

ballbot.dev

816만 유저의 서비스를 지탱해온

소프트웨어 엔지니어 김수빈입니다.

프로젝트 보기

GitHub ↗

Overview

2 Years

넛지헬스케어 타임스프레드, 링커리어 팀 백엔드

- 19억 건의 테이블을 점검 시간 1시간 내에 마이그레이션
- 분당 1,500건의 요청을 받는 서비스 리팩토링
- 월 1,000만 원을 넘기던 AWS 비용을 30% 절감
- 메모리 32GB 를 다 쓰고 멈추던 API 서버 정상화

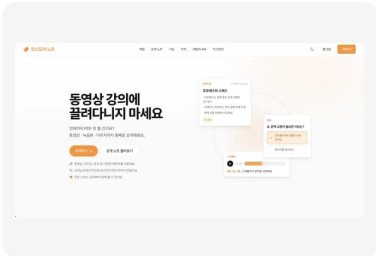
2 Awards

- 코스모의 노트, 전국 대회 최우수상 1위 달성
- HEALIX, 1,500명 규모 전국 해커톤 2위 달성

4.43 / 4.5

강원대학교 컴퓨터공학과 재학

Projects



2025 ~ 현재

 코스모의 노트

밀린 동영상 강의를 빠르게 따라잡기 위한 영상 요약 서비스입니다. 음성뿐 아니라 화면 속 슬라이드까지 읽고, 구독료 없이 쓴 만큼만 받도록 만들었습니다.

cosmonote.site ↗

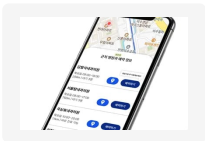


2026 ~ 현재

 enqor

실시간 영상·음성으로 전문가와 1:1 상담하는 서비스입니다. 초 단위 통화 정산과, AI 에이전트의 결과물을 사람 병목 없이 검증하는 하네스를 만들었습니다.

[App Store](#) ↗ [Google Play](#) ↗

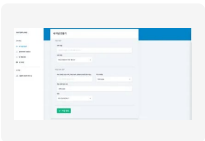


2024

HEALIX

증상을 바탕으로 가장 가깝고 적절한 병원을 추천하는 서비스입니다. 공공데이터포털의 병원 정보와 앱 내 GPS 데이터를 AI 에 전달하게 만들었습니다.

[GitHub](#) ↗



2023

WATERFLAKE

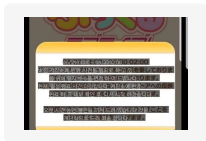
간단한 설치로 외부 접속이 막힌 게임 서버에 접속 가능한 도메인을 발급해주는 서비스입니다. TCP 터널링과 SRV 레코드를 핵심으로 사용했습니다.

[waterflake-api](#) ↗

[waterflake-front](#) ↗

[waterflake-tunnel](#) ↗

[waterflake-plugin](#) ↗



2017 ~ 2018

스크린번역기

휴대폰을 흔들면 화면을 캡처해 오버레이로 번역문을 보여줍니다. 중학생 때 처음 만들어 수익화까지 성공했습니다.

[GitHub](#) ↗

Experience

2021.12 ~ 2023.12

Nudge Healthcare

Backend Engineer

타임스프레드, 링커리어 팀



19억 건의 테이블을 점검 시간 1시간 내에 마이그레이션

웁길 데이터에 우선순위를 정하고, 언제든 되돌릴 수 있는 마이그레이션 스크립트를 준비했습니다.



분당 1,500건의 요청을 받는 서비스 리팩토링

DynamoDB 와 Redis 로 다시 설계해, 6시간에 한 번 갱신되던 랭킹을 실시간으로 움직이게 했습니다.



월 1,000만 원을 넘기던 AWS 비용을 30% 절감

값싸고 성능 좋은 Graviton 인스턴스로 옮기고, 인스턴스 증감 규칙을 다시 잡았습니다.

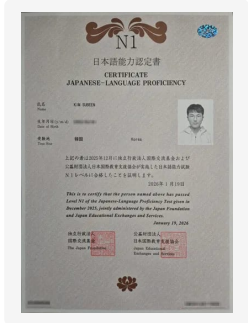


메모리 32GB 를 다 쓰고 멈추던 API 서버 정상화

아무 오류도 남기지 않고 메모리를 모두 써버리던 문제를, 상용 서버의 메모리 덤프를 떼서 잡았습니다.



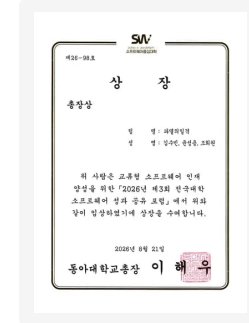
Certificates



2026.02
JLPT N1
일본국제교류기금



2026.09
정보처리기사
한국산업인력공단



2026
제3회 전국대학 소프트웨어 성과
공유포럼
최우수상 1등
동아대학교 총장상
코스모의 노트로 전국 13개의 팀이
모인 대회에서 수상했습니다.



2024
멋쟁이사자처럼 대학 12기 중앙 해커톤
최우수상 2등
영림원소프트랩 특별상
HEALIX로 전국 55개 학교
1,500여 명이 모인 대회에서
수상했습니다.



2025
국가우수장학(이공계)
과학기술정보통신부장관 증서
전국에서 1,000여 명만
선발되는 국가우수장학금
(이공계)을 받았습니다.



2025
**Daangn Builder's
Camp**
당근마켓 수료증
당근마켓이 20여 명만 뽑는
해커톤에 선발되었습니다.

2013 ~ 2018

초·중학교

○ 2014

리눅스 환경에서 마인크래프트 24시간 서버 구축

VPS 를 빌려 서버를 열고, 최고 동시접속자 120명을 기록했습니다.

● 2015

초등학교 졸업

장래희망으로 적은 컴퓨터 프로그래머의 꿈이 지금까지 이어지고 있습니다.

○ 2015

텍스팩 커뮤니티 개발

PHP 로 동작하는 소모임 서비스를 개발해 가입자 100여 명을 모았습니다.

○ 2017

스크린번역기 개발

휴대폰을 흔들면 화면 위에 번역 오버레이를 띄워주는 앱을 개발해 다운로드 1,000건과 수익화를 경험했습니다.

2018 ~ 2021

선린인터넷고등학교

소프트웨어과

● 2018

모바일콘텐츠경진대회 수상

스크린번역기로 교내 대회에서 2등을 수상했습니다.

강원대학교

컴퓨터공학과

2021.12 ~ 2023.12 넗지헬스케어

Backend Engineer
타임스프레드, 링커리어 팀

○ 2019

VR 게임 개발

언리얼 엔진을 활용해 **VRTetris, Tooth, Unrevived** VR 게임을 개발했습니다.

○ 2020

모바일 게임 개발

언리얼 엔진을 활용해 **MyRunnerGame** 모바일 게임을 개발했습니다.

2021

● 2021

KNU 코딩플랫폼 개발

강원대학교 SW중심대학사업단에서 진행한 코딩 플랫폼 구축 사업에서 팀장 역할을 맡았습니다.

● 2022

메모리 32GB 를 다 쓰고 멈추던 API 서버 정상화

● 2022

월 1,000만 원을 넘기던 AWS 비용을 30% 절감

○ 2023

19억 건의 테이블을 점검 시간 1시간 내에 마이그레이션

○ 2023

분당 1,500건의 요청을 받는 서비스 리팩토링

2024 ~ 현재

강원대학교

컴퓨터공학과

- 
- 2024 **멋쟁이사자처럼 대학 12기 중앙 해커톤 2등**
HEALIX로 전국 55개 학교 1,500여 명이 모인 대회에서 수상했습니다.
 - 2024 **국제교류처 버디 프로그램**
일본에서 온 교환학생 한 명을 맡아 학교생활을 돕고, 동아리에서 좋은 추억을 만들 수 있도록 도왔습니다.
 - 2025 **국가우수장학(이공계)**
전국에서 1,000여 명만 선발되는 국가우수장학금(이공계)을 받았습니다.
 - 2025 **일본 돗토리대학교 교환학생**
일본에서 한 학기를 지내며 다도부에서 전통 문화를 배웠습니다.
 - 2026 **제3회 전국대학 소프트웨어 성과 공유포럼 1등**
코스모의 노트로 전국 13개의 팀이 모인 대회에서 수상했습니다.
 - 2026 **Align Networks 리드 엔지니어**
enqor 프로젝트 개발 전체를 담당하고 있습니다.

2025 ~ 현재

코스모의 노트

동영상, PDF 등 수업 자료 요약 서비스

cosmonote.site ↗

참가 인원

1명(개인)

개발 기간

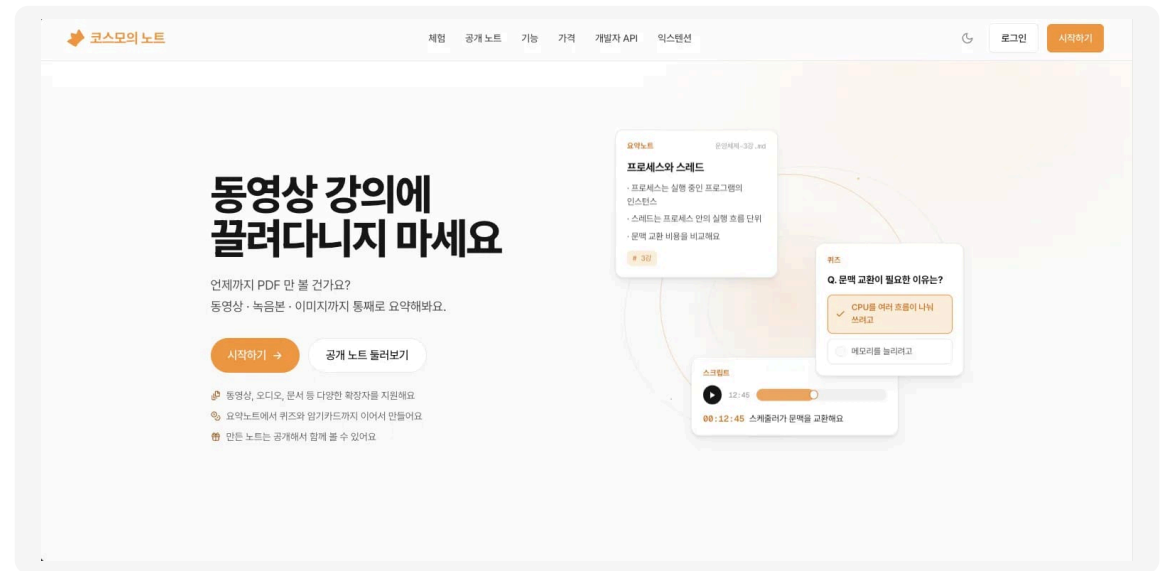
2025.05 ~ 현재

주요 기술

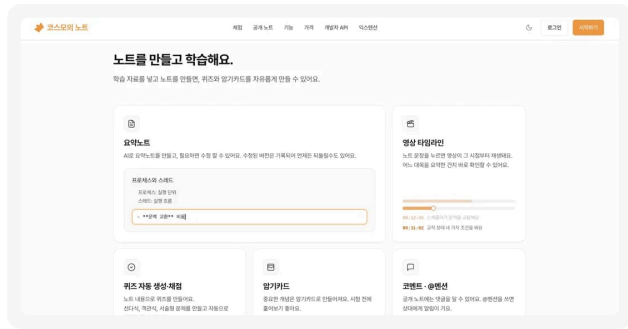
Cloudflare Workers, R2, Containers

수상

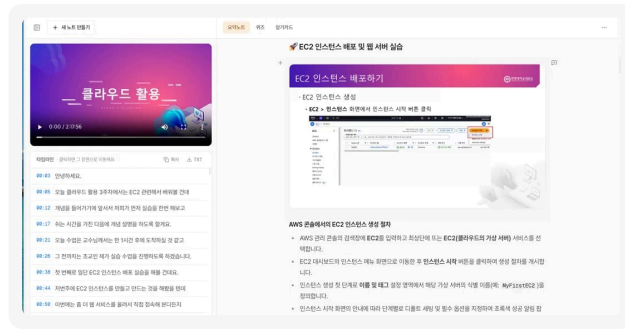
전국대학 소프트웨어 성과 공유포럼 최우수상



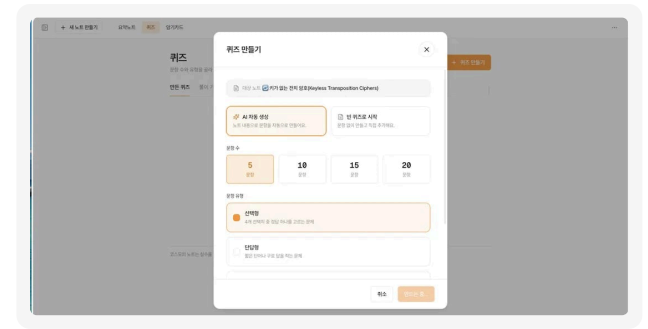
랜딩



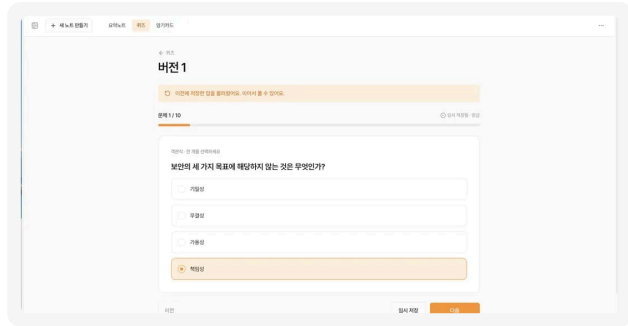
기능



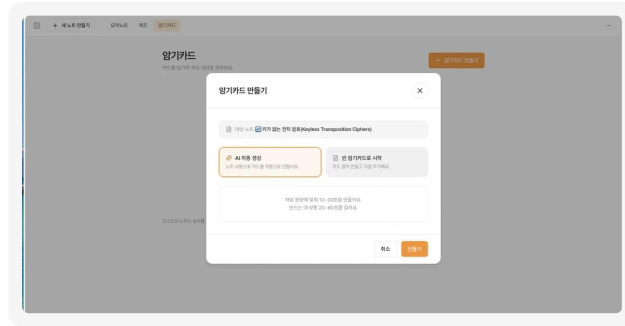
요약노트



퀴즈 만들기



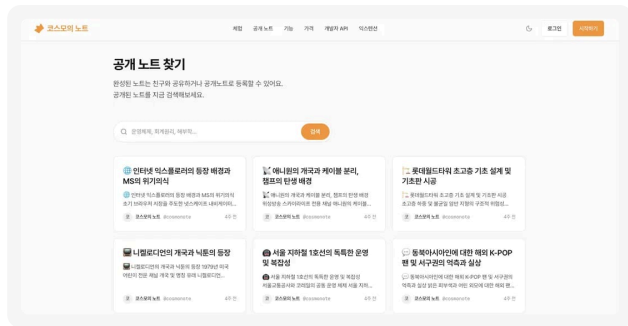
퀴즈 풀기



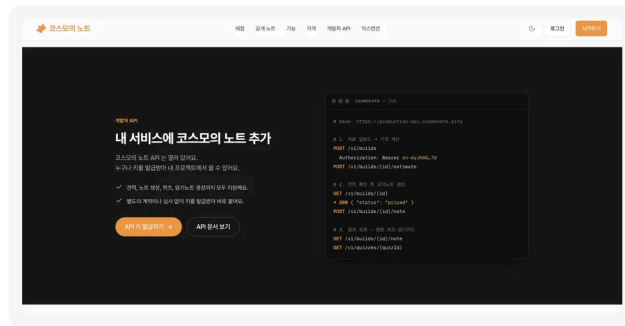
암기카드 만들기



암기카드 학습



공개 노트



개발자 API

Highlights

- 화면 속 슬라이드까지 읽어 하나의 타임라인으로 만들기
- 사용자 한 명을 위해 24시간 고성능 머신을 띄워둬야 하는 비용 문제 해결하기
- 플래그십 모델을 고효율 모델로 바꾸면서 노트 품질은 지켜내기

Motivation

부전공인 클라우드학과는 수업이 대부분 동영상 강의로 진행됐습니다. 기초 과목은 이미 아는 내용이 태반인데도, 시험과 과제 때문에 아는 강의를 처음부터 다시 틀어놓고 있어야 했습니다.

한동안은 강의 영상을 내려받아 타사 요약 서비스에 넣어 썼습니다. 시험 2주 전에 몰아 쓰고 두 달을 손도 안 대는 게 학생의 사용 패턴인데, 월 구독제라 쓰지 않는 날에도 비용을 지불해야 했습니다. 영상을 불러오는 것부터 요약까지 한 번에 끝내면서, Pay As You Go 형태의 서비스를 만들어보자는 게 코스모의 노트의 시작이었습니다.

History

2025.05

코스모스 다운로더



처음에는 강의 시청 화면 오른쪽에 다운로드 버튼을 붙여주는 브라우저 익스텐션을 만들었습니다. 버튼을 누르면 익스텐션이 웹 통신 내용을 읽고, 스트리밍 파일의 주소를 서버로 보냅니다. 서버는 ts 스트리밍 파일을 내려받아 mp4 로 인코딩한 뒤, 다운로드 링크를 주는 간단한 구조였습니다.

익스텐션을 만들어 배포한 뒤 커뮤니티에 게시하니, 학우 480명 이상이 좋아요를 눌렀습니다. 다운로드만 가능한 프로그램임에도 뜨거운 관심을 받았다는 점은 이 불편함이 저만의 문제는 아니라는 뜻이었습니다.

2025.08

요약 기능 추가

9주차 [4월29일 - 5월05일]

5/5(월)<6/18(수) 대체>, 5/8(목) 2일치를 동영상 강의로 대체합니다.

5/11(일) 까지 수강해야 출석으로 인정합니다.

▶ OS 9주차-1 2025-04-29 00:00:00 ~ 2025-05-11 23:59:59, 20:28

다운로드 AI 노트로 요약

▶ OS 9주차-2 2025-04-29 00:00:00 ~ 2025-05-11 23:59:59, 24:22

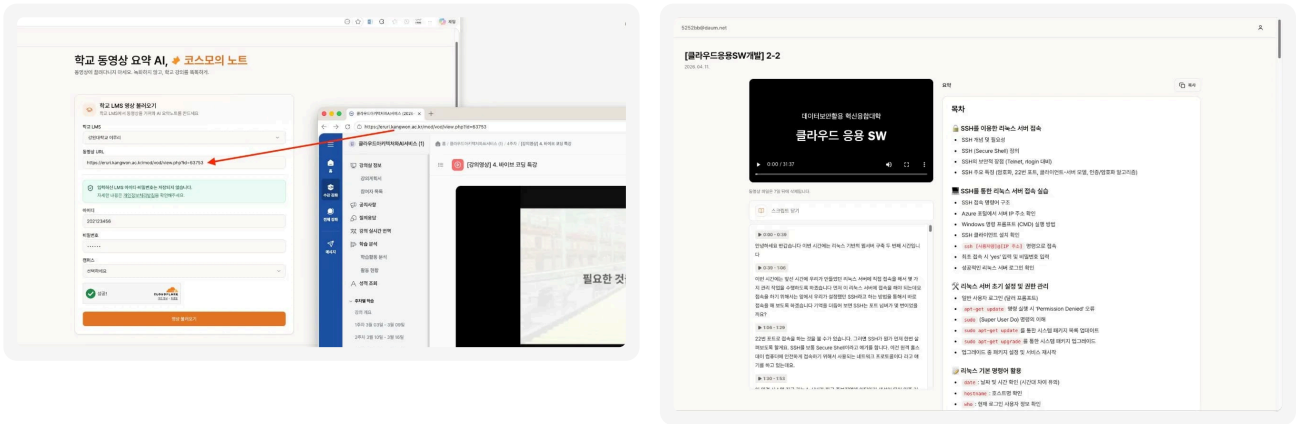
다운로드 AI 노트로 요약

AI 노트 생성 기능을 붙이고, 다운로드 버튼을 강의 홈 화면으로 옮겼습니다. 버튼을 누르면 Cloudflare Browser 로 서버가 직접 강의 사이트에 접속하도록 바꿨습니다. 로그인은 익스텐션이 넘겨준 세션 키를 서버 쪽 브라우저에 심어 우회했습니다.

익스텐션이 하던 일을 하나씩 서버로 옮겨두는 작업이었습니다. 익스텐션을 없애고 완전히 웹 서비스로 동작하게 만드는 게 목표였습니다. 더불어 결제를 연동해 수익화 구조도 만들어봤습니다.

2026.04

익스텐션 제거

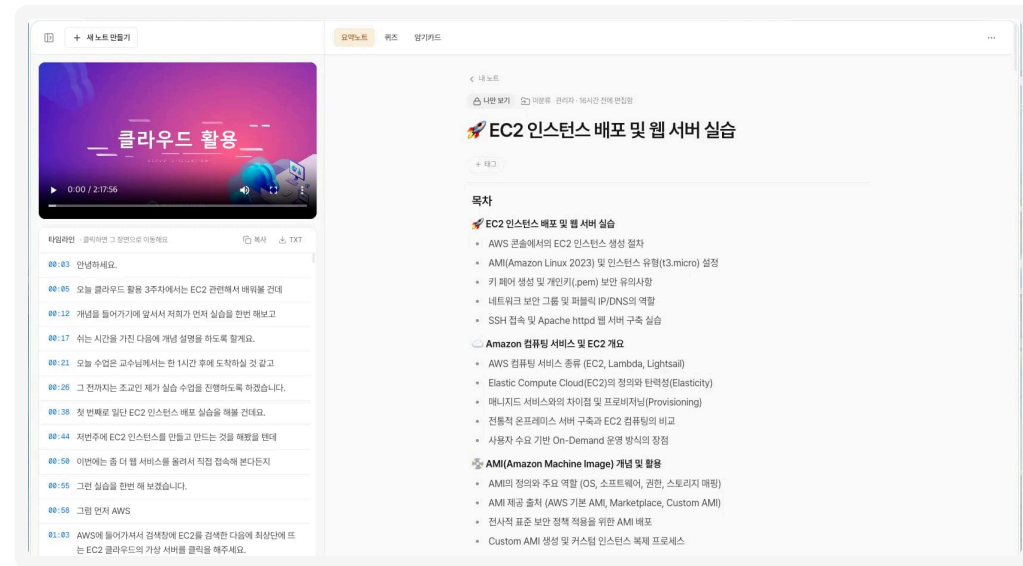


교환학생을 다녀온 뒤, 다시 서비스를 업데이트하기 시작했습니다. 가장 큰 목표는 브라우저 익스텐션을 완전히 제거하는 일이었습니다. 브라우저 익스텐션은 사용자의 브라우저에 깊게 접근할 수 있다는 점은 좋지만, 모바일에서는 전혀 동작하지 않고 특정 브라우저에 종속되는 문제가 있습니다. 무엇보다 익스텐션을 아는 사람이 별로 많지 않았습니다.

그래서 웹 사이트만으로 동작하도록 대대적으로 고쳤습니다. 학교 사이트 로그인 정보와 강좌 URL 을 입력받으면 Cloudflare Browser 로 직접 로그인하고, 패킷을 탐지해 영상을 찾아내는 구조입니다. 로그인 정보는 저장하지 않게 만들었습니다. 소프트웨어는 복잡해지고 버그도 늘었지만, 익스텐션을 없애니 사용성은 비교할 수 없이 나아졌습니다.

2026.09

범용 영상 요약 사이트로 리뉴얼



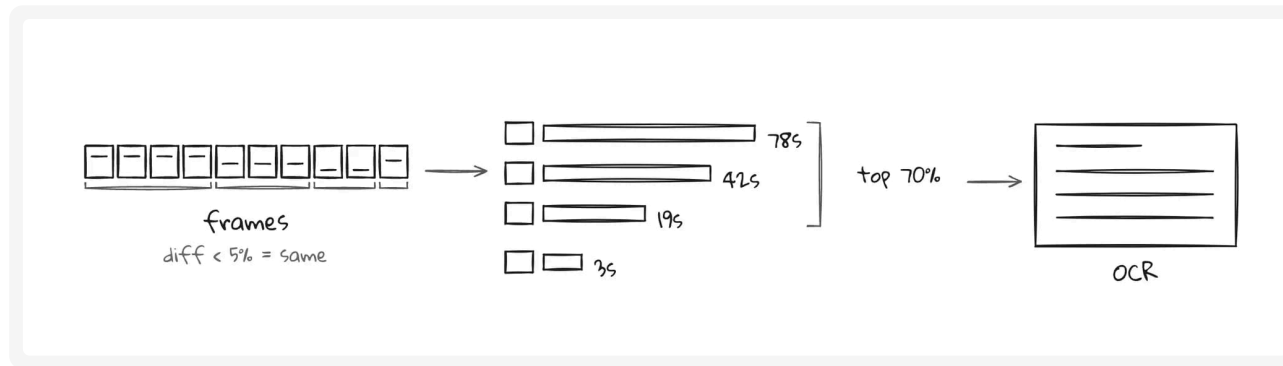
기존 구조는 학교 사이트에서만 영상을 불러올 수 있었습니다. 성장에 걸림돌이기도 했지만, 저작권이 있는 학교 강의를 가공해 돈을 받는 일이라 법적인 리스크를 피할 수 없었습니다.

비록 매출이 발생하기 시작해 기뻐했지만, 학교 사이트 연동 기능을 폐기했습니다. 다만, 이 서비스의 정체성인 만큼 완전히 버리지는 못하고 익스텐션으로 되돌렸습니다. 동영상과 녹음본, 이미지, 문서를 직접 올리는 방식으로 리뉴얼하여 현재 모습에 이르렀습니다.

Architecture

화면 속 슬라이드까지 읽어 하나의 타임라인으로 만들기

대부분의 강의 요약 서비스는 음성을 STT 로 받아쓴 다음, 그 텍스트를 요약합니다. 그런데 수업 영상에서 정작 중요한 것은 프레젠테이션에 많이 있고, 강의자가 이를 읽지 않고 넘어가는 경우도 많습니다. 슬라이드에서 나온 수식, 도표가 그대로 유실되는 문제가 발생합니다.



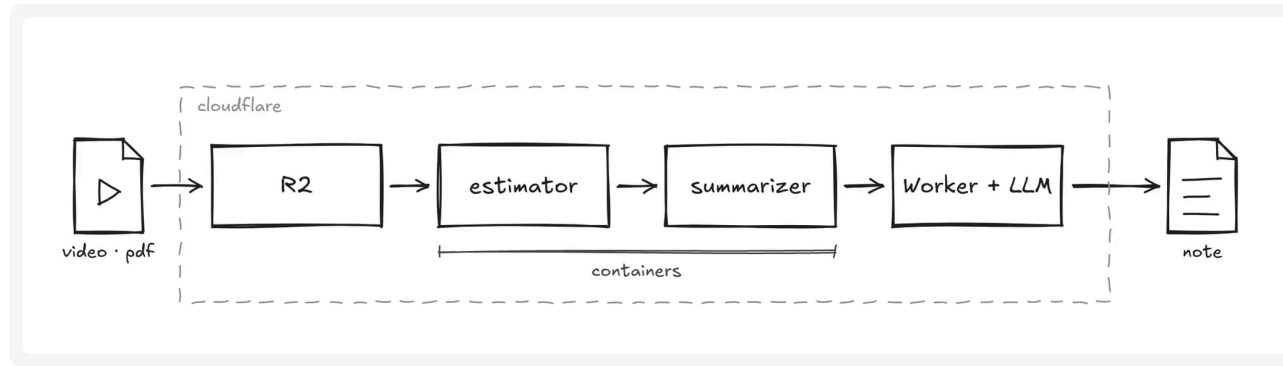
변화가 없는 구간을 묶어 오래 머문 화면만 읽는 흐름

따라서 코스모의 노트는 영상 프레임을 분석하는 로직을 넣었습니다. summarizer 는 ffmpeg 를 이용해 프레임의 변화를 추적합니다. 프레임의 변화율이 5% 미만이라면 같은 프레임으로 보고, 같은 프레임이 오랫동안 유지되는 경우를 랭킹으로 만듭니다. 그리고 상위 70% 에 해당하는 프레임을 찾아 OCR 을 수행합니다.

STT 결과와 OCR 결과를 하나의 타임라인으로 병합합니다. 마지막으로 Worker 에 이 타임라인을 전달하고, 사전에 튜닝된 프롬프트로 요약노트를 생성해냅니다.

사용자 한 명을 위해 24시간 고성능 머신을 띄워둬야 하는 비용 문제 해결하기

코스모의 노트는 ffmpeg 로 영상 프레임을 처리해야 하기 때문에, 그 순간에는 꽤 좋은 머신을 요구합니다. 고성능 인스턴스를 상시로 띄워두면 아무도 사용하지 않는 시간에도 비용이 발생합니다. 따라서 영상 처리 기능은 서버리스 구조를 채택하여 비용을 크게 절감했습니다.



업로드부터 노트 생성까지

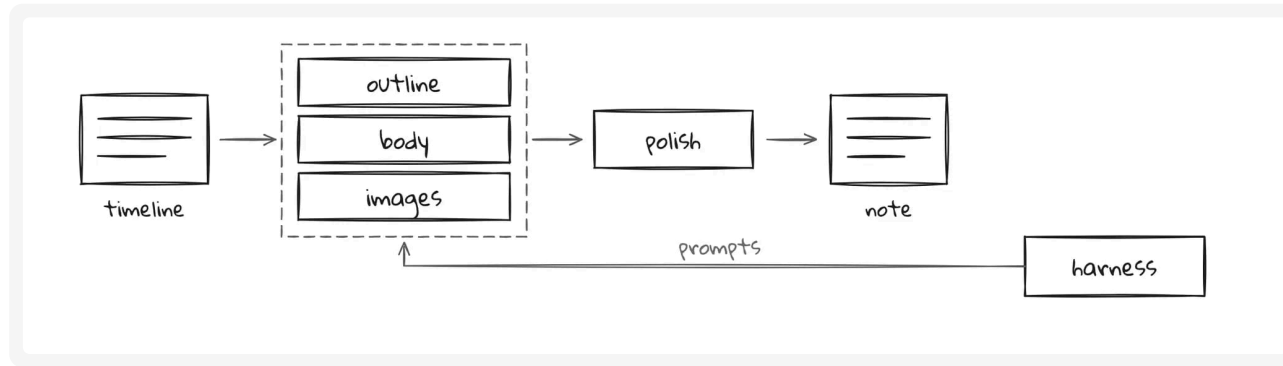
Workers 는 최대 100MB 까지만 업로드를 받을 수 있습니다. 동영상의 100MB 미만인 경우는 거의 없으므로, 서버는 버킷에 바로 업로드할 수 있는 Presigned URL 을 발급해 클라이언트에 제공합니다. 파일은 클라이언트에서 R2 에 바로 업로드됩니다.

업로드된 파일은 estimator 가 검사합니다. 깨지거나 잘못된 파일이 아닌지 확인하고 문제가 없으면 샘플링해 비용을 예측합니다. 사용자는 예측된 비용을 결제해야 노트 생성이 시작됩니다. 비록 예측된 금액보다 분석해야 할 양이 많으면 손해를 볼 수 있지만, 평균 비용에 배수를 곱하게 하여 이 문제를 최소화했습니다.

이 구조가 그대로 요금제로 만들어졌습니다. 2시간 영상에 약 1,300원이 듭니다. 타사 서비스가 3시간 영상을 분석하는 데 월 7,900원을 요구하니, 시험 기간에 몰아 쓰는 학생들 대상으로는 매우 합리적인 제도입니다. 인프라 또한 사용한 만큼만 청구되는 구조이기에 유지 가능한 시스템입니다.

플래그십 모델을 고효율 모델로 바꾸면서 노트 품질은 지켜내기

개발 초기에는 클로드 같은 플래그십 모델을 사용했습니다. 프롬프트 튜닝이나 타임라인을 크게 정리하지 않아도 만족스러운 결과가 나왔지만, 노트 한 편을 생성하는 데 돈이 너무 많이 들었습니다. 2시간짜리 영상을 클로드로 요약해주고 1,300원만 받는 것은 자선단체가 아니면 어려운 일입니다.



한 번에 시키지 않고 나눠 맡기는 파이프라인

지금은 Gemini, DeepSeek 같은 저비용 고효율 모델을 사용하고 있습니다. 대신 한 번에 모든 작업을 시키지 않고 목차 생성, 본문 생성, 이미지 배치, 퇴고 등으로 파이프라인을 나눴습니다. 작은 모델은 특히 컨텍스트가 커지면 품질이 급격히 떨어지는데, 목차를 먼저 만들고 한 항목씩 쓰게 일을 나누면 품질이 크게 떨어지지 않습니다. 또한 캐싱을 통해 비용이 일부 절감되기도 합니다.

컨텍스트 때문에 품질이 떨어지는 문제는 해결했지만, 사고 능력의 차이가 남았습니다. 그래서 클로드 Fable 같은 플래그십 모델과 지금 파이프라인의 결과를 비교하는 하네스를 만들어 정기적으로 실행하고 있습니다. 비록 Fable 만큼의 성능이 나오지는 못하지만, 비용 대비 좋은 품질이 나오고 있습니다.

2026 ~ 현재



1:1 전문가 상담 서비스

App Store [↗](#) Google Play [↗](#)

참가 인원

3명

개발 기간

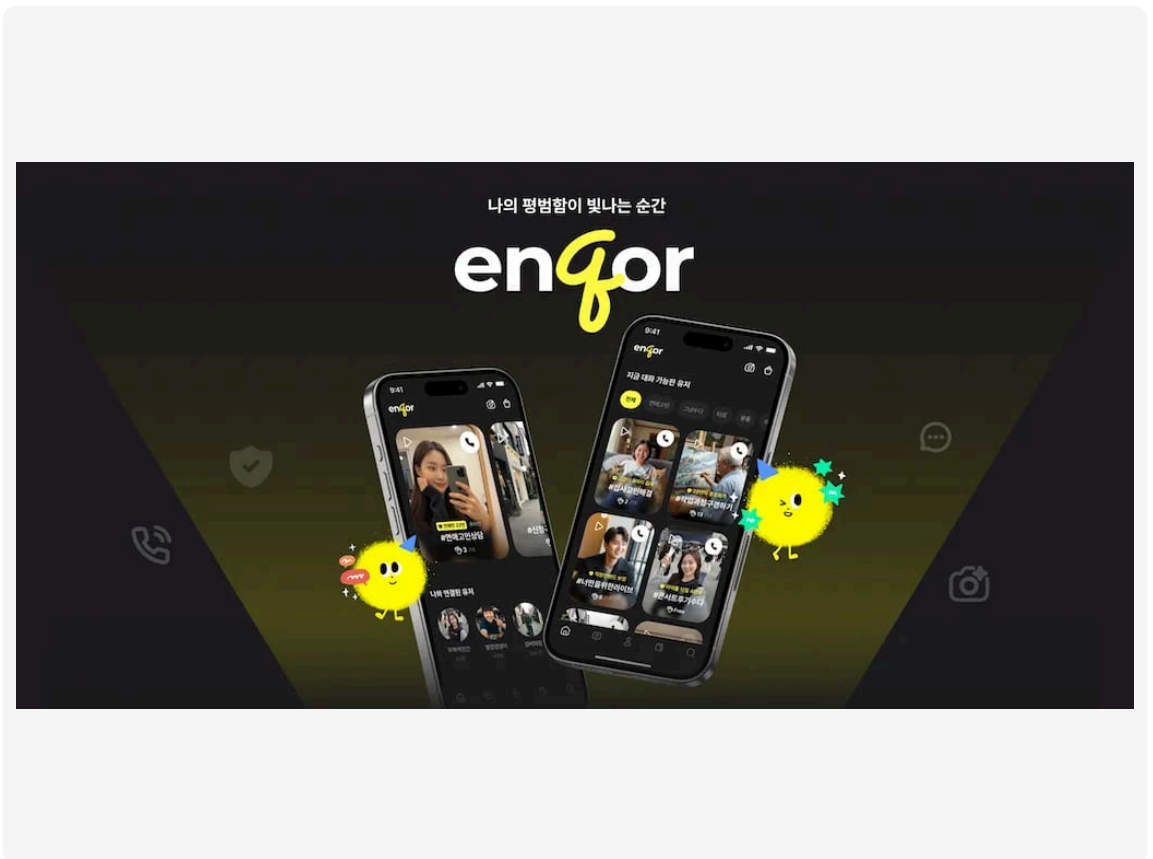
2026.04 ~ 현재

담당

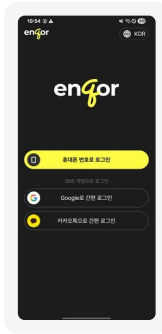
앱 및 서버 개발

주요 기술

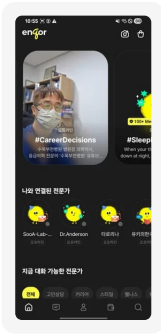
Agora(스트리밍), NestJS(GraphQL), Expo



enqor



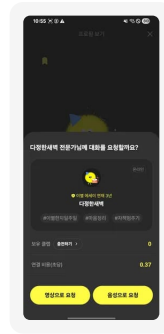
로그인



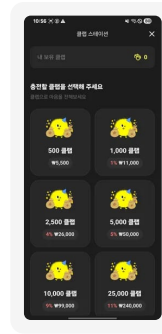
전문가 탐색



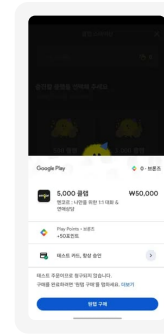
전문가 프로필



대화 요청



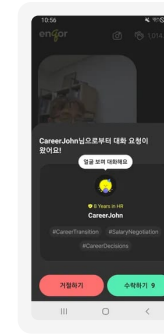
클럽 충전



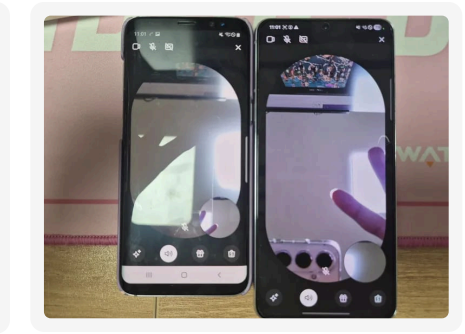
결제



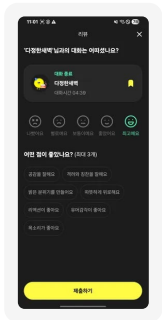
연결 중



요청 수신



통화



리뷰

Highlights

- 안정적으로 통화가 연결된 경우만 초 단위로 비용 정산하기
- AI 의 응답을 사람 병목 없이 검증하기

Motivation

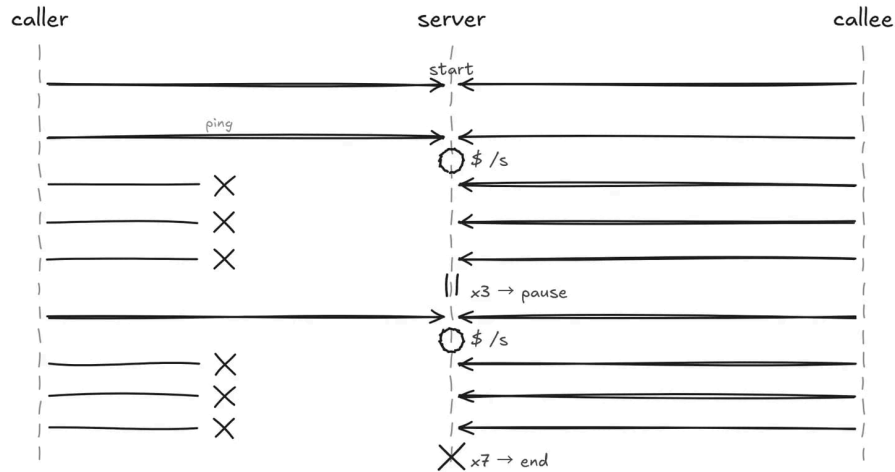
요즘은 고민이 생기면 ChatGPT 에게 먼저 물어봅니다. 그런데 정작 마음이 복잡할 때는 AI 특유의 영혼 없는 대화에 금세 지치고, 의로나 입시처럼 책임이 따르는 상담에서는 AI 의 답을 그대로 믿기도 어려웠습니다. 몇몇 대화는 아직 사람과 해야 합니다.

의사 같은 전문가부터 연애를 오래 해본 사람까지, 각자 자신 있는 분야로 다른 사람의 고민을 들어주고 수익을 만드는 자리가 필요하다고 봤습니다. 대화 플랫폼을 개발하기 위해 Align Networks 에 합류해 앱과 서버 개발을 맡고 있습니다. 다만, 엔코르는 사람이 직접 프로그래밍하지 않았습니다. 기획서와 디자인을 AI 에이전트가 읽을 수 있게 만들고, 에이전트가 내놓은 결과물을 검증하는 환경을 만드는 게 제 일이었습니다.

Architecture

안정적으로 통화가 연결된 경우만 초 단위로 비용 정산하기

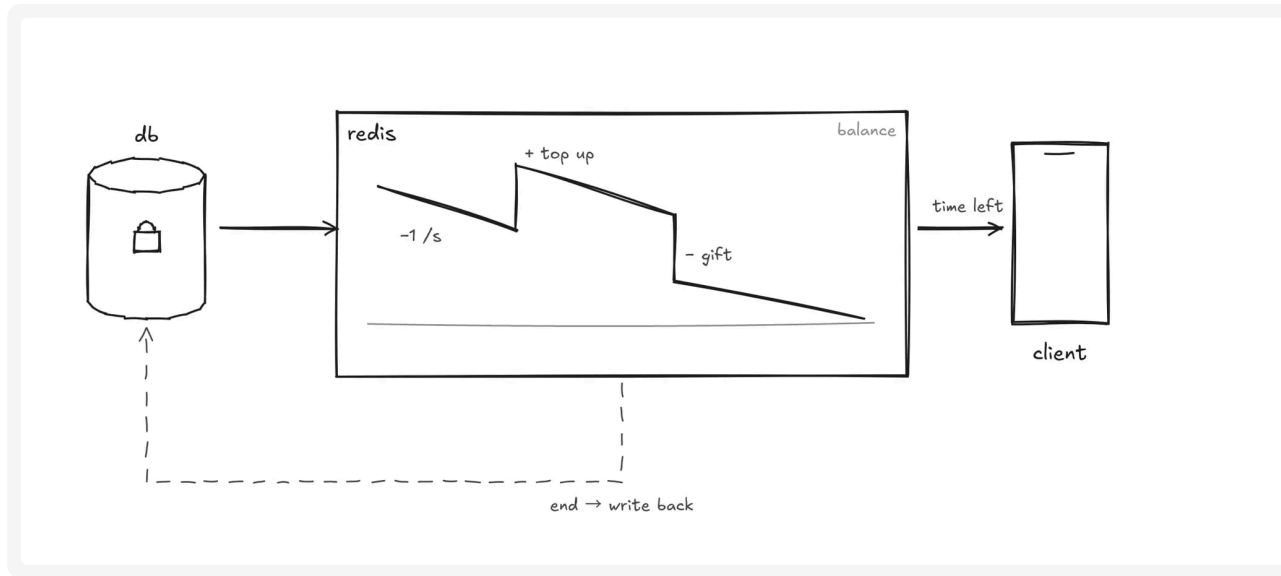
엔코르의 핵심 기능은 영상·음성 통화입니다. 스트리밍은 Agora 를 써서 어려움이 없었지만, 정산 파이프라인은 직접 만들어야 하는 과제였습니다. 모바일은 통신 상태가 나빠지는 일이 잦은 만큼, 정상적으로 이루어진 시간만 정산해야 했고, 초 단위 청구까지 요구되었습니다.



핑으로 정의한 안정적인 통화

먼저 '안정적인 통화'가 무엇인지 정해야 합니다. 이는 서버와 클라이언트가 주고받는 핑으로 정의했습니다. 어느 한 참여자라도 핑이 3번 누락되면 연결에 문제가 있다고 판단하고 정산을 중단합니다. 7번 누락되면 통화가 끊어졌다고 판단하고 세션을 종료하게 만들었습니다.

다음 문제는 매초 정산이라는 요구사항입니다. 처음에는 전체 통화 시간에서 불안정했던 시간을 빼고, 통화가 끝난 뒤 데이터베이스를 한 번 업데이트할 생각이었습니다. 그런데, 요구사항이 생각 이상으로 복잡했습니다. 통화 중에 캐시(앱 안의 재화, '클랩')를 충전할 수 있어야 했고, 충전한 캐시로 서로 선물도 할 수 있으며, 통화 가능 시간이 20초 미만으로 줄면 통화가 곧 끝난다고도 알려줘야 합니다. 충전, 소비, 조회가 모두 실시간으로 이뤄져야 하므로 남은 캐시를 읽기 쓰기가 가벼운 어딘가에 저장하고 처리하는 게 간단하겠다고 판단했습니다.

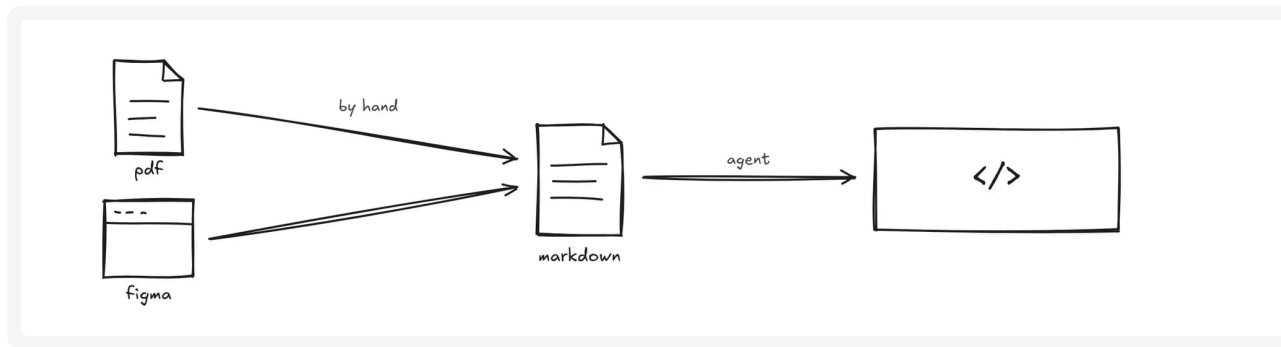


통화 세션의 잔액

통화가 시작되면 발신자 캐시를 수정할 수 없게 만듭니다. 현재 캐시를 읽어 Redis 의 통화 세션에 복제해두면 이 값이 발신자가 쓸 수 있는 캐시의 원본이 됩니다. 매초 이 값을 줄여 정산하고, 충전이나 선물이 발생하면 이 값을 계속 업데이트합니다. 충전은 결제 기록을 남겨야 하므로 DB 에 추가로 남깁니다. 남은 통화 시간도 이 값을 바탕으로 계산해 클라이언트에 내려줍니다. 통화가 끝나면 DB 에 남은 잔액을 업데이트합니다. 충전·선물을 빼면 통화 한 건의 DB 조작은 시작과 끝 두 번뿐이라, 복잡한 로직을 직관적으로 구현할 수 있었습니다.

AI 의 응답을 사람 병목 없이 검증하기

기획팀이 작성한 파일은 PDF 로 된 기획서와, Figma 로 작성된 디자인 파일입니다. 사람이 읽고 수정하기에는 편하지만 에이전트에게 그대로 제공하면 어느 Figma 노드에 어떤 설명이 들어 있는지 알 수 없습니다. 그래서 Figma 노드와 PDF 페이지를 직접 짝지어 Markdown 으로 작성했고, 곳곳에 메모도 적어봤습니다.

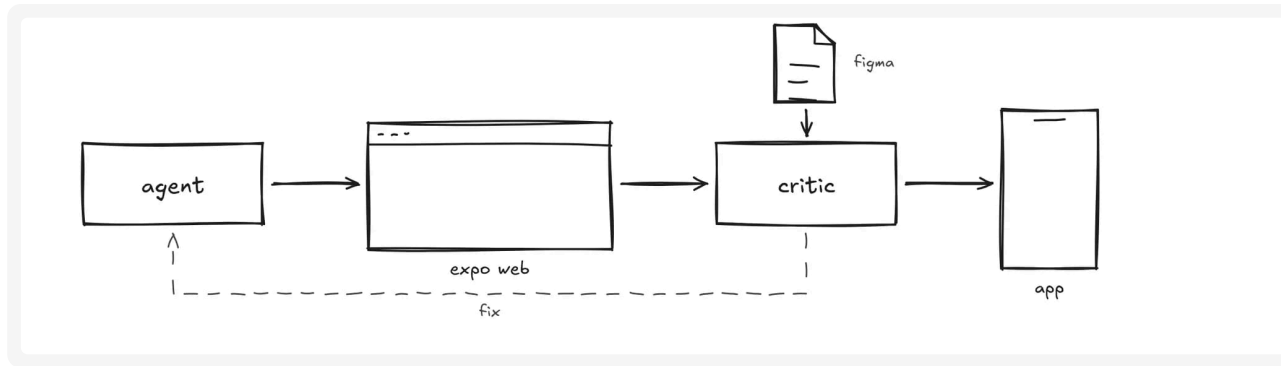


기획서에서 코드까지

이렇게 완성한 약 800줄짜리 기획서를 가지고 AI 에이전트에게 프로젝트를 끝까지 완성하라고 했습니다. 30여 분 만에 프로젝트 하나가 나왔고, 실행해보니 그 결과는 엉망이었습니다. 어느 정도 예상은 했지만 실행조차 안 될 줄은 몰랐고, 겨우 고쳐 로그인 화면에 진입하니 Figma 에 정의된 디자인은 전혀 반영되어 있지 않았습니다. 로그인 같은 기능도 전부 모킹으로 구현된 순 엉터리 프로젝트였습니다.

이 문제를 고문님께 털어놓았는데, '하네스 엔지니어링'이라는 개념을 소개해주셨습니다. 인터넷에 검색해보니 'AI 가 올바른 행동을 할 수 있게 환경을 만드는 것'이라는 모호한 답변만이 나왔습니다. 개념이 완전히 이해되지는 않았지만 고문님 조언을 듣고 혹, 비판적 리뷰어, 테스터를 하나씩 세팅하다 보니 어느새 저만의 하네스가 완성되어 있었습니다.

하네스에서 가장 어려운 것은 '성공 조건'을 정의하는 일입니다. 예를 들어 Figma 에 선언된 디자인대로 퍼블리싱했는지 판정하려면, 그 디자인과 퍼블리싱된 화면이 일치하는지 평가하는 로직이 필요합니다. 처음에는 앱 화면을 캡처해 단순히 비슷한지만 검사했고, 독립적인 서브에이전트가 이 비교를 담당했습니다. 나쁘지 않았지만 그렇다고 좋은 결과물도 아니었습니다. 여기저기 아이콘이 다르고 텍스트 위치가 어긋났습니다. 무엇보다 앱 화면을 캡처해 Figma 와 대조하는 작업이 매우 느리고 토큰을 많이 소비했습니다.



웹으로 먼저 퍼블리싱하는 하네스

이 문제는 expo web 으로 해결했습니다. 먼저 구현하려는 화면을 expo web 으로 퍼블리싱하면, headless 브라우저를 열어 결과를 어렵지 않게 확인할 수 있습니다. 창 크기만 바꾸면 모바일·태블릿 해상도에서 깨지는 영역이 있는지도 바로 보입니다. 이렇게 웹에서 검증이 끝난 코드를 앱으로 옮기는 순서로 바꾸니 평가가 훨씬 싸졌습니다. Figma 와 웹으로 퍼블리싱된 화면이 얼마나 비슷한지는 비판적 에이전트가 판정하되, 아이콘을 직접 생성하지 못하도록 규칙을 넣었고, Figma MCP 에 문제가 생겨 동작하지 않는 경우 하네스가 아예 멈추도록 하는 preflight 절차도 추가했습니다. 그 결과 간단한 화면은 거의 완벽한 수준으로, 복잡한 화면도 70% 정도의 완성도로 나왔습니다.

Retrospective

이번 경험에서, AI 에게 좋은 자료와 환경을 주면 그만큼의 결과가 나온다는 것을 알았습니다. 그리고 AI 가 만들어낸 결과도 '어떻게' 검증하느냐보다는 '어디까지' 검증할지를 생각하는 방향이 옳다고 느꼈습니다.

금융, 우주 산업, 방위 산업처럼 사고 한 번이 참사인 도메인은 모든 코드를 전부 꼼꼼히 검증해야 할 것입니다. 하지만 이번 프로젝트처럼 대부분은 그렇지 않습니다. 개발자는 AI 가 지금 무슨 일을 하고 있고, 이번 일이 끝나면 어디까지 검증해야 하고, 어떤 파일들이 바뀌길 기대하는지 정도만 알면 충분하다고 생각합니다. 그 안에서도 단순한 작업을 AI 가 하고 있다면 몇 번 둘러보고 테스트를 마칠 수도 있고, 핵심 로직이면 설계까지 꼼꼼히 논해야 합니다.

개발자가 코드를 작성하는 사람에서 판단하고 책임지는 사람으로 바뀌었다는 것을, 이 프로젝트에서 확인했습니다.

2024

HEALIX

증상 기반 병원 찾기 서비스

GitHub 7

참가 인원

6명

개발 기간

2024.07 ~ 2024.08

담당

백엔드 리드

주요 기술

ChatGPT API, NestJS

증상 분석부터 나와 가까운 병원까지,

HEALIX

Zeegle

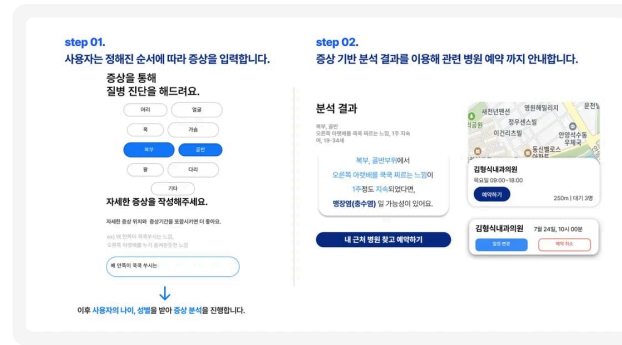
김수빈 김원호 김재연 이진호 조준환 최혜선



HEALIX



기획 배경



증상 입력과 병원 예약

Motivation

사람들은 어디가 아플 때 어느 병원에 가야 할지 모르는 경우가 많습니다. 당장 생각해봐도 열이 나는 감기에 걸리면 내과를 가야 할지 이비인후과를 가야 할지 모르겠고, 엉덩이 같은 생소한 부위가 아파오면 그 고민은 더더욱 커집니다. 과거에는 인터넷에서 검색해 적절한 과를 찾고, 지도에서 병원을 찾아야 했습니다. HEALIX 는 이 불편함을 해결하고자 개발되었습니다.

Video



HEALIX 시연 영상

<https://www.youtube.com/watch?v=p3cPmhgZELg>

2023

WATERFLAKE

간단한 설치만으로 마인크래프트 도메인 서버 열기

[waterflake-api](#) ↗ [waterflake-front](#) ↗

[waterflake-tunnel](#) ↗ [waterflake-plugin](#) ↗

참가 인원

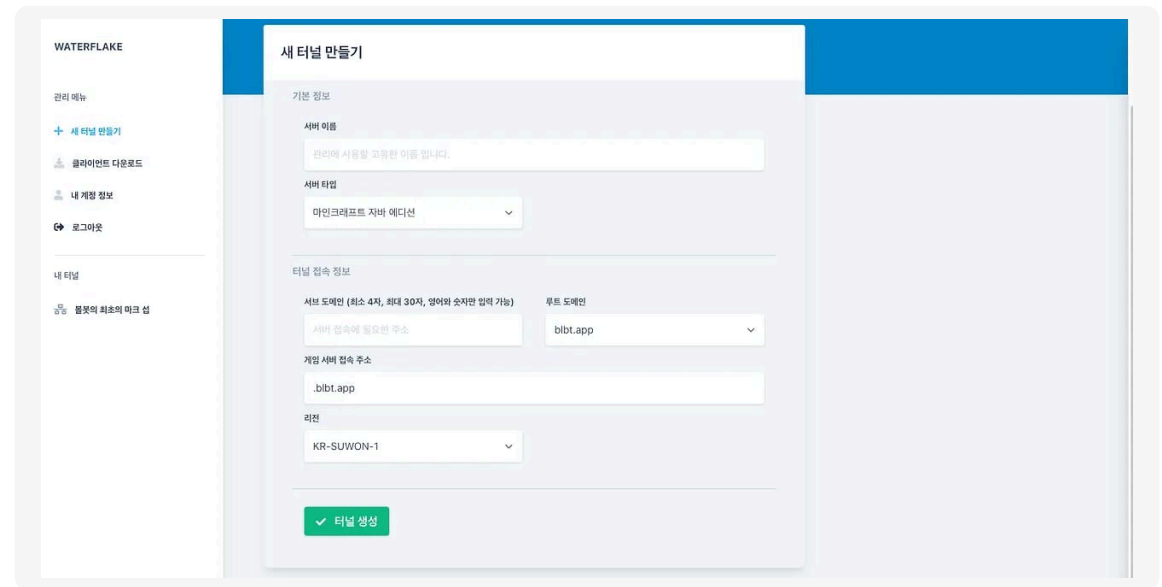
1명(개인)

개발 기간

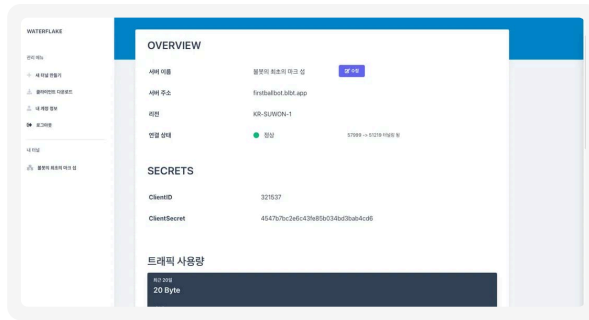
2023.03 ~ 2023.05

주요 기술

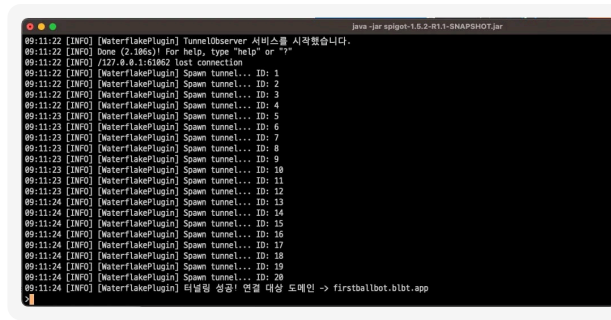
NestJS, Java



터널 만들기

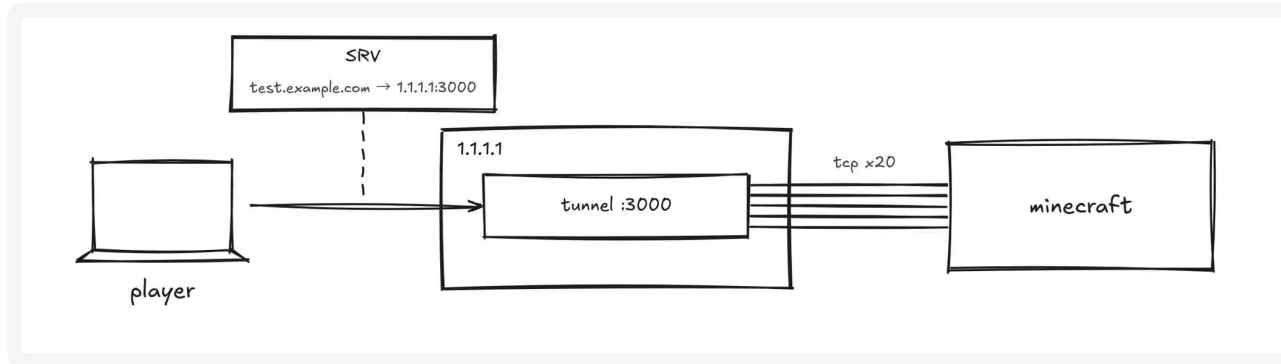


터널 대시보드



Architecture

SRV 레코드로 서브도메인이 특정 IP 의 포트를 가리키도록 자동 설정하기



도메인에서 게임 서버까지

사용자가 WATERFLAKE 웹 화면에서 터널을 생성하면 Cloudflare API 를 통해 SRV 레코드를 생성합니다. 그리고 터널링을 담당하는 가상 머신에 랜덤한 포트로 터널링 서버를 띄웁니다. 터널링 서버는 두 개의 연결을 받는데, 한 개는 SRV 레코드를 통해 들어오는 연결, 한 개는 마인크래프트 서버 쪽으로 나가는 연결입니다. 이 두 연결을 파이프해 외부에서 들어오는 패킷이 자연스럽게 마인크래프트 서버로 나갈 수 있습니다. 사용자에게는 도메인 주소만 입력하면 연결되는 서버처럼 보입니다.

게임 서버와 터널링 서버 사이를 TCP 소켓으로 터널링하기

터널링 서버와 게임 서버 간 TCP 터널은 20개의 소켓으로 관리됩니다. 게임 플레이어가 서버에 접속하면 이 소켓 중 하나를 점유합니다. 플레이어가 게임 서버를 나가면 해당 소켓은 폐기되고, 새 소켓을 생성합니다. 이 주기가 반복되어 소켓의 개수를 관리합니다. 다행히 마인크래프트 서버 설정에는 최대 접속 가능한 인원이 정해져 있기 때문에, 이 값만큼만 소켓을 관리하면 운영에 문제가 없었습니다.

19억 건의 테이블을 점검 시간 1시간 내에 마이그레이션

배경

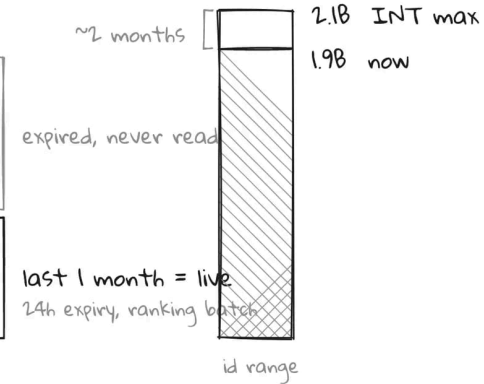
타임스프레드에는 랜덤 상자라는 기능이 있습니다. 광고를 보면 캐시가 무작위로 담긴 상자가 지급되고, 그 상자를 열면 캐시가 들어오는 단순한 기능이었습니다. 상자 하나는 luckybox 테이블의 한 행이었습니다. 사용자가 늘면서 이 테이블에 19억 건이 쌓였는데, 기본 키가 INT 로 선언되어 있어 상한인 21억까지 두 달 정도밖에 남지 않은 상태였습니다. 이 테이블을 BIGINT 로 선언한 새 테이블로 옮겨야 했습니다.

첫 번째 시도

먼저 서비스의 특징을 조사했습니다. 상자는 24시간이 지나면 만료되고, 랭킹을 처리하는 배치 프로그램은 최근 1개월 데이터만 읽고 있었습니다. 점검 시간 안에 이 데이터만 옮기면 서비스를 다시 띄울 수 있다는 뜻이었습니다.

luckybox 1.98 rows

id INT	cash	created_at
1	300	2021-12-01
2	100	2021-12-01
...	...	...
...	...	...
1,899,999,998	500	2023-06-19
1,899,999,999	1,000	2023-06-30
1,900,000,000	300	2023-06-30



19억 건 가운데 실제로 읽히는 것은 최근 한 달치뿐

팀장님과 논의해 평일 새벽에 서비스를 1시간 멈추고, 미리 준비한 마이그레이션 스크립트를 실행하기로 했습니다. 스크립트는 아래 작업을 순서대로 수행합니다.

- 기존 테이블의 이름을 RENAME 해 연결을 끊습니다.
- 새 테이블을 BIGINT 로 다시 만듭니다.
- 혹시 모를 중복을 막기 위해 PK 가 22억부터 시작하게 합니다.
- 1개월 전 데이터가 몇 번 ID 인지 알아냅니다.
- MySQL 프로시저로 100건씩 배치로 데이터를 복사합니다.

스크립트를 짜면서 가장 중요하게 본 것은 '롤백이 가능한가'였습니다. 작업 도중 문제가 생기면 즉시 멈출 수 있어야 했고, 진행 상황도 눈으로 볼 수 있어야 했습니다. 데이터 삽입은 쿼리 한 줄로도 작성할 수 있었지만, 진행 상황을 알 수 없고 중간에 멈출 수도 없어 작업을 잘게 쪼개기로 했습니다.

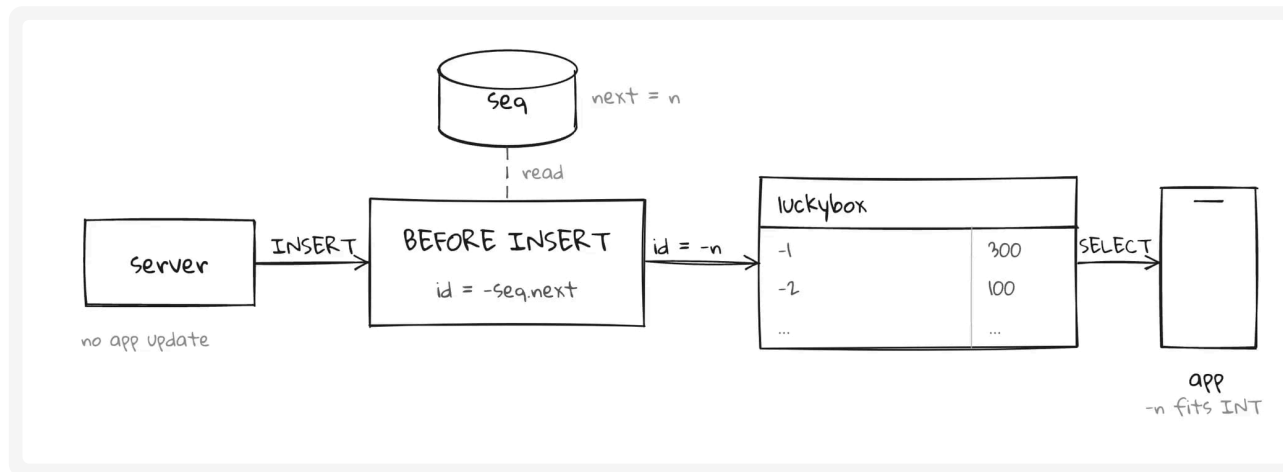
마이그레이션은 무사히 진행됐습니다. 1개월치 데이터가 들어갔고, 새 상자는 22억부터 만들어지기 시작했습니다. 그런데 어째서인지 안드로이드 앱에서 크래시가 나기 시작했습니다. 남은 점검 시간이 많지 않아 준비해둔 순서대로 롤백했고, 서비스는 이전 모습으로 돌아갔습니다.

원인

이유는 단순했습니다. 앱에서도 ID 를 INT 로 다루고 있어 21억이 넘는 값을 처리하지 못한 것입니다. 서버만 고쳐서 될 문제가 아니었습니다. 다른 방법을 찾아야 했습니다.

두 번째 시도

팀장님과 해결책을 논의하다가 문득 INT 에는 음수 영역도 있다는 사실이 떠올랐습니다. 다행히 unsigned 로 선언되어 있지 않아, PK 를 음수로 저장하는 방법을 두고 이야기를 나눴습니다. 팀장님께서도 좋은 방법이라고 하셔서 곧바로 작업에 들어갔습니다.



트리거가 PK 를 음수로 바꿔 끼우는 흐름

BEFORE INSERT 트리거를 걸어, 값이 들어올 때 시퀀스 테이블에서 다음 값을 읽어 PK 를 음수로 바꿔 넣도록 했습니다. 테스트 서버에서 서버와 앱이 음수 PK 를 모두 문제없이 처리하는 것을 확인하고, 새벽에 트리거를 설치했습니다.

그날 이후 삽입되는 모든 데이터의 PK 가 음수로 들어갔고, 앱에서도 문제가 생기지 않았습니다. 이 조치로 당장의 장애를 막고 약 2년의 리팩토링 시간을 벌었습니다.

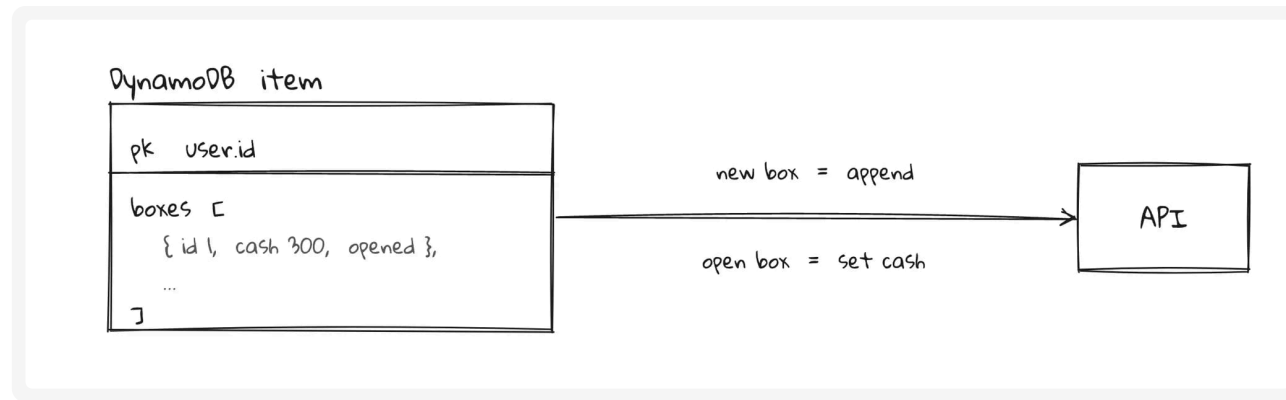
분당 1,500건의 요청을 받는 서비스 리팩토링

배경

앞선 마이그레이션으로 시간을 번 직후부터 다음 작업을 준비했습니다. 먼저 앱이 ID 를 정수로 다루던 것을 String 으로 바꿨고, 그 배포가 사용자의 99% 에 닿을 때까지 기다렸습니다. 마침 운영팀에서 랜덤 상자의 확률 조정 기능을 파악하기 어려우니 리뉴얼하자는 요청이 왔고, 대표님께서도 승인해주셔서 드디어 리팩토링에 착수할 수 있었습니다. 상자 서비스의 /luckybox 엔드포인트는 그때 분당 1,500건 안팎의 요청을 받고 있었습니다.

상자를 어디에 저장할 것인가

먼저 상자를 어디에 저장할지 고민했습니다. 예전 설계는 상자를 전부 MySQL 에 저장해, 상자를 만들 때마다 INSERT 가, 상자를 열 때마다 UPDATE 가 발생했습니다. 이를 DynamoDB 로 옮기고, 파티션 키로 읽고 쓰는 속도가 빠르다는 장점을 최대한 활용하기로 했습니다.



파티션 키 하나에 상자를 배열로

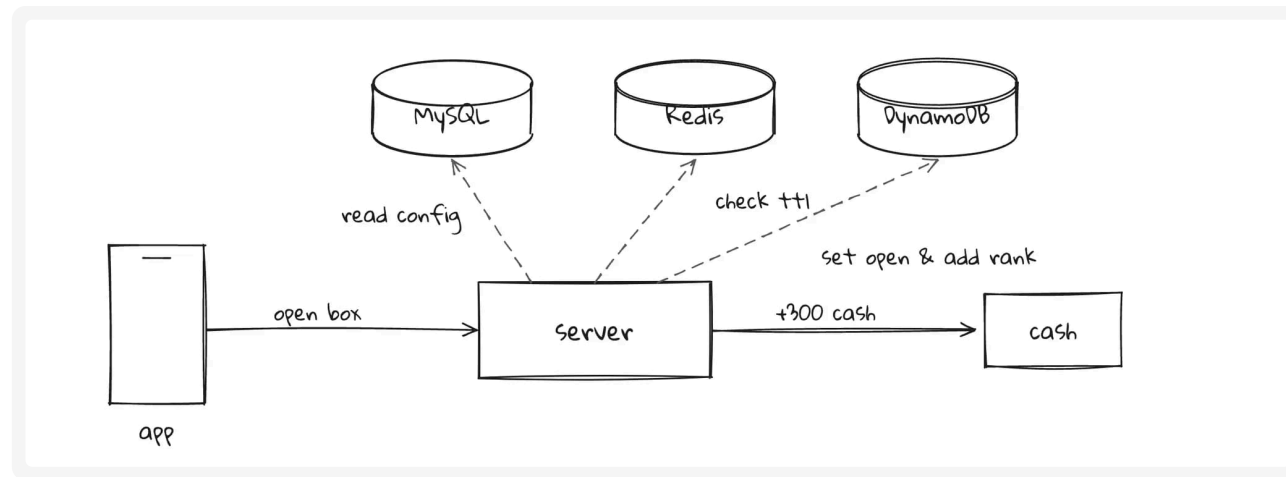
상자가 생기면 배열에 값을 더하고, 상자를 열면 그 항목만 고치면 됐습니다. 겉으로 보기에 MySQL 과 다르지 않지만 파티션 키 하나로 조회하고 수정하기 때문에 훨씬 빨랐습니다. 또 사용자가 그날 처음 접속하면 쌓여 있던 데이터를 지우게 해, 이전처럼 값이 끝없이 늘어나는 일도 막았습니다.

새로 생긴 정책으로, 한 사용자는 최대 20개의 상자만 열 수 있다는 제약도 있었습니다. 이 또한 상자 배열의 length 만 세면 깔끔하게 해결할 수 있었습니다.

랭킹과 설정은 어디에 둘 것인가

랭킹 시스템도 새로 만들었습니다. Redis 의 ZSET 은 중복 없는 키를 점수 순으로 정렬해두고, 순위를 읽고 갱신하는 작업에 $O(\log n)$ 의 시간복잡도를 가집니다. 그리고 고액 당첨자를 일정 기간 다시 당첨되지 못하게 막아야 한다는 요구사항도 있었는데, 당첨된 사용자의 ID 를 TTL 을 걸어 Redis 에 저장해두면 데이터베이스에서 당첨 기록을 조회할 필요 없이 간단하게 해결할 수 있었습니다.

당첨 확률 같은 설정 값은 MySQL 에 남겼습니다. 관리자 콘솔과의 연동을 고려해야 했기 때문입니다. 상자를 열 때마다 MySQL 을 한 번 읽어야 한다는 단점이 있지만, 설정을 실시간으로 반영할 수 있고 조회만 하므로 큰 부담은 아니라고 판단했습니다.



상자를 열 때 저장소 셋을 도는 순서

결과

완성된 시스템은 조금씩 트래픽을 받았습니다. 아침에는 10% 정도만 받으면서 지켜보다가, 문제가 없어 오후에 100%로 올렸습니다. MySQL의 부하가 내려갔고, 무엇보다 랭킹에 활기가 돌아왔습니다. 6시간에 한 번 갱신되던 값이 실시간으로 동작했기 때문입니다. 운영팀에서도 당첨 확률을 설정하기가 편해졌다며 만족스러워했습니다.

문제를 발견해 팀장님께 보고한 날부터 마지막 리팩토링까지 꼬박 1년이 걸렸습니다. 제가 쓴 코드가 조금씩 트래픽을 받아 100%까지 배포되는 모습을 지켜본 경험은 아직도 잊히지 않는 개발의 즐거움입니다.

월 1,000만 원을 넘기던 AWS 비용을 30% 절감

배경

타임스프레드는 서버 비용으로 매달 1,000만 원 넘게 쓰고 있었습니다. 팀장님께서 이 비용을 줄일 방법을 찾아보라고 하셨습니다.

어디에 돈이 나가고 있는가

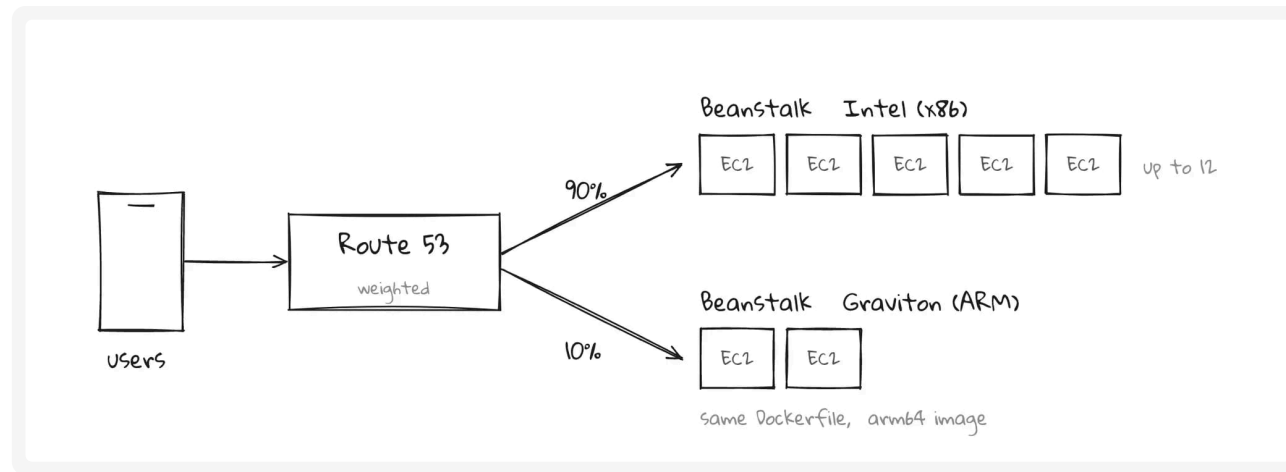
Cost Management 에서 확인해보니 비용이 가장 많이 나가는 곳은 세 군데였습니다.

- RDS
- Beanstalk (EC2)
- 네트워크 트래픽

RDS 는 이미 부하가 높아 손댈 여지가 없었으므로, EC2 에서 줄일 곳을 찾았습니다. 마침 ARM 계열의 Graviton 인스턴스가 새로 나왔고, 같은 성능을 20% 정도 싸게 낼 수 있다는 AWS 의 발표를 보고 우리 서비스에 적용해보기로 했습니다.

Graviton 으로 옮기기

먼저 Graviton 인스턴스를 한 대 만들어 서버를 띄워봤습니다. 운영체제 버전과 아키텍처가 바뀌어 기존 방식으로는 뜨지 않았지만, 다행히 Dockerfile 로 빌드하니 ARM 아키텍처에서도 안정적으로 동작했습니다. 그래서 기존 CI/CD 파이프라인을 고쳐 GitHub Actions 에서 크로스 빌드로 ARM 이미지를 만들게 했고, Graviton 인스턴스로 도는 Beanstalk 을 새로 만들어 그 이미지를 쓰게 했습니다.



Route 53 가중치 라우팅으로 트래픽의 10% 만 새 인스턴스에

테스트 서버에서 QA 팀과 함께 서버가 제대로 도는지 확인했고, 아무 문제가 없었습니다. Route 53 의 가중치 라우팅으로 새 서버에 트래픽의 10% 만 흘려보냈고, 큰 문제가 나오지 않아 최종적으로 모든 트래픽을 Graviton 인스턴스가 받도록 했습니다.

인스턴스 증감 규칙 다시 잡기

며칠 뒤에는 CPU 사용량 데이터를 보고 시간대별 스케일 아웃 규칙을 조정했습니다. 피크 타임에 최대 12대까지 늘어나던 인스턴스를 8대로 줄였고, 야간에는 1대만 돌게 했습니다. 인스턴스 교체와 스케일 규칙 조정을 합쳐 비용을 30% 정도 줄일 수 있었습니다.

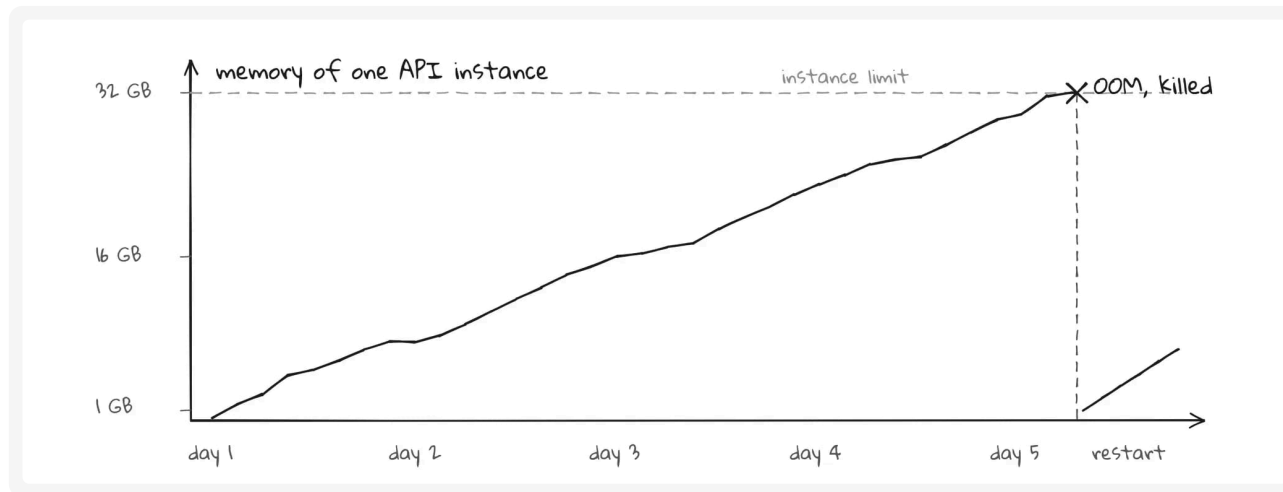
메모리 32GB 를 다 쓰고 멈추던 API 서버 정상화

배경

입사하고 4개월쯤 지난 시점이었습니다. 머신마다 메모리 사용량이 이상할 만큼 제각각이었습니다. 어떤 머신은 20GB 넘게 쓰고 있고, 어떤 머신은 1GB 정도만 쓰고 있었습니다. 그리고 거의 모든 머신이 나흘에서 닷새쯤 지나면 강제로 종료되고 재시작됐습니다.

원인 찾기

처음에는 재부팅 규칙이 있는 줄 알았습니다. 다른 팀의 백엔드 서버는 실제로 일주일에 한 번 머신을 재부팅한다고 했습니다. 하지만 인스턴스가 메모리를 20GB 씩 쓰는 것은 재부팅 규칙과 상관없이 무언가 잘못된 상태였습니다.



인스턴스 한 대의 메모리, 닷새 만에 1GB 에서 32GB 까지

며칠 동안 모니터링한 결과, API 서버 어딘가에서 메모리 누수가 일어나 인스턴스의 메모리를 전부 쓰고 마지막에 OOM 으로 강제 종료된다는 사실을 알았습니다. 그런데 Sentry 에도 AWS 모니터링 도구에도 원인이 될 만한 흔적이 남지 않아, 디버깅에 난항을 겪었습니다.

메모리 덤프 뜨기

팀장님께서 상용 서버에 메모리 덤프를 떠서 조사해보자는 아이디어를 주셨습니다. 솔직히 내키지 않았습니다. 덤프를 뜨는 동안 서비스 장애율이 올라갈 수도 있고, 덤프를 가져와도 읽을 줄 몰라 유의미한 결과를 찾기는 어려울 것 같았습니다. 그래도 달리 방법이 없어 우선 해보기로 했습니다.

찾아보니 메모리 덤프 도구가 몇 가지 있었습니다. 팀장님께 공유드리고, 상용 서버에 직접 접속해 두 가지 도구로 메모리를 스캔했습니다. 첫 번째 도구에서는 건질 것이 없었지만, 두 번째 도구에서 같은 오류가 여러 번 발생한 것으로 보이는 덤프 로그를 발견했습니다.

원인과 해결

앱 푸시 발송을 병렬로 바꿔둔 코드가 문제였습니다. 병렬 처리를 위해 스레드를 만드는데, 스레드 안에서 exception 이 발생하면 그 스레드가 회수되지 않고 그대로 남아 메모리 누수가 쌓이고 있었습니다. 스레드 내부의 처리 로직에 try/catch 를 붙여 배포했고, 그 뒤로 모든 인스턴스가 1GB 정도만 쓰는 정상 상태로 돌아왔습니다.

처음에는 해결책이 도저히 떠오르지 않아 스트레스를 받았습니다. 결국 직접 열어서 들여다보면 답은 생각보다 가까운 곳에 있다는 것을 배운 작업이었습니다.