 열어 한글 깨짐이나 내용 누락이 없는지 검증합니다 [91] [164] .
- [ ] **아카이브 용량 관리**: 서버 `archive/` 폴더에 날짜별 보고서/요약 파일들이 누적되므로, 용량을 모니터링합니다 [165] . Cloud Run 컨테이너는 상태 비저장이지만, 만약 외부 스토리지를 연결해 저장하는 경우 주기적으로 오래된 파일을 압축 보관하거나 삭제하여 공간을 확보합니다 [165] .
- [ ] **로그 및 모니터링**: Cloud Run **로그에 오류나 경고**가 없는지 발행 직후 확인합니다 [166] . 또한 모니터링 지표(메모리, CPU, 요청 레이턴시 등)를 주시하여, 자원 사용이 한계를 넘으면 Cloud Run 인스턴스의 스펙을 조정합니다 [166] . 특히 새로운 요약 기능 도입으로 메모리 사용이 늘었을 수 있으므로 관찰이 필요합니다.
- [ ] **토큰 및 접근 보안 확인**: 프론트엔드에 설정된 ADMIN_TOKEN과 백엔드 환경변수의 값이 **항상 일치**하는지 확인합니다 [167] . 401 에러가 반복 발생하면 토큰 불일치나 만료를 의심하고 바로 수정합니다. 외부에서 토큰 없이 API 접근을 시도한 로그가 있다면 원인을 파악하고, 필요한 경우 ADMIN_TOKEN을 변경하거나 IP 차단을 검토합니다 [167] .
- [ ] **데이터베이스 연동 점검 (옵션)**: 만약 DB를 도입하여 사용 중이라면, 배포 후 **DB 연결에 문제가 없는지** 확인합니다 [168] . Cloud Run이 VPC를 통해 DB에 잘 접속하는지, 마이그레이션이 정상 수행되었는지, 쿼리 성능에 문제는 없는지 살펴봅니다 [168] . 테이블에 데이터 입력이 잘 되고 누락이 없는지도 함께 체크합니다.
- [ ] **CI/CD 동작 확인**: 최신 코드를 push 했을 때 GitHub Actions 등의 CI가 정상적으로 트리거되어, **테스트 통과 후 자동 배포**까지 이루어지는지 주기적으로 확인합니다 [169] . (예: 작은 내용 수정 커밋 → CI 파이프라인 확인) CI 로그에 에러는 없었는지, 배포 완료 알림이 도착했는지 모니터링합니다.
- [ ] **문서 및 인계사항 최신화**: 모든 변경사항이 README와 운영 가이드 문서에 반영되었는지 마지막으로 점검합니다 [170] . 스크린샷이나 예시 값들이 오래된 것은 없는지, 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 [170] .
- [ ] **향후 개선 로드맵 검토**: 남은 장기 과제들(예: 다중 사용자 권한 관리, AI 추천 기능, 검색 기능 등)이 별도 로드맵 문서로 관리되고 있다면, 이번 릴리스 후 해당 항목들을 정리합니다 [171] . 이번 배포 과정에서 발견된 추가 개선 아이디어나 TODO도 기록해두고, 차기 버전에 반영할 계획을 수립합니다 [171] .

以上 (이상). 모든 과정을 완료하면 QualiJournal 관리자 시스템의 **최종 통합 작업**이 완료됩니다. 이제 문서를 잘 보관하고, 향후 인수인계 시 본 가이드북을 참고하여 원활한 프로젝트 지속이 이루어지길 바랍니다. 1 172

---

1 3 4 30 31 35 36 54 55 56 58 59 65 66 1012_2QualiJournal 관리자 시스템 작업 가이드북.pdf
file://file-Ad1vex4HDdK79hKp9LXwzj

2 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 32 33 34 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 57 60 61 62 63 64 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 1012_1QualiJournal 관리자 시스템 최종 단계별 작업 가이드북.pdf
file://file-8wm4iV9GGuoV7FL4ABABAZ