

프로젝트 주제	LLM 활용 내부 고객 업무 효율성 향상을 위한 구글 API 전문 개발자 지원 AI 기반 문서 검색 시스템
원본 데이터 (텍스트)	<p>구글 API 공식 문서 (Google for developers)</p> <ul style="list-style-type: none">- OAuth 2.0 / Google Identity- People API- Google Drive API- Google Sheets API- Gmail API- API Reference- YouTube Data API- Google Maps- Firestore REST API- Firebase Authentication- BigQuery API <p>기업 내부 문서 : OpenAI 합성 데이터셋(자체 생성)</p>
데이터 전처리 과정	<p>[구글 API 공식 문서] OPEN AI API를 사용해서, 구글 API 문서 데이터를 QA셋(jsonl파일)로 변환하였습니다. 구글 API 문서 데이터 원문을 벡터 DB에 넣게 되었을 때, 코드나 문맥이 잘리면서 데이터의 품질이 떨어질 수 있기 때문에 QA셋으로 가공을 하기로 결정하였습니다.</p> <p>원문 데이터를 벡터 DB에 넣는 작업도 따로 진행 중이므로 QA셋과 원문 데이터를 각각 벡터 DB에 넣은 후 검색 성능을 비교해볼 예정입니다.</p> <p>(1) 각 문서 1개당 최대 10개의 Q&A셋을 생성</p> <ul style="list-style-type: none">● OPEN AI API를 통해 구글 API 문서 데이터(txt 파일)을 순회● 각 문서 1개당 최대 10개의 Q&A셋을 생성하고, 생성된 Q&A셋을 JSONL 파일로 저장 <p>→ 문서 길이 등을 고려하지 않고, 한 문서당 최대 10개의 Q&A셋을 만들다 보니, 문서 길이가 긴 경우 문서의 핵심 정보가 Q&A 생성 과정에서 누락되는 문제가 발생했습니다.</p> <p>(2) 페어 단위 Q&A셋 생성</p> <ul style="list-style-type: none">● 청크 분할 및 오버랩 적용: 위 문제 해결을 위해, 긴 문서도 정보 손실 없이 처리할 수 있도록 청크(Chunk) 단위로 분할하고 경계 영역 정보를 보존하기 위해 오버랩(Overlap)을 적용했습니다.● 정보 누락 최소화 및 정확성 확보: 페어 단위로 처리하여 청크 간 중복을 검토하고,질문-답변 생성이 불필요한 페어는 저장하지 않음으로써 최종 결과물의 정확성과 유용성을 확보했습니다.
DB 사용 용도	<p>[VectorDB - Chroma] API 문서 및 내부 문서의 임베딩 벡터 저장 RAG(검색 증강 생성) 시스템의 핵심 요소로, 사용자 질문에 가장 관련성 높은 문서를 찾아 LLM에 제공하는 역할 의미 검색 + 키워드 검색 지원 권한 기반 필터링 적용</p> <p>[MySQL (관계형 DB)] 사용자 계정, 직급, 권한 관리 (sLLM(Qwen3-8B)의 권한 기반 필터링에 사용)</p>

	채팅 저장 및 관리 (GPT-4o 및 sLLM과의 전체 대화 내역) 커뮤니티 Q&A, 대화 내역 카드 저장, API 키 관리
사용 데이터	외부 데이터: Google Developers 공식 문서 <ul style="list-style-type: none">• OAuth 2.0 / Google Identity• People API• Google Drive API• Google Sheets API• Gmail API• YouTube Data API• Google Maps• Firestore REST API• Firebase Authentication• BigQuery API 내부 데이터: 기업 내부 문서 RAG용 전처리 데이터: Google Developers 공식 문서 QA, 기업 내부 문서 QA (말투 포함)
Vector DB 구조	[구글 API 공식 문서 QA] <pre>[2] id=58ef51d2-61f3-40db-ae23-81e6a7e0b1ad similarity=72.1% (distance=0.2791) Q: Google의 CardDAV 프로토콜을 사용하여 연락처를 삽입하는 방법은 무엇인가요? A: 클라이언트 애플리케이션은 VCard 3.0 형식의 새 연락처가 포함된 POST 요청을 실행하여 연락처를 삽입할 수 있습니다. 응답에는 새 연락처의 ID가 포함됩니다. -> sources: ["https://developers.google.com/people/carddav?hl=ko#synchronizing_contacts"] -> tags: people -> last_verified: 2025-08-19 -> source_file: people_v1_troubleshoot-authentication-authorization.txt [3] id=cf76f249-b1bd-4233-92ad-337210e2278f similarity=71.4% (distance=0.2862) Q: Google의 CardDAV API를 사용하여 새 연락처를 생성하는 방법은 무엇인가요? A: 클라이언트 애플리케이션은 VCard 3.0 형식의 새 연락처가 포함된 POST 요청을 실행하여 새 연락처를 생성할 수 있습니다. 응답에는 새 연락처의 ID가 포함됩니다. -> sources: ["https://developers.google.com/people/carddav"] -> tags: people -> last_verified: 2025-08-19 -> source_file: people_v1_troubleshoot-authentication-authorization.txt</pre>
Django 내부 RDB 구조	<pre>graph LR api_key --> user approval_log --> user user --> card user --> chat_session card --> card_message card --> chat_message chat_session --> chat_message</pre>

RDB 조회

- RDB에 테스트용 더미데이터 저장 후 조회

```
sqlite> PRAGMA table_info('user');
cid  name      type      notnull  dflt_value  pk
---  ---
0    password  varchar(128)  1        0
1    last_login datetime    0
2    is_superuser bool        1
3    id        varchar(50)  1
4    email     varchar(255) 1
5    name      varchar(100) 1
6    phone     varchar(30)  1
7    gender    varchar(20)  1
8    birthday  date         1
9    rank      varchar(50)  1
10   department varchar(100) 0
11   status    varchar(20)  1
12   created_at datetime     1
13   updated_at datetime     1
14   is_active  bool        1
15   is_staff  bool        1
sqlite> PRAGMA table_info('approval_log');
cid  name      type      notnull  dflt_value  pk
---  ---
0    id        INTEGER    1        1
1    action    varchar(20) 1
2    reason    varchar(500) 0
3    created_at datetime     1
4    user_id   varchar(50)  1
sqlite> PRAGMA table_info('api_key');
cid  name      type      notnull  dflt_value  pk
---  ---
0    id        INTEGER    1        1
1    name      varchar(100) 1
2    secret_key varchar(255) 1
3    created_at datetime     1
4    user_id   varchar(50)  1
sqlite> PRAGMA table_info('chat_session');
cid  name      type      notnull  dflt_value  pk
---  ---
0    id        INTEGER    1        1
1    title     varchar(200) 1
2    mode      varchar(20)  1
3    created_at datetime     1
4    user_id   varchar(50)  1
sqlite> PRAGMA table_info('chat_message');
cid  name      type      notnull  dflt_value  pk
---  ---
0    id        INTEGER    1        1
1    role      varchar(20) 1
2    content    TEXT         1
3    created_at datetime     1
4    session_id bigint       1
sqlite> PRAGMA table_info('card');
cid  name      type      notnull  dflt_value  pk
---  ---
0    id        INTEGER    1        1
1    title     varchar(200) 1
2    created_at datetime     1
3    user_id   varchar(50)  1
4    session_id bigint       0
sqlite> PRAGMA table_info('card_message');
cid  name      type      notnull  dflt_value  pk
---  ---
0    id        INTEGER    1        1
1    position  INTEGER    1
2    card_id   bigint     1
3    message_id bigint     1
```

```
sqlite>
sqlite> SELECT id,email,name,phone,gender,birthday,rank,department,status
...> FROM user
...> LIMIT 3;
u001|u001@example.com|홍길동|010-1111-1111|male|1990-01-01|사원|콘텐츠|approved
u002|u002@example.com|이영희|010-1111-1112|female|1992-02-02|대리|영업지원|pending
u003|u003@example.com|박철수|010-1111-1113|male|1988-03-03|과장|고객지원|approved
sqlite>
sqlite> SELECT *
...> FROM approval_log
...> WHERE action='rejected';
34|rejected|신원 조회 불가|2025-08-22 08:23:17|u004
39|rejected|직급 불일치|2025-08-22 08:23:17|u009
sqlite>
sqlite> SELECT id, session_id, role, content
...> FROM chat_message
...> WHERE content LIKE '%안녕하세요%';
31|1|user|안녕하세요
32|1|assistant|안녕하세요, 무엇을 도와드릴까요?
sqlite>
sqlite> SELECT u.id AS user_id, u.name, k.name AS key_name, k.secret_key
...> FROM api_key AS k JOIN user AS u ON u.id = k.user_id
...> WHERE u.name = '홍길동';
u001|홍길동|key1|sk_001
sqlite>
```