ChatGPT

QualiJournal 관리자 시스템 Cloud Run 배포 및 운영 가이드

1. 완료된 작업 (Completed Tasks)

[DEV] KPI 자동 갱신 구현

- **개발 내용:** 관리자 대시보드에서 주요 지표(KPI)가 수동 새로고침 없이 주기적으로 갱신되도록 기능을 구현했습니다. 이는 백엔드에서 일정 간격마다 KPI 데이터를 다시 계산하거나, 프론트엔드에서 주기적으로 API를 호출하는 방식으로 처리됩니다. 예를 들어 **일일 발행 건수**, **승인/보류 기사 수** 등의 지표가 자동으로 업데이트됩니다.
- **운영 확인:** 운영자는 대시보드에 접속하여 일정 시간 경과 후 KPI 값이 변동하는지 확인함으로써 자동 갱신 기능이 정상 동작함을 검증할 수 있습니다. 또한 필요한 경우 수동으로 갱신 트리거를 제공하여 즉시 KPI를 다시 불러올 수 있도록 했습니다.
- **유의 사항:** KPI 자동 갱신 주기는 환경 변수로 설정되며(예: `KPI_REFRESH_INTERVAL=300`으로 5분 간격 설정), 운영 환경에서는 해당 변수 값을 조절하여 성능과 최신성 사이의 균형을 맞출 수 있습니다. (실제 환경 변수 값은 GUID 형식의 예시로 대체)

[DEV] 슬라이더 임계값 설정 기능

- **개발 내용:** 커뮤니티 등 **콘텐츠 점수 임계값(score_threshold)**을 관리자 UI에서 슬라이더로 조정할 수 있게 개선했습니다. 이를 통해 편집자는 코드 수정 없이도 필터링 임계값을 즉시 변경하여 콘텐츠 품질 기준을 높이거나 낮출 수 있습니다 [1]. 예를 들어 점수 임계값을 3.5에서 4.0으로 올리면, 해당 점수 미만의 커뮤니티 글은 자동으로 리스트에서 제외됩니다.
- **운영 사용:** 운영자(편집자)는 관리자 화면의 설정 섹션에서 슬라이더 UI를 움직여 임계값을 변경할 수 있습니다. 슬라이더 옆에는 현재 설정된 값이 표시되어 즉각 확인 가능합니다. 변경 사항은 실시간으로 백엔드 `config.json` 또는 데이터베이스 설정에 반영되어 이후 수집된 콘텐츠에 적용됩니다 [1].
- **유의 사항:** 너무 자주 임계값을 조정하면 하루 기준이 일관되지 않을 수 있으므로 가급적 발행 주기(일일) 단위로 조정하고 추이를 관찰합니다. 슬라이더 기본 범위는 0~5 사이로 설정되어 있으며, 필요 시 코드에서 범위를 조정할 수 있습니다.

[DEV/OPS] Cloud Run 배포 완료

- **개발 내용:** QualiJournal 관리자 애플리케이션을 Docker 컨테이너로 패키징하여 **Google Cloud Run**에 배포했습니다. Cloud Run은 요청 시 컨테이너를 실행하는 서버리스 플랫폼으로, 높은 수준의 확장성과 관리형 인프라를 제공합니다 [2]. 배포 과정에서 Docker 이미지를 빌드하고, Google Artifact Registry에 이미지를 저장한 후 Cloud Run 서비스를 생성했습니다.
- **배포 절차:** (로컬 PowerShell 예시 기준)

```
PS C:\Users\user\Desktop\퀄리저널> gcloud builds submit --tag asia-northeast1-docker.pkg.dev/<PROJECT_ID>/qualijournal/admin:v1 .
# 소스 디렉토리의 코드를 빌드하여 Artifact Registry에 v1 태그로 이미지 업로드
# 예상 출력: SUCCESS - Image uploaded to [asia-northeast1-docker.pkg.dev/.../admin:v1]

PS C:\Users\user\Desktop\퀄리저널> gcloud run deploy QualiJournalAdmin `
    --image asia-northeast1-docker.pkg.dev/<PROJECT_ID>/qualijournal/admin:v1 `
```

```
    --region asia-northeast1 --platform managed --allow-unauthenticated
# Cloud Run 서비스 생성 및 이미지 배포
# 예상 출력:  Deploying... done. Service [QualiJournalAdmin] deployed with URL: https://qualijournaladmin-...a.run.app
```

위 예시에서는 `<PROJECT_ID>`와 서비스 이름 등을 실제 값으로 교체해야 합니다. `--allow-unauthenticated` 옵션을 지정하면 별도 인증 없이도 접근 가능하나, 관리자 시스템에는 자체 인증 토큰이 적용되므로 외부 노출에 주의해야 합니다.

- **운영 확인:** 배포 완료 후 Cloud Run 콘솔 또는 CLI로 서비스 상태를 확인합니다:

```
PS C:\Users\user\Desktop\퀄리저널> gcloud run services describe QualiJournalAdmin --region asia-northeast1
# 출력에서 URL 및 최근 배포 시간이 표시되며, 원하는 경우 리비전별 환경 변수 확인 가능
```

Cloud Run 서비스 URL(예: `https://qualijournaladmin-<hash>-asia-northeast1.run.app`)을 브라우저에서 열어 관리자 로그인 페이지 또는 대시보드 화면이 뜨는지 확인합니다. **스모크 테스트**를 통해 핵심 기능 (로그인, 데이터 조회 등)이 정상 동작하면 배포가 성공적으로 완료된 것입니다.

[DEV/OPS] 토큰 기반 인증 적용

- **개발 내용:** 관리자 시스템 보안을 강화하기 위해 **토큰 기반 인증**을 도입했습니다. 사용자가 로그인하면 JWT(JSON Web Token)와 같은 **액세스 토큰**이 발급되며, 이후 요청에는 해당 토큰을 포함하여 인증을 수행합니다. 백엔드에서는 토큰을 검증하여 유효한 관리자만 접근할 수 있게 합니다.
  이를 구현하기 위해 서버 측에 토큰 서명용 비밀키(`ADMIN_JWT_SECRET`)를 설정했고, 클라이언트(프론트엔드)는 로그인 시 획득한 토큰을 로컬 스토리지나 쿠키에 보관합니다. 모든 후속 API 호출 시 HTTP Header의 `Authorization: Bearer <TOKEN>` 형태로 토큰을 포함하여 보내도록 수정되었습니다.
- **환경 변수 설정:** 실제 비밀키나 API 키 등 민감한 값은 소스 코드에 직접 두지 않고, Cloud Run 서비스의 **환경 변수** 또는 **Secret Manager**를 통해 주입했습니다. 예를 들어 OpenAI API 키가 필요하다면 환경 변수 `OPENAI_API_KEY=123e4567-e89b-12d3-a456-426614174000` (예시 GUID)로 설정하고 코드에서는 `os.getenv("OPENAI_API_KEY")`로 불러쓰도록 구현했습니다. Cloud Run에 환경 변수를 설정하려면 다음과 같은 명령을 사용할 수 있습니다:

```
PS C:\Users\user\Desktop\퀄리저널> gcloud run services update QualiJournalAdmin `
    --update-env-vars "ADMIN_JWT_SECRET=00000000-0000-0000-0000-000000000000"
# (예시) JWT 서명용 시크릿 키를 안전하게 설정
```

  설정 후 **Cloud Run 재배포**가 트리거되며 새로운 환경 변수가 적용됩니다.
- **운영 영향:** 운영자는 이제 관리자 시스템에 접속할 때 최초 **로그인 과정**을 거쳐 토큰을 발급받아야 합니다. JWT 기반 인증 덕분에 세션 관리가 불필요하며, 토큰의 만료 시간 정책으로 추가적인 보안 설정이 가능합니다. 만약 토큰 유효기간이 만료되면 사용자는 재로그인해야 하며, 이 과정에서 Refresh 토큰 전략이 있다면 자동 연장도 가능합니다.
- **주의:** 발급된 토큰은 절대 노출되거나 Git 등에 커밋되지 않도록 해야 합니다. 또한, 토큰 검증 시 **발급자(issuer)**와 **만료(expiration)**를 반드시 체크하여 만료된 토큰이나 위조된 토큰이 통과되지 않도록 해야 합니다.

[OPS] 스모크 테스트 완료 및 결과

- **테스트 내용:** Cloud Run 배포와 신규 기능 적용 후 주요 기능에 대한 **스모크 테스트**를 진행했습니다. 스모크 테스트는 시스템의 가장 기본적이고 중요한 기능들이 제대로 동작하는지 확인하는 절차입니다. 로그인, KPI 표시, 기사 수집 및 승인, 발행 미리보기 등의 시나리오를 간략히 테스트했습니다.
- **수행 방법:** 운영자는 배포된 **QualiJournalAdmin** 서비스 URL에 접속하여 다음 항목들을 점검했습니다:
- **로그인 페이지 로드:** 로그인 화면/UI가 정상 표시되고, 등록된 관리자 계정으로 로그인 시 대시보드로 이동되는지 확인.
- **KPI 표시:** 대시보드에 오늘의 발행 키워드 수, 수집된 기사 수, 승인된 기사 수 등의 KPI가 자동 표시되고 일정 시간 후 갱신되는지 확인.
- **슬라이더 조작:** 설정 페이지에서 임계값 조절 슬라이더를 움직였을 때 즉시 화면에 반영되고, 새로운 값으로 설정이 저장되는지 확인.
- **기사 수집/승인 플로우:** 테스트 키워드를 입력하여 기사 수집 → 승인 → 발행 미리보기까지의 간이 흐름을 실행해 보고, 오류 없이 동작하는지 확인.
- **결과:** 위 핵심 기능들이 모두 **정상 동작**함을 확인했습니다. 특히, Cloud Run 환경에서 예상치 못한 권한 또는 경로 이슈는 발견되지 않았으며, 토큰 인증을 포함한 보안 기능도 의도대로 작동했습니다. 스모크 테스트 중 발견된 경미한 UI 레이아웃 문제는 추후 테마 점검 단계에서 수정될 예정입니다.

2. 현재 문제 (WIF 인증 오류 및 해결) [DEV]

Workload Identity Federation (WIF) 이슈 개요

현재 CI/CD 파이프라인(예: GitHub Actions)을 통해 Cloud Run에 배포할 때 **Workload Identity Federation**(WIF) 방식의 인증을 사용하고 있습니다. WIF는 외부 ID 제공자(예: GitHub)로부터 OIDC 토큰을 받아 GCP 서비스 계정으로 교환함으로써, **서비스 계정 키 없이** 안전하게 배포 권한을 부여하는 방법입니다 [3] [4].
문제가 되는 오류 메시지 예시는 다음과 같습니다:

```
ERROR: (gcloud.run.deploy) PERMISSION_DENIED: The caller does not have
permission.
This command is authenticated as principal://iam.googleapis.com/projects/
PROJECT_NUMBER/locations/global/workloadIdentityPools/POOL_ID/subject/
repo:org/repo:ref:refs/heads/main ...
```

위 오류는 GitHub Actions 워크플로우에서 WIF로 발급된 임시 자격으로 `gcloud run deploy`를 실행할 때 발생한 것입니다. `principal://.../subject/repo:...` 형식의 주체로 인증되었으나, **배포 권한이 충분하지 않아** 거부된 것을 나타냅니다 [5]. 특히 추가 로그에서는 **Cloud Storage 버킷에 대한 접근 거부 (storage.buckets.get)**, 또는 **Artifact Registry 권한 부족** 등의 메시지도 포함되었습니다 [6]. 이는 Cloud Run 소스 배포 시 내부적으로 **소스 스테이징 버킷**과 **빌드 이미지 레지스트리**에 접근하기 때문입니다.

원인 분석

WIF를 통해 외부 주체(예: GitHub Action)가 GCP 리소스에 접근하려면, 해당 주체에 대응되는 **Workload Identity Pool** 및 **Provider**가 구성되어 있어야 합니다. 현재 GitHub OIDC Provider는 설정되었고, GitHub 저장소에서 워크플로우가 OIDC 토큰을 받아오는 것까지는 성공한 상황입니다. 그러나, **권한 부여(IAM)** 단계에서 누락이 있어 오류가 발생했습니다 [7]. 즉, WIF로 인증된 주체(또는 그가 가장할 서비스 계정)에 Cloud Run 배포 및 관련 리소스 접근 권한이 충분히 주어지지 않았습니다.

해결 방안: IAM 권한 설정

1. **배포용 서비스 계정 확인 또는 생성:** 가능한 권장 방식은 GitHub Actions 전용의 GCP **서비스 계정** (예: `gha-deploy-sa`)을 만들고, GitHub WIF 주체가 이 서비스 계정을 **가장(impersonate)**하도록 구성하는 것입니다. 해당 서비스 계정이 없다면 생성합니다:

```
PS C:\> gcloud iam service-accounts create gha-deploy-sa --display-name "GitHub Deploy SA"
# 예상 출력: Created service account [gha-deploy-sa].
```

2. **WIF 풀 구성 및 바인딩:** 앞서 만든 서비스 계정에 GitHub OIDC 주체를 연결합니다. `principalSet:` URI는 WIF Pool ID와 Provider, 그리고 조건으로 GitHub 저장소를 지정합니다. 예를 들어 main 브랜치 배포만 허용:

```
PS C:\> gcloud iam service-accounts add-iam-policy-binding gha-deploy-sa@<PROJECT_ID>.iam.gserviceaccount.com `
    --role roles/iam.workloadIdentityUser `
    --member "principalSet://iam.googleapis.com/projects/<PROJECT_NUMBER>/locations/global/workloadIdentityPools/<POOL_NAME>/providers/<PROVIDER_NAME>/attribute.repository/<ORG>/<REPO>:ref:refs/heads/main"
# 예상 출력: Updated IAM policy for service account [gha-deploy-sa@...].
```

   위 명령으로 특정 GitHub 리포지토리의 main 브랜치에서 오는 OIDC 토큰에 대해 해당 서비스 계정을 **사용할 권한**(Workload Identity User)을 부여합니다.
3. **필수 IAM 역할 부여:** 이제 실제 배포 작업에 필요한 권한들을 서비스 계정 `gha-deploy-sa`에 할당합니다. Cloud Run 배포에는 다음 권한(역할)이 필요합니다 [8] [9]:
4. Cloud Run 관리자 권한: `roles/run.admin` – Cloud Run 서비스 생성/업데이트 권한
5. Cloud Run 런타임 서비스계정 사용자: `roles/iam.serviceAccountUser` – (만약 Cloud Run 서비스에 별도 런타임 SA를 지정했다면) 해당 SA로 Cloud Run을 실행할 수 있는 권한
6. Cloud Build 빌드 실행 권한: `roles/cloudbuild.builds.editor` – `gcloud run deploy --source` 시 Cloud Build를 사용하여 빌드
7. Artifact Registry 푸시 권한: `roles/artifactregistry.writer` (또는 admin) – 컨테이너 이미지를 Artifact Registry에 저장
8. Cloud Storage 접근 권한: `roles/storage.objectAdmin` – 빌드 과정의 임시 버킷(예: `run-sources-*`)에 대한 읽기/쓰기
   이러한 역할은 최소 권한 원칙하에 서비스 계정에 바인딩합니다:

```
PS C:\> gcloud projects add-iam-policy-binding <PROJECT_ID> --member "serviceAccount:gha-deploy-sa@<PROJECT_ID>.iam.gserviceaccount.com" --role "roles/run.admin"
PS C:\> gcloud projects add-iam-policy-binding <PROJECT_ID> --member "serviceAccount:gha-deploy-sa@<PROJECT_ID>.iam.gserviceaccount.com" --role "roles/cloudbuild.builds.editor"
PS C:\> gcloud projects add-iam-policy-binding <PROJECT_ID> --member "serviceAccount:gha-deploy-sa@<PROJECT_ID>.iam.gserviceaccount.com" --role "roles/storage.objectAdmin"
... (필요한 모든 역할에 대해 반복)
# 각 명령 실행 시 'Updated IAM policy for project' 메시지 확인
```

참고: Artifact Registry 권한은 리포지토리 단위로 부여할 수도 있습니다. 예를 들어 특정 AR 리포지토리 경로에 writer 권한을 주어 과도한 권한 부여를 피할 수 있습니다 [10] .

9. **GitHub Actions 워크플로우 수정:** GitHub Actions에서 `google-github-actions/auth` 사용 시, 위에서 만든 서비스 계정으로 **가장하도록 설정**합니다. 예를 들어:

```
- name: Authenticate to GCP
  uses: google-github-actions/auth@v2
  with:
    workload_identity_provider: projects/<PROJECT_NUMBER>/locations/global/
workloadIdentityPools/<POOL_NAME>/providers/<PROVIDER_NAME>
    service_account: gha-deploy-sa@<PROJECT_ID>.iam.gserviceaccount.com
```

   이렇게 하면 GitHub Action이 OIDC 토큰으로 `gha-deploy-sa` 서비스계정의 단기 크레덴셜을 얻어 사용하게 됩니다. 이후 `google-github-actions/deploy-cloudrun@v2` 스텝에서 자동으로 해당 인증이 활용되어 배포가 진행됩니다.

10. **재시도 및 검증:** 설정 변경 후 GitHub Actions 파이프라인을 재실행합니다. 여전히 실패한다면 오류 메시지를 확인하여 **누락된 권한**이 없는지 점검합니다. 예컨대 `Permission 'run.routes.invoke' denied`가 나오면 Cloud Run 서비스를 **호출**하는 권한(Invoker)이 필요함을 의미합니다. 배포 과정에는 필요 없지만, 만약 파이프라인에서 Cloud Run을 트리거하거나 검증 호출을 한다면 `roles/run.invoker`를 추가로 줄 수 있습니다.

위 설정을 완료하면 WIF를 통한 CI/CD 인증이 올바르게 작동하여, GitHub Actions에서 **서비스계정 키 없이도** Cloud Run 배포를 자동화할 수 있습니다 [7] . 정리하면, WIF 이슈 해결의 핵심은 **옳은 주체에 올바른 IAM 권한을 부여**하는 것입니다. 특히 Cloud Run 도메인 매핑 시 storage.buckets.get 에러 등은 `Storage Object Admin` 권한 부여로 해결되었고, 최종적으로 배포 파이프라인이 **Permission Denied 없이 성공**하도록 조치했습니다.

3. 남은 작업 (Next Tasks)

3.1 [OPS] 커스텀 도메인 연결 (Custom Domain Mapping)

현재 Cloud Run 서비스는 기본 제공 도메인(`*.run.app`)으로 운영되고 있습니다. 사용자 친화적 URL 제공과 브랜딩을 위해 **커스텀 도메인**을 연결할 예정입니다. Cloud Run은 자체적으로 커스텀 도메인 매핑 기능을 지원하며, 이를 통해 예를 들어 `admin.qualijournal.com`과 같은 도메인을 서비스에 매핑할 수 있습니다. **작업 순서**는 다음과 같습니다:

1. **도메인 소유권 확인:** 처음 사용하는 커스텀 도메인은 프로젝트에 소유권이 인증되어 있어야 합니다.

```
PS C:\> gcloud domains list-user-verified
# 이미 인증된 도메인 목록 조회. 만약 `qualijournal.com`이 없다면 아래 명령으로 인증 시작.

PS C:\> gcloud domains verify qualijournal.com
# Search Console을 통해 도메인 소유권 인증 절차를 시작 (브라우저 또는 출력 안내에 따라 DNS TXT 레코드 추가 등 진행).
# 예상 출력: 다음 URL에서 도메인 소유권을 확인하세요: https://search.google.com/search-console/...
```

   이 과정에서는 Google Search Console을 통해 해당 도메인의 **TXT 레코드**를 등록하는 등의 절차를 거쳐야 합니다. 도메인 등록업체의 DNS 설정에 안내된 TXT 레코드를 추가한 뒤, 소유권이 확인되면 콘솔에 반영됩니다

[11] [12].
Tip: `gcloud domains verify` 명령 출력의 안내에 따라 Search Console에서 확인을 완료하면, `gcloud domains list-user-verified`로 `Verified` 상태를 확인할 수 있습니다.

2. **도메인 매핑 생성:** 도메인 인증이 완료되면 Cloud Run 서비스에 도메인을 매핑합니다. 현재 이 기능은 gcloud **beta** 명령으로 제공됩니다 [13].

```
PS C:\> gcloud beta run domain-mappings create --service QualiJournalAdmin --domain admin.qualijournal.com --region asia-northeast1
# 커스텀 도메인 admin.qualijournal.com을 서비스 QualiJournalAdmin에 연결 (asia-northeast1 리전 기준).
# 예상 출력: Creating...done. Created [admin.qualijournal.com] -> [QualiJournalAdmin]
#         resourceRecords:
#         - name: admin
#           rrdata: ghs.googlehosted.com.
#           type: CNAME
```

위 출력 예시에서 보이듯이, Cloud Run은 SSL 인증서 발급을 자동으로 시도하며, 도메인 매핑 객체에 필요한 DNS 레코드 정보가 포함됩니다. `resourceRecords` 필드에서 `name`이 `admin`이고 `rrdata`가 `ghs.googlehosted.com.`인 **CNAME 레코드**가 제시되는데 [14], 이는 `admin.qualijournal.com` 도메인을 가리키도록 설정해야 하는 대상입니다.

3. **DNS 레코드 설정:** 도메인 등록처의 DNS 설정에 접속하여 **CNAME 레코드**를 추가합니다.

4. 호스트(Name): `admin` (또는 전체 도메인을 지정하는 경우 빈 값/@)

5. 값(Data): `ghs.googlehosted.com`
Cloud Run 도메인 매핑의 경우 Google이 관리하는 부하분산기의 주소로 트래픽을 안내하기 위해 위와 같은 CNAME을 사용합니다 [14]. DNS에 레코드를 추가한 후 전파되기까지 수분 ~ 수시간 소요될 수 있습니다. 전파 여부는 `nslookup admin.qualijournal.com` 등으로 확인 가능합니다.

6. **SSL 인증서 발급 대기:** Cloud Run은 도메인 매핑 생성 후 LetsEncrypt 기반의 **HTTPS 인증서**를 자동 발급합니다. 일반적으로 약 15분 정도 소요되며, 최대 24시간까지도 걸릴 수 있습니다 [15]. 이 기간 동안 Cloud Run 콘솔의 도메인 매핑 상태가 "Provisioning Certificate"로 표시됩니다.

```
PS C:\> gcloud beta run domain-mappings describe --domain admin.qualijournal.com --region asia-northeast1
# status 필드에서 certificate 상태 확인 (Provisioning 중 / Active 등).
```

인증서 발급이 완료되면 Cloud Run 서비스는 커스텀 도메인에 대해 **HTTPS 트래픽을 수신**하기 시작하며, 콘솔에 녹색 자물쇠 아이콘이 표시됩니다.

7. **검증:** 브라우저에서 `https://admin.qualijournal.com`으로 접속하여 관리자 시스템이 정상 열리는지 확인합니다. 만약 인증서 오류가 발생한다면 DNS 설정이 올바른지, 인증서 발급 상태를 재확인해야 합니다. Cloud Run의 커스텀 도메인은 자동으로 HTTPS만 허용하므로, 별도의 SSL 세팅은 필요 없지만, **DNS 제공자에 따라** CNAME 대신 A 레코드 설정이 필요할 수 있습니다 (Console에 안내된 경우 따라 설정). 일반적인 경우 위 CNAME 설정으로 충분합니다.

> **Note:** Cloud Run의 도메인 매핑 기능은 현재 프리뷰 단계로, 일부 제약(지연 발생 가능, 특정 리전 제한 등)이 있습니다 [16] [17]. 트래픽량이 많거나 커스텀 도메인에 대한 세밀한 제어가 필요하면 **Cloud Load Balancing**이나 **Firebase Hosting**을 프론트로 사용하는 것도 고려 가능합니다 [18] [19]. 그러나 소규모 서비스의 경우 Cloud Run 자체 매핑으로도 충분하며, Google이 **인증서 자동 갱신**까지 관리해주므로 편리합니다.

3.2 [DEV/OPS] Cloud Scheduler를 통한 자동화 (일정 작업 스케줄링)

QualiJournal 시스템의 **일일 발행 파이프라인**을 완전 자동화하기 위해 Cloud Scheduler를 도입합니다. 기존에는 Windows 작업 스케줄러 등을 통해 하루 한 번 스크립트를 실행했지만 [20], Cloud Scheduler를 사용하면 GCP 환경에서 정해진 크론(cron) 표현식으로 HTTP 요청이나 Cloud Pub/Sub 메시지를 보낼 수 있습니다. 여기서는 **매일 오전 9시**에 Cloud Run의 특정 엔드포인트를 호출하여 기사 수집→발행을 트리거한다고 가정하겠습니다.

준비 1: Cloud Run 엔드포인트 준비 [DEV]
백엔드에 스케줄러가 호출할 **API 엔드포인트**를 마련해야 합니다. 예를 들어 `POST /api/publish-today` 또는 특정 키워드를 지정할 수 있다면 `POST /api/publish/오늘의키워드` 등의 REST API를 만들어 두었습니다 [21]. 이 엔드포인트는 적절한 인증을 요구하도록 하며(내부 토큰 검증 등), 호출 시 당일 발행 프로세스를 비동기로 시작하도록 구현합니다. 이미 `orchestrator.py` 기반으로 일련의 과정을 실행하는 기능이 있으므로, 이를 웹 요청으로 트리거할 수 있게 래핑한 것입니다. 해당 API에 대한 인증은 내부 토큰 또는 Cloud Scheduler의 OIDC 토큰으로 대체할 수 있습니다.

준비 2: 스케줄러 전용 서비스 계정 생성 [OPS]
Cloud Scheduler가 Cloud Run(또는 Cloud Function) HTTP 목표를 호출할 때, **권한이 있는 서비스 계정**으로 요청하는 것이 좋습니다 [22] [23]. 따라서 Scheduler 용도로 최소 권한을 가진 서비스 계정 `scheduler-invoker`를 생성합니다:

```
PS C:\> gcloud iam service-accounts create scheduler-invoker --display-name "Scheduler Invoker SA"
# 예상 출력: Created service account [scheduler-invoker].
```

이 서비스 계정에 Cloud Run 서비스를 **invoke**할 수 있는 권한을 부여합니다:

```
PS C:\> gcloud run services add-iam-policy-binding QualiJournalAdmin `
    --region asia-northeast1 `
    --member "serviceAccount:scheduler-invoker@<PROJECT_ID>.iam.gserviceaccount.com" `
    --role "roles/run.invoker"
# 예상 출력: Updated IAM policy for service [QualiJournalAdmin].
```

위 설정으로 `scheduler-invoker` 계정이 Cloud Run 서비스를 호출(invoke)할 수 있게 됩니다. (만약 Cloud Run을 **인증 필요** 모드로 운용 중이라면 이 계정만 invoke 권한을 가지고, 익명 호출은 불가하게 통제됩니다.)

준비 3: Cloud Scheduler 작업 생성 [OPS]
이제 실제 스케줄러 Job을 생성합니다. Cron식 `0 0 9 * * *` (매일 09:00)을 사용하고, HTTP 타겟으로 Cloud Run의 엔드포인트 URL을 지정합니다. 아울러 위에서 만든 서비스 계정을 OIDC 인증 토큰과 함께 사용하도록 설정합니다 [23]:

```
PS C:\> gcloud scheduler jobs create http daily-publish-job `
    --location asia-northeast1 `
    --schedule="0 9 * * *" `
    --http-method POST `
    --uri="https://qualijournaladmin-<hash>-asia-northeast1.run.app/api/publish-today" `
    --oidc-service-account-email="scheduler-invoker@<PROJECT_ID>.iam.gserviceaccount.com" `
    --oidc-token-audience="https://qualijournaladmin-<hash>-asia-northeast1.run.app"
# 예상 출력: Created job [daily-publish-job].
```

명령 요소 설명:
- `--schedule`: Cron 표현식 또는 간단 표현으로 일정 지정. (예: `"every 24 hours"`로도 가능하지만, 위처럼 명시)
- `--uri`: 호출할 대상 URL. Cloud Run 커스텀 도메인을 사용하는 경우 해당 URL을 넣어도 되지만, **OIDC audience** 설정 이슈로 Cloud Run의 기본 URL을 권장합니다 [24]. 위 예시는 Cloud Run 기본 도메인을 사용.
- `--oidc-service-account-email`: 이 서비스 계정의 신원으로 OIDC ID 토큰을 발행하여 요청에 `Authorization: Bearer <token>` 헤더를 포함합니다 [23]. Cloud Run 측에서는 이 토큰의 발행자가 Google이며 audience가 자신과 일치함을 자동 검증하고(인증 필요 모드일 경우), 또한 Invoker 권한까지 확인하여 최종 요청을 수락합니다 [22].
- `--oidc-token-audience`: 토큰의 audience(대상)를 지정. 일반적으로 Cloud Run 서비스의 URL로 설정하며, **이 값은 Cloud Run에서 기대하는 값과 일치해야** 합니다. 커스텀 도메인을 쓴다면 해당 URL로 지정하고, Cloud Run 서비스 설정에서 JWT Audience로 그 값을 허용해야 할 수 있습니다. 기본적으로 Cloud Run은 자체 URL을 기본 audience로 삼습니다.

작업 생성 후 스케줄러가 제대로 설정되었는지 확인합니다:

```
PS C:\> gcloud scheduler jobs describe daily-publish-job --location asia-northeast1
# 예상 출력: schedule: 0 9 * * * / state: ENABLED / lastAttemptTime: ... / nextRunTime: ...
```

주의: Cloud Scheduler 지역(`--location`)은 가급적 Cloud Run 서비스와 동일한 리전에 가깝게 설정했습니다 (asia-northeast1). 이렇게 해야 네트워크 지연을 줄일 수 있습니다.

준비 4: 권한 및 테스트 [DEV/OPS]
Cloud Scheduler 작업이 생성되면 GCP가 정해진 시각에 자동으로 Cloud Run 엔드포인트를 호출합니다. 첫 실행 전에 수동으로 **"Run now"** (콘솔에서) 또는 `gcloud scheduler jobs run daily-publish-job` 명령으로 즉시 실행해 볼 수 있습니다. 이때 Cloud Run 로그를 모니터링하여 `/api/publish-today` 요청이 도착했고 정상 처리되었는지 확인합니다.
Cloud Run이 인증 필요 모드라면, Scheduler가 보내는 OIDC 토큰이 유효한지, 해당 서비스 계정에 Invoker 권한이 있는지만 맞으면 자동 처리됩니다. 만약 Cloud Run이 **익명 접근 허용** 상태라면, 사실 OIDC 설정 없이도 호출은 가능하나 보안을 위해 OIDC 방식을 유지하는 것이 바람직합니다. 추가로, 백엔드에서 **Scheduler 전용 보안 토큰**을 검사하도록 구현했다면 (예: 요청 헤더에 미리 공유된 비밀 토큰 포함), Scheduler 설정의 `--headers` 옵션을 사용해 맞춰줘야 합니다.

이제 Cloud Scheduler를 통해 **일일 자동 발행**이 구현되었습니다. 정해진 시간마다 운영자가 수동 개입하지 않아도 수집→발행 프로세스가 실행되며, 실행 결과는 Slack 알림이나 이메일로 통지하도록 Cloud Run 내부 로직을 추가 구현할 수 있습니다 (예: 완료 후 Slack Webhook 호출). 기존 보고서에서도 Windows 작업 스케줄러 자동화 및 알림 방안이 제시되었는데, 이를 GCP 서비스로 대체한 것입니다 [25].

3.3 [DEV/OPS] 백업 자동화 (데이터 보존)

QualiJournal 시스템은 **아카이브** 폴더에 일자별 결과물(JSON, HTML, Markdown)을 저장해두고 있습니다 [26]. 이 데이터들은 시간이 지날수록 용량이 쌓이며, 혹시 모를 분실에 대비해 정기적인 백업이 필요합니다 [27]. Cloud Run 컨테이너의 로컬 저장소는 휘발성이므로, 기존처럼 파일 시스템에만 보관하는 것은 위험합니다. 이에 **Cloud Storage**를 활용한 백업 전략을 세웁니다.

- **Cloud Storage 버킷 생성 [OPS]:** 우선 백업 파일을 보관할 GCS 버킷을 생성합니다 (예: 버킷 이름 `qualijournal-archive-backup`).

```
PS C:\> gsutil mb gs://qualijournal-archive-backup
# 예상 출력: Creating gs://qualijournal-archive-backup/...
```

  버킷 위치를 서비스 리전과 동일(또는 인근)하게 선택하여 성능을 최적화합니다.

- **발행 데이터 업로드 [DEV]:** 일일 발행 작업이 끝난 후, **아카이브 파일들을 Cloud Storage에 업로드**하는 기능을 구현합니다. 예를 들어 Python 백엔드에서는 Google Cloud Storage 클라이언트 라이브러리를 사용해 `archive/YYYY-MM-DD_keyword.json` 등을 버킷에 저장할 수 있습니다. 간단한 예:

```
from google.cloud import storage

client = storage.Client()
bucket = client.bucket("qualijournal-archive-backup")
for fname in ["latest.json", archive_filename]:
    blob = bucket.blob(f"2025-10/{fname}")
    blob.upload_from_filename(f"archive/{fname}")
```

  위 코드는 예시이며, 실제로는 날짜별 디렉터리 구조를 만들어 업로드하거나, **저장 시점의 타임스탬프**를 붙여서 파일명을 중복없이 관리합니다 [27]. 업로드가 성공하면 GCS 상에 파일 크기와 MD5 해시 등을 확인하여 무결성을 검증할 수 있습니다.

- **기존 자료 백업 [OPS]:** 만약 로컬에 과거 아카이브 데이터가 있다면, 초기 설정 시 한 번 수동 백업을 권장합니다. 로컬 PC의 `C:\Users\user\Desktop\퀄리저널\archive\` 폴더 내용을 모두 GCS로 올려두면 안전합니다:

```
PS C:\Users\user\Desktop\퀄리저널> gsutil -m rsync -r .\archive gs://qualijournal-archive-backup
# 예상 출력: Copying file://archive/2023-12-01_keyword.json [Content-Type=application/json]...
#        [###] 100% Done
```

  이로써 과거 데이터도 클라우드에 백업됩니다.

- **정기 백업 자동화 [OPS]:** 백업도 Cloud Scheduler를 이용해 **주기적**으로 수행 가능합니다. 다만, 발행 직후 바로 업로드하도록 구현했다면 추가 스케줄은 필요 없을 수 있습니다. 구현하지 못한 경우, 예를 들어 매주 일요일 새벽에 백업 스크립트를 실행하도록 Cloud Scheduler + Cloud Functions를 구성할 수 있습니다. 또는 Cloud Build의 트리거를 사용해 특정 시간에 저장소로부터 데이터를 읽어 버킷에 쓰는 방법도 있습니다.

가장 간단한 형태로, 위 **발행 API 호출 뒤** 백업을 같은 프로세스에서 처리하도록 했다면 별도의 Scheduler는 불필요합니다. 운영자는 주기적으로 GCS 버킷에 데이터가 누락 없이 쌓이는지 확인만 하면 됩니다.

- **데이터 보관 및 보안:** 백업 버킷에 **Object Versioning**을 켜 두면 파일 수정/삭제 이력이 남아 혹시 실수로 지워도 복구할 수 있습니다. 또한 버킷에 **Retention Policy**를 적용해 일정 기간(예: 1년) 이전의 데이터는 자동 삭제하거나, 별도로 내보내 장기보관(예: Coldline Storage)하는 것도 고려합니다. 버킷 권한은 최소한으로 설정하고, 민감정보가 없다면 편집팀에서 읽기 전용 액세스를 갖도록 IAM 설정할 수 있습니다.

요약하면, 백업 자동화를 통해 **컨텐츠 자산의 안정적 보관**이 가능해졌습니다. 운영자는 더 이상 로컬 서버 디스크 용량을 걱정하지 않아도 되며, GCP 상에 안전하게 축적되는 데이터를 기반으로 향후 트렌드 분석이나 검색 기능 확장도 용이해질 것입니다 [28] .

3.4 [DEV] UI 테마 점검 및 개선

마지막으로 **UI 테마**를 최종 점검하고 있습니다. 새로운 관리자 웹 UI는 QualiJournal의 브랜드 이미지와 편집 편의성을 고려하여 디자인되었으며, **라이트/다크 모드 지원** 및 **반응형 디자인**을 적용했습니다 [29] . 남은 작업은 세부적인 스타일 튜닝과 일관성 확보입니다:

- **컬러 스키마 확인:** 디자인 가이드에 맞춰 주요 색상이 사용되고 있는지 확인합니다. 예를 들어 브랜드 컬러(로고 색상 등)가 버튼이나 헤더에 반영되었는지, 다크 모드에서 글자 색상 대비가 충분한지 등을 살핍니다. 필요하면 `tailwind.config.js` 또는 CSS 변수 등에 색상 값을 조정합니다.

- **폰트 및 레이아웃:** 한글 콘텐츠 가독성을 위해 웹폰트 적용 여부(예: 나눔고딕 또는 본고딕 등)를 점검하고, 표나 카드 레이아웃이 화면 크기에 따라 깨지는 부분이 없는지 확인합니다. **반응형 레이아웃** 테스트를 위해 데스크톱, 태블릿, 모바일 해상도에서 UI를 확인하고, CSS 그리드/플렉스 아이템이 줄 바꿈 등 예상대로 작동하는지 확인합니다.

- **다크 모드 전환:** 시스템 다크 모드 설정 또는 UI 토글로 다크 모드 전환 시 색상이 적절히 반전되는지 확인합니다 (배경은 어두워지고, 텍스트는 밝게 등). 다크 모드에서 일부 아이콘이나 이미지가 안 보이는 경우 대체 자산을 쓰거나 CSS 필터를 적용합니다. 라이브러리를 썼다면 제공하는 테마 옵션을 활용합니다 [29] .

- **UX 세부 개선:** 편집자 피드백을 수렴하여, 예를 들어 **hover 효과**, **툴팁 표시**, **로딩 스피너** 등 사용자 경험을 향상시키는 요소를 추가합니다. 또한 키보드 단축키(기사 승인 등)가 의도대로 동작하는지 다시 한 번 통합 테스트합니다 [30] .

- 예: 기사 리스트에서 ↑/↓ 키로 항목 간 이동, Enter로 승인, Space로 고정 등의 단축키가 제대로 적용되고 충돌 없이 작동하는지 확인.

- 승인/보류 버튼을 눌렀을 때 즉각 시각적 피드백(색상 변화나 메시지)이 있는지 등.

- **마무리 문서화:** UI가 확정되면 스크린샷을 캡처하여 **운영 매뉴얼**에 최신 UI를 반영합니다. 편집자들이 참고할 수 있도록 주요 기능(기사 검색/필터, 코멘트 입력, 발행 미리보기 등)의 사용법을 문서화합니다 [31] . 이는 추후 신규 편집자 교육이나 시스템 인수인계에 도움이 됩니다.

테마 점검을 거쳐 작은 스타일 이슈들을 수정하면, QualiJournal 관리자 시스템은 디자인 면에서도 **완성도 높은 모습**을 갖추게 됩니다. 개발팀은 프론트엔드 빌드를 갱신하여 Cloud Run에 재배포하고, 운영자는 새 테마에서 **가독성**과 **편의성**이 충분한지 최종 확인합니다. 필요시 색상이나 UI 관련 피드백을 추가 반영하고, 이후에는 **버전 태깅**을 통해 이 상태를 기준점으로 삼아 향후 기능 추가를 진행할 수 있습니다.

본 가이드에서는 **QualiJournal 관리자 시스템**의 개발부터 배포, 그리고 운영 자동화와 보안/도메인 연계까지 전 과정을 다루었습니다. 요약하면, 현재까지 KPI 자동 갱신, 임계값 슬라이더 조정, 토큰 인증, Cloud Run 배포 등의 핵심 개선 작업이 완료되었고, 남은 과제로 WIF 인증 오류 수정, 커스텀 도메인 연결, 스케줄러/백업 자동화, UI 테마 마무리 등이 식별되었습니다. 해당 과제들을 순차적으로 해결함으로써, **총괄 개발자**는 시스템의 기술 스택과 아키텍처를 최신 클라우드 환경에 맞게 정비할 수 있고, **운영자(편집팀)**는 보다 안정적이고 편리해진 쿼리저널 플랫폼에서 일일 발행 작업을 이어갈 수 있을 것입니다.

참고 자료:

- Google Cloud Run 공식 문서 – 커스텀 도메인 매핑 가이드 [13] [14]
- Google Cloud Scheduler 공식 문서 – HTTP 타겟 스케줄링 및 OIDC 인증 [23] [22]
- Google Cloud IAM/Workload Identity Federation – 배포 파이프라인 연동 가이드 [8] [7]
- QualiJournal 시스템 개선 제안서 – 백업 및 자동화 필요성 언급 [27] [25]

---

[1] [26] 1004_2A고도화report.pdf
file://file-Ko3WmavWUUxUmLMC1s6fso

[2] Cloud Run | Google Cloud
https://cloud.google.com/run

[3] [4] Configure Workload Identity Federation with deployment pipelines | IAM Documentation | Google Cloud
https://cloud.google.com/iam/docs/workload-identity-federation-with-deployment-pipelines

[5] [6] [7] [8] [9] [10] Use Github workflow to deploy to cloud run with workload identity provider without a service account - Serverless Applications - Google Developer forums
https://discuss.google.dev/t/use-github-workflow-to-deploy-to-cloud-run-with-workload-identity-provider-without-a-service-account/191784

[11] [12] [13] [15] [16] [17] [18] [19] Mapping custom domains | Cloud Run | Google Cloud
https://cloud.google.com/run/docs/mapping-custom-domains

[14] Deploying Domain Mappings in GCP with GitLab CI | AirQuill
https://airquill.io/2022/11/12/deploying-domain-mappings-gcp-gitlab/

[20] [21] [29] [30] [31] 1003_4UI.pdf
file://file-Gcxm7FdLSYTHtsLZnDwtiJ

[22] Use authentication with HTTP targets | Cloud Scheduler | Google Cloud
https://cloud.google.com/scheduler/docs/http-target-auth

[23] Verifying Cloud Scheduler requests in Google Cloud Run with TypeScript · Jack Cuthbert
https://jackcuthbert.dev/blog/verifying-google-cloud-scheduler-requests-in-cloud-run-with-typescript

[24] Cloud Scheduler doesn't work with custom Cloud Run domain
https://discuss.google.dev/t/cloud-scheduler-doesnt-work-with-custom-cloud-run-domain/87920

[25] [27] [28] 1004_3A작업계획report.pdf
file://file-S3G4TQFPei3GeAJcXjN6B3