외장 뇌 소유자, 이건희 AI Workflow · AnyStack

2024년부터 **LLM과 함께 코딩**하며, **생각을 구조화하고 논리를 코드로 구현**하고 있습니다.
LangGraph로 **AI Agent**를 개발하며 **기존 아키텍처**(ReAct, Plan And Execute)의 **한계**를 **고민**하고,
이를 보완한 **새로운 구조**(Plan-Decision-Execute)를 만들어 **서비스**에 **적용**한 경험이 있습니다.
LLM을 통해 다양한 스택을 빠르게 익히고 적용하는 강점을 살려 **해결하고자 하는 문제**를,
Spring Boot, React, Next.js 로 **기획**부터 **개발**, **배포**까지 **직접 완성**한 경험이 있습니다.
몰입 속에서 새로운 가치를 만들어내는 순간을 즐깁니다.

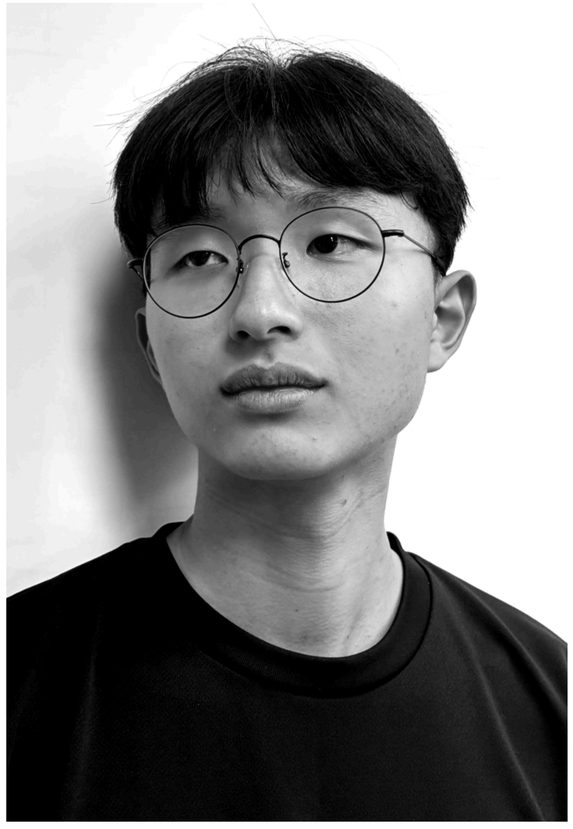
Contact Me

Github: tumblecat44

Email: hello@jason-lee.dev

More About: jason-lee.dev

Phone: +82 10 9491 6725



Main Stack

Python | FastAPI | LangGraph | LangChain

Sub Stack

TypeScript | React | Kotlin | Spring Boot | Android

My Activity

- 대구소프트웨어마이스터고등학교 9기 인공지능소프트웨어과 | 2024. 3. ~ 2027. 2. (졸업 예정)
- 2024학년도 교내 소프트웨어 해커톤 대상
- 2024학년도 프로그래밍 과목 교과우수상
- Topcit 23회 정기 평가 - 712점 (4수준)

Make Human More Creative

반복적이고 단순한 일에 **사람의 시간과 창의성이 낭비**되고 있다고 생각합니다.

LLM의 대중화로 이제 **"기계가 할 일"**과 **"사람이 해야 할 일"**의 경계가 분명해졌습니다.

Sync와 **Commitly**, **시험 친구 피티형**을 통해 **'기계가 대신할 일'**을 **자동화하는 AI 서비스**를 만들어왔습니다.

사람이 **더 창의적인 문제 해결**과 **사고에 집중**할 수 있는 **세상을 만드는 것이 제 목표**입니다.

AI Projects I've Built

[Team Project] Sync : 비개발자와 개발자간의 협업을 위한 AI 기반 프로젝트 관리 SaaS 2025.03 ~

- **Plan & Execute, ReAct 아키텍처의 한계를 보완한 Plan–Decision–Execute 구조를 직접 고안·적용한 AI 에이전트 DeepSync 개발.**
  - Plan 단계에서 사용할 도구, goal 등 전반적인 계획을 세움
  - Decision 단계에서 여러 선택지 중 최적의 행동을 선택 후 Execute, 반복
  - LangGraph 를 활용하여 Agent Workflow 구조화
  - Context 압축, Input 최소화 등 토큰 최적화 된 효율적인 멀티턴 대화 구축
  - StreamWriter, StreamingResponse 를 활용한 SSE 반환
- **AWS SES, Lambda, S3 를 활용한 이메일 기반 AI 이슈 생성 기능 개발**
  - PDF 를 분석하여 LLM 이 직접 이미지를 이해하고 활용하는 기능 구현
- **오류 체크 후 자동 수정 기능이 탑재 된 ERD 자동 생성 Workflow 개발**

[Solo Project] Commitly : 깃허브 커밋 기록 기반 AI 회고록 자동화 서비스 2025.01 ~ 2025.02

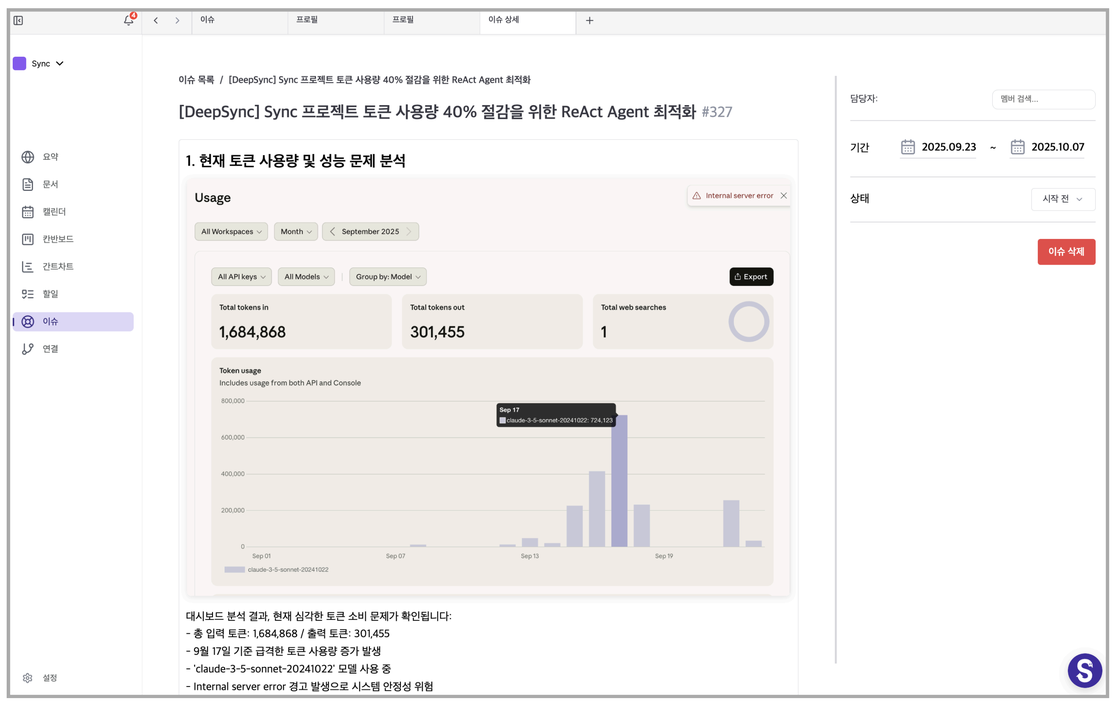
- **Github GraphQL 로 커밋기록을 가져온 후 Spring AI 를 활용한 회고록 자동 생성 기능 개발**
- **기획부터 개발 배포까지 혼자 끝낸 React, Spring Boot 활용한 풀스택 프로젝트**
- **실제로 모집한 사용자: 49명**
- **깃허브 오픈소스로 공개** github.com/Commitly

[Solo Project] 시험 친구 피티형 : PDF 기반 시험 문제 생성 서비스 2025.06

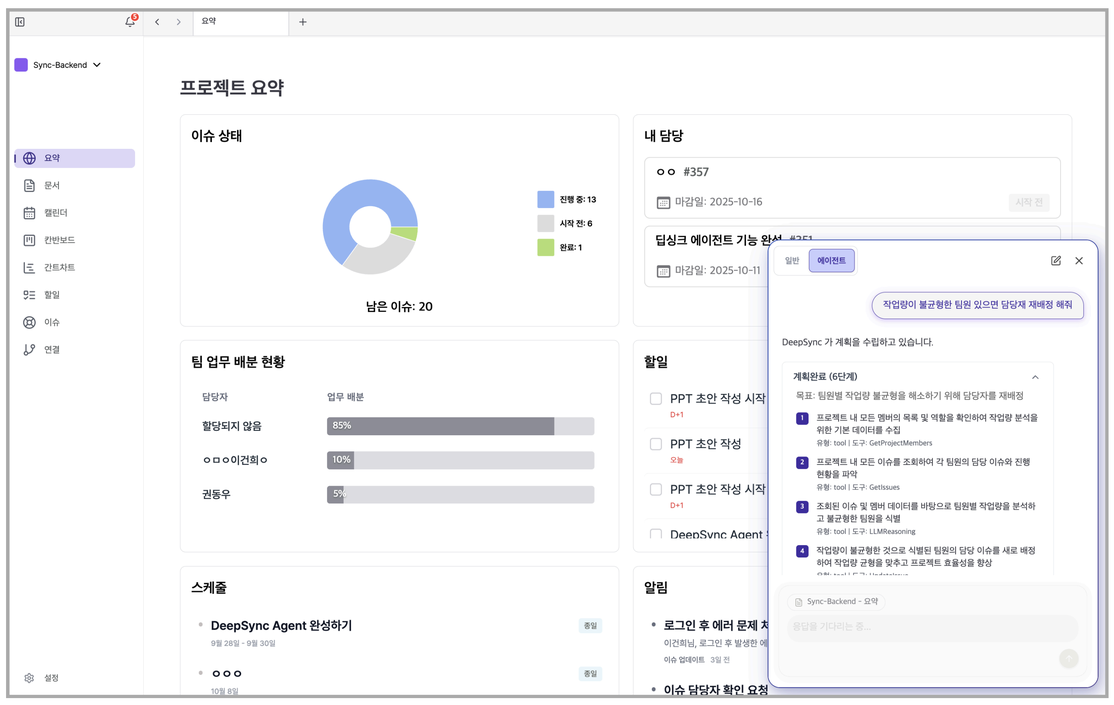
- **Next.js, LangChain JS 를 활용한 풀스택 프로젝트**
- **RecursiveCharacterTextSplitter, MemoryVectorStore 등을 활용한 PDF 분석 End To End 파이프라인 구축**
- **깃허브 오픈소스로 공개** github.com/tumblecat44/exam-friend-pt-bro
- **Vercel을 활용한 배포** exam-friend-pt-bro.vercel.app

Main Project - Sync

2025.03 ~



Email 기능으로 DeepSync 가 생성한 이슈



AI Agent DeepSync 작동 화면

내가 기여한 AI 개발 스택

Python | FastAPI | LangGraph | LangChain

내가 기여한 프론트 개발 스택

React | TypeScript | Electron | Zustand | Zod

개요

Sync는 비개발자와 개발자 간의 협업을 위한 AI 기반 프로젝트 관리 SaaS 입니다.

노션과 비슷한 실시간 마크다운 문서 작성을 통한 “기획"

기획을 바탕으로 AI 자동 ERD 생성을 통한 “검증"

깃허브 연동을 통한 자동 이슈 상태 변경을 통한 “실행"

캘린더, 실시간 칸반보드, 간트차트, 할일, 이슈를 통한 “관리"

소프트웨어 개발 전 단계에서 사용이 가능한 All In One 협업툴 입니다

나의 역할

팀장을 맡아 Sync 개발의 전과정을 전부 관여했습니다.

LangGraph 와 LangChain 을 활용하여 Sync 의 AI 기능을 전부 개발했습니다.

Sync 의 품질을 위해 AI 개발을 넘어서 React 를 사용한 프론트엔드 탭 기능, 전역 상태 관리 등을 개발했습니다.

피그마를 사용해 디자인시스템을 개발하였습니다.

마감기한을 정하고 팀원에게 할일을 정해주는 PM 역할도 맡았습니다.

Main Project - Sync

참여 인원

프론트엔드:

- 2025.03 ~ 2025.08: 4명
- 2025.08 ~ : 3명

백엔드

- 2025.03 ~ : 3명 (본인 포함)

Sync - Plan-Decision-Execute DeepSync 구현

개요

DeepSync 는 Sync 에서 작동하는 AI Agent 입니다.

기존에 ReAct 와 Plan And Execute 아키텍처를 사용아며 개발했지만 실사용 서비스에 적용하기에는 문제가 있어 단점을 보완하고 장점을 합친 아키텍처인 Plan-Decision-Execute 아키텍처를 고안해 DeepSync 에 적용했습니다.

ReAct 아키텍처의 문제점

ReAct 아키텍처의 가장 큰 문제점은 **과도하게 소비되는 토큰** 입니다. 초기 DeepSync 는LangGraph 의 create_react_agent 를 사용하여 구현됐습니다. 하지만 때 도구 호출 기록을 저장하면서 Reasoning 까지 하다 보니 “이슈를 수정해달라”는 간단한 요청에도 때 136원 이상의 돈이 소비가 되어 테스트를 하거나 실제로 사용하기 부담이 됐습니다.

또한 create_react_agent 를 사용하여 내부 커스터마이징의 제한이 있었고 주요 기능이 과도하게 블랙박스로 구현이되어 테스트 하기도 힘들었습니다.

Plan And Execute 아키텍처의 문제점

Plan And Execute 아키텍처의 가장 큰 문제점은 Conditional 관리와 과도하게 소요되는 Planning 시간입니다.

Plan 단계에서 모든 상황을 사전에 예측해야 하므로 계획이 정적으로 고정되고, 실시간 적응이 어렵습니다.

또한 LLM이 전체 워크플로우를 한 번에 시뮬레이션하기 때문에 토큰 비용과 지연이 과도하게 증가합니다.

결과적으로 실제 SaaS 환경에서는 비용·속도·유연성 모두 비효율적이었습니다.

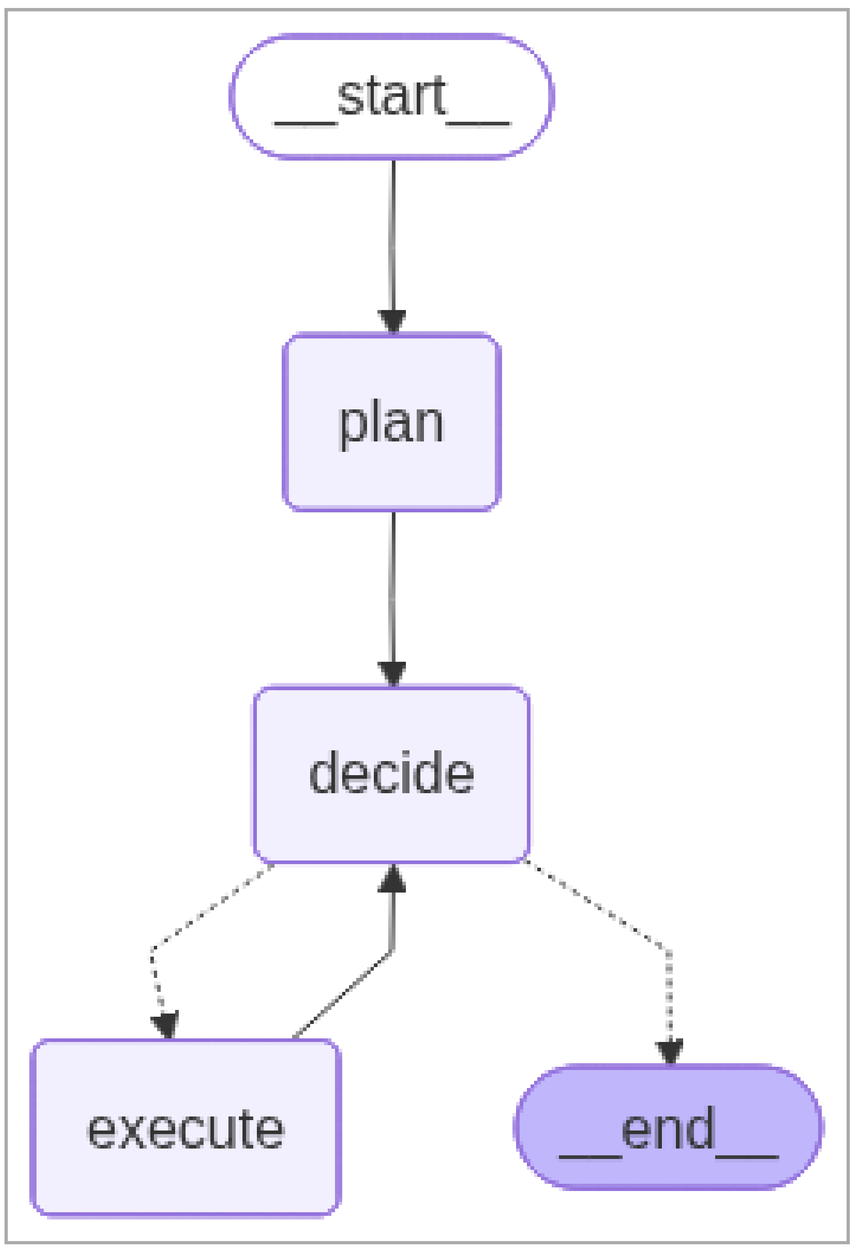
Main Idea Of Decision

Decision 단계는 실행 전후의 상태를 평가해 계획을 실시간으로 조정하는 중간 제어 레이어입니다.

LLM이 단순히 결과를 관찰하는 데 그치지 않고, 상황에 따라 수행·생략·우회 여부를 판단합니다.

이를 통해 Plan의 정적 한계를 보완하고, 동적 분기(Conditional Branching) 와 Self-Correction 을 가능하게 합니다.

Sync - Plan-Decision-Execute DeepSync 구현



AI Agent DeepSync 작동 그래프

Why LangGraph?

2번째로 제작한 DeepSync 는 Plan And Execute 아키텍처로 제작했습니다. (Decision 아키텍처는 세 번째 입니다.)
LangGraph를 쓴다고 해도 동적 컴파일이 불가능 하므로 LangGrpah 사용보다 While 문으로 Plan 을 순회하는 형식을 선택했습니다.
하지만 State 관리 지옥과 모니터링 지옥에 빠져 순수 개발 시간이 늘어나는 문제가 있었습니다.
이 문제를 또 겪지 않고자 LangGraph 를 사용해 LangSmith 의 모니터링 기능을 유용히 잘 쓰고자 했습니다.

Plan 단계

```
data: {"type": "plan_completed", "plan": {"goal": "To convert the project plan into actionable development tasks, create issues for each task, and assign them to project member
s.", "steps": ["ReadPage", "AnalyzeFeature", "GenerateIssue", "CreateIssue", "AssignMember", "SendNotification"]}, "total_steps": 6}

data: {"type": "decision_start", "message": "다음 Step 결정"}

data: {"type": "decision_completed", "message": "결정 완료", "reason": "The page content is missing, which is necessary to proceed with the subsequent steps of analyzing features
and generating issues.", "next_step": "ReadPage"}
```

프롬프트에 명시된 Tool 을 이용해 Tool 을 반환한 Response

Plan 단계에서는 수행해야 할 작업의 목록(Steps) 만 정의합니다.

각 Step 간의 순서는 명시하지 않고, Decision 단계에서 현재 상황(State)에 따라 다음에 실행할 Step을 동적으로 선택하기로 설계 했습니다.

이를 통해 정적인 순서 기반 플로우 대신,

AI 가 직접 판단하는 조건에 따라 유연하게 분기되는 Agent 워크플로우(Adaptive Workflow) 를 구현했습니다.

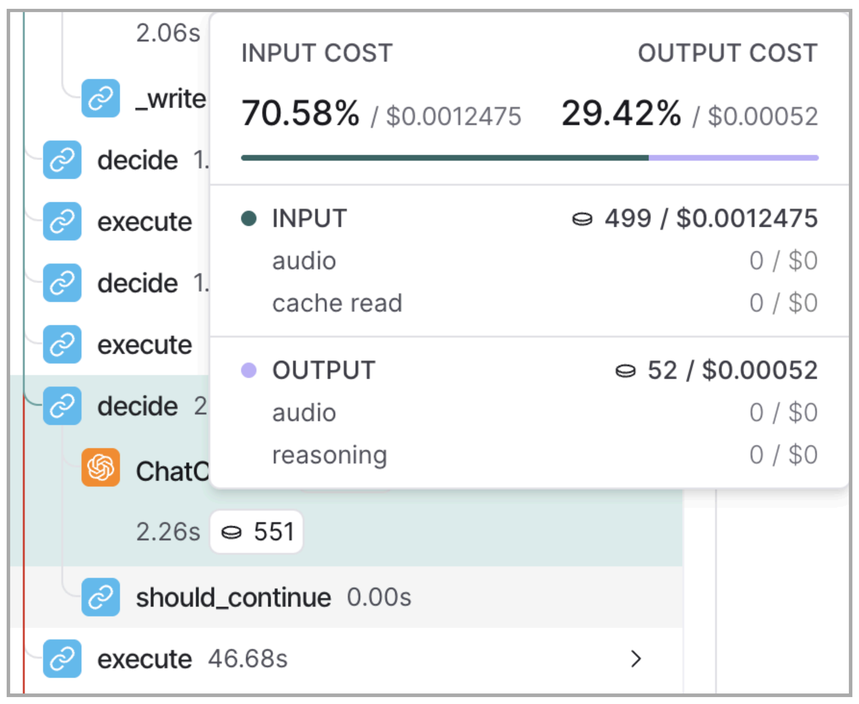
Sync - Plan-Decision-Execute DeepSync 구현

Decision 단계

```
n_completed", "message": "결정 완료", "reason": "The page content is missing, which is necessary to proceed with the subsequent steps
, "next_step": "ReadPage"}

art", "step_type": "tool", "step_number": 1, "purpose": "The page content is missing, which is necessary to proceed with the subsea
 issues.", "tool": "ReadPage"}
```

현재 상태를 판단 후 다음 Step 을 결정을 반환한 Response



시스템 프롬프트만 있어 토큰이 절약되는 모습

Decision 단계는 LLM이 현재 상태를 분석하고 다음에 수행할 Step 을 동적으로 판단하는 제어 레이어입니다.

Plan의 순서를 따르되, 상황에 따라 생략·재시도·분기(Self-Correction) 를 수행하며

DeepSync의 워크플로우를 유연하고 효율적으로 제어합니다.

Decision 은 이 아키텍처의 핵심으로, 매 Step마다 State가 누적되더라도 전체 문맥을 다시 전달하지 않고 필요한 정보만 참조하기 때문에 평균 600토큰 이하로 작동하는 고효율 제어 단계입니다.

Execute 단계

다른 정보 전달 없이 State 의 정보만으로 Execute 되는 모습

Execute 단계는 Decision이 선택한 Step을 실제로 실행하는 단계입니다.

이미 모든 데이터가 State에 저장되어 있기 때문에,

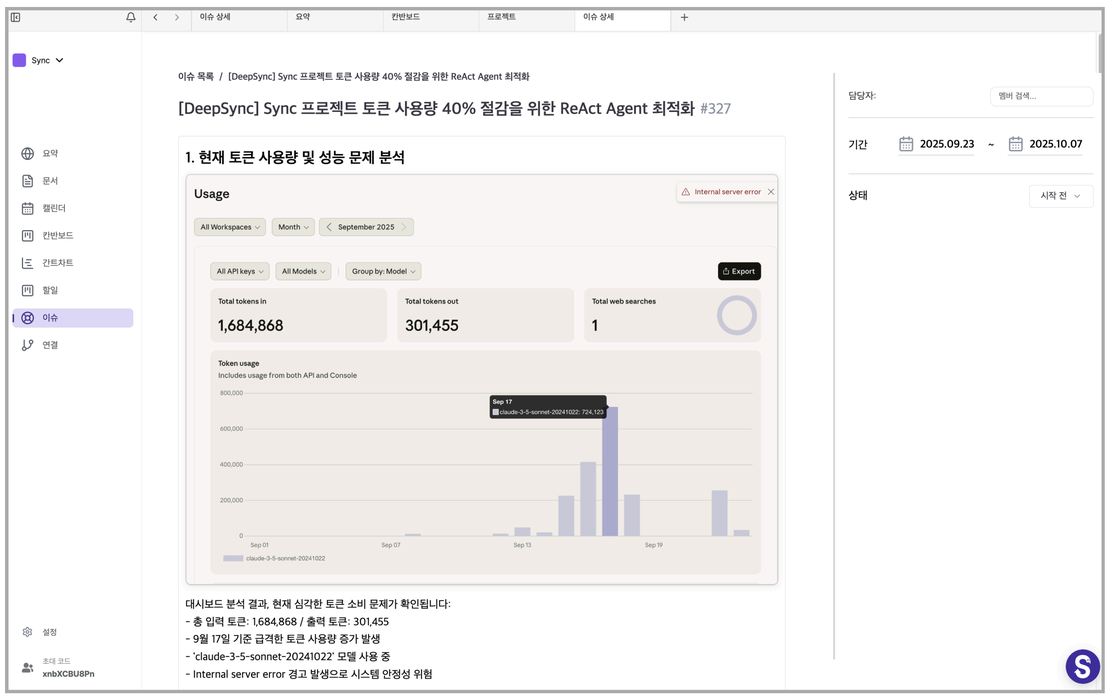
별도의 프롬프트나 추가 컨텍스트 전달 없이 즉시 실행이 가능합니다.

이를 통해 매 Step마다 불필요한 토큰 사용을 줄이고,

DeepSync의 도구 호출(예: 이슈 생성, 담당자 할당 등)을

가볍고 빠르게 수행할 수 있습니다.

Sync - 이메일 기반 AI 이슈 자동 생성



개요

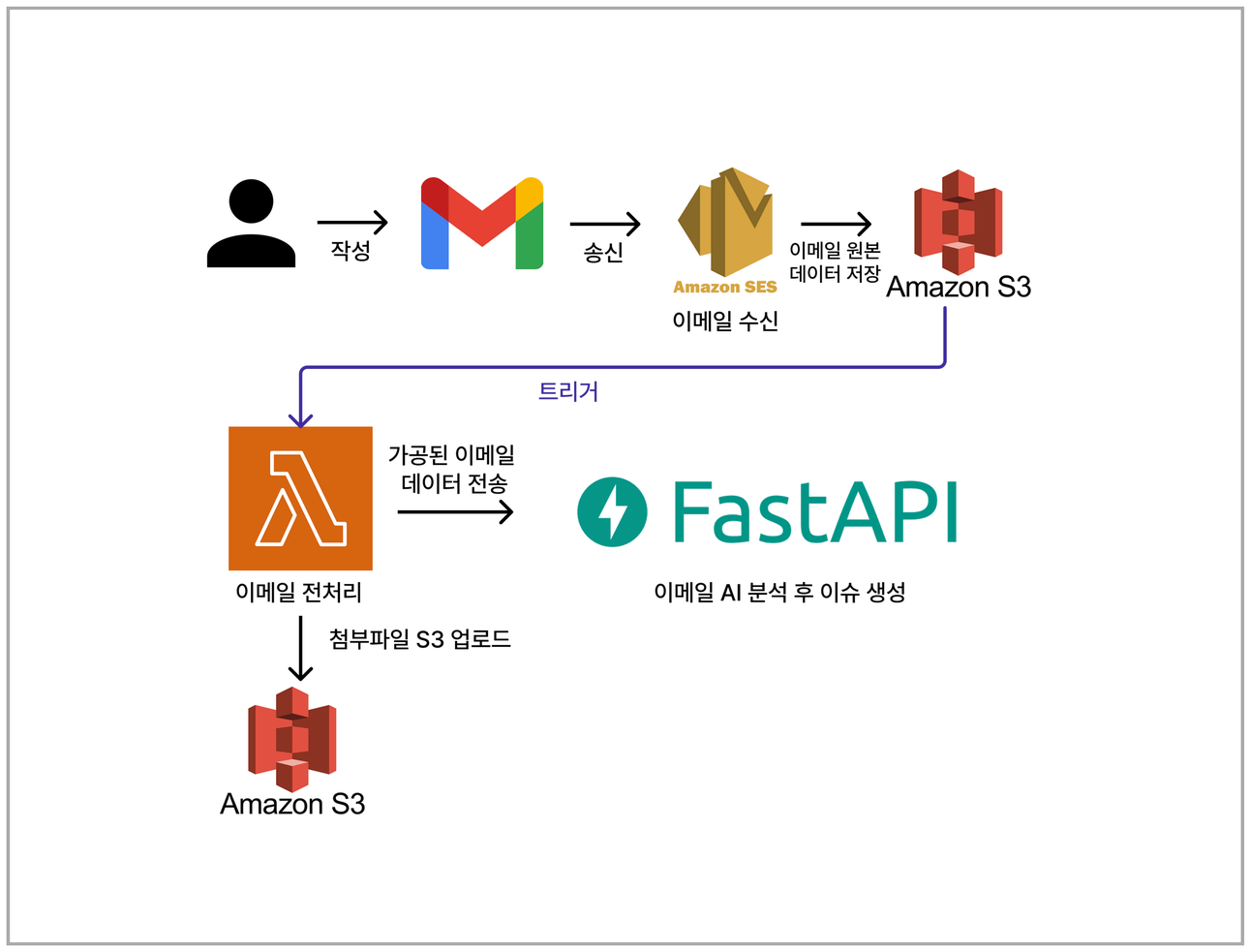
기획자가 슬랙 DM으로 “이거 버그인 것 같은데...”라고 보내거나, 이메일로 스크린샷 몇 장과 함께 문제를 설명하는 상황을 해결하기 위해, 이메일을 활용한 직관적이고 통합된 이슈 생성 기능을 개발했습니다.

FastAPI 기반 AI 서버에서 이메일의 제목, 본문, 첨부파일을 분석해 명확하고 구체적인 이슈로 자동 변환하도록 구현했습니다. 이 기능을 통해 이메일 한통으로 이슈를 생성할 수 있는 UX 를 제공했습니다.

이 기능의 기술적 구현 과정은 블로그에 자세히 정리했습니다.

velog - 이메일 한통으로 티켓 생성까지, AWS SES와 Lambda를 활용한 이메일 기반 AI 티켓 생성 기능

전체 플로우



Sync - 이메일 기반 AI 이슈 자동 생성

AWS 생태계를 활용한 아키텍처

AWS SES → Lambda → S3 → FastAPI로 이어지는 완전 자동화 파이프라인을 구축했습니다.

단순히 텍스트만 처리하는 것이 아니라, PDF·이미지·텍스트 파일을 모두 이해하고 분석할 수 있는 멀티모달 AI 시스템으로 구현했습니다.

PyMuPDF를 활용한 PDF 전처리

PDF 분석은 PyMuPDF 기반으로 구현했습니다.

각 페이지를 순회하며 page.get_images(full=True)로 이미지 리스트를 추출하고,

fitz.Pixmap으로 이미지를 PNG 바이트로 변환했습니다.

너무 작거나 의미 없는 장식용 이미지는 should_process_image() 함수를 통해 필터링하여

처리 속도와 효율을 최적화했습니다.

GPT-4o Vision API를 통한 이미지 관련성 판단

추출된 이미지는 GPT-4o Vision API로 전송되어,

"이 이미지가 실제로 이슈 해결에 도움이 되는가?"를 자동으로 판단합니다.

이메일 제목과 본문을 함께 컨텍스트로 제공하여

에러 스크린샷·UI 문제 이미지는 TRUE, 로고·장식 이미지는 FALSE로 분류했습니다.

이미지별로 설명·S3 URL·활용 팁 등을 메타데이터로 구조화하여

이후 이슈 생성 단계에서 활용할 수 있도록 했습니다.

이미지 해시 기반 캐싱 시스템

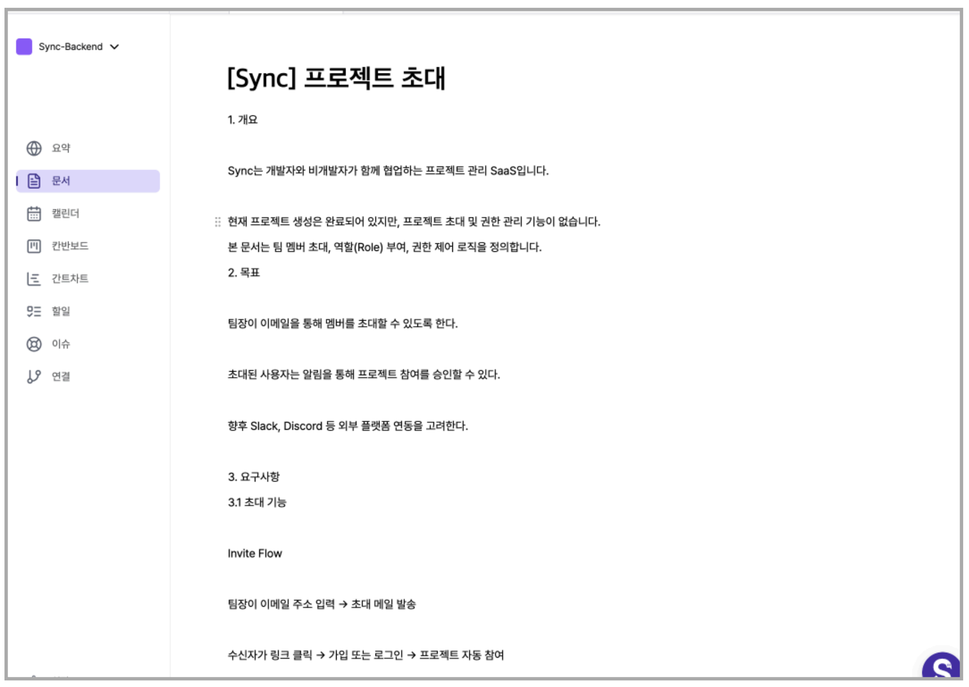
이미지 분석 속도를 높이기 위해 get_image_hash()로 이미지의 고유 식별자를 생성하고,

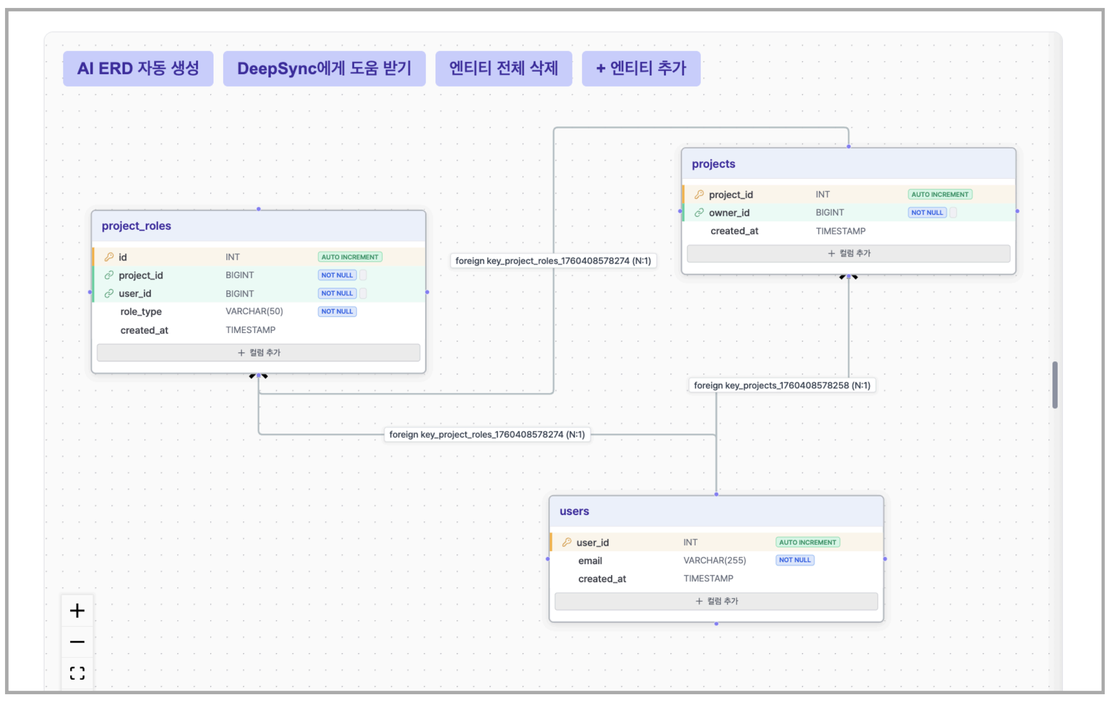
이메일 컨텍스트를 조합한 캐시 키를 만들어 중복 분석을 방지했습니다.

같은 이미지라도 다른 이메일 내용일 경우 서로 다른 캐시로 처리되도록 설계했으며,

캐시가 1,000개를 초과하면 LRU(Least Recently Used) 방식으로 오래된 항목부터 제거했습니다.

Sync - 기획 기반 자동 ERD 생성





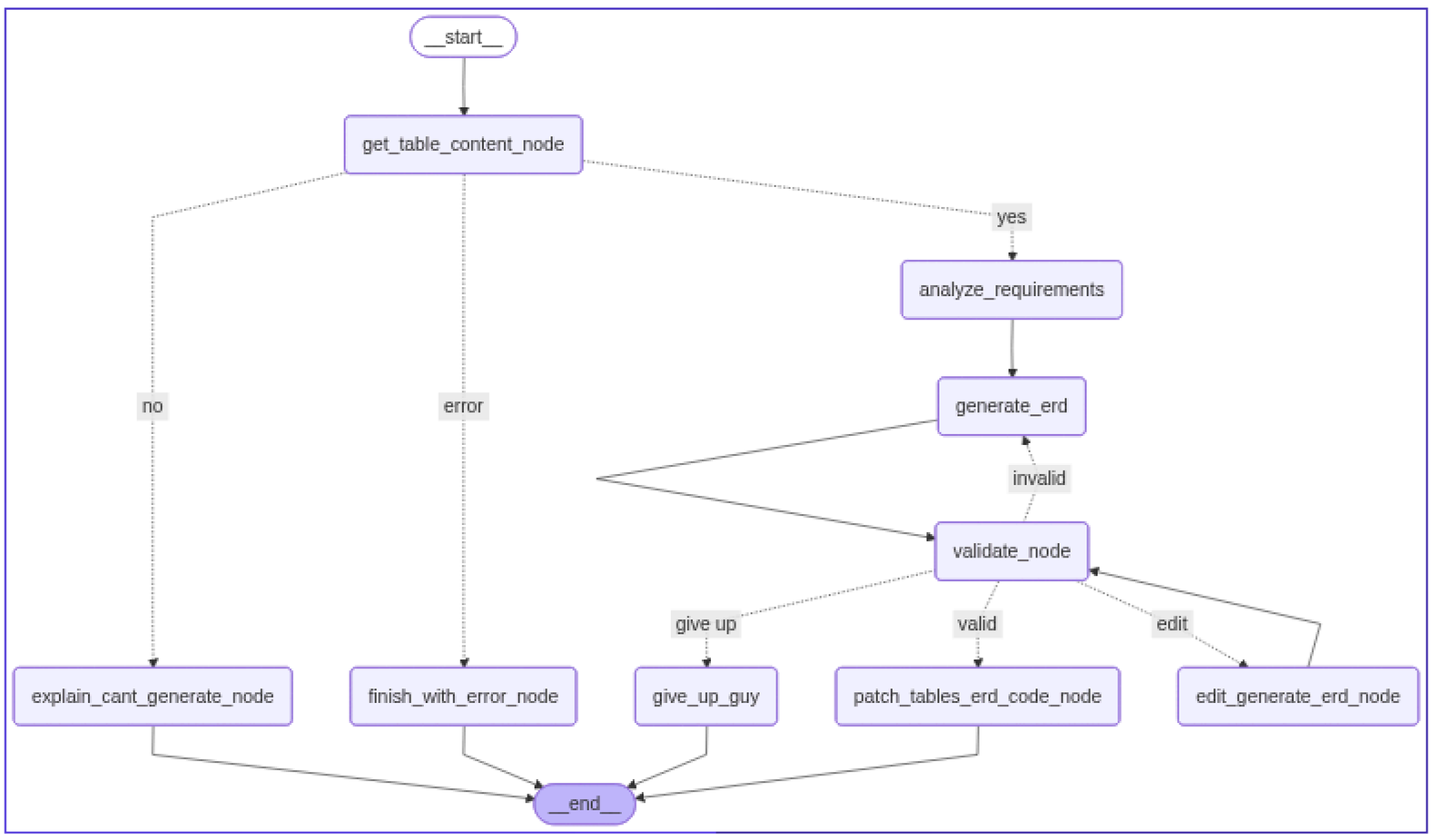
개요

기획을 바로 검증할 수 있게 ERD 로 생성해주는 기능 입니다.

기능명세서의 내용을 요약한 후 LLM 이 추론 하여 SQL 코드로 반환해주는 로직을 가지고 있습니다.

vaildate_node 에서 sqlglot 를 활용해 파싱을 시도하고 실패할 시 edit_generate_erd_node 로 이동해 Self-Feedback 루프를 실행합니다.

전체 플로우



LangGraph 기반 워크플로우

Sync - 트러블 슈팅 1 - DeepSync Rate Limit


```
sues have not yet been created. The next logical step is to create these issues.", "tool": "CreateIssue"}

data: {"type": "step_completed", "step_number": 4, "step_type": "tool", "result": {"success": true, "step_type": "tool", "tool_name": "CreateIssue", "tool_result": {"status": "c
ompleted", "data": null, "error": null}, "purpose": "The issues to create are available, but the created_issue_ids are missing, indicating that the issues have not yet been crea
ted. The next logical step is to create these issues."}, "success": true}

data: {"type": "decision_start", "message": "다음 Step 결정"}

data: {"type": "decision_completed", "message": "결정 완료", "reason": "All previous steps have been completed except for assigning members to the created issues. Since project me
mbers and assigned members data is missing, assigning members is the logical next step to ensure tasks are allocated to team members.", "next_step": "AssignMember"}

data: {"type": "step_start", "step_type": "tool", "step_number": 5, "purpose": "All previous steps have been completed except for assigning members to the created issues. Since
project members and assigned members data is missing, assigning members is the logical next step to ensure tasks are allocated to team members.", "tool": "AssignMember"}

data: {"type": "step_completed", "step_number": 5, "step_type": "tool", "result": {"success": true, "step_type": "tool", "tool_name": "AssignMember", "tool_result": {"status":
"completed", "data": null, "error": null}, "purpose": "All previous steps have been completed except for assigning members to the created issues. Since project members and assig
ned members data is missing, assigning members is the logical next step to ensure tasks are allocated to team members."}, "success": true}

data: {"type": "decision_start", "message": "다음 Step 결정"}

data: {'error': 'Error code: 429 - {'error': {'message': 'Rate limit reached for gpt-4o in organization                          on tokens per min (TPM): Limit 30000, Used 3
0000, Requested 488. Please try again in 975ms. Visit https://platform.openai.com/account/rate-limits to learn more.', 'type': 'tokens', 'param': None, 'code': 'rate_limit_excee
ded'}}'}
```


문제 상황: 과도한 비동기 호출로 인한 Rate Limit 문제

DeepSync의 '기획 기반 자동 이슈 생성' 기능을 구현하는 과정에서 OpenAI Rate Limit 문제가 발생했습니다.

이 기능은 Analyze Agent가 기획 내용을 시나리오 단위로 세분화한 뒤,

각 시나리오를 개발자가 이해하기 쉬운 이슈로 변환하는 구조 입니다.

하지만 시나리오가 20개 이상으로 세분화될 경우 한 요청당 약 1,000토큰을 사용해

GPT-4o의 TPM(30,000)을 초과 해 Agent 실행이 중단되는 문제가 있었습니다.

문제 원인: 비동기 API 처리

다량의 이슈의 정보를 for 순회를 통해 동기식으로 생성한다면 과도한 시간이 소요되므로 asyncio 의 process_single_feature() 를 사용하여 여러개의 비동기 Task 를 생성한 후 asyncio.gather() 로 수집하는 형식으로 개발했습니다. 이 때문에 20개 이상의 시나리오가 동시에 실행되어 TPM 을 넘게 되는 문제가 발생했습니다.

해결방안 1: 간단한 타협

이슈를 생성해주는 GenerateIssue는 최적화를 한다고 해도 30,000 토큰을 넘게 Output 에 나올 확률이 높았습니다.

그래서 비용과 성능, TPM 을 비교해서 가장 적합한 모델인 gpt-5-mini 를 선택했습니다.

해결방안 2: 비동기 → 배치 처리

GenerateIssue 후에 호출 될 수 있는 도구인 AssignMember 도 이슈 수에 비례하여 Rate Limit 가

발생했습니다. AssignMember 는 자동으로 이슈에 적합한 양수인을 할당해주는 기능입니다.

하지만 이 도구는 한번 호출이 2~3 초 안으로 짧아,

비동기로 전부 호출하는 방식이 아닌 시나리오의 총개수 / 4 한 값 만큼 배치 처리를 하도록 수정했습니다.

Sync - 트러블 슈팅 1 - DeepSync Rate Limit

느낀점

배치 처리와 모델 변경으로 문제가 해결 됐지만 근본적으로는 Rate Limit 를 늘리는 것이 문제 해결 인것 같습니다.

검색을 해보니 Queue 및 토큰 버킷 알고리즘을 활용한 해결방법도 있어서 시도 해봐야겠다는 생각이 들었습니다.

Sync - 트러블 슈팅 2 - 이메일 기능

문제 상황: 한글 인코딩 문제

이메일이 FastAPI 서버로 정상적으로 전송되었지만, 일부 이메일의 한글이 깨져서 표시되는 문제가 발생했습니다.
이로 인해 AI가 본문 내용을 정확히 분석하지 못하는 상황이 종종 발생했습니다.

문제 원인: 비동기 API 처리

이메일 표준은 다양한 인코딩 방식(UTF-8, CP949, EUC-KR 등)을 지원하고,
또한 RFC 2047 규격을 따라 헤더를 별도로 인코딩합니다.
이러한 혼합 인코딩 구조로 인해 서버에서 한글이 올바르게 복원되지 못했습니다.

해결방안 : 멀티 디코딩 구현

decode_header() 함수를 통해 RFC 2047 디코딩을 우선 수행하고,
디코딩 실패 시 UTF-8 → CP949 → EUC-KR → ISO-8859-1 순으로
자동 복원(fallback)하는 다중 인코딩 복원 시스템을 구현했습니다.
이를 통해 어떤 이메일 클라이언트로 전송하더라도 한글이 100% 정확히 복원되었고,
AI가 모든 이메일을 완전한 형태로 분석할 수 있게 되었습니다.

Sync - 트러블 슈팅 2 - 이메일 기능

문제 상황: 첨부파일 전송 시 이메일 누락

초기 개발 단계에서 이메일 수신 시스템으로 Mailgun, Mailjet, SendGrid 등의
서드파티 서비스의 웹훅 기능을 사용했지만, 첨부파일과 함께 이메일 전송 시 누락되는 문제가 있었습니다.

문제 원인: 첨부파일 1MB 제한

이들 서비스의 인바운드 기능이 대부분 유료 플랜 전용이며,
무료 플랜에서는 첨부파일 용량이 1MB로 제한되어
실제 서비스 환경에서 첨부파일이 포함된 이메일을 처리할 수 없었습니다.

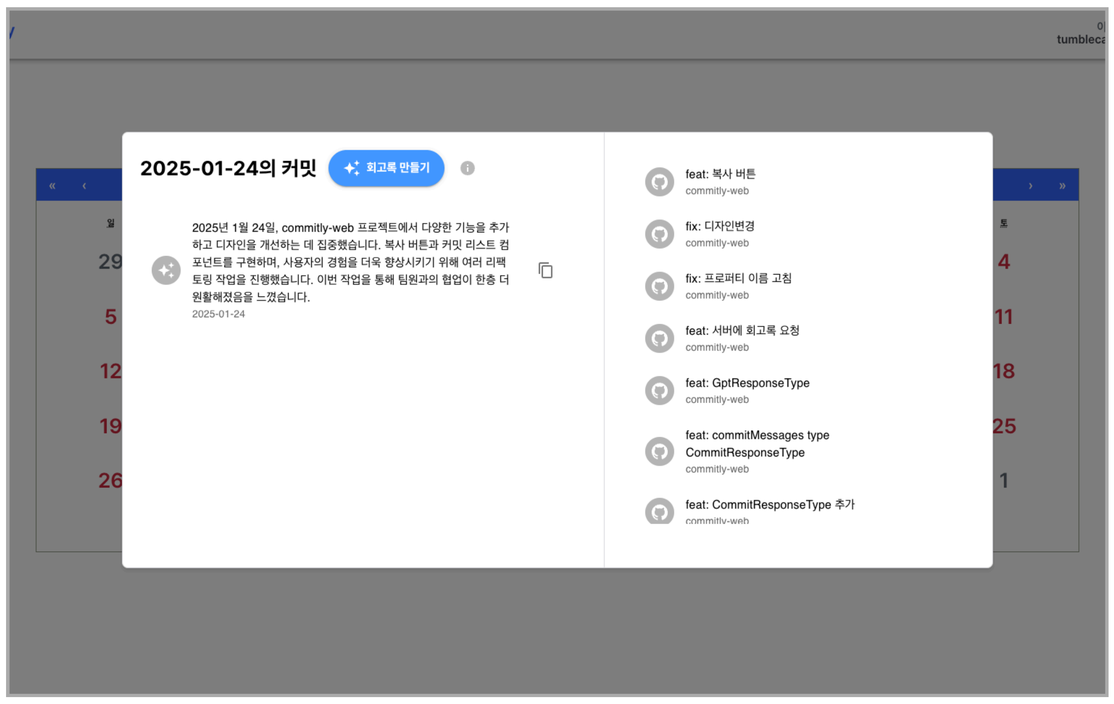
해결방안 : 기술 스택 변경

비용 효율성과 안정성을 고려해 Mailjet → AWS SES로 전환했습니다.
AWS SES를 통해 이메일을 수신하고, Lambda 트리거로 전처리한 뒤
S3에 저장된 데이터를 FastAPI 서버로 전달하는 구조로 재설계했습니다.
이를 통해 첨부파일 크기 제한과 누락 문제를 완전히 해결하고,
안정적인 SES → Lambda → S3 → FastAPI 파이프라인을 구축했습니다.

Side Project - Commitly

2025.03 ~

Commitly 랜딩 페이지



좌측: 커밋 기록 기반 회고록 생성, 우측: 해당 날짜의 커밋 기록

내가 사용한 개발 스택

Spring Boot | Spring AI | Kotlin | React | TypeScript

개요

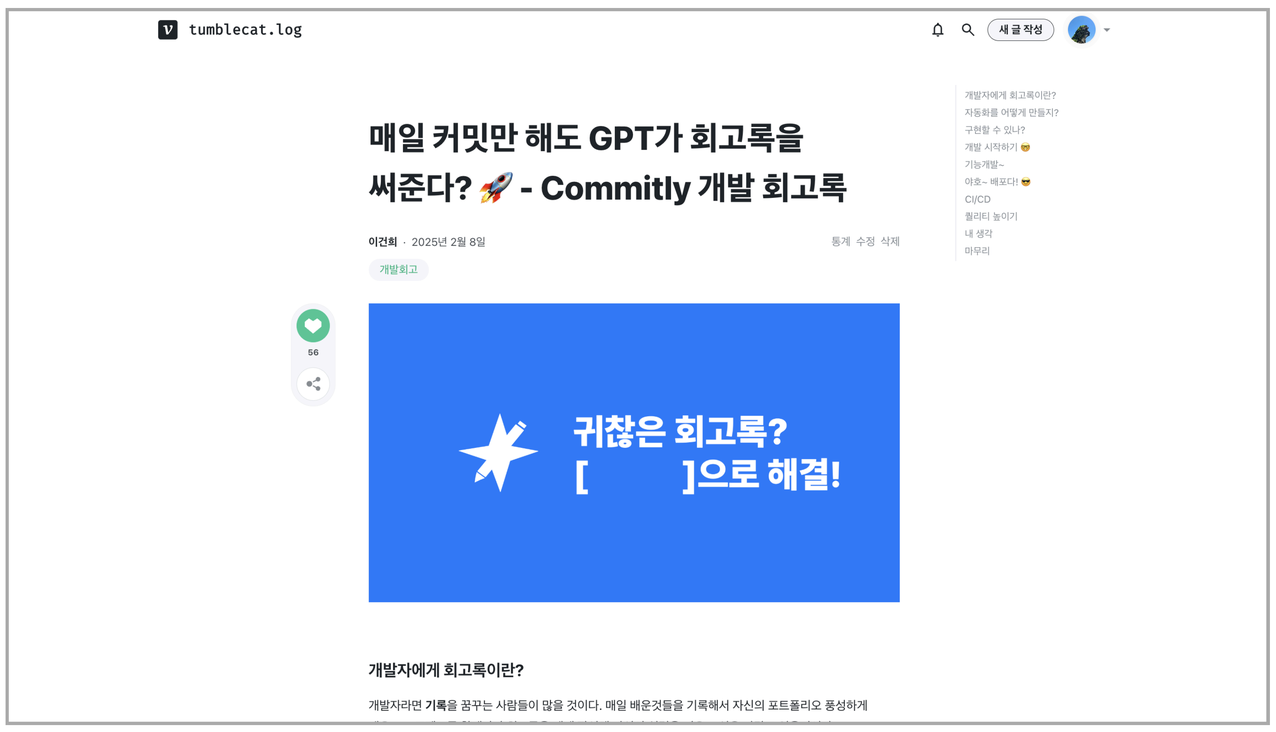
Commitly 는 깃허브 커밋 기록 바탕 AI 회고록 자동 생성 서비스입니다.

교내 동아리 B1ND 활동을 하며 회고록에 반복되는 회고록 작성에 부담을 느낀 것에서 시작 되었습니다.

주요 기능으로는 GitHub OAuth 로그인, GraphQL을 통한 커밋 데이터 수집 및 분석, 자동 회고 생성, UI 제공 입니다.

MS Clarity, Google Analytics 를 활용해 사용자 모니터링을 했으며 실제로 랜딩페이지에서 사람들이 전달되지 않는 클릭을 지속적으로 하는 문제를 파악했고 그 문제를 해결한 성과가 있습니다.

개발한 회고록을 velog 에 올려 좋아요 56개를 받았으며 이를 통해 49명의 사용자를 모았습니다.



Velog 에 작성된 회고록

Side Project - Commitly

2025.01 ~ 2025.02

느낀 점

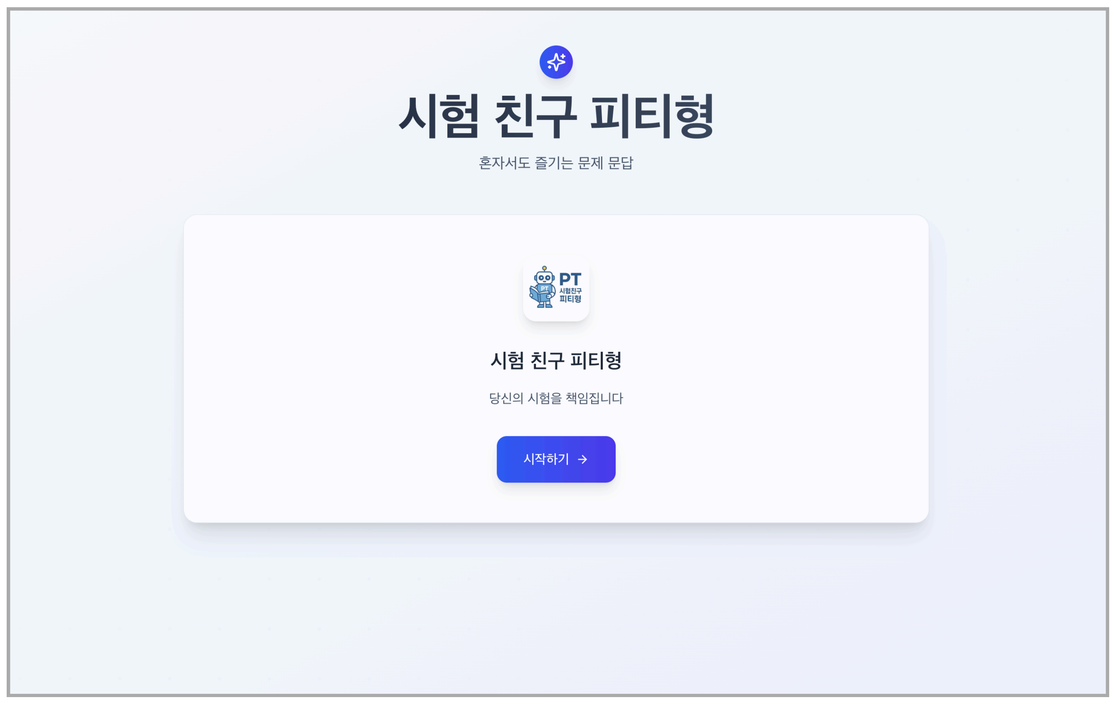
Commitly 는 처음으로 “내가 느낀 문제를 해결하기 위해 만든 서비스” 입니다. 이 경험으로 일상의 불편함을 유심히 관찰하게 됐고 이후 “시험친구 피티형"과 “Sync”의 아이디어를 고민할 때 정말 도움이 됐습니다.
CommitIly를 개발하기 전 까지는 Android 개발만 하여 React 와 Spring Boot 의 지식이 없었습니다.
그래서 개발을 할 때 ChatGPT에게 물어가며 서비스를 완성해나갔습니다.
이를 통해 어떤 기술 스택이든 ChatGPT 와 함께라면 “기획"부터 “서비스"까지 전부 만들 수 있겠다는 자신감을 얻게 되었습니다.
이 경험으로 “새로운 기술 스택” 에 대한 공포와 거부감은 전부 사라지게 되었고 모르는 분야여도 “까짓 것 한번 해보죠 뭐” 하는 사고방식을 가지게 됐습니다.
실제로 Sync 를 기획할 때 이 경험을 바탕으로 한번도 안써봤던 기술스택인 FastAPI 와 LangChain, LangGraph 을 선택하게 됐습니다.

아쉬운 점

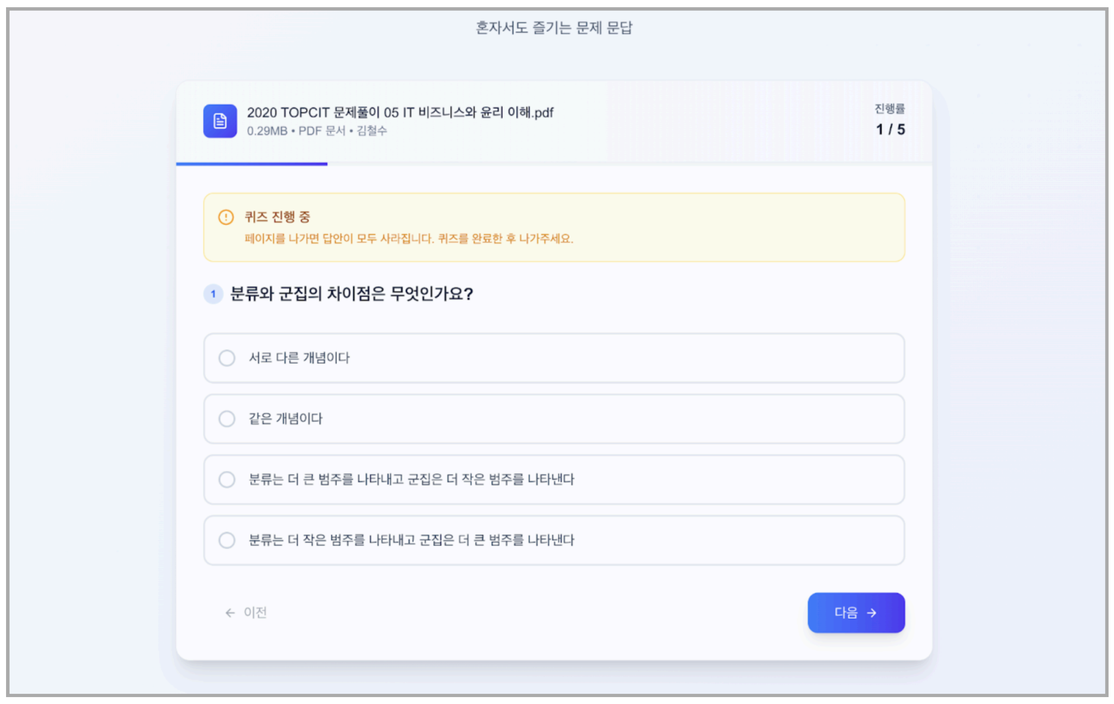
처음 개발할 때 “프로덕트" 보단 “공부"에 목적을 두고 진행하여 완성한 후 유지보수를 전혀 하지 않은 체 방치되어 정말 아쉽습니다.
2025년의 겨울방학에 창업을 목표로 유저를 모으는 서비스를 만드는 걸 생각하고 있습니다.

Side Project - 시험친구 피티형

2025.06



시험친구 피티형 랜딩 페이지



소프트웨어 공학 PDF 를 넣고 생성된 문제

내가 사용한 개발 스택

Next.js | LangChain.js | JavaScript

개요

시험친구 피티형은 PDF 기반 시험문제 대비 자동 문제 생성 서비스 입니다.

Next.js 를 사용하여 JS 코드 하나로 풀스택 개발을 하는 경험을 했고 Vercel 과 Next.js 의 연동성을 알게 됐습니다.

LangChain.js 를 활용하여 RAG 와 PDF 전처리를 했습니다.

디스코드 웹훅을 추가해 시험을 친 사람을 디스코드에서 확인할 수 있게 만들었습니다.

실제 디스코드 웹훅

Side Project - 시험친구 피티형

2025.06

기획 이유

시험기간에 친구들끼리 문제를 내고 맞추는 모습을 자주 봤습니다.

친구들에게 문제를 내달라고 하기에는 반이 너무 조용하거나 공부를 하고 있어 말을 못 걸 때에는 문제 문답을 못하는 문제(Problem) 을 알게 됐습니다.

여기서 LLM 이 친구를 대신할 수 있지 않을까? 아이디어를 얻게 됐고 대구소프트웨어마이스터고등학교의 대부분의 과목은 학습지를 PDF로 제공해준다는 것과 결합하여 PDF 기반 AI 문제 생성 및 시험 서비스를 만들게 됐습니다.

느낀 점

시험친구 피티형을 개발하며 PDF 처리에 감을 익히게 됐습니다. 이 경험으로 Sync 의 이메일 기반 AI 이슈 생성을 구현할 수 있게 됐습니다.

JS LangChain 과 Python LangChain 에는 차이가 있다는 것을 알게 됐습니다

사용자 문의사항, 시험 로깅을 디스코드를 활용해 기능을 만들었습니다. 이를 통해 “기능" 에는 꼭 DB 나 서버 같은 기술이 필요한게 아니라는 것을 느끼게 됐습니다.

아쉬운 점

2025학년도 1학기 웹프로그래밍 기초 수행평가로 낸 프로젝트라 이 서비스도 “프로덕트”로 발전시키지 않아서 아쉽습니다.