필요한 기술은 빠르게 익히고,
목적을 달성하는 데 집중하는 개발자 이건희입니다

익숙한 한 가지 기술에 얽매이지 않고, **문제 해결**에 가장 적합한 기술 스택을 선택 합니다.
최적의 서비스를 제공하기 위해 **처음 써보는 기술 스택**을 사용하여 **기획부터 개발**, **배포** 까지 진행한 경험이 있습니다.

Introduce

Github : tumblecat44

Velog : tumblecat

Email: leegeh1213@gmail.com

Phone : 010-9491-6725

LinkedIn : 이건희

Stack

Web

- **React**
- Next.js
- TypeScript
- JavaScript

BackEnd

- **Kotlin**
- **Spring Boot**
- **MySQL**
- Java
- **FastAPI**
- **Python**

AI

- **LangGraph**
- **LangChain**
- Pinecone
- **LangSmith**
- **Python**
- RAG

Android

- **Jetpack Compose**
- **Kotlin**
- CMP
- Coroutine
- Koin
- XML

Deployment

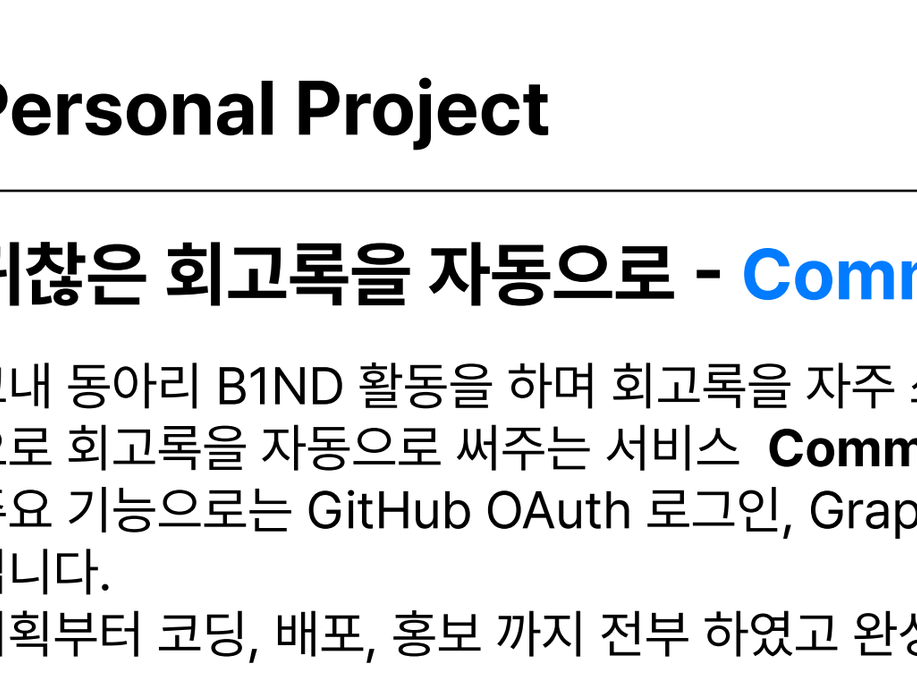
- **Docker**
- **Docker Compose**
- **Github Action**

Personal Project

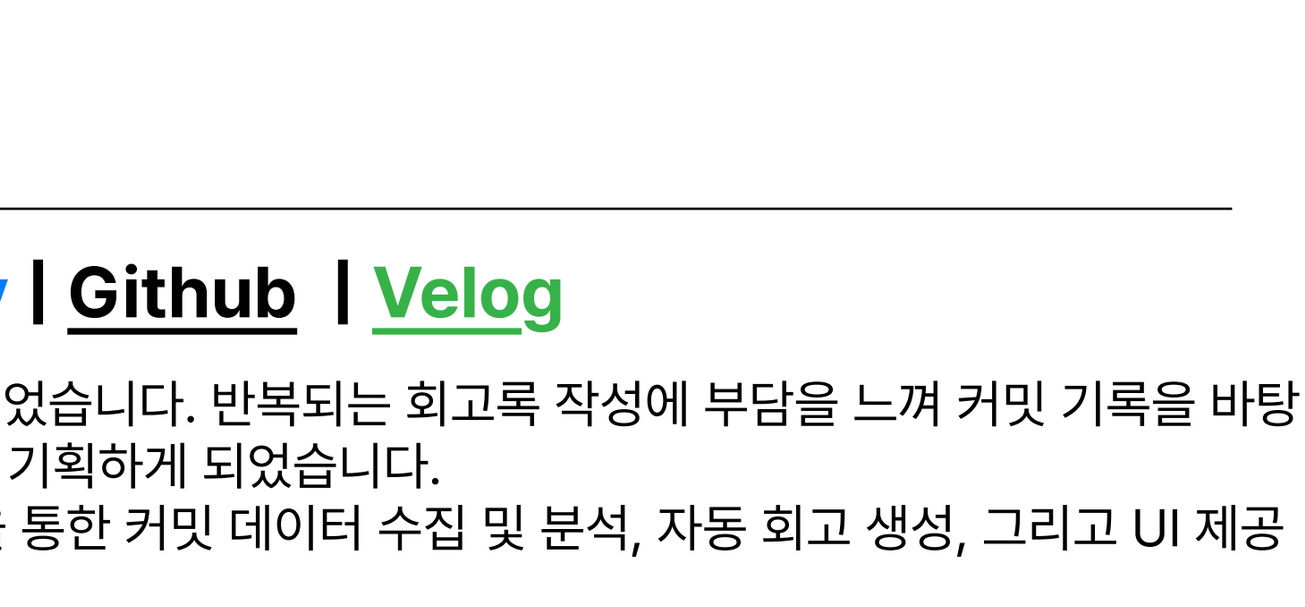
귀찮은 회고록을 자동으로 - Commitly | Github | Velog

교내 동아리 B1ND 활동을 하며 회고록을 자주 쓰게 되었습니다. 반복되는 회고록 작성에 부담을 느껴 커밋 기록을 바탕으로 회고록을 자동으로 써주는 서비스 **Commitly를** 기획하게 되었습니다.
주요 기능으로는 GitHub OAuth 로그인, GraphQL을 통한 커밋 데이터 수집 및 분석, 자동 회고 생성, 그리고 UI 제공입니다.
기획부터 코딩, 배포, 홍보 까지 전부 하였고 완성하고 난 후 회고록을 velog 에 올려 좋아요 52개를 받았습니다.

아키텍처



실제 화면



주요 성과

- 사용자 행동을 분석하기 위해 **Microsoft Clarity**와 **Google Analytics**를 도입했습니다. Clarity의 히트맵과 세션 리플레이를 분석한 결과, 온보딩 화면의 리뷰 섹션에서 **사용자 이탈률**이 높고 클릭이 거의 발생하지 않는 문제를 발견했습니다. 이에 따라 리뷰 섹션을 제거한 결과, 사용자들이 자연스럽게 메인 화면으로 이동하여 서비스를 더욱 원활하게 이용하는 개선 효과를 얻었습니다.
- 가져온 커밋 기록을 바탕으로 GPT 가 더 나은 회고록을 작성할 수 있도록 프롬프트 엔지니어링을 적용하였습니다.

트러블 슈팅

배포 후 Mixed Content Error 발생

- 배포 후 웹에서 서버를 호출할 때 Mixed Content Error 발생하였습니다.
- 원인을 찾는 중 Chrome 정책으로 HTTP와 HTTPS 간의 통신을 차단한다는 것을 알게 됐습니다.
- EC2에 구입한 도메인을 연결하여 HTTPS 를 적용 시켜 해결하였습니다.

배포 시간이 비효율적임

- 배포를 수동으로 EC2에 올려 개발 시간 중 많은 부분을 배포가 차지 하였습니다.
- GitHub Actions를 도입하여 CI/CD 자동화 하였습니다.

불필요한 보일러플레이트 코드

- GPT API 를 수동으로 호출하여 많은 보일러플레이트 코드가 발생하는 것을 확인하였습니다. 해결방법을 찾는 중 Spring AI 라이브러리를 사용하면 보일러플레이트를 줄일 수 있다는 것을 확인하여 도입하였습니다.

Team Project

비개발자와 개발자를 위한 AI 기반 프로젝트 관리 SaaS - Sync 2025.03~

Sync 는 교내 프로그램 '**소프트웨어 나르샤**'로 진행중인 프로젝트 입니다. **웹 4명**, **백엔드 3명**으로 이루어져있으며 **팀장**과 **AI** 서버 개발을 맡고 있습니다.

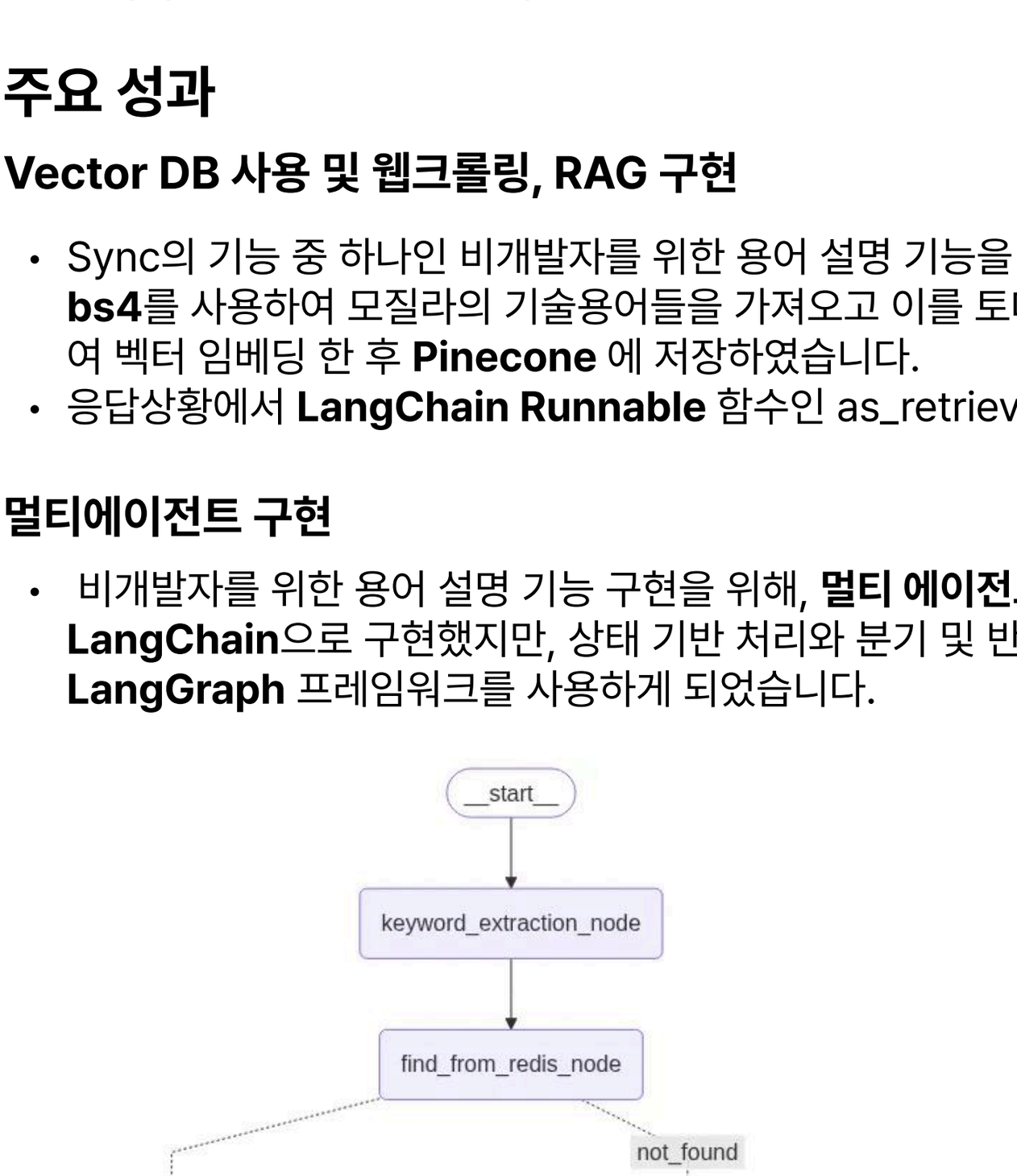
주요 성과

Vector DB 사용 및 웹크롤링, RAG 구현

- Sync의 기능 중 하나인 비개발자를 위한 용어 설명 기능을 위해 Vector DB 와 웹 크롤링, RAG 를 사용했습니다. **bs4**를 사용하여 모질라의 기술용어들을 가져오고 이를 토대로 **Langchain** 의 **OpenAIEmbeddings** 을 사용하여 벡터 임베딩 한 후 **Pinecone** 에 저장하였습니다.
- 응답상황에서 **LangChain Runnable** 함수인 as_retriever() 를 사용하여 **RAG** 를 구현했습니다.

멀티에이전트 구현

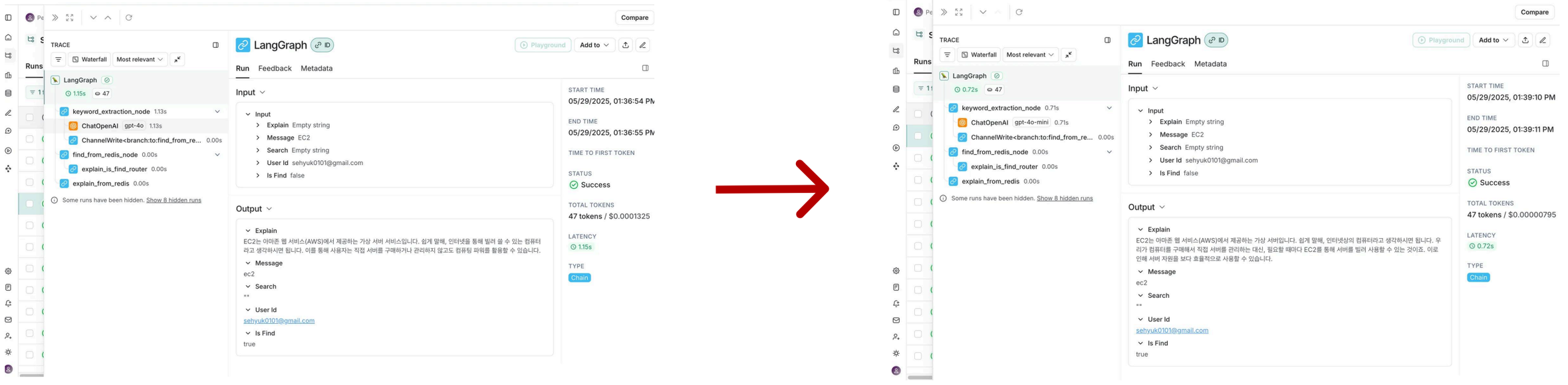
- 비개발자를 위한 용어 설명 기능 구현을 위해, **멀티 에이전트** 시스템을 개발하였습니다. 초기에는 초기에는 **LangChain**으로 구현했지만, 상태 기반 처리와 분기 및 반복 로직이 필요해짐에 따라 **LangChain** 팀에서 개발한 **LangGraph** 프레임워크를 사용하게 되었습니다.



- **keyword_extraction_node** 는 주어진 문장 중 주요 용어를 추출하는 노드입니다.
- **find_from_redis_node** 는 Redis에 기존에 저장된 설명 데이터를 **검색**하는 노드입니다. 매 응답마다 LLM에게 답을 요청할 시 발생하는 경제적인 **비효율**과 **응답지연**을 해결하기 위해 캐싱을 도입했습니다.
- **read_search_node** 는 LLM 에 이미 학습 된 단어인지 **판단**하는 노드입니다.
- **web_search_node** 는 **Tavily** 를 사용하여 웹 크롤링을 하는 노드입니다.
- **explain_with_web_node** 는 웹 크롤링 결과를 바탕으로 **RAG** 하여 결과를 반환하는 노드입니다..

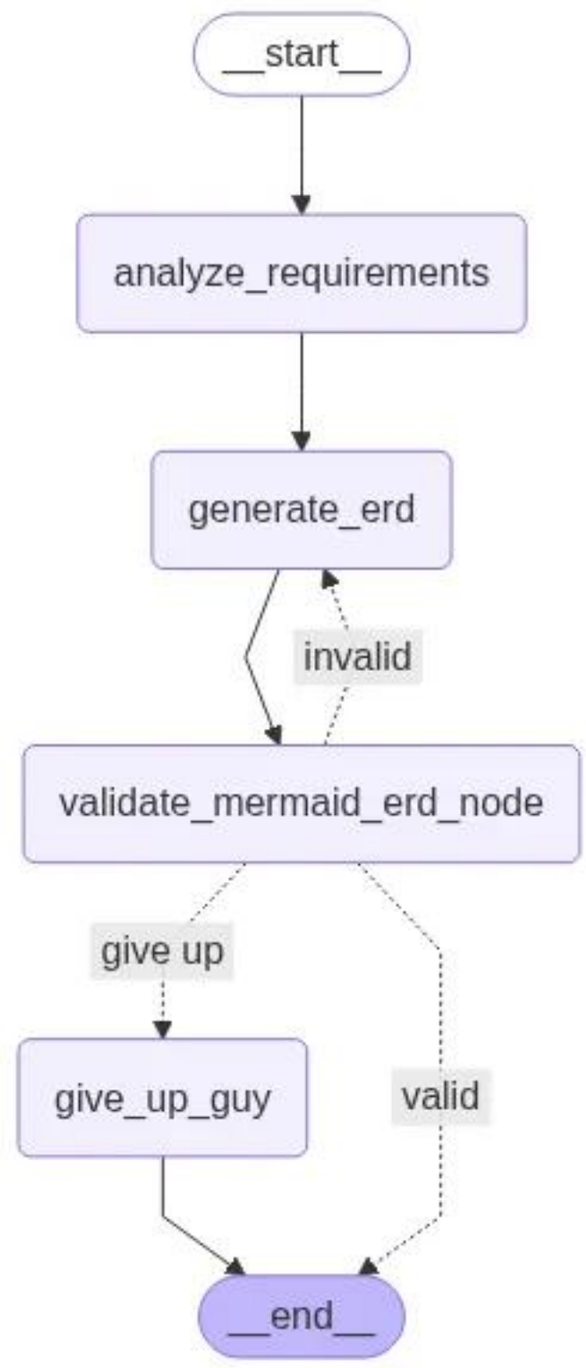
응답 지연 시간 개선

- 기존 keyword_extraction_node 는 **GPT-4o**를 사용했습니다. 그러나 단순한 기능을 수행하는데에 비해 지연시간이 길었습니다. 그래서 **GPT-4o-mini** 로 모델을 변경했습니다. 이를 통해 약 0.4초 이상 지연시간을 감소했습니다.



기능 명세서 기반 ERD 생성 기능 구현

- 작성된 기능명세서를 바탕으로 **Mermaid.js ERD** 시각화 코드를 반환하는 기능을 구현했습니다.



- **analyze_requirements** 노드를 통해 방대한 기능명세서 중 핵심 도메인만 추출합니다.
- **generate_erd** 를 통해 Mermaid.js 로 시각화 할 수 있는 ERD 시각화 코드를 반환합니다.
- **vaildate_mermaid_erd_node** 를 통해 코드가 문법에 맞고 오류가 없는지 검사합니다.
- 이 단계에서 코드에 문제가 있다면 다시 **generate_erd** 노드로 돌려보냅니다.
- 특정 횟수 이상이 지나도 생성이 안될 시 에러를 반환합니다.

트러블 슈팅

ERD 생성 그래프 무한 순회 문제

- ERD 생성 중 문법에 맞지않는 코드가 반환될 시 그 전 노드로 반환하도록 설계했습니다. 그러나 계속 생성에 실패할 시 반환으로 넘어가지 않고 **무한 순환**이 일어났습니다. 이 문제를 해결하기 위해 **state** 에 **count** 변수를 추가하여 **vaildate_mermaid_erd_node** 에서 ERD 생성을 다섯번 이상 시도했을 시 에러를 반환하도록 구현했습니다.

응답 신뢰도 문제

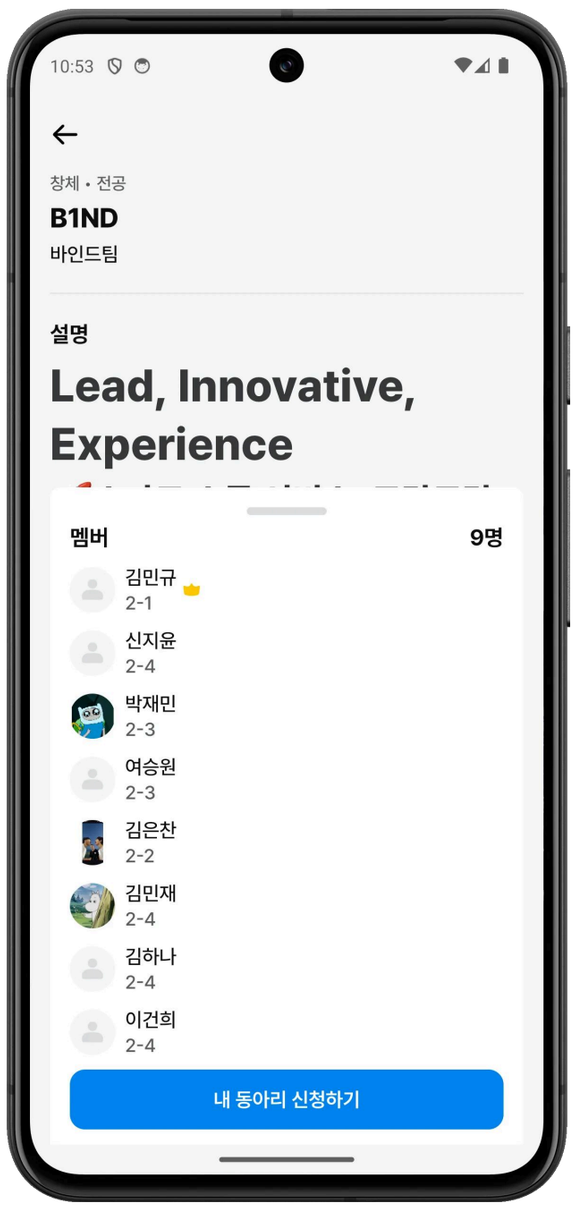
- 빠른 응답 속도를 위해 캐싱을 도입 했습니다. 다만 신뢰도가 떨어지는 답변이 캐싱되어 설명으로 반환되는 문제가 있었습니다. 이를 위해 **“응답이 좋아요", “응답이 싫어요"** 기능을 추가했습니다. 응답에 긍정할 시 redis 에 들어가있는 키의 만료시간을 갱신하고 부정적 반응을 보일 시 키를 삭제하도록 구현했습니다.

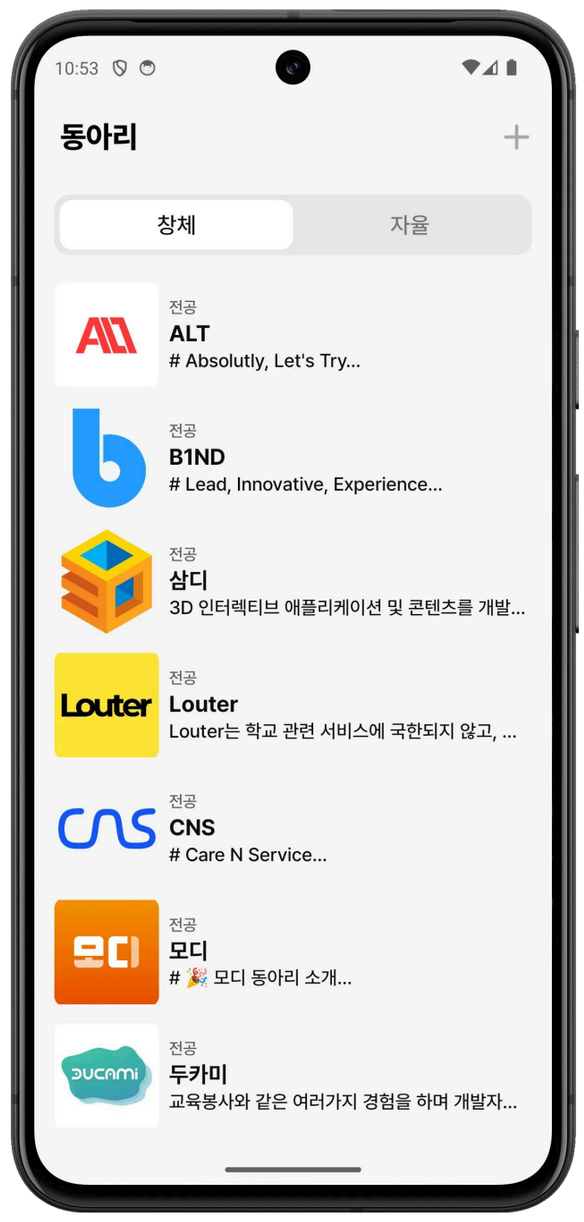
LangGrpah 환경에서의 SSE 구현의 문제

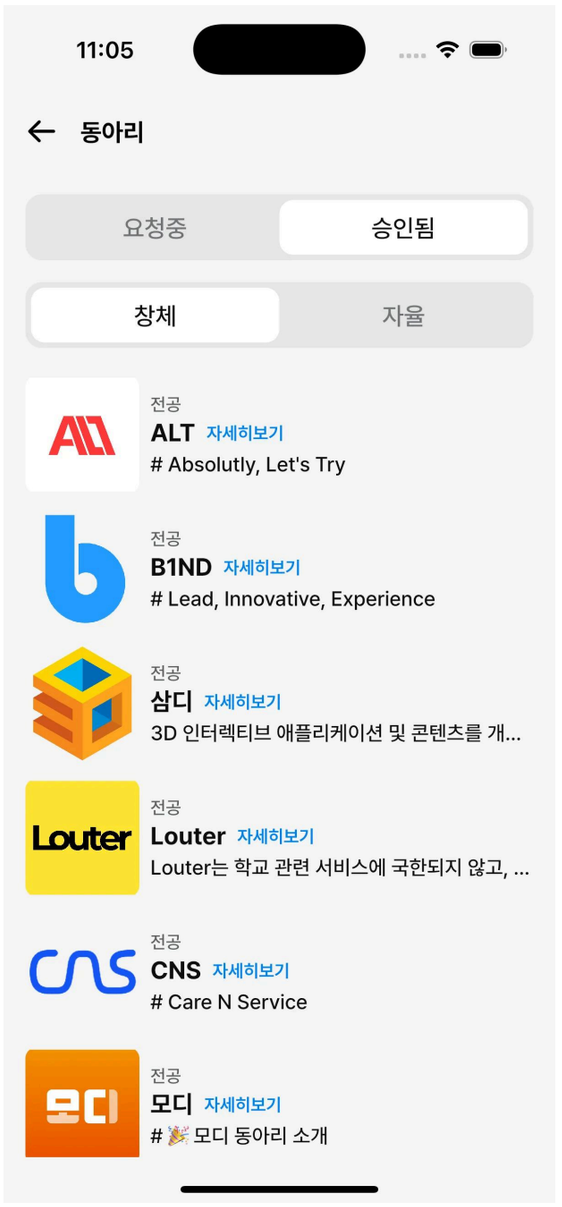
- 기존 LangChain 환경에서는 **astream() Runnable** 과 **yield Generator**, **StreamResponse** 를 통해 **SSE** 를 구현할 수 있었습니다. **(작성한 블로그)** 그러나 LangGrpah StateGraph 환경에서는 state 를 반환해야하는데 도중에 청크를 반환하는 방법을 찾기 힘들었습니다. 그러던 중 langgraph.type 에 StreamWriter 가 있는 것을 확인했고 이를 통해 SSE 구현을 했습니다.

교내 스마트 스쿨 서비스 - 도담도담 dodamdodam-android 2025. 02 ~

도담도담은 교내 동아리 **B1ND**에서 운영 및 유지보수하는 스마트 스쿨 플랫폼 입니다.
3월의 교내 동아리 신청 기간에 사용할 동아리 조회, 신청 기능과 동아리 승인 기능을 개발 하였습니다.







주요 성과

- MVVM + 멀티 모듈 + CMP 환경에서 개발을 진행하였습니다.
- Koin 을 이용한 의존성 분리로 모듈과 클래스간의 결합도를 줄였습니다.

트러블 슈팅

- 동아리 소개 화면에 마크다운 렌더링 기능이 필요했습니다.
- 처음에는 Jetpack Compose 전용 라이브러리인 https://github.com/jeziellago/compose-markdown을 사용하려 했습니다.
- 그러나 CMP(Compose Multiplatform) 환경에서는 정상적으로 동작하지 않는 문제가 발생했습니다.
- 원인 분석
  a. compose-markdown이 Android Compose 전용으로 설계되었기 때문에, Multiplatform Compose에서는 정상적으로 동작하지 않았습니다.
  b. expect/actual을 활용한 MPP(Multiplatform Project) 지원이 없었습니다.
  c. 실제 코드 실행 시 "Unresolved reference: Markdown" 등의 컴파일 오류가 발생하였습니다.
- 해결 시도
  - 직접 마크다운 렌더러를 구현하는 방법 고려
    - commonMain에서 HTML 변환 후 AnnotatedString으로 변환하는 방식
    - 하지만, MarkdownParser 등을 직접 구현해야 했고, 마감 기한 내에 완료하기 어려웠습니다.
  - 다른 라이브러리 탐색
    - https://github.com/mikepenz/multiplatform-markdown-renderer 발견
    - 해당 라이브러리는 CMP(Multiplatform) 지원하였습니다.
    - Android, iOS, Desktop 등 다양한 환경에서 실행 가능
- 적용 및 결과
- multiplatform-markdown-renderer를 사용하여 CMP 환경에서도 정상적으로 마크다운이 렌더링이 됐습니다.
- 빠르게 적용할 수 있었고, 이슈 없이 마감 기한을 맞출 수 있었습니다.

수상경력

- **대소고 프로그래밍 교과 우수상(1등)**
- **대소고 해커톤 대상**
- **제6회 청소년 ICT창업가 캠프 본선 대회 은상**
- **TOPCIT 460 (3수준)**