ChatGPT

QualiJournal 관리자 시스템 작업 가이드북

개요

QualiJournal 관리자 시스템의 백엔드(FastAPI)와 프론트엔드(HTML/JS) 개발자 및 운영자를 위한 가이드북입니다. 이 문서는 Google Cloud Run 환경에서 어플리케이션을 배포하고 관리하는 과정을 안내하며, **ADMIN_TOKEN**, **API_KEY**, **DB_URL** 등의 환경 변수를 사용합니다. 현재 게이트 슬라이더, KPI 자동 새로고침, 인증 토큰 적용, 테마/인코딩 개선, 코드 리팩터링 등의 작업이 **완료**된 상태이며, 이 가이드북에서는 **남은 작업들**에 중점을 두어 다룹니다. 각 작업 섹션은 **[목적]**, **[준비 파일]**, **[실행 절차]**, **[예상 결과]**, **[주의사항]** 형식으로 구성되어 있으므로, 순서대로 따라하면서 개발 및 운영 작업을 완성해보세요.

완료된 작업 목록 (참고용): - **게이트 슬라이더 구현 완료** - **KPI 자동 새로고침 완료** - **인증 토큰 (ADMIN_TOKEN) 적용 완료** – 이제 모든 관리자 API 호출에 토큰이 필요합니다. - **테마 및 인코딩 개선 완료** - **백엔드/프론트엔드 코드 리팩터링 완료**

이제 남은 과제들에 대해 자세히 살펴보겠습니다.

일일 보고서 생성 (POST `/api/report`)

목적:
일일 보고서는 시스템의 **일별 주요 지표와 활동 요약**을 생성하는 기능입니다. 관리자는 이 엔드포인트를 호출하여 하루 동안의 KPI 통계나 로그를 집계한 **Daily Report**를 생성할 수 있습니다. 예를 들어, 전일 대비 사용자 수 증가, 생성된 뉴스 카드 수, 조회수 등의 지표를 요약해 보여줄 수 있습니다. 이 기능은 운영자가 매일 손쉽게 서비스 현황을 파악하거나 보고서를 저장할 수 있도록 돕습니다.

준비 파일:
- 백엔드 FastAPI 라우트 파일: `/app/routes/report.py` (또는 유사한 위치의 라우터 모듈) – 새로운 엔드포인트를 추가합니다.
- 데이터베이스 관련 모듈: `models.py` 또는 서비스 로직 파일 – 보고서 생성을 위해 조회해야 할 데이터 소스 (예: User, Article, Stats 테이블 등).
- 환경설정: `.env` 또는 환경 변수 (DB_URL 등) – DB 연결 정보를 통해 쿼리를 실행합니다.
- 프론트엔드 파일: `admin.html` 혹은 관련 JS 파일 – **"일일 보고서 생성"** 버튼 클릭 시 fetch 요청을 보내는 기능을 추가합니다.

실행 절차:
1. **백엔드 엔드포인트 구현:** FastAPI에 POST `/api/report` 라우트를 추가합니다. 이 라우트는 관리자 토큰을 인증 헤더로 받아야 하며(예: `Authorization: Bearer <ADMIN_TOKEN>`). 내부 구현에서는 데이터베이스로부터 필요한 정보를 집계합니다. 예를 들어, 전날 생성된 콘텐츠 수, 신규 사용자 수, 주요 이벤트 로그 등을 쿼리합니다. 그런 다음 이 정보를 하나의 **요약 객체(JSON)**로 만듭니다.
- FastAPI에서 의존성 주입 또는 미들웨어로 **토큰 인증**을 이미 완료한 상태이므로, 새로운 라우트에도 동일한 인증이 적용됩니다 (예: `Depends(auth_required)` 형태 또는 미들웨어에서 헤더 검증).
- 예시 코드 (구조):

```
from fastapi import APIRouter, Depends, HTTPException
from app.auth import verify_admin_token  # 토큰 검증 의존성 (이미 구현됨 가정)
```

```
from app.db import SessionLocal, get_stats_for_today  # DB 세션 및 통계 함수

router = APIRouter()

@router.post("/api/report")
def create_daily_report(token: str = Depends(verify_admin_token)):
    # DB 세션 열기
    db = SessionLocal()
    try:
        stats = get_stats_for_today(db)  # 예: 오늘 날짜의 통계 데이터 집계
    finally:
        db.close()
    if not stats:
        raise HTTPException(status_code=404, detail="No data for report")
    # 보고서 결과 구성
    report = {
        "date": stats.date.strftime("%Y-%m-%d"),
        "new_users": stats.new_users,
        "new_posts": stats.new_posts,
        "total_views": stats.total_views,
        # 필요에 따라 더 많은 필드 추가
    }
    return report
```

위 코드에서 `verify_admin_token` 의존성은 ADMIN_TOKEN 검증을 수행하고, `get_stats_for_today`는 예시로 DB에서 금일(또는 전일)의 지표를 집계하는 함수입니다. 2. **API 테스트 (로컬):** 로컬에서 서버를 실행하고 curl로 엔드포인트를 테스트합니다. Uvicorn 등을 통해 애플리케이션을 구동한 뒤, 관리자 토큰을 포함하여 호출합니다.
- 예시:

```
uvicorn app.main:app --reload # FastAPI dev 서버 실행
```

```
# 일일 보고서 API 호출 (로컬)
curl -X POST "http://localhost:8000/api/report" \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer ADMIN_SECRET_TOKEN"
```

토큰이 올바르지 않으면 401 에러가 반환되어야 합니다. 올바른 토큰과 함께 호출하면 JSON 형태의 일일 보고서 데이터가 응답됩니다. 3. **프론트엔드 연동:** 관리자 웹 UI에 "일일 보고서 생성" 버튼을 만든 후, 해당 버튼 클릭 시 자바스크립트로 `/api/report`에 fetch 요청을 보냅니다.
- JS 예시 (토큰은 로그인 시 저장된 값을 사용한다고 가정):

```
async function generateDailyReport() {
  const token = localStorage.getItem('adminToken'); // ADMIN_TOKEN 저장되어 있다고 가정
  const res = await fetch('/api/report', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
```

```
        'Authorization': `Bearer ${token}`
      }
    });
    if (res.ok) {
      const data = await res.json();
      console.log("일일 보고서:", data);
      alert(`일일 보고서 생성 완료: 신규 사용자 ${data.new_users}명, 신규 포스트 ${data.new_posts}건 등
`);
      // TODO: UI상에 data를 표나 카드 형태로 표시 가능
    } else {
      alert("보고서 생성 실패: " + res.status);
    }
}
```

위 코드로 `/api/report` 호출 후 결과를 브라우저 콘솔 및 alert으로 출력합니다. 실제로는 DOM에 요소를 추가하여 결과를 보여주는 방식으로 구현할 수 있습니다. 4. **결과 확인:** 프론트엔드에서 버튼을 눌러 보고서 생성이 트리거되면, 백엔드 로그 또는 브라우저 네트워크 탭에서 200 응답과 JSON payload를 확인합니다. 이후 UI에 해당 데이터가 표시되도록 추가 작업을 수행합니다.

예상 결과:
- **백엔드:** `/api/report` 호출 시 **HTTP 200** 응답과 함께 JSON 형식의 보고서 데이터가 반환됩니다. 예시 응답:

```
{
  "date": "2025-10-12",
  "new_users": 5,
  "new_posts": 12,
  "total_views": 340,
  "top_post": "QualiJournal 플랫폼 출시 소식"
}
```

만약 해당 날짜에 데이터가 없다면 404 또는 빈 데이터가 반환될 수 있습니다.
- **프론트엔드:** 관리 화면에서 "일일 보고서 생성" 버튼을 클릭하면, 최신 데이터로 갱신된 일일 보고서 정보가 화면에 표시됩니다. 예를 들어 "신규 사용자: 5명, 게시물: 12건, 총 조회수: 340회" 등의 요약이 나타납니다. 또한 개발자 도구 콘솔에도 JSON 객체가 출력됩니다.

주의사항:
- **인증 필수:** ADMIN_TOKEN이 헤더에 포함되지 않으면 **401 Unauthorized**가 반환되어야 합니다. 이 점을 확인하기 위해 테스트 시 잘못된 토큰이나 토큰 없이 호출하여 동작을 검증하세요.
- **데이터 정확성:** 집계에 사용하는 데이터의 기준 시간이 명확해야 합니다(예: 자정 기준 전일 데이터 등). 타임존이 다르면 일별 합산이 어긋날 수 있으므로, **서버 시간대** 또는 쿼리 시 날짜 처리에 유의하세요 (서울 시간 기준 일자 등 설정 검토).
- **자동 스케줄링:** Cloud Run은 상시 구동형이 아니므로, 만약 일일 보고서를 **정해진 시간에 자동 생성**하려는 경우 **Cloud Scheduler** 등의 서비스로 이 엔드포인트를 호출하는 방식을 고려해야 합니다. 이 때도 인증 토큰을 전달해야 하므로, 예약 호출에 토큰을 포함하거나 Cloud Scheduler와 Cloud Run 사이에 IAM Authorizer를 사용하는 등 추가 설정이 필요합니다.
- **성능:** 보고서 집계 쿼리가 복잡하거나 데이터량이 많은 경우 **실행 속도**에 유의하세요. 필요한 경우 인덱스 최적화나 캐시를 활용하고, 응답 지연이 발생하면 백엔드 작업을 비동기로 처리한 뒤 결과를 가져오는 방식도 검토합니다.
- **보관:** 생성된 보고서를 DB에 저장해야 하는 요구사항이 있는지 확인하세요. 현재는 호출 시마다 실시간 계산하여 반환하지만, 필요하다면 **리포트 테이블**에 저장하고 관리자가 이력을 조회할 수 있게 할 수 있습니다.

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

목적:
뉴스 카드 요약/번역 기능은 관리자가 뉴스 기사나 텍스트를 빠르게 요약하고 필요한 경우 번역하여 **"뉴스 카드"** 형태로 제공하기 위함입니다. 예를 들어, 영문으로 된 뉴스 기사를 입력하면 한국어로 **핵심 내용 요약**을 생성하고, 특정 문장을 선택하여 번역할 수도 있습니다. 이로써 운영자는 원본 기사의 내용을 효율적으로 파악하고, 다국어로 콘텐츠를 활용할 수 있습니다 [1].

- `/api/enrich/keyword` 엔드포인트는 **전체 기사나 주요 내용**을 요약하는 데 사용됩니다. (이름은 keyword지만, 요청 본문에 기사 전문 또는 핵심 문장을 담아 요약을 얻는 용도로 해석합니다.)
- `/api/enrich/selection` 엔드포인트는 사용자가 **특정 선택 영역(텍스트)**을 지정하여 번역하거나 요약할 때 사용됩니다. 일반적으로 selection은 문장이나 단락 등 비교적 짧은 텍스트로, 번역 기능에 더 초점을 둡니다.

두 기능 모두 외부 AI API를 활용하여 자연어 처리(요약, 번역)를 수행하며, `API_KEY` 환경 변수를 통해 해당 API의 인증 키를 주입받습니다. (예: OpenAI API 키 등)

준비 파일:
- 백엔드 FastAPI 라우트 파일: `/app/routes/enrich.py` 등 – `/api/enrich/keyword`와 `/api/enrich/selection` POST 핸들러 구현. - 외부 API 연동 모듈: `openai_helper.py` 또는 직접 라우트 파일 내 – OpenAI 등 요약/번역 API를 호출하는 코드.
- OpenAI의 GPT API를 사용할 경우, `openai` 라이브러리 (또는 `requests`)를 통해 HTTP 호출.
- 대안으로 Naver Papago API 등을 사용할 경우, 해당 API 키와 호출 로직.
- 환경 변수: **API_KEY** – 외부 요약/번역 API의 Key. OpenAI를 예로 들면 OpenAI API 키가 .env에 저장되고 `os.getenv("API_KEY")`로 로드. - 프론트엔드 파일: 관리 UI의 뉴스 콘텐츠 화면 JS – 기사 입력 영역과 "요약" 버튼, 텍스트 선택 이벤트와 "번역" 버튼 등에 대한 처리 추가.

실행 절차:

1. **외부 API Key 설정:** 우선 `.env` 파일 또는 Cloud Run 환경 변수에 `API_KEY`를 설정합니다. (OpenAI를 사용한다면 OpenAI Dashboard에서 키를 발급받아 설정). 로컬 개발 시 `.env`에 `API_KEY=<발급받은 키>`를 추가하고, 백엔드 초기화 시 이를 불러오도록 합니다.
2. Python에서 예시:

```
import os, openai
openai.api_key = os.getenv("API_KEY")
```

3. 또는 API 키를 직접 HTTP 요청 헤더/URL에 넣는 서비스라면, 그에 맞게 준비합니다.

4. **백엔드 구현 – 요약 엔드포인트 (`/api/enrich/keyword`):**
   이 엔드포인트는 입력으로 기사 전문 또는 요약 대상 텍스트를 받고, **요약 결과**를 반환합니다. FastAPI에서 `from pydantic import BaseModel`을 이용해 요청 바디 모델을 정의하면 유용합니다. 예를 들어:

```
class ArticleInput(BaseModel):
    text: str
```

라우트 함수는 `ArticleInput`을 받아 text 필드를 추출하고, AI 요약 API에 요청을 보냅니다.

5. **OpenAI GPT 사용 예:**

```
import openai
from fastapi import APIRouter, Depends
from app.auth import verify_admin_token

router = APIRouter()

@router.post("/api/enrich/keyword")
def summarize_article(data: ArticleInput, token: str = Depends(verify_admin_token)):
    text = data.text
    if not text or len(text) < 10:
        return {"summary": ""}  # 혹은 HTTP 400 에러
    # OpenAI API 호출: 프롬프트 생성
    prompt = f"Summarize the following news article in Korean:\n\"\"\"\n{text}\n\"\"\""
    try:
        response = openai.Completion.create(
            engine="text-davinci-003",  # 또는 최신 모델 엔진
            prompt=prompt,
            max_tokens=1024,
            temperature=0.5,
        )
    except Exception as e:
        # 오류 처리 (API 오류 등)
        raise HTTPException(status_code=502, detail=str(e))
    summary_text = response.choices[0].text.strip()
    return { "summary": summary_text }
```

위 코드는 입력된 기사를 한국어로 요약하도록 프롬프트를 구성한 예시입니다. (`text-davinci-003` 모델을 사용하여 프롬프트를 던짐). **API 요청 비용과 토큰 제한**을 고려해 `max_tokens` 등을 적절히 설정합니다. 요약 결과는 JSON으로 감싸 `{"summary": "...요약문..."}` 형태로 반환합니다.

6. **대안 (Papago 등):** 만약 외부 API로 **요약 기능이 없다면**, OpenAI 대신 자체 알고리즘이나 간단한 heuristic으로 키워드를 추출하는 방법도 있지만, 여기서는 AI 요약을 기본 가정합니다. Papago API는 번역만 제공하므로 요약에는 사용할 수 없습니다.

7. **언어 처리:** 프롬프트에 `"in Korean"`처럼 명시하여 결과를 한국어로 받을지, 혹은 `"Summarize in the same language"` 등으로 할지 결정합니다. **영문 기사를 한국어로 요약**하려면 위 예시처럼 지시할 수 있고, **한글 기사 요약**이라면 `"한글로 입력된 기사를 간략히 요약해줘"`와 같이 한국어로 요청해도 됩니다. 필요 시 언어 감지를 통해 조건부로 프롬프트를 조정할 수도 있습니다.

8. **백엔드 구현 – 번역 엔드포인트 (`/api/enrich/selection`):**
이 엔드포인트는 짧은 텍스트 조각을 번역하거나 요약하여 반환합니다. 일반적으로 selection은 번역 용도로 쓰일 가능성이 높으므로, 여기서는 **번역 기능** 중심으로 설명합니다. 요청 바디는 `{ text: "..."}`로 전달되고, 응답은 로 전달합니다.

9. **OpenAI 사용 예:** OpenAI의 GPT는 번역도 가능하므로, ChatGPT API (gpt-3.5-turbo 등)을 써서 간단히 번역할 수 있습니다.

```
@router.post("/api/enrich/selection")
def translate_selection(data: ArticleInput, token: str = Depends(verify_admin_token)):
    text = data.text
    if not text:
        raise HTTPException(status_code=400, detail="No text provided")
    prompt = f"Translate the following text to Korean:\n\"\"\"\n{text}\n\"\"\""
    try:
        response = openai.Completion.create(
            engine="text-davinci-003",
            prompt=prompt,
            max_tokens=512,
            temperature=0.0
        )
    except Exception as e:
        raise HTTPException(status_code=502, detail="Translation API error")
    translated = response.choices[0].text.strip()
    return { "translation": translated }
```

위에서는 간단히 주어진 텍스트를 한국어로 번역하도록 프롬프트를 구성했습니다. 필요에 따라 target language를 파라미터로 받을 수도 있습니다 (예: 한국어↔영어 토글).

10. **Papago API 사용 예:** Naver Papago 번역 API를 사용하려면, Papago에서 애플리케이션을 등록하고 Client ID/Secret을 받아야 합니다. 그런 다음 `requests`로 Papago 엔드포인트에 POST 호출 (source, target 언어 코드와 텍스트)하고 결과를 받아오면 됩니다. 이때 Papago API 자격증명을 다루기 위해 **추가 환경변수** (예: PAPAGO_CLIENT_ID, PAPAGO_SECRET)를 설정해야 합니다.

11. **응답 생성:** 반환은 JSON으로 `{"translation": "...번역문..."}` 형식으로 합니다. 번역 결과 문자열 양 끝 공백을 `.strip()`으로 정리하는 등 후처리를 합니다.

12. **프론트엔드 연동:**

13. **기사 요약:** 예를 들어 관리 UI에서 기사를 등록하거나 편집하는 화면이 있다고 할 때, "요약 생성" 버튼을 만들어 `articleTextarea.value` 등을 백엔드로 보내고, 응답받은 summary를 별도의 필드나 모달에 표시합니다.

```
async function requestSummary() {
  const articleText = document.getElementById('articleContent').value;
  const res = await fetch('/api/enrich/keyword', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': 'Bearer ' + localStorage.getItem('adminToken')
    },
    body: JSON.stringify({ text: articleText })
  });
  if (res.ok) {
    const data = await res.json();
    document.getElementById('summaryResult').innerText = data.summary || "(요약 결과 없음)";
  } else {
    alert("요약 실패: " + res.status);
```

```
  }
}
```

이와 같이 요약 버튼 클릭 시 `requestSummary()`를 호출하도록 UI 이벤트를 바인딩합니다. 요약 결과는 `#summaryResult` 영역에 넣어 사용자가 확인할 수 있게 합니다.

14. **텍스트 번역:** UI에서 사용자가 기사 내용 일부를 드래그 선택한 후 "번역" 버튼을 누르면, 선택된 텍스트를 가져와 API에 보냅니다. DOM에서 선택된 내용을 얻는 방법은 `window.getSelection().toString()` 등을 활용할 수 있습니다. 예를 들어:

```
async function translateSelection() {
  const selectedText = window.getSelection().toString();
  if (!selectedText) {
    alert("번역할 텍스트를 먼저 선택하세요.");
    return;
  }
  const res = await fetch('/api/enrich/selection', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': 'Bearer ' + localStorage.getItem('adminToken')
    },
    body: JSON.stringify({ text: selectedText })
  });
  if (res.ok) {
    const data = await res.json();
    console.log("번역 결과:", data.translation);
    // 예: 번역 결과를 모달 팝업으로 표시
    showModal("번역 결과", data.translation);
  } else {
    alert("번역 실패: " + res.status);
  }
}
```

여기서 `showModal(title, content)`은 미리 정의된 모달 창 표시 함수라고 가정합니다. 선택한 원문과 번역문을 함께 보여주도록 구현할 수도 있습니다.

15. **통합 테스트:** 실제 기사 텍스트(영어/한글)를 넣고 UI에서 요약/번역 기능을 실행해 봅니다. 예를 들어 영문 기사 한 단락을 복사해 넣고 "요약 생성"을 눌렀을 때 한국어 요약 결과가 몇 줄로 나오는지 확인합니다. "번역"의 경우 임의의 영어 문장을 선택하여 번역 결과가 정확히 출력되는지 확인합니다.

예상 결과:
- **요약 결과:** 영문으로 작성된 긴 뉴스 기사를 입력하고 `/api/enrich/keyword`를 호출하면, 핵심 내용이 담긴 **짧은 한국어 요약문**이 반환됩니다. 예시: 원문 5~6문단의 기사가 입력되었을 때 응답 JSON:

```
{
  "summary": "이 기사는 2022년 주식시장의 혼조세와 2023년 연준의 금리 결정에 대한 투자자들의 신중한 태도를 다루고 있습니다. 많은 현금이 시장 외곽에 머물러 있으며, 만약 랠리가 지속된다면 매수세가 촉발될 수 있다
```

```
는 전망을 제시합니다. 전체적으로 향후 시장 반등 가능성과 투자 심리에 대한 긍정적인 견해를 보여줍니다."
}
```

(위 예시는 요약된 한국어 내용을 가상의 형태로 제시한 것입니다.)
- **번역 결과:** 짧은 문단이나 문장을 `/api/enrich/selection`으로 보내면 해당 문장의 번역이 반환됩니다. 예를 들어 원문 선택: "Markets are cautious ahead of the Fed's decision." -> 응답:

```
{
 "translation": "연준의 결정에 앞서 시장은 신중한 태도를 보이고 있다."
}
```

번역 결과는 정확한 문맥을 반영하여 적절한 한국어 (또는 요청한 타겟 언어) 문장으로 나타납니다. 프론트엔드에서도 이 문자열을 받아 화면에 표시합니다.

주의사항:
- **API 키 보안:** 외부 API 호출에 사용되는 `API_KEY`는 코드에 하드코딩하지 말고 **환경 변수**로 주입해야 합니다. 운영 환경에서는 Cloud Run의 Secret Manager 연동이나 환경 변수 설정을 통해 주입하고, `.env` 파일은 절대 깃 등에 커밋하지 않도록 합니다.
- **요약 품질:** AI를 사용한 요약 결과는 때때로 원문 정보를 누락하거나 부정확할 수 있습니다. **중요 보고서에 활용하기 전에 사람의 검수**가 필요할 수 있음을 유념하세요. 또한, 한국어 요약의 어투나 번역투가 어색하면 프롬프트를 조정하거나 후처리 규칙(예: 존칭/반말 통일)을 적용하세요.
- **한글 입력 요약:** 한글 기사를 요약할 경우 OpenAI 등에 한국어로 요약을 요청하면 되지만, 모델에 따라 한국어 응답이 영어보다 품질이 낮을 수 있습니다. 필요하면 한국어로 질문을 작성하여 요청하거나, 영어로 번역한 후 요약하는 2단계도 고려할 수 있습니다.
- **번역 대상 언어:** 현재 예시로 English -> Korean 번역을 다루었지만, 요구에 따라 반대로 한국어->영어 번역이나 다국어 지원이 필요할 수 있습니다. 이 경우 UI에서 언어 옵션을 제공하거나, `/api/enrich/selection` 엔드포인트에 `target_lang` 파라미터를 추가해 처리할 수 있습니다.
- **API 호출 비용/속도:** 요약/번역 API 호출에는 응답 지연이 몇 초 발생할 수 있고 비용이 발생합니다. **대용량 기사**를 처리할 때는 토큰 한도에 유의하세요. OpenAI Davinci 모델의 경우 한 번에 약 4000 tokens 한도이므로, 그 이상의 텍스트는 잘라서 여러 번 요약하거나 GPT-4 8k/32k 모델을 활용해야 합니다. 또한, Cloud Run의 요청 타임아웃(기본 5분)을 넘지 않도록 해야 하며, 일반적으로 요약/번역은 10~30초 내에 끝나게 하는 것이 좋습니다.
- **예외 처리:** 외부 API 오류나 타임아웃에 대비하여 try/except로 예외를 잡고 적절한 오류 메시지를 반환하세요. 예를 들어 OpenAI API rate limit 초과 시 429 오류, Papago API 키 오류 시 401 등이 발생할 수 있으므로, FastAPI에서 HTTPException으로 매핑해주면 프론트엔드에서 알맞게 대응할 수 있습니다.
- **텍스트 전처리:** 모델 입력 전에 불필요한 공백이나 특수문자를 정리하고, 필요한 경우 기사 내 광고 문구나 저작권 문구 등을 제거하여 **요약 정확도**를 높이세요.
- **데이터 사용 동의:** 만약 내부적으로 민감한 기사나 미공개 자료를 AI API에 보내는 경우, 해당 서비스(OpenAI 등)의 데이터 사용 정책을 확인해야 합니다. 기본적으로 OpenAI API에 보낸 내용은 학습에 사용되지 않지만(2023년 이후 정책), **회사 내부 데이터의 외부 전송**에 대한 보안 정책을 검토하세요.

Export UI 연동 (GET `/api/export/md`, `/api/export/csv`)

목적:
관리자는 서비스의 데이터를 **Markdown 파일**이나 **CSV 파일**로 내보내어 저장하거나 보고할 수 있어야 합니다. Export 기능의 목적은 예를 들어 **일일 보고서**나 **운영 로그, 데이터 테이블** 등을 파일로 쉽게 추출하는 것입니다. Markdown 형식은 리포트를 문서화하여 공유하기에 좋고, CSV는 스프레드시트에서 열어볼 수 있는 형식입니다. **Export UI 연동** 작업에서는 백엔드의 export API 구현과 프론트엔드의 버튼/링크 동작을 연결합니다.

준비 파일:
- 백엔드 FastAPI 라우트 파일: `/app/routes/export.py` – `/api/export/md`와 `/api/export/csv`에 대한 GET 핸들러 구현. - 데이터 구성 모듈: 보고서 데이터나 내보낼 컨텐츠를 준비하는 함수. 예: `generate_markdown_report()` 또는 DB에서 특정 테이블 데이터를 CSV로 변환하는 로직. - 프론트엔드 파일: `admin.html` 또는 JS – "Markdown로 내보내기", "CSV로 내보내기" 버튼 추가 및 클릭 시 동작 구현. - 기타: Markdown 템플릿이 필요한 경우 템플릿 파일(.md 템플릿) 또는 CSV 헤더 정의.

실행 절차:

1. **백엔드 – Markdown Export 엔드포인트 (`/api/export/md`):**
   이 엔드포인트는 보통 정적 파일 다운로드를 처리합니다. 호출 시 최신 보고서나 데이터 요약을 **Markdown 문자열**로 생성하여 파일로 응답합니다.
2. FastAPI에서 파일 다운로드 응답을 주는 방법은 `FileResponse`를 쓰거나 `Response` 객체에 적절한 헤더를 추가하는 것입니다. 여기서는 동적으로 파일 내용을 생성하므로, **StreamingResponse**나 **Response**를 사용합니다.
3. 예시 구현:

```
from fastapi import Response
import datetime

@router.get("/api/export/md")
def export_markdown(token: str = Depends(verify_admin_token)):
    # 1) 내보낼 Markdown 컨텐츠 생성
    today = datetime.date.today().strftime("%Y-%m-%d")
    md_content = "# 일일 보고서 (" + today + ")\n"
    md_content += "- 신규 사용자: 5명\n"
    md_content += "- 신규 포스트: 12건\n"
    md_content += "- 총 조회수: 340회\n"
    md_content += "\n*Generated by QualiJournal Admin*"
    # 2) 응답 헤더 설정 (파일 다운로드)
    headers = {
        'Content-Type': 'text/markdown; charset=utf-8',
        'Content-Disposition': f'attachment; filename="report_{today}.md"'
    }
    return Response(content=md_content, headers=headers, media_type='text/
markdown')
```

   위 코드는 예시로 하드코딩된 통계를 넣었지만, 실제로는 앞서 `/api/report`와 동일한 데이터를 이용해 Markdown 문서를 구성하면 됩니다. **Content-Disposition** 헤더에 `attachment; filename="...md"`를 지정하면 브라우저가 강제로 다운로드를 진행합니다.

4. Markdown 내용은 필요에 따라 서식을 추가할 수 있습니다 (테이블 형식 등). 예를 들어 `md_content`에 `| 항목 | 값 |\n|---|---|\n| 신규 사용자 | 5명 | ...`처럼 표 형태로 작성하거나, 추가 설명을 문단으로 넣을 수 있습니다.

5. **백엔드 – CSV Export 엔드포인트 (`/api/export/csv`):**
   CSV 내보내기는 대량의 표형 데이터를 다루기에 적합합니다. 예를 들어 **전체 사용자 목록**이나 **포스트 목록**을 CSV로 뽑는 용도로 사용할 수 있습니다.

6. 구현 방법은 Markdown과 유사하지만, **CSV 포맷**에 맞게 데이터를 구성해야 합니다. Python 표준 라이브러리의 `csv` 모듈을 사용하면 편리합니다.
7. 예시 구현 (간단한 테이블 가정):

```
import csv, io

@router.get("/api/export/csv")
def export_csv(token: str = Depends(verify_admin_token)):
    # DB로부터 가져온 데이터 예시 (컬럼: date, new_users, new_posts, total_views)
    data = [
        ("2025-10-10", 3, 10, 300),
        ("2025-10-11", 4, 8, 280),
        ("2025-10-12", 5, 12, 340),
    ]
    output = io.StringIO()
    writer = csv.writer(output)
    # 헤더 작성
    writer.writerow(["date", "new_users", "new_posts", "total_views"])
    # 데이터 작성
    writer.writerows(data)
    csv_content = output.getvalue()
    output.close()
    headers = {
        'Content-Type': 'text/csv; charset=utf-8',
        'Content-Disposition': 'attachment; filename="export.csv"'
    }
    return Response(content=csv_content, headers=headers, media_type='text/csv')
```

위 코드는 예시 데이터를 사용했지만, 실제로는 DB 쿼리를 통해 필요한 레코드를 얻어와 `data` 리스트를 채운 뒤 CSV로 작성합니다. `io.StringIO()`를 이용하면 메모리 상에서 문자열 버퍼를 만들어 파일 I/O 없이도 CSV를 작성할 수 있습니다.

8. **주의:** csv.writer는 기본적으로 쉼표로 구분하고 줄바꿈으로 행 구분을 합니다. 만약 값에 쉼표가 포함될 경우 자동으로 따옴표로 감싸주지만, 데이터에 개행이나 특수문자가 있으면 Excel 등에서 읽을 때 문제가 될 수 있으니 필터링하거나 `quoting=csv.QUOTE_MINIMAL` 등의 옵션을 사용할 수 있습니다.

9. **프론트엔드 – Export 버튼/링크 구현:**
   파일 다운로드를 위해 프론트엔드에서는 두 가지 방식을 고려할 수 있습니다.

10. **방법 A:** 단순 `<a>` 링크에 URL을 걸고, 클릭 시 다운로드. 이 경우 **인증 토큰을 포함**해야 하는데, Authorization 헤더는 일반 링크 클릭으로는 보낼 수 없습니다. 토큰을 URL 파라미터로 붙이는 것은 보안상 바람직하지 않습니다. 따라서 이 방법은 **불가**하거나, 토큰이 쿠키에 있다면 가능하겠지만 우리는 쿠키 인증을 사용하지 않으므로 생략합니다.
11. **방법 B: JavaScript fetch**로 API 응답(파일)을 받아 Blob 형태로 저장 후 다운로드 트리거. 이 방법을 사용하면 Authorization 헤더를 쉽게 넣을 수 있고, 응답을 수동으로 처리하여 파일 저장이 가능합니다.
12. 여기서는 방법 B를 적용합니다. Export 버튼에 클릭 이벤트를 달아서, 해당 형식으로 fetch -> blob -> 다운로드 흐름을 구현합니다.

```
function downloadFile(url, filename) {
  const token = localStorage.getItem('adminToken');
```

```
    fetch(url, {
      headers: { 'Authorization': 'Bearer ' + token }
    })
    .then(response => {
      if (!response.ok) throw new Error("네트워크 응답 실패");
      return response.blob();
    })
    .then(blob => {
      // Blob을 사용하여 임시 다운로드 링크 생성
      const downloadUrl = URL.createObjectURL(blob);
      const a = document.createElement('a');
      a.href = downloadUrl;
      a.download = filename;
      document.body.appendChild(a);
      a.click();
      a.remove();
      URL.revokeObjectURL(downloadUrl);
    })
    .catch(error => {
      alert("파일 다운로드 실패: " + error.message);
    });
}

// "Markdown 다운로드" 버튼 onClick
document.getElementById('btnExportMd').addEventListener('click', () => {
  const dateStr = new Date().toISOString().slice(0,10);
  downloadFile('/api/export/md', `report_${dateStr}.md`);
});
// "CSV 다운로드" 버튼 onClick
document.getElementById('btnExportCsv').addEventListener('click', () => {
  downloadFile('/api/export/csv', 'export.csv');
});
```

위 코드에서 `downloadFile` 함수는 공통으로 fetch 요청을 수행하고, Blob 데이터를 다운로드 처리합니다. `URL.createObjectURL`을 사용하여 브라우저 메모리상 Blob을 가리키는 임시 링크를 만들고, `<a>` 태그를 통해 다운로드를 트리거합니다.

13. HTML 버튼 예시:

```
<button id="btnExportMd">Markdown로 내보내기</button>
<button id="btnExportCsv">CSV로 내보내기</button>
```

이 버튼들이 앞서 JS 이벤트와 연결되어 동작하게 됩니다.

14. **기능 테스트:** 브라우저에서 Export 버튼을 눌러보세요. Markdown 버튼 클릭 시 파일 저장 대화창(혹은 브라우저 자동 다운로드)이 나타나고, 지정한 `report_<날짜>.md` 파일이 다운로드됩니다. 메모장이나 마크다운 뷰어로 열어 내용(보고서)이 정상적으로 보이는지 확인합니다. CSV 버튼도 마찬가지로 `export.csv` 파일 다운로드 후 Excel이나 텍스트 에디터로 열어 내용과 형식(쉼표로 컬럼 구분)이 올바른지 확인합니다.

예상 결과:
- **Markdown 내보내기:** 관리화면에서 Markdown 버튼을 클릭하면, 예를 들어 `report_2025-10-12.md` 파일이 다운로드됩니다. 파일 내용을 열어보면 다음과 비슷한 Markdown 문서가 있습니다:

```
**# 일일 보고서 (2025-10-12)**
- 신규 사용자: 5명
- 신규 포스트: 12건
- 총 조회수: 340회

*Generated by QualiJournal Admin*
```

Markdown이기 때문에 마크다운 뷰어나 GitHub 등에 업로드하면 서식이 적용됩니다. (헤더 글씨 크기, 리스트 표시 등)
- **CSV 내보내기:** CSV 버튼을 누르면 `export.csv` 파일이 내려받아지고, 열어보면 UTF-8 인코딩으로 데이터 행이 나옵니다. 예:

```
date,new_users,new_posts,total_views
2025-10-10,3,10,300
2025-10-11,4,8,280
2025-10-12,5,12,340
```

각 줄이 각각의 레코드를 나타내며, 쉼표로 구분된 컬럼은 Excel에서 셀로 인식됩니다. Excel로 열었을 경우 한글이 깨지지 않고 잘 보이는지 (인코딩 확인)도 체크하세요.

주의사항:
- **인증 및 권한:** Export 엔드포인트 역시 인증이 필요합니다. 프론트엔드 fetch 코드가 Authorization 헤더를 세팅하므로 정상 동작하나, 혹시 토큰이 없을 경우 서버는 401을 반환합니다. UI에서는 이 상황에 대한 처리 (예: 로그인 페이지로 리다이렉트)를 할 수도 있습니다.
- **파일명 및 형식:** Content-Disposition 헤더의 파일명을 동적으로 생성할 때, 공백이나 특수문자가 있으면 다운로드 시 문제가 될 수 있습니다. 가능하면 영어와 숫자, 밑줄 등만 사용하세요. 한글 파일명은 브라우저별 인코딩 이슈가 있으므로 피하거나 URL 인코딩을 고려해야 합니다.
- **대용량 데이터:** CSV로 수천 건 이상의 데이터를 한 번에 보내는 경우, 응답 크기가 매우 커질 수 있습니다. Cloud Run의 응답 스트림 한도를 초과하지는 않겠지만, 네트워크 전송에 시간이 걸릴 수 있습니다. 이때 **StreamingResponse**를 사용하여 chunk 단위로 전송하거나, 미리 생성한 파일을 Cloud Storage에 올려두고 다운로드 링크를 제공하는 방법도 고려할 수 있습니다. 현재 시나리오에서는 적정 수준의 데이터 (수백~수천 행)라 가정합니다.
- **인코딩:** UTF-8로 응답을 보내고 있으나, Excel에서 CSV를 열 때 기본 인코딩을 다르게 인식하면 한글이 깨질 수 있습니다. 이 문제를 피하려면 BOM(Byte Order Mark)을 추가하는 방법도 있습니다. `Response(content='\ufeff' + csv_content, ...)`처럼 문자열 앞에 `\ufeff`를 붙이면 Excel이 UTF-8로 인식합니다. Markdown의 경우는 텍스트 에디터 대부분 UTF-8을 잘 인식하니 큰 문제는 없습니다.
- **보안:** Export된 파일에 **민감정보**(예: 사용자 이메일, 전화번호 등)가 포함되지 않도록 주의하세요. 관리자만 받는다고 해도 파일 유출 위험을 고려해야 합니다. 필요한 경우 컬럼을 제한하거나 익명화하세요.
- **유효성 검사:** 요청에 따라 Export 대상 범위를 지정할 수도 있습니다 (예: 특정 날짜 범위의 데이터만 CSV로). 현재 구현은 고정 전제이지만, 확장 계획이 있다면 쿼리 파라미터 (`/api/export/csv?from=2025-10-01&to=2025-10-12`) 등을 파싱하여 해당 기간의 데이터만 추출하는 기능을 고려하세요.

통합 테스트 및 README/매뉴얼 정비

목적:
마지막 개발 단계에서는 전체 시스템에 대한 **통합 테스트**를 수행하고, 프로젝트의 **README 및 운영 매뉴얼**을 최신 상태로 업데이트합니다. 통합 테스트의 목표는 백엔드와 프론트엔드 기능이 조화롭게 동작하는지, 특히 새로 구현한 일일 보고서 생성, 요약/번역, Export 등의 기능이 요구사항에 맞게 작동하는지 검증하는 것입니다. README/매뉴얼 정비의 목표는 개발자 및 운영자가 프로젝트를 설정, 배포, 사용하는 방법을 명확히 알 수 있도록 **문서화**하는 것입니다.

준비 파일:
- **테스트 코드**: `/tests/` 디렉토리 (예: `test_main.py`, `test_api_endpoints.py` 등) – pytest를 활용하여 API 응답을 검증하는 테스트 스크립트.
- **테스트 설정**: `conftest.py` – 테스트용 설정 (예: 테스트 DB URL, 토큰 등)을 정의.
- **README.md** (프로젝트 루트) – 프로젝트 설명, 설치 방법, 사용 방법, 배포 방법 등을 기술.
- **운영 매뉴얼** (필요 시 별도 문서) – 운영자가 시스템을 관리하는 데 필요한 절차 (예: 환경변수 변경, 장애 대응)를 정리.

실행 절차:

1. **백엔드 통합 테스트 작성:** pytest를 사용하여 API 단위 및 통합 테스트를 진행합니다. 우선 FastAPI의 `TestClient`를 이용해 백엔드 API를 호출하고 응답을 검증하는 방식을 채택합니다 [2] [3].
2. **준비:** `pip install pytest`로 pytest를 설치하고, 필요한 경우 `pip install httpx` (TestClient 내부사용) 등을 설치합니다.
3. **예시 테스트 (토큰 인증 및 기능 확인):**

```
from fastapi.testclient import TestClient
from app.main import app  # FastAPI 애플리케이션 객체

client = TestClient(app)
ADMIN_TOKEN = "TEST_ADMIN_TOKEN"  # 테스트용 토큰 (실제 app에서 ADMIN_TOKEN을 이 값으로 간주하도록 설정해야 함)

def test_report_endpoint_success(monkeypatch):
    # 가짜 토큰 인증 우회: 항상 통과
    monkeypatch.setenv("ADMIN_TOKEN", ADMIN_TOKEN)
    # 또한, get_stats_for_today 등 내부 함수에 대해 더미 데이터를 반환하도록 patch할 수도 있음
    response = client.post("/api/report", headers={"Authorization": f"Bearer {ADMIN_TOKEN}"})
    assert response.status_code == 200
    data = response.json()
    assert "date" in data and "new_users" in data

def test_report_endpoint_unauthorized():
    response = client.post("/api/report")  # 토큰 없이 호출
    assert response.status_code == 401

def test_enrich_summary_and_translation(monkeypatch):
    monkeypatch.setenv("ADMIN_TOKEN", ADMIN_TOKEN)
    monkeypatch.setenv("API_KEY", "DUMMY_API_KEY")
    # 실제 외부 API 호출을 테스트에서 수행하기 어렵다면, openai.Completion.create 등을 monkeypatch로 가짜 함수 대체
```

```
    # 예: monkeypatch.setattr(openai.Completion, "create", fake_completion_func)
    article = "This is a test article. It has multiple sentences."
    res1 = client.post("/api/enrich/keyword", json={"text": article},
headers={"Authorization": f"Bearer {ADMIN_TOKEN}"})
    assert res1.status_code == 200
    summary = res1.json().get("summary", "")
    assert isinstance(summary, str)
    # selection - 번역 테스트 (fake response or simple assumption)
    res2 = client.post("/api/enrich/selection", json={"text": "Hello"},
headers={"Authorization": f"Bearer {ADMIN_TOKEN}"})
    assert res2.status_code == 200
    translation = res2.json().get("translation", "")
    assert isinstance(translation, str)
```

위 테스트들은 간략히 동작 여부만 확인하고, 실제 내용 검증은 생략했습니다. 외부 API 연동 부분은 테스트 환경에서 호출하지 않도록 monkeypatch로 대체하거나, `if app.env == "test": ...`와 같이 분기처리하는 방법도 있습니다. **중요**: 외부 API 호출을 테스트에서 그대로 하면 키 노출 및 요금 발생 우려가 있으므로, 가짜 응답을 만들어주는 방법(mocking)을 적용하세요.

4. **실행:** 터미널에서 `pytest` 명령을 실행하면 모든 테스트가 실행됩니다. **CI 설정**에도 pytest를 포함시켜 푸시마다 테스트가 돌도록 설정하면 좋습니다.

5. **커버리지:** `pytest --cov=app` 옵션을 주면 코드 커버리지 측정이 가능합니다. 통합 테스트를 충분히 작성하여 새로운 API들의 커버리지가 확보되도록 합니다.

6. **프론트엔드 통합 테스트 (선택):** 만약 가능하다면 Cypress 같은 도구로 UI 테스트를 작성할 수도 있습니다. 여기서는 개발 리소스를 고려하여 수동으로 브라우저에서 주요 시나리오를 점검했다면, UI 테스트 자동화는 필수가 아닐 수 있습니다. 대신 테스트 체크리스트를 만들어 수동 테스트 결과를 정리해도 좋습니다 (예: 구글 시트로 각 기능별 테스트 케이스와 결과 기록).

7. **README 업데이트:** 프로젝트 루트의 README.md 파일을 열어 아래 사항을 최신화/추가합니다.

8. 프로젝트 개요: QualiJournal Admin 시스템의 목적과 기능 요약.
9. 시스템 구조: 백엔드(FastAPI)와 프론트엔드(HTML/JS) 구조, 사용 기술 스택(FastAPI, SQLite 또는 PostgreSQL, etc.).
10. **설치 및 실행 방법:** 개발 환경에서 실행하는 법 (예: `git clone` -> `.env 설정` -> `docker-compose up` 또는 `uvicorn main:app` 등). 데이터베이스 초기 설정 (마이그레이션이나 샘플 데이터 로드) 절차가 있으면 추가.
11. **환경 변수 설명:** ADMIN_TOKEN, API_KEY, DB_URL 각각에 대해 무엇을 의미하며 어떻게 설정하는지 기술합니다. (예: ADMIN_TOKEN은 관리자 인증을 위한 토큰 값으로, 임의의 복잡한 문자열을 설정; API_KEY는 OpenAI API 키 등 외부 서비스 키; DB_URL은 PostgreSQL 연결 문자열 등 [4]).
12. **실행 예시:** 주요 API의 사용 방법 (어떤 endpoints가 있고 어떻게 호출하는지). 개발자는 Swagger UI 등을 통해 API를 시도해볼 수 있으므로, `/docs` 경로를 활성화했다면 README에 안내합니다. 운영자는 Postman 등을 통해 테스트할 수도 있으니, 토큰 넣는 방법 등을 병기하면 좋습니다.
13. **테스트 및 품질:** pytest로 테스트를 돌리는 방법, linters나 code formatters 사용하는 게 있다면 명시.
14. **배포:** Cloud Run 배포 관련 핵심 내용 (GitHub Actions를 통해 자동 배포되는지, 수동 배포하려면 어떡해야 하는지) 간략히 적습니다. 또한 CI/CD 배지나 링크를 걸어두면 가시성이 좋습니다.
15. **매뉴얼/FAQ:** 자주 겪는 문제와 해결책 (아래 주의사항에 언급할 내용을 정리).

16. **라이선스/Authors:** 필요하면 추가.

17. **운영 매뉴얼 작성:** README에 모두 넣기 어렵거나, 운영자를 위한 상세 지침이 필요하면 별도 문서를 작성합니다 (예: `docs/Operations.md` 또는 위키 페이지). 여기에는:

18. **환경 변수 변경 방법:** Cloud Run 콘솔에서 환경 변수를 수정하는 방법, 재배포 방법.
19. **장애 발생 시 대응:** Cloud Run 로그 확인 방법, 오류 메시지 해석, 재시작 방법.
20. **성능 모니터링:** Cloud Run 메트릭스나, 추가로 설정했다면 Stackdriver Logging/Monitoring 확인 절차.
21. **보안 설정:** (만약 VPC나 DB 접근 IP 제한 등을 했다면 그에 대한 내용).
22. **데이터베이스 백업/복원:** 운영 DB를 백업하거나 복원하는 방법 (Cloud SQL 사용 시 안내 등).

23. 이러한 운영 이슈 관련 내용을 적어둡니다.

24. **팀 공유:** 작성된 README와 매뉴얼을 팀원들에게 공유하여 피드백을 받습니다. 혹시 누락된 부분이나 잘못된 부분이 있으면 수정하고, 실제 문서만 보고 새로운 개발자가 환경을 셋업할 수 있는지 검증해보는 것도 좋습니다.

예상 결과:
- 모든 통합 테스트가 **통과**하여 `================  X passed in Y seconds ================` 와 같은 메시지가 pytest 결과로 나타납니다. 특히 새로운 기능들 (`/api/report`, `/api/enrich/*`, `/api/export/*`)에 대한 테스트가 추가되었으므로, 기능 변경이 있어도 빨리 이상을 감지할 수 있습니다.
- README.md를 비롯한 문서들을 최신 상태로 정비하여, Git 리포지토리 최상단에서 바로 확인할 수 있습니다. 이를 본 개발자/운영자는 이 프로젝트를 어떻게 실행하고, 배포하고, 사용하는지 한눈에 알 수 있습니다. 예를 들어 README 앞부분에 다음과 같은 내용이 들어갑니다:

```
## 환경 변수 설정
다음 변수를 .env 파일이나 Cloud Run 환경에 설정해야 합니다:
- **ADMIN_TOKEN**: 관리자 인증 토큰 (예: `ADMIN_TOKEN=super-secret-token123`)
- **API_KEY**: 외부 API 키 (예: OpenAI 키를 넣거나 개발 시 빈 값)
- **DB_URL**: 데이터베이스 연결 문자열 (예: `postgresql://user:pass@host:5432/dbname`)

## 로컬 실행 방법
1. 레포지토리 클론 및 dependencies 설치...
2. `.env` 파일 생성 및 환경변수 기입...
3. `uvicorn app.main:app --reload` 명령으로 개발 서버 실행...
4. 브라우저에서 `http://localhost:8000` 접속 및 기능 확인...
```

또한 Cloud Run 배포, CI/CD 설명도 간략히 포함되어, 협업자들이 동일한 절차로 작업할 수 있게 됩니다. 운영 매뉴얼에는 예를 들어 "환경 변수 수정 시 Cloud Run 재배포 필요 - Cloud Console에서 서비스 수정 -> 새 리비전 배포 -> 트래픽 라우팅 확인" 등이 상세히 기술됩니다.

주의사항:
- **테스트 환경 분리:** 테스트 시 실제 DB나 API를 사용하지 않도록, **테스트 전용 DB** (예: SQLite 메모리 DB)와 **테스트용 설정**을 사용하세요. pytest 실행 시 `--env=test` 등의 플래그를 주어 애플리케이션이 테스트 모드로 동작하게 할 수도 있습니다. 본 프로젝트에서는 DB_URL을 별도로 세팅하거나, SQLite 메모리로 강제 전환하는 등의 조치를 취해야 실제 운영 DB를 손상시키지 않습니다.
- **MonkeyPatch와 시크릿:** pytest에서 monkeypatch로 환경변수를 넣는 경우, 애플리케이션 초기화 시 이미 os.getenv로 값을 가져갔다면 소용없을 수 있습니다. 이럴 땐 **테스트용 .env 파일**을 만들어 로딩하거나, 또는 애플리케이션이 startup 시 environment를 다시 읽도록 구조를 바꿔야 할 수도 있습니다. - **외부 API Mocking:** OpenAI 등 API 키를 테스트에 노출하거나 실제 호출하면 안 됩니다. unittest.mock 패치를 활용하여 OpenAI 함수를 호출하면 임의의 고정된 응답을 주도록 하세요. 이렇게 하면 비용도 들지 않고, 예측 가능한 결과로 테스트할 수 있습니다.
- **문서 최신화 지속성:** README와 매뉴얼은 코드 변경에 따라 계속 업데이트되어야 합니다. 특히 **배포 방법**이나 **환경 변**

수 키 이름 등이 변경되면 바로 문서에도 반영하세요. 문서가 오래된 정보로 남아있으면 새로운 개발자나 운영자가 큰 혼란을 겪을 수 있습니다.
- **코드 주석/정리:** 테스트와 문서 작성을 하면서 발견된 사소한 버그나 TODO 주석, 불필요한 로그 출력 등이 있으면 정리합니다. 예를 들어 print로 디버깅하던 코드는 제거하고, logging 모듈로 필요한 정보만 로깅하도록 정돈합니다. README에 언급했지만 코드에 없는 기능은 없는지 점검합니다.
- **사용자 매뉴얼:** 만약 최종 사용자(관리자가 아닌 일반 사용자)를 위한 기능 설명이 필요하다면 별도의 사용자 매뉴얼도 고려합니다. 하지만 QualiJournal **관리자 시스템**이라면, 이 문서 자체가 운영자에게 충분한 가이드가 될 것입니다.
- **종합 QA:** 기능 테스트는 가능한 한 다양한 경로를 따라 해보세요. 예를 들어, Export를 토큰 없이 호출, 잘못된 URL로 호출, 요약 기능에 매우 긴 텍스트 넣기 등 엣지 케이스를 수동으로라도 테스트해보면 품질을 높이는 데 도움이 됩니다.

보안 설정 및 환경변수 적용 (.env / Cloud Run)

목적:
운영 환경에서 안전하게 시스템을 동작시키기 위해 **보안 설정**을 강화하고, **환경 변수 관리**를 철저히 합니다. 이 작업에서는 개발 시 편의를 위해 사용하던 설정을 프로덕션 수준으로 끌어올리는 것이 목적입니다. 주요 내용은 `.env` 파일로 로컬 개발 시 환경변수를 로드하고, Cloud Run 배포 시 해당 환경변수를 설정하여 코드에 주입하는 방법입니다. 또한 불필요한 디버그 옵션을 끄고, 중요한 정보가 로그 등에 남지 않도록 조치합니다.

준비 파일:
- **.env 파일**: 프로젝트 루트에 존재 (git에 커밋되지 않도록 .gitignore에 포함) – 로컬 개발용 환경변수 키=값을 저장. - **환경 변수 로딩 코드**: 예를 들어 `app/core/config.py` 또는 `main.py`에서 `python-dotenv` 라이브러리를 사용하거나, Pydantic Settings 모델을 사용. - **Cloud Run 설정**: `cloudrun.yaml` (있다면) 또는 배포 스크립트 – Cloud Run에 환경변수를 설정하는 부분. - **CI/CD 파일**: GitHub Actions workflow 등에서 환경변수 (특히 API 키, DB URL)을 안전하게 주입하는 설정 (GitHub Secrets로 관리). - **기타 설정**: FastAPI `debug` 모드 플래그, CORS 설정, HTTPS 리다이렉트 설정 등 보안 관련 설정 파일.

실행 절차:

1. **.env 파일 활용 (로컬 개발):** 개발 시에는 `.env` 파일에 필요한 모든 환경변수를 지정하고, 앱 시작 시 자동 로드되게 합니다. 방법은 두 가지입니다:
2. **python-dotenv 사용:** `pip install python-dotenv` 후, 애플리케이션 시작 직후 `from dotenv import load_dotenv; load_dotenv()`를 호출하면 현재 디렉토리의 .env 파일을 읽어 `os.environ`에 주입합니다. 이 방식은 간단하나, Cloud Run 등 실제 배포환경에서는 .env를 사용하지 않으므로, `load_dotenv`는 로컬 개발시에만 쓰는 게 좋습니다.
3. **Pydantic BaseSettings 사용:** FastAPI 권장 방식으로, Pydantic의 Settings 모델을 정의하여 .env를 자동으로 읽도록 할 수 있습니다. 예를 들어:

```
from pydantic import BaseSettings
class Settings(BaseSettings):
    admin_token: str
    api_key: str = ""
    db_url: str
    class Config:
        env_file = ".env"
settings = Settings()
ADMIN_TOKEN = settings.admin_token
API_KEY = settings.api_key
DB_URL = settings.db_url
```

이렇게 하면 `.env`에 대응되는 값들이 없으면 오류를 내거나, 기본값 (api_key 같이 빈 문자열 기본 허용)으로 설정됩니다.

4. **.env 예시:**

```
ADMIN_TOKEN=super-secret-token-1234
API_KEY=sk-xxxxxxxxxxxxxxxxxxx
DB_URL=postgresql://qualijournal:password@localhost:5432/quali_db
```

개발 시 DB_URL은 로컬 DB (SQLite 파일 경로나 Docker로 올린 PostgreSQL)를 가리키고, 운영 시는 Cloud Run에 별도 설정합니다.

5. **Cloud Run 환경변수 설정:** Cloud Run에 배포할 때는 .env를 사용하지 않고, 직접 환경변수를 지정해야 합니다. 여러 가지 방법이 있습니다:

6. **gcloud CLI 이용:** 배포 명령에 `--set-env-vars` 옵션을 추가하면 환경변수를 지정할 수 있습니다 [5]. 예를 들어:

```
gcloud run deploy qualijournal-admin \
 --image gcr.io/your-project/qualijournal:latest \
 --platform managed --region=asia-northeast3 \
 --set-env-vars ADMIN_TOKEN=super-secret-
token-1234,API_KEY=sk-...,DB_URL=postgresql://user:pw@host/db
```

이와 같이 한 번에 여러 변수를 콤마로 구분해 전달합니다 [6]. Cloud Run은 이 설정으로 새 Revision을 만들고, 이후 업데이트 시 기존 값들을 유지하거나 교체합니다. 참고로 `--set-env-vars`를 쓰면 명시하지 않은 기존 변수들은 제거되니, 모든 필요한 변수를 한꺼번에 적어줘야 합니다 [7].

7. **Cloud Console 이용:** Cloud Run 웹 콘솔에서 해당 서비스 -> 수정 -> 변수 & 시크릿 탭에서 키/값을 입력하는 방법도 있습니다. UI에서 `.env` 파일 내용을 한번에 붙여넣어 일괄 생성할 수도 있습니다 [8] [9].
8. **YAML 배포:** `gcloud run services replace` 명령으로 YAML에 env를 정의해 배포할 수도 있습니다 (운영에 익숙한 경우 선택).
9. Cloud Run에 환경변수를 설정하면, 컨테이너 내부에서 `os.getenv("ADMIN_TOKEN")` 식으로 값을 가져올 수 있습니다. 따라서 코드 수정 없이, 로컬에서는 .env, Cloud Run에서는 설정된 env var로 값을 얻습니다.
10. **CI/CD Secrets 설정:** 만약 GitHub Actions로 배포 자동화 시, **민감한 값**들은 GitHub Secrets에 저장해두고, workflow 파일에서 참조해야 합니다. 예를 들어 workflow YAML에서:

```
env:
 ADMIN_TOKEN: ${{ secrets.ADMIN_TOKEN }}
 API_KEY: ${{ secrets.API_KEY }}
 DB_URL: ${{ secrets.DB_URL }}
```

그런 다음 `gcloud run deploy ... --set-env-vars "ADMIN_TOKEN=${{ secrets.ADMIN_TOKEN }},API_KEY=${{ secrets.API_KEY }},DB_URL=${{ secrets.DB_URL }}"` 형태로 전달할 수 있습니다. (주의: GH Actions에서 gcloud CLI 사용 시 문자열 인코딩에 유의, 쉼표로 구분하거나 `--set-env-vars^@^KEY=VAL@KEY2=VAL2` 방식을 쓸 수 있습니다).

GitHub Secrets에 민감 정보를 저장해두면 repository collaborator만 접근 가능하며, Actions에서만 decrypt되어 사용되므로 비교적 안전합니다.

11. **기타 보안 설정:**
12. **HTTPS 강제:** Cloud Run 서비스는 기본 domain이 https://로 제공되므로 별도 설정은 필요 없습니다. 만약 Cloud Run Custom Domain을 사용한다면, SSL 인증서가 적용됐는지 확인하세요.
13. **인증 강화:** 현재 ADMIN_TOKEN으로 단순 토큰 인증을 하고 있습니다. 추후 이 토큰이 유출될 위험을 줄이기 위해 **더 복잡한 토큰으로 변경**하거나 **주기적 로테이션**을 고려하세요. 또한 요청 로그 등에 토큰이 남지 않도록, FastAPI 로그나 에러메시지에 토큰값 출력이 없는지 확인합니다.
14. **CORS 설정:** 관리 시스템이 동일한 도메인에서 동작하면 상관 없지만, 다른 도메인에서 API를 호출할 가능성이 있다면 FastAPI에 CORS 미들웨어를 적용해야 합니다. 운영 환경에서는 필요한 오리진만 허용하도록 제한하는 것이 좋습니다.
15. **Secret Manager 사용 권장: 중요**: 데이터베이스 비밀번호나 API 키 등의 **민감 정보는 환경 변수에 직접 저장하지 않는 것이 안전합니다** [4]. Google Cloud Secret Manager에 값을 저장해 두고, Cloud Run 서비스에서 이를 참조하도록 설정할 수 있습니다. 이렇게 하면 값이 암호화되어 관리되고, 필요 시 rotation도 용이합니다. 다만, Secret Manager 연동 시에는 약간의 설정이 필요하며, 환경변수보다는 접근 시 I/O가 발생한다는 점을 참고하세요. 현재는 간단히 환경변수로 다루지만, 장기적으로는 Secret Manager 도입을 고려합니다.
16. **DB 보안:** DB_URL이 가리키는 데이터베이스 (예: Cloud SQL) 접근을 제한해야 합니다. Cloud Run에서 Cloud SQL을 쓰는 경우, Cloud Run 서비스를 특정 VPC에 연결하고 Cloud SQL 권한을 부여하거나, Cloud SQL Auth Proxy를 사용하는 방식으로 **DB 인스턴스가 외부에 노출되지 않도록** 구성하세요. 이 내용은 인프라에 속하므로 간략히 언급만 하지만, 운영 보안의 중요한 부분입니다.
17. **로그 민감정보 마스킹:** 혹시 로그인이나 기타 API에서 비밀번호 같은 민감 정보가 로그에 찍히고 있지 않은지 확인합니다. FastAPI 기본 로그는 요청 경로와 상태 정도만 남기므로 괜찮지만, `print()`로 디버깅하면서 토큰 등을 찍었다면 지워야 합니다.
18. **디버그 모드 해제:** FastAPI 앱을 생성할 때 `debug=True` 설정은 개발시에만 쓰고, 운영 배포 시에는 False로 설정합니다. Debug=True일 경우 예외 발생 시 스택트레이스를 브라우저에 노출하는데, 해커에게 내부 정보를 줄 수 있으므로 **꼭 꺼야** 합니다. Uvicorn을 실행할 때도 `--env-file` 옵션 등으로 환경을 구분하거나, APP_ENV 검사를 통해 debug 모드를 분기하면 좋습니다.
19. **모니터링 및 알림:** 보안에는 사고 대응도 포함됩니다. Cloud Run의 에러율이나 응답 시간 모니터링을 설정해 두고, 필요 시 알림(Webhook, Email 등)을 받도록 Cloud Monitoring을 활용하세요. 예를 들어 5xx 오류가 연속 발생하면 Slack으로 알림이 가도록 세팅하면 문제를 빠르게 인지할 수 있습니다.
20. **적용 및 확인:**
21. 로컬에서는 .env로 잘 동작하는지 (예: ADMIN_TOKEN 값을 .env에서 변경하면 서버 재기동 후 새로운 토큰으로 인증되는지) 확인합니다.
22. Cloud Run에 새 환경변수를 설정한 뒤 배포하고, 실제 서비스 URL에 접속하여 기능을 한 번 모두 테스트합니다. 특히 외부 API_KEY가 제대로 적용되어 요약/번역이 동작하는지, DB_URL이 올바르게 설정되어 DB 연결이 성공하는지 확인합니다. Cloud Run의 **Revisions** 페이지에서 최신 Revision의 Variables 섹션을 열어 변수들이 정확히 반영됐는지도 체크합니다.

예상 결과:
- **로컬 .env:** 개발 PC에서 `uvicorn` 또는 Docker로 서버를 띄웠을 때, `.env` 파일의 값들이 정상적으로 반영되어 서비스 작동합니다. 예를 들어 .env의 ADMIN_TOKEN을 일부러 틀리게 하고 API 호출해보면 401이 나고, 맞게 하면 200이 나오는 것으로 확인 가능하며, DB_URL을 변경하여 다른 테스트 DB를 가리키게 할 수도 있습니다.
- **Cloud Run 환경변수:** Cloud Run에 배포 후 해당 서비스 **Variables & Secrets** 설정에 ADMIN_TOKEN, API_KEY, DB_URL이 목록에 나타납니다. Cloud Run 서비스 URL에 대해 (Postman 등을 사용해) ADMIN_TOKEN을 헤더로 보낸 API 요청이 인증 성공하는지 테스트합니다. 또한, Cloud Run 로그를 통해 앱이 올바른 DB에 연결되었는지 (`Connected to db...`와 같은 로그가 있다면 확인) 파악합니다.
- **보안 강화 측면:** 외부에서 인증 없이 접속을 시도하면 모두 실패하며(allow-unauthenticated=false 설정 시 Cloud Run이 아예 401 처리, 아니면 앱 레벨에서 401), 관리용 페이지도 안전하게 보호됩니다. API 키 등의 비밀은 Git 저장

소나 클라이언트 코드에 노출되지 않고, 서버 환경변수로만 존재합니다. Google Cloud 프로젝트의 IAM을 확인했을 때 Cloud Run에 접근할 수 있는 권한은 최소한으로 설정되어 있습니다.

주의사항:
- **환경변수 변경 시 재배포:** Cloud Run에서는 환경변수를 수정하면 새로운 Revision이 생성됩니다 [10]. 만약 Cloud Console에서 수정을 했다면 수동으로 새 Revision을 트래픽에 라우팅해야 할 수 있습니다. `gcloud run deploy` 명령으로 업데이트하면 자동으로 새 Revision 활성화됩니다. 이때 짧은 순간이지만 배포 중에 무중단으로 트래픽이 이전 Revision에서 새 Revision으로 넘어갑니다.
- **민감 정보 취급:** 환경변수에 넣었다고 해도 Cloud Run 콘솔에 평문으로 값이 보입니다. 프로젝트 편집자 이상 권한을 가진 이는 그 값을 볼 수 있으므로, **조직 내 권한 관리**도 중요합니다. 앞서 언급한 Secret Manager를 쓰면 콘솔에서도 값을 숨길 수 있습니다.
- **CI 로그 유출 방지:** GitHub Actions에서 `--set-env-vars`에 값을 직접 넣으면, 만약 Actions 로그에 그 명령줄이 출력될 경우 비밀이 노출될 위험이 있습니다. 다행히 GCP CLI는 기본적으로 명령과 환경 변수를 로그에 남기지 않지만, 실수로 `echo $API_KEY` 등을 하면 안 됩니다. Actions에서 `set-env-vars`에 secrets 넣는 것은 괜찮으나, 혹시 우려되면 **deploy-cloudrun Action**을 사용하면 secrets를 직접 취급하지 않고도 배포 가능합니다.
- **백엔드 Rate Limiting:** 보안의 하나로 관리 API에 Rate Limit(분당 요청 수 제한)을 둘 필요는 보통 없지만, 악의적 공격 대비로 IP 제한이나 Rate Limit 미들웨어를 적용할 수 있습니다. FastAPI에는 직접적인 미들웨어는 없고, Cloud Run 레벨에서 IP 제한은 어렵지만 Cloud Armor 등 연계 서비스를 고려할 수 있습니다. - **정기 감사:** 환경변수 값 (특히 토큰, 키)은 주기적으로 바꾸는 것이 이상적입니다. ADMIN_TOKEN의 경우 운영자가 주기적으로 새로운 값을 설정하고 팀에 공지하여 바꾸도록 하면 유출 위험을 낮출 수 있습니다. 키 rotation에 따른 재배포도 용이하도록 CI/CD 파이프라인을 잘 준비해두세요.

배포 (CI/CD: GitHub Actions, gcloud)

목적:
마지막으로, 애플리케이션을 **Google Cloud Run**에 원활하게 배포하기 위한 CI/CD 파이프라인을 설정/확인합니다. CI/CD를 통해 코드 변경이 자동으로 컨테이너 빌드 및 배포까지 이어지도록 하거나, 수동 배포 절차(gcloud CLI 사용)를 숙지함으로써 운영 효율성을 높입니다. 이 과정에서는 **GitHub Actions**를 이용한 Cloud Run 배포를 중점으로 설명하고, 대안으로 gcloud CLI 수동 배포 방법도 함께 다룹니다.

준비 파일:
- **Dockerfile**: 프로젝트 루트에 존재 – FastAPI 앱 컨테이너화를 위한 Dockerfile. Cloud Run은 컨테이너 이미지를 실행하므로 필수입니다. - **GitHub Actions Workflow**: `.github/workflows/deploy.yml` (예시 이름) – CI 단계(테스트)와 CD 단계(이미지 빌드/배포)를 정의한 YAML. - **Google Cloud 프로젝트 정보**: GCP 프로젝트 ID, Artifact Registry 또는 Container Registry repo 정보, Cloud Run 서비스 이름, Cloud Run 실행 지역. - **인증 정보**: GitHub에 저장된 GCP 서비스계정 키(또는 OIDC Workload Identity), 또는 로컬에서 gcloud 인증된 상태 (수동 배포 시). - **배포 스크립트**: (옵션) 배포를 편하게 하는 쉘스크립트 (예: `deploy.sh`).

실행 절차:

1. **Dockerfile 검토:** Dockerfile이 최신 코드에 맞게 제대로 작성되었는지 확인합니다. 예를 들어:

```
FROM python:3.10-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
```

```
COPY . .
ENV PYTHONUNBUFFERED=1  UVICORN_WORKERS=1
EXPOSE 8000

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

위는 간단한 예시입니다. **주의**: Cloud Run이 자동으로 포트를 지정할 경우, uvicorn에서 포트를 하드코딩하지 않고 `$PORT` 환경변수를 이용하는 편이 좋습니다. 예를 들어 CMD를 `["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "${PORT}"]`로 하면 Cloud Run이 할당한 포트로 실행됩니다. (일반적으로 Cloud Run은 `$PORT` 환경변수를 8080으로 설정함) 또한, 이미지 크기를 줄이기 위해 `slim` 이미지를 사용하고, 불필요한 파일(.env, **pycache**) 등이 COPY되지 않도록 .dockerignore 설정도 확인하세요.
Dockerfile 수정 후에는 **로컬에서 빌드 및 실행 테스트**를 해보세요:

```
docker build -t qualijournal-admin:test .
docker run -d -p 8000:8000 -e ADMIN_TOKEN=devtoken -e API_KEY=devkey -e
DB_URL=sqlite:///test.db qualijournal-admin:test
```

로컬 Docker 컨테이너로 뜬 애플리케이션에 접속해 보고 (/docs 열어보기 등) 문제 없으면 넘어갑니다.

2. **GitHub Actions CI/CD 설정:**

3. **서비스계정 준비:** GitHub Actions에서 GCP에 배포하려면 권한이 필요한데, 안전한 방법은 **Workload Identity Federation (WIF)**을 이용하는 것입니다. 이 방법 설정이 복잡하다면, 서비스 계정 키(JSON)을 발급받아 GitHub Secrets에 저장하는 방법도 있습니다. (WIF는 키 없이 OIDC로 인증하며, 권장되는 현대적인 방법 [11]). 간단히 설명하면:
   - GCP에서 Deploy용 서비스 계정을 생성하고 Cloud Run 관리자 (`roles/run.admin`), Artifact Registry Writer (`roles/artifactregistry.writer`), 서비스 계정 사용자 (`roles/iam.serviceAccountUser`) 권한을 줍니다 [12].
   - WIF를 쓰는 경우 그 서비스 계정을 워크로드 풀에 연결하고, GitHub OIDC provider 설정을 해야 합니다 [13]. 이 가이드 범위를 넘어서므로, 여기서는 **JSON 키 방법** 예시를 들겠습니다 (보안상 WIF가 좋지만).
   - JSON 키를 받았다면, GitHub 리포지토리 Settings > Secrets에 `GCP_SA_KEY`라는 이름으로 키 내용을 등록합니다.
   - 또한 Secrets에 `GCP_PROJECT_ID`, `CLOUD_RUN_SERVICE` (서비스명), `CLOUD_RUN_REGION`, 그리고 앞서 언급한 ADMIN_TOKEN, API_KEY, DB_URL도 저장해 둡니다.
4. **Workflow 작성:** `.github/workflows/deploy.yml` (혹은 main.yml에 통합) 파일을 작성/수정합니다. Push 시 자동배포하려면, trigger를 `on: push` (main 브랜치에)로 설정합니다. 예:

```
name: CI-CD

on:
 push:
  branches: [ "main" ]

jobs:
 build-and-deploy:
  runs-on: ubuntu-latest
```

```
  steps:
   - name: Checkout code
     uses: actions/checkout@v3

   - name: Set up Cloud SDK
     uses: google-github-actions/setup-gcloud@v1
     with:
       service_account_key: ${{ secrets.GCP_SA_KEY }}
       project_id: ${{ secrets.GCP_PROJECT_ID }}

   - name: Build and push to Artifact Registry
     run: |
       gcloud auth configure-docker ${GCP_ARTIFACT_REGISTRY_HOST} -q
       docker build -t ${GCP_ARTIFACT_REGISTRY_HOST}/${{ secrets.GCP_PROJECT_ID }}/
qualijournal-admin:${{ github.sha }} .
       docker push ${GCP_ARTIFACT_REGISTRY_HOST}/${{ secrets.GCP_PROJECT_ID }}/
qualijournal-admin:${{ github.sha }}

   - name: Deploy to Cloud Run
     run: |
       gcloud run deploy ${{ secrets.CLOUD_RUN_SERVICE }} \
         --image ${GCP_ARTIFACT_REGISTRY_HOST}/${{ secrets.GCP_PROJECT_ID }}/
qualijournal-admin:${{ github.sha }} \
         --region ${{ secrets.CLOUD_RUN_REGION }} --platform managed \
         --set-env-vars ADMIN_TOKEN=${{ secrets.ADMIN_TOKEN }},API_KEY=$
{{ secrets.API_KEY }},DB_URL=${{ secrets.DB_URL }} \
         --allow-unauthenticated=false
```

위에서 `${GCP_ARTIFACT_REGISTRY_HOST}`는 Artifact Registry 도메인 (예: `asia-northeast3-docker.pkg.dev` 등)이고, Cloud Run 서비스명/지역은 secrets에서 불러옵니다. `--allow-unauthenticated=false`는 Cloud Run이 Cloud IAM 인증을 요구하게 하지만, 우리 앱은 자체 토큰인증을 하므로 사실 필수는 아닙니다. (false로 하면 외부에서 토큰이 있어도 Cloud Run 레벨에서 401이 될 수 있는데, 이 경우 Cloud Run IAM에 allUsers 권한을 주지 않고도 보호할 수 있으나, 서비스계정 인증이나 Cloud IAM OIDC 토큰을 써야 해서, ADMIN_TOKEN 인증과 이중이 되어버림. **권장**: allow-unauthenticated를 **true**로 두고 앱 레벨에서만 인증 체크를 하는 것이 일반적입니다. 여기서는 false로 두되, 실제 운용을 고려해 결정하세요.)

5. **단계 설명:** checkout -> gcloud 세팅(서비스계정 인증) -> docker build & push -> gcloud run deploy. 이 pipeline이 실행되면 수 분 내에 Cloud Run에 새 revision이 배포됩니다.
6. **테스트 단계 포함:** 위 예에서는 CI(테스트) 생략했지만, 실무에서는 steps 맨 앞에 `- name: Run tests; run: pytest` 같은 단계를 추가하여 **테스트 통과 시에만 배포**하게 하는 것이 좋습니다. 또는 `if: success()` 조건으로 분기할 수도 있습니다.

7. **Workflow 실행 확인:** 새로운 커밋을 main 브랜치에 푸시하고, GitHub Actions 탭에서 워크플로우가 정상 진행되는지 모니터링합니다. 만약 실패하면 로그를 보고 수정합니다 (예: gcloud 인증 실패, 권한 부족 등 해결).

8. **gcloud CLI를 통한 수동 배포 (대안):** CI/CD를 갖추기 전이거나, 긴급 수정시 로컬에서 배포해야 할 때를 대비해 수동 방법도 알아둡니다.

9. **이미지 빌드/푸시:** GCP Artifact Registry를 사용한다고 가정하면, 먼저 로컬 터미널에서:

```
gcloud auth login # GCP 인증 (또는 이미 로그인 상태이면 생략)
gcloud config set project <YOUR_PROJECT_ID>
# Artifact Registry에 Docker 인증
gcloud auth configure-docker ${REGION}-docker.pkg.dev
# 도커 빌드
docker build -t ${REGION}-docker.pkg.dev/<YOUR_PROJECT_ID>/<REPO_NAME>/
qualiadmin:latest .
docker push ${REGION}-docker.pkg.dev/<YOUR_PROJECT_ID>/<REPO_NAME>/
qualiadmin:latest
```

Artifact Registry의 REPO_NAME은 미리 콘솔에서 만들었어야 합니다. (`gcloud artifacts repositories create` 명령으로 생성 가능).

10. **Cloud Run 배포:** 이미지를 push했다면:

```
gcloud run deploy qualiadmin \
 --image ${REGION}-docker.pkg.dev/<YOUR_PROJECT_ID>/<REPO_NAME>/
qualiadmin:latest \
 --platform managed --region ${REGION} \
 --set-env-vars ADMIN_TOKEN=<token>,API_KEY=<key>,DB_URL=<url> \
 --allow-unauthenticated
```

위에서 필요한 변수와 옵션을 적절히 채웁니다. 혹은, 이전 Revision에서 설정한 env를 유지하려면 `--update-env-vars` 옵션으로 특정 키만 업데이트할 수도 있습니다. Cloud Run 서비스와 이미지를 연결해 주면 몇십 초 내로 새 Revision이 뜹니다.

11. **배포 확인:** `gcloud run services describe qualiadmin --region ${REGION}` 명령으로 현재 활성 Revision과 환경변수를 확인합니다. 또는 Cloud Run 웹 콘솔에서 배포가 반영되었는지 UI로 봅니다.

12. **배포 결과 테스트:** 새로운 기능이 포함된 버전이 Cloud Run URL에서 정상 동작하는지 최종 점검합니다. 예를 들어:

13. Cloud Run URL (예: https://qualiadmin-xxxxxx-uc.a.run.app)에 웹 브라우저로 접속해 관리자 UI 페이지를 띄워봅니다. (만약 frontend가 별도 호스팅이 아니라 backend에서 제공된다면, static files 세팅 필요 등 확인).
14. API 테스트: Postman으로 Cloud Run URL의 `/api/report`를 호출해보고 200 응답 받기 (Authorization 헤더에 ADMIN_TOKEN 포함).
15. UI에서 Export 버튼 눌러보기 등 모든 통합 기능을 실제 Cloud 환경에서 한번 실행해 봅니다.
16. 로그 확인: `gcloud logs read --project=<ID> --service=qualiadmin --region=${REGION}` 명령으로 Cloud Run 로그를 Tail 해보거나, 콘솔 Logs 화면에서 ERROR 레벨 로그가 없는지 확인합니다.

예상 결과:
- **CI/CD 파이프라인:** 새로운 커밋이 발생할 때마다 GitHub Actions Workflow가 실행되고, 성공적으로 Cloud Run에 배포 완료됩니다. Actions 로그 마지막에 `Deployed service [qualiadmin] to [https://qualiadmin-<hash>-<region>.a.run.app]` 등의 메시지가 보여집니다. (또는 deploy-cloudrun action을 썼다면 output으로 URL을 주기도 합니다 [14].)
- **Cloud Run 서비스:** Cloud Run 콘솔에 들어가 보면 Revisions 탭에 최근 배포 시각의 revision이 있고, 트래픽 100%가 할당되어 있습니다. 서비스 URL을 통해 새 기능들이 반영된 것을 확인합니다. 예를 들어, 이전에는 일일 보고

서 버튼이 작동안했는데 이제 동작하고 API 결과도 제대로 표시됩니다.
- **배포 시간:** GitHub Actions를 통한 전체 빌드+배포 시간이 수분 내로 (보통 5~7분) 완료됩니다 [15]. 추후 최적화를 통해 3분 내외로 단축할 수도 있지만, 현재 pipeline도 무난한 편입니다.

주의사항:
- **배포 트러블슈팅:** 만약 Actions에서 배포가 실패하면, 오류 내용을 면밀히 살펴야 합니다. 예를 들어 인증 오류 시 서비스계정 권한 재검토, Docker build 오류 시 Dockerfile 수정, `--allow-unauthenticated` 관련 경고 시 플래그 조정 등을 합니다.
- **Cloud Run 서비스 이름 주의:** Cloud Run 서비스명을 변경하면 URL도 달라지고, IAM 권한도 새로 설정해야 합니다. 이 가이드에서는 기존 서비스 업데이트를 다뤘으므로, 새로 서비스 생성하는 경우 `--allow-unauthenticated`를 true로 해야 웹에서 바로 접속 가능합니다 (false면 Cloud Run IAM에 사용자 추가하거나 ID token을 받아 호출해야 함). 개발 편의상 Admin 시스템이지만, 내부용이라면 false로 두고 사전에 정해둔 Google 계정만Invoker로 추가하는 방법도 있습니다.
- **GitHub Actions 보안:** Actions YAML에 secrets를 삽입할 때 **주의**: push나 run 명령에서 secret값이 노출되지 않도록 해야 합니다. GitHub Actions 자동 마스킹이 기본 제공되지만, 실수로 `echo $ADMIN_TOKEN` 같은 걸 하면 로그에 나오니 조심합니다. 또한 PR에서 비밀 유출되지 않도록 워크플로우 권한을 설정(예: fork된 리포지토리의 PR에서는 배포 job이 돌지 않게)해야 합니다.
- **빌드 최적화:** 앞서 Reddit 사례에서도 나오듯이, Cloud Build (gcloud builds submit)나 Kaniko 캐시 등을 활용하면 CI 시간을 줄일 수 있습니다 [16] [17]. 현재 pipeline에서는 Docker 레이어 캐시가 매번 없어서 느릴 수 있는데, Actions의 cache액션을 써서 `~/.cache`나 `docker layer cache`를 보존하도록 개선 가능입니다. 다만, 초심자에게는 복잡할 수 있으므로 추후 개선 과제로 둡니다.
- **모니터링:** 배포 후에도 CI/CD 파이프라인이 잘 동작하는지 주기적으로 모니터링하세요. GitHub Actions 실패 알림을 Slack이나 이메일로 받을 수 있으면 유용합니다.
- **버전 태그:** 현재는 `${{ github.sha }}` (커밋 해시)로 이미지를 태그했지만, `:latest`로 덮어쓰는 전략도 있습니다. 다만 해시 태그는 이전 버전 롤백이 쉽고, 추적이 명확하다는 장점이 있습니다. Production에는 태그를 `v1.2.3` 같은 버전으로 관리하는 것도 방법입니다.
- **종료:** 불필요한 과거 리비전이 많이 쌓이면 비용이 발생할 수 있습니다. Cloud Run은 기본적으로 최근 100개까지 리비전을 보관합니다. `gcloud run revisions list`로 확인하고, 오래된 Revision은 트래픽 0%여도 완전히 삭제되지 않을 수 있으니 `gcloud run revisions delete`로 지울 수 있습니다. 또는 Cloud Run 설정에서 "Revision history limit"을 설정하세요.
- **마무리 테스트:** 배포까지 모두 끝났다면, 프로젝트 전반에 걸친 모든 기능을 실제 운영 URL에서 한번 더 종합적으로 테스트하는 것을 권장합니다. 그리고 안정판으로 간주되면 태그를 찍어 Release를 만들고, 이후 변경사항이 있으면 버전업을 해가는 형식으로 프로젝트를 관리하면 좋습니다.

---

[1] 영문 뉴스기사 크롤링 후 ChatGPT(OpenAI API) 기사 요약 및 한글 번역하기 - 테디노트
https://teddylee777.github.io/python/news-article/

[2] [3] Testing - FastAPI
https://fastapi.tiangolo.com/tutorial/testing/

[4] [5] [8] [9] [10] Configure environment variables for services | Cloud Run | Google Cloud
https://cloud.google.com/run/docs/configuring/services/environment-variables

[6] [Google Cloud] Cloud Run 서비스 간 (인터넷 구간) 통신하기 | by Kiwon Lee | google-cloud-apac | Medium
https://medium.com/google-cloud-apac/google-cloud-cloud-run-%EC%84%9C%EB%B9%84%EC%8A%A4-%EA%B0%84%EC%97%90-%EC%9D%B8%ED%84%B0%EB%84%B7-%EA%B5%AC%EA%B0%84%EC%9C%BC%EB%A1%9C-%ED%86%B5-%EC%8B%A0%ED%95%98%EA%B8%B0-e5833f5edacb

[7] 작업 환경 변수 구성 | Cloud Run Documentation | Google Cloud
https://cloud.google.com/run/docs/configuring/jobs/environment-variables?hl=ko

11 16 17 Github actions가 google cloud run에 배포하는데 7분 정도 걸리는데, 이거 정상인가? : r/devops
https://www.reddit.com/r/devops/comments/1fthk0i/github_actions_to_google_cloud_run_takes_about_7/?tl=ko

12 13 Deploy to Cloud Run with GitHub Actions | Google Cloud Blog
https://cloud.google.com/blog/products/devops-sre/deploy-to-cloud-run-with-github-actions/

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

15 How to deploy Cloud Run services with GitHub Actions - YouTube
https://www.youtube.com/watch?v=DMCi7WWTtX0