안녕하세요!
Quest를 좋아하는
개발자 **홍의민**입니다.



- **Motto.** 고민은 짧게, 행동은 빠르게
- **Email.** elipair0106@gmail.com
- **Education.** 세종대학교 컴퓨터공학과 · AI Advanced Microdegree
- **Certification.** SQLD (2026.03 취득)
- **Awards.** Google I/O Extended 1st Place 외 수상·선정 9건
- **Shipped.** 경찰과도둑 · Block Drag · RuleBook

메일 보내기

PROJECTS

메인 프로젝트

경찰과도둑

2025.10 – 2026 · Flutter · Android & iOS

친구들과 밖에서 노는 GPS 술래잡기, 경도

역할 Flutter 앱 설계·구현 · GitHub Actions/Fastlane 배포 자동화 · UX 개선

수상 세종 창업 아이디어 리그 대상 외 4건

기능 실시간 위치 공유 · QR 체포·초대 · 팀별 채팅 · 자동 게임 판정 · 2~150명 플레이





Website Google Play App Store GitHub

자세히 보기 →

그 외의 프로젝트

Party Passport Google I/O Extended: Hack the Beat 1st Place
파티에서 QR로 만난 사람을 스탬프로 모으고, 서로 고른 사람만 매칭되는 웹앱
React · TypeScript · Vite · 2026.08 · Live Demo · GitHub · Instagram 행사 기록 · LinkedIn 행사 기록

passQL App Store 출시 · 웹 서비스
AI가 함정 오답을 만들고, 보기 SQL을 샌드박스 DB에서 실행해 확인하는 SQLD 학습 앱
Flutter · Riverpod · React · TypeScript · 2026.04 – 2026 · App Store · Website · GitHub

트립게더 세종대 창업캠프 대상 롯데 청년기업가대상 본선
SNS 게시물을 공유하면 장소 정보를 정리하는 AI 여행 가이드
Flutter · 2025.09 – 2025.11 · GitHub

Stocker Seeds 팀 프로젝트 우수상
주식투자를 배우는 핀테크 교육 앱 · 팀 돔황촤 코스피
Flutter · 2025.05 – 2025.11 · GitHub

Block Drag Google Play · App Store 출시
공간 감각을 활용하는 퍼즐 게임
Flutter · Supabase · 2025.05 – 2025.07 · Google Play · App Store

RuleBook Google Play · App Store 출시
출석·벌칙·벌금을 관리하는 앱
Flutter · 2025.03 – 2025.06 · Google Play · App Store

완익세종 SW-AI 해커톤 장려상
AI 기반 진로-교과목 로드맵 추천 서비스 · 팀 완익세종
React · Next.js · 2025.12.23 – 2025.12.24 · GitHub

우주공부선 스터디 목표 설정·학습 기록·공유 플랫폼 · 2025.12 – 2026

냥생뭐였니? 고양이를 테마로 한 웹 콘텐츠 · 2024.11 – 2024.12

플래닝 로스트아크 정보를 제공하는 Discord 봇 · 2024.11 – 2025.01

은샘침례교회 교회 홈페이지와 API 서버 — 실제 운영 중 · 2026.06 – 2026.09

HOW I WORK

기획 웹 서버 DB 클라우드 AI 에이전트

Flutter

스토어 출시 앱 3개 · 경찰과도둑 앱 설계

Front만 알 땐, API가 외국어였습니다.

그래서 서버를 직접 띄워 보고(Oracle Cloud), 백엔드는 AI 에이전트와 함께 짜 봤습니다.

지금은 교회 홈페이지를 DB부터 배포까지 혼자 만들어 운영합니다.

이제 백엔드 팀원과 같은 언어로 말합니다.

사용자 관점
경찰과도둑

사용자가 답답하다면요?

Lock은 걸어도, Block은 말고.

잡힌 사람이 감옥을 못 찾자 지도와 채팅을 열어 두자고 제안 →

디자인 협업
경찰과도둑

버튼은 어디에 둬요?

Right는 Right에.

긍정 버튼에 색을 넣고 오른쪽으로 옮기자고 디자이너에게 제안 →

책임감
경찰과도둑

제 실수를 찾으면요?

제가 세운 Gate, 제가 걷습니다.

제가 넣은 배터리 설정 요구가 진입 장벽이 되자, 원인을 고쳐 없앰 →

문제 해결
경찰과도둑

같은 불만이 계속되면요?

Count 말고 Cause를.

튜토리얼 노출 횟수를 세 번 바꾼 뒤, 강제하는 구조를 걷어냄 →

팀워크
에브리페스타

안 되는 기능은요?

Disable 말고 Delete.

눌러도 반응 없는 버튼을 지우자고 제안했고, 팀 규칙이 됨 →

디테일
Big-Brother

경고는 언제 띄워요?

늦게 오는 Alert는 Alert가 아니니까.

관제원 창에 경고 카드를 바로 띄우고, 영상이 밀리지 않게 자리를 고정 →

AX-NATIVE

AI 에이전트와 함께 일하는 방식을 도구로 만들고, 실제 팀 프로젝트에 적용합니다.

Second Brain Template

회의·문서·이슈·결정을 지식으로 쌓고, 코드와의 충돌 감지·리포트·반복 이슈 탐지까지 한 흐름으로 묶는 프로젝트별 세컨드 브레인 템플릿

경찰과도둑 저장소 31문항 1차 벤치마크 — 코드만 있을 때 → 볼트가 있을 때

정확도	'왜 이렇게 정했나' 질문	없는 경로 인용
0.71 → **0.97**	0.0 → **1.0**	8.3% → **0.8%**

벤치마크를 어떻게 했는지 보기 → `npx github:EM-H20/second-brain-template`

Claude Code · Codex · Gemini · Obsidian · 의존성 0

적용 중인 프로젝트

- **경찰과도둑** 앱·서버·웹 3개 저장소의 회의·이슈·설계를 한 볼트로 관리
- **passQL** 웹·앱·서버·AI 모노레포의 이슈 428개와 결정 212개를 한 볼트로
- **캡스톤 Big-Brother** CCTV 위험 상황 관제와 VLM 자연어 설명 — FE·BE·VLM 지식을 한곳에
- **에브리페스타** 대학 축제 정보 서비스 — 결정·이슈·교훈을 기록하고 축제 데이터 볼트를 따로 분리

AWARDS

1st Place	Google I/O Extended: Hack the Beat · GDG Campus Korea · Party Passport	2026
대상	세종대학교 SW중심대학 · 세종 창업 아이디어 리그 · 경찰과도둑	2026
우수상	세종대학교 창업아이템 경진대회 (22팀 중 6팀 수상) · 경찰과도둑	2026
우수 창업 아이템상	세종대학교 하반기 입주공모전 · 경찰과도둑	2026
특별상	세종대학교 중앙학술동아리 아롬 데모데이 · 경찰과도둑	2026
선정	세종대학교 창업 동아리 SSUP · 경찰과도둑	2026

대상	세종대학교 창업캠프 · 트립게더	2025
본선 진출	제2회 신격호 롯데 청년기업가대상 · 트립게더	2025
우수상	연합동아리 Seeds 팀 프로젝트 · Stocker	2025
장려상	제13회 세종대학교 SW-AI 해커톤 · 완익세종	2025

CONTACT

함께 일해요

채용·협업 제안은 메일로 연락해 주세요.

메일 보내기

elipair0106@gmail.com GitHub LinkedIn Tistory 블로그

Instagram @e_m_hong

© 2026 Hong Eui-min · Elipair

경찰과도둑

2025.10 – 2026 · Flutter · Android & iOS

세종 창업 아이디어 리그 대상 세종대 창업아이템 경진대회 우수상 입주공모전 우수 창업 아이템상 아롬 데모데이 특별상 창업 동아리 SSUP 선정

친구들과 밖에서 노는 GPS 술래잡기, 경도

Website **Google Play** **App Store** **GitHub**





Flutter · Riverpod · Dio/Retrofit · Google Maps · WebSocket(STOMP) · Firebase

[경찰과도둑] 공식 홍보영상 — 동심은 그대로, 더 쫄깃하게

배경

경찰과 도둑을 좋아해 직접 모여 뛰던 사람들이 "이걸 앱으로 하면 더 재밌겠다"는 생각으로 시작한 프로젝트입니다. **경찰과 도둑**은 오프라인 술래잡기에 지도, 위치 공유, 실시간 게임 상태를 더한 Flutter 앱입니다. 호스트가 지도 위에 플레이 구역과 감옥을 그리고, 한 방에 최대 150명까지 참여합니다.

팀은 Flutter 앱, Spring Boot 게임 서버, Next.js 공식 사이트를 저장소별로 나눠 개발했고, 저는 Flutter 앱의 구조 설계와 대부분의 기능 구현을 맡았습니다. 코드 구현은 AI 에이전트(Claude Code)에게 맡기고, 설계·결정·검증은 직접 했습니다. 밖에서 뛰는 게임이라 화면이 꺼져도 위치와 연결이 유지돼야 했습니다.

맡은 역할

- **앱 구조 설계** — Clean Architecture, Riverpod, GoRouter, Dio/Retrofit
- **지도 SDK 선정** — 글로벌 사용자를 전제로 초반에 Google Maps 확정 (초기 SDK 연동은 팀원)
- **서비스 도입** — 소셜 로그인, 딥링크, FCM, Remote Config, Analytics, Crashlytics, AdMob
- **좌표 격자** — 지도를 칸으로 나눠 핑 위치를 팀 채팅에 자동 연결
- **방 만들기·게임 진행** — 지도 위 구역·감옥 그리기, QR 체포, 재진입 시 경로 복원
- **채팅·커뮤니티** — 인게임 채팅 UI 개선·신고·차단, v3 커뮤니티(모집글·모임 채팅방, 알림, 강퇴). 커뮤니티 채팅은 STOMP로 채팅 목록 실시간화(유저당 알림 채널·안 읽은 개수)와 고정 공지를 직접 연결. 게임용 STOMP 연결 기반은 팀원이 구현
- **백그라운드 위치·연결 유지** — iOS 백그라운드 위치, Android Foreground Service
- **잠금 화면 게임 현황** — iOS Live Activity, Android 게임 진행 알림
- **배포 자동화** — GitHub Actions·Fastlane으로 iOS·Android 배포, PR CI 테스트
- **디스코드 브레인 봇** — 팀 디스코드에서 말로 물으면 지식 볼트를 찾아 근거와 함께 답함 (Second Brain Template)

과정

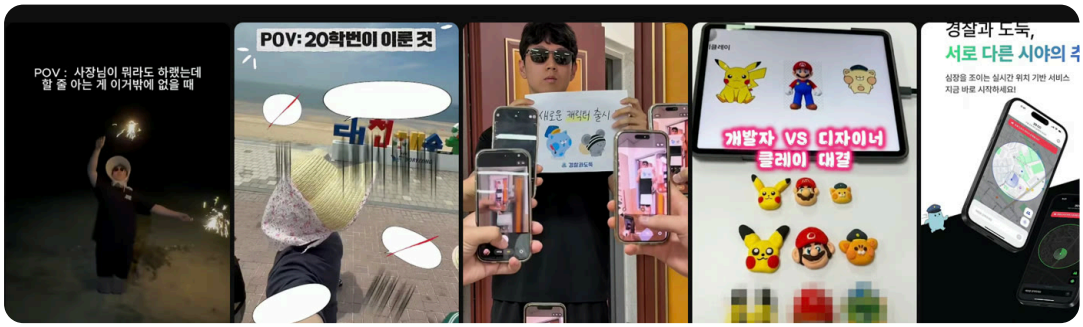
앱 개발자 두 명이 동시에 개발할 수 있게 구조부터 정했습니다. 처음에는 두 명이 앱을 맡았고, 제가 제안한 Feature-First 구조와 Clean Architecture로 도메인 계층은 인터페이스만 두고 데이터 계층이 이를 구현하게 해(의존성 역전), 기능을 나눠 병렬로 개발했습니다. 다른 팀원은 원래 웹 개발자로, 앱 개발을 돕기 위해 합류한 상태였습니다. 후반으로 가면서 홍보와 검색 노출(SEO·AEO)을 위한 웹사이트가 필요해졌고, 앞으로 도메인도 필요할 것으로 보여 그 팀원은 웹사이트와 어드민 콘솔 개발로 보내고 앱은 제가 이어서 맡았습니다. 앱 안 약관 열람, 동의 흐름, 약관 개정은 제가 맡았습니다. 약관·크레딧을 웹 정본으로 두고 앱이 웹뷰로 여는 구조는 팀원이 맡았습니다.

첫 이슈는 기능이 아니라 배포였습니다. 앱 저장소의 1번 이슈가 iOS TestFlight 배포 자동화였고, 이어서 Android 릴리스 서명과 Play Store 배포를 붙였습니다. 이후 빌드 환경 문제로 배포가 여러 번 깨져, Fastlane 설치를 Bundler 방식으로 통일했습니다.

잠금 화면에서도 게임이 보이게 했습니다. 휴대폰을 주머니에 넣고 뛰는 게임이라, 앱을 열지 않아도 다음 위치 공개 시각과 도주 중인 도둑 수를 보여 줍니다. iOS 확장 기능은 따로 서명돼야 해서 TestFlight 배포 설정도 함께 고쳤습니다.



광진구 청춘대로 학교 대항전 참가자를 모으려고 QR 박스를 쓰고 거리로 나간 영상 (팀 인스타그램)



유튜브 @cops._.nrobbers 쇼츠 (조회수 1천 · 3.4천 · 1.4천 · 160 · 1.7천, 2026.10.22 기준)

일한 방식

사용자 관점 · HOW I WORK ↑

Lock은 걸어도, Block은 말고.

잡힌 사람도 지도는 보게 하자고 제안했습니다.

상황 축제를 열흘 앞둔 리허설에서, 잡힌 사람이 감옥을 찾지 못했습니다. 닫을 수 없는 체포 안내창이 지도를 가렸기 때문입니다. 채팅은 참가자 목록 속 좁은 창에 있어서, 감옥에서 채팅할 수 있다는 것도 잘 몰랐습니다.

한 일 화면이 한 단계 깊어지더라도 채팅은 전체 화면으로 빼자고 했습니다. 잡혀도 지도는 보이게 하고, 그 위에 철창을 덮자고도 했습니다. 팀원이 "그러면 버튼이 안 눌릴 것처럼 보인다"고 반대해서, 감옥 밖을 어둡게 하는 쪽으로 정했습니다.

결과 9월 13일에 배포하고 보니, 화면 전체가 어두워져 갇혔다는 느낌이 약했습니다. 그래서 버튼을 철창보다 위에 두어 그대로 눌리게 하고, 철창을 다시 넣었습니다. 9월 19일 축제 전에 나갔습니다.

디자인 협업 · HOW I WORK ↑

Right는 Right에.

확인 창의 긍정 버튼을 오른쪽으로 옮기자고 했습니다.

상황 디자인 시안에서는 확인 창의 긍정 버튼이 왼쪽에 있었고, 두 버튼의 색도 같았습니다.

한 일 디자이너에게 긍정 버튼에 색을 넣고 오른쪽으로 옮기자고 제안했습니다. 오른손잡이가 많으니, 자주 누르는 버튼을 오른쪽에 두자는 이유였습니다.

결과 디자이너가 시안을 고쳤고, 저는 이 규칙을 앱의 공용 확인 창에 넣었습니다. 지금 앱의 확인 창 31곳이 이 창을 씁니다. 버튼이 두 개인 19곳은 모두 왼쪽이 회색 취소, 오른쪽이 색이 들어간 확인입니다.

책임감 · HOW I WORK ↑

제가 세운 Gate, 제가 걷습니다.

제가 넣은 배터리 설정 요구를, 원인부터 고쳐 없앴습니다.

상황 Android에서 화면을 끄면 게임 연결이 끊겼습니다. 저는 기기 배터리 정책 탓으로 보고, 배터리 설정을 "제한 없음"으로 바꿔야 방에 들어오게 막았습니다. 약 3개월 뒤, 이 요구가 가장 큰 진입 장벽이 됐습니다.

한 일 지도·음악 앱은 이런 설정을 요구하지 않는다는 점에서 다시 봤습니다. 진짜 원인은 CPU가 잠드는 것이었고, CPU를 깨워 두도록 고쳤습니다.

결과 배터리 설정 요구를 없앴습니다(v2.4.8). 게임에 들어가는 조건은 위치 권한 하나만 남았습니다.

문제 해결 · HOW I WORK ↑

Count 말고 Cause를.

튜토리얼은 횟수를 바꾸다가, 구조를 바꿨습니다.

상황 "튜토리얼을 강제한다, 답답하다"는 피드백이 계속 들어왔습니다. 팀별 한 번, 팀과 상관없이 한 번, 별도 페이지로 노출 방식을 세 번 바꿨지만 같은 불만이 이어졌습니다.

한 일 횟수가 아니라 구조가 문제라고 봤습니다. 건너뛸 수 없는 안내와 닫을 수 없는 창이 게임 직전에 연달아 떴습니다. 이 안내를 모두 걷어내고, 설치 후 한 번만 보는 온보딩 화면으로 바꿨습니다.

결과 9월 2일 v3.0.1로 배포했습니다. 온보딩은 처음 한 번, 네 장으로 끝납니다.

기술적 결정

지도는 초반에 Google Maps로 확정

상황 개발 초반에 지도 SDK를 정해야 했고, 처음 후보는 카카오·네이버 지도였습니다.

결정 한국어·영어·일본어 지원과 해외 사용자까지 목표로 하자고 제안했습니다. 해외에서도 똑같이 동작하는 Google Maps로 초반에 확정했습니다.

대안 카카오·네이버 지도는 국내 중심이라 글로벌 목표와 맞지 않아 제외했습니다.

결과 어느 나라에서든 같은 지도 위에 플레이 구역과 감옥을 원형·폴리곤으로 그립니다. 42개국에서 사용됐습니다.

백그라운드를 도입하고, 배터리 설정 요구는 원인부터 고쳤습니다

상황 앱에는 세 상태가 있습니다. 화면을 보는 foreground, 홈으로 나가거나 화면을 끈 background, 앱을 종료한 terminated입니다. foreground에서는 API와 웹소켓이 잘 동작했지만, background로 넘어가면 OS가 앱 프로세스를 멈춰 웹소켓이 끊기고 API 요청도 나가지 않았습니다.

문제 앱을 다시 열어야 이벤트가 보였습니다. 서버에는 background로 가기 직전의 위치만 남아, 경찰이 도둑의 옛 위치를 보고 혼선이 컸습니다.

결정 필요성을 제가 먼저 제기해 도입했습니다. Android는 Foreground Service, iOS는 background location 모드로 프로세스와 위치 전송을 유지했습니다. 위치를 상시 수집하게 되어 개인정보처리방침과 이용약관도 함께 개정했습니다.

후속 Android에서 화면을 끄면 STOMP 하트비트가 끊겨 게임이 중단되는 문제가 남았고, 처음에는 사용자에게 배터리 "제한 없음" 설정을 요구해 우회했습니다. 3개월 뒤 이 요구가 가장 큰 진입 장벽이 됐습니다. iOS는 추가 설정 없이 동작하고 내비게이션·음악 앱도 이런 설정을 요구하지 않는다는 점에서 원인을 다시 봤습니다.

원인 Foreground Service만으로는 CPU가 잠들고 있었습니다.

재결정 프로세스는 Foreground Service로, CPU는 `PARTIAL_WAKE_LOCK`으로 따로 깨워 뒀습니다. Wake Lock은 정상 종료, 태스크 스와이프, 4시간 타임아웃 중 하나로 반드시 해제됩니다.

결과 배터리 차단 다이얼로그가 사라지고, 게임 진입 조건은 위치 권한 하나만 남았습니다.

초대는 링크 하나로 — 앱과 웹이 나눠 맡은 딥링크

상황 초대 코드를 직접 입력하거나 앱 안 스캐너로 QR을 찍어야 했고, 기본 카메라로 찍으면 의미 없는 텍스트만 보였습니다.

문제 링크로 앱을 바로 열려면 도메인에 검증 파일이 있어야 했는데, 그 도메인은 웹 팀원이 맡은 Next.js 저장소에 있었습니다. 게다가 카카오톡 같은 인앱 브라우저에서는 OS가 링크를 가로채지 않아, 앱이 있어도 스토어로 넘어갔습니다.

대안 Firebase Dynamic Links는 2025년 8월 서비스가 종료돼, 처음부터 App Links·Universal Links 같은 OS 네이티브 딥링크로 설계했습니다.

결정 앱은 `app_links`로 링크를 받고, 로그인 전에 받은 초대는 로그인 후 방으로 이어 줍니다. 웹의 `/join` 페이지는 인앱 브라우저를 빠져나와 앱을 깨웁니다. 웹 팀원에게는 검증 파일 배치와 폴백 페이지 코드를 담은 핸드오프 문서를 넘겼습니다.

결과 대기방 QR도 같은 링크로 바꿔, 카카오톡으로 받은 링크나 기본 카메라로 찍은 QR 하나로 방에 바로 들어오게 됐습니다.

카드 탭 체포 대신 QR 대면 체포, 그리고 30초 만료

상황 경찰이 목록에서 도둑 카드를 눌러 체포하다 보니, 잘못 누르거나 멀리서 체포하는 일이 생겼습니다.

결정 도둑이 띄운 QR을 경찰이 카메라로 찍어야만 체포되게 바꿨습니다. 찍는 행위 자체가 같은 자리에 있다는 증거라 확인 모달도 없앴습니다.

후속 QR에 참가자 ID만 있어 찍어 둔 사진으로 언제든 체포할 수 있었기에, 만료 시각을 넣어 30초가 지나면 거부되게 했습니다.

대안 서버 서명은 위조 QR까지 막지만 백엔드 변경이 필요합니다. 함께 뛰는 사람들끼리의 게임에서 위조는 현실적인 위협이 아니라고 보고 넣지 않았습니다.

결과 체포는 실제로 마주친 상태에서만 성립하고, 만료 경계는 단위 테스트로 고정했습니다.

REST는 조회·명령, STOMP는 실시간 이벤트

상황 체포·탈옥 같은 요청과 위치 공개·체포 알림 같은 이벤트가 한 게임 안에서 동시에 오갑니다.

결정 요청은 REST로 보내고, 이벤트는 STOMP로 받도록 나눴습니다.

결과 연결이 끊겼다 복구되면 REST 상태 스냅샷으로 놓친 상태를 보정합니다. 이 재연결 복구는 팀원이 구현했습니다.

심사 거부로 배운 위치 표시등

상황 표시등 없이 백그라운드에서 위치를 수집했다가 Apple 심사에서 거부됐습니다.

결정 iOS 위치 인디케이터 표시를 정책 요건으로 받아들여 넣었습니다. 게임이 끝난 뒤에도 표시등이 남아 위치를 계속 수집하는 것처럼 보이는 문제는 위치 구독 해제 시점을 바로잡아 고쳤습니다.

교훈 "백그라운드 위치 수집은 사용자에게 보이는 신호와 함께 설계한다"를 팀 규칙으로 기록했습니다. 이후 스토어 심사 대응의 기준이 됐습니다.

지도에 격자를 깔아 "어디"를 한마디로

상황 "어디로 가는 중"이라고 쓰면 추상적이라 알아듣기 어려웠고, 뛰면서 채팅을 칠 틈도 없었습니다. 위치를 알리는 핑은 2.5초면 사라져 거의 쓰이지 않았습니다.

계기 한동안 붙잡고 있었지만 좋은 방법이 떠오르지 않았습니다. 그러다 누워서 쉬던 중 천장에 앉아 있는 파리를 보다가, 지도에 좌표를 붙이면 되겠다는 생각이 떠올라 바로 만들어 팀에 보여 줬습니다.

결정 지도 위에 체스판처럼 격자를 깔고, 칸마다 "D5"처럼 이름을 붙였습니다. 열은 왼쪽부터 알파벳, 행은 위에서부터 숫자라서 "D5"는 D열 5행입니다. 격자를 켠 채 핑을 찍으면 "D5에서 도둑을 봤어요"가 팀 채팅에 자동으로 올라가서, 핑을 놓쳐도 기록이 남습니다.

조정 칸 크기는 플레이 구역의 크기에 맞춰 자동으로 정해집니다. 격자는 켠 사람에게만 적용해, 원하지 않는 사람의 채팅이 늘지 않게 했습니다.

구상 나중에 보이스 채팅이나 무전기가 생기면 "D5에 둘, E6으로 이동"처럼 말로 브리핑하기도 쉬워집니다.

서비스를 도입한 이유

- **소셜 로그인** — Firebase Authentication. 일반 회원가입보다 접근성이 높고 가입 이탈이 적다는 사례를 근거로 제안했습니다. Google·Apple 로그인 후 백엔드 JWT를 발급받습니다.
- **딥링크** — 링크 하나로 바로 유입되면 가입·유입이 늘어난다는 사례를 근거로 제안했습니다. Google이 소개한 쇼피 사례에서는 딥링크 도입 후 앱으로 바로 들어오는 쇼핑 트래픽이 약 20%에서 50%로 늘고 결제가 126% 증가했고(Think with Google), 딥링크 캠페인의 전환율이 일반 링크보다 51% 높다는 Moburst 수치(Airbridge)와 유입 35% 증가(AppsFlyer)도 있었습니다. 넣어서 잃을 것은 없고 얻을 것만 있다고 판단했습니다. 앱 수신부는 제가, 웹 랜딩은 웹 팀원이 맡았고, 새 경로를 열 때 앱과 웹을 한 작업 단위로 보는 협업 규칙을 만들었습니다. 자세한 내용은 위 딥링크 항목입니다.
- **FCM** — 당장 필요한 기능은 아니었지만 APNs 설정 등 준비할 것이 많아 초기에 한 번에 세팅했습니다. 이후 게임 시작·인게임 채팅·커뮤니티 알림에 쓰였고, 채팅 푸시는 30초에 1건으로 묶어 알림 과다를 막았습니다.
- **Remote Config** — 이벤트 배너·점검·광고 on/off를 앱 재배포 없이 바꾸려고 도입했습니다. 설정이 없거나 실패하면 꺼지게 했습니다.

- **Analytics** — 로그인부터 이탈까지 어디서 사용자가 빠지는지 데이터가 없어 도입했습니다. 계측 실패가 앱 흐름을 막지 않게 했습니다.
- **AdMob** — 수익화 수단이 없어 게임 종료 전면광고와 커뮤니티 네이티브 광고를 넣었습니다. 광고가 실패해도 게임은 진행되고(fail-open), Remote Config로 원격에서 끌 수 있습니다.
- **Crashlytics** — 앱이 갑자기 꺼졌을 때 원인을 모아 보여 주는 도구입니다. 화면을 그리다 난 오류와 잡히지 않은 비동기 오류를 릴리즈 빌드에서만 수집하고, 디버그 빌드에서는 끕니다. Crashlytics 설정이 실패해도 앱은 계속 뜹니다.

결과

- iOS App Store·Google Play에 출시했고, 한국어·영어·일본어를 지원합니다.
- Firebase Analytics 기준 2026년 1월 1일부터 10월 12일까지 새 사용자 840명이 들어왔고, 한국·미국·일본 등 42개 국가에서 앱을 사용했습니다.
- 탈퇴를 제외하고 남아 있는 사용자 수는 딥링크·QR 전환을 마친 뒤 156명(2026.06.18)에서 660명(2026.10.19)으로 약 4배가 됐습니다.
- 잠금 화면 게임 현황(iOS Live Activity·Android 게임 진행 알림)을 v3.1.20으로 배포했습니다.
- 수상: 세종 SW 창업 아이디어 리그 대상, 세종대 창업아이템 경진대회 우수상, 입주공모전 우수 창업 아이템상, 아롬 데모데이 특별상, SSUP 선정
- 오프라인 행사 참여: 서울게임타운, 서울 대학생 연합 페스티벌, 광진구 청춘대로

배운 점

- **사용자에게 설정 변경을 요구하기 전에, 같은 종류의 앱은 어떻게 동작하는지 먼저 확인해야 합니다.** 내비게이션·음악 앱은 배터리 설정을 요구하지 않았고, 실제 원인은 CPU가 잠드는 것을 놓친 제 구현이었습니다.
- **보안 대책은 막을 위협의 범위부터 정해야 합니다.** 막을 대상을 찍어 둔 QR 사진의 재사용으로 정하니, 서버 서명까지는 필요 없다고 판단할 수 있었습니다.
- **백그라운드 위치 수집은 사용자에게 보이는 신호와 함께 설계해야 합니다.** 표시등 없이 올렸다가 심사에서 거부된 뒤, 이를 팀 규칙으로 남겼습니다.

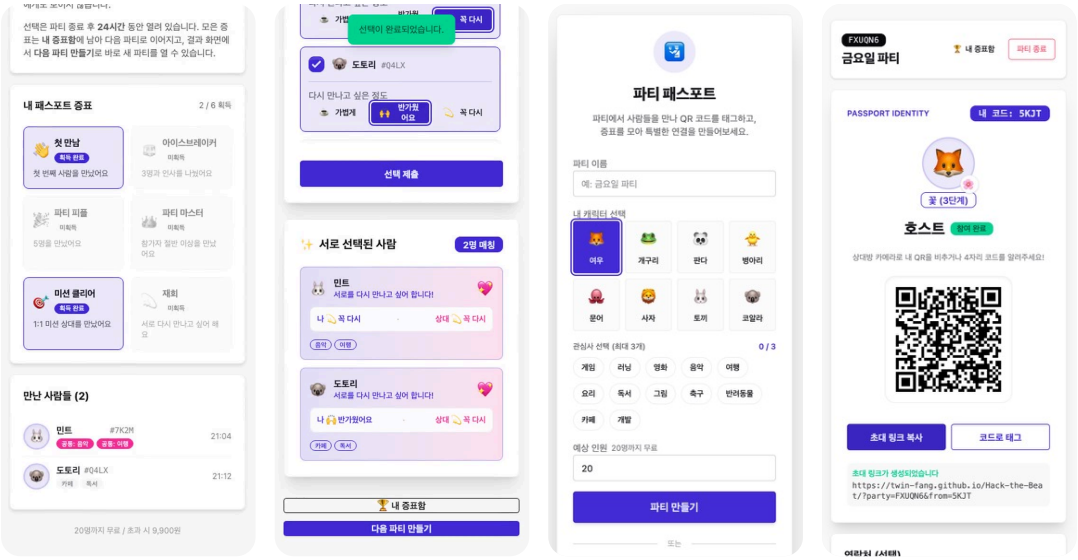
Party Passport

2026.08 · React · TypeScript · Vite

Google I/O Extended: Hack the Beat 1st Place

파티에서 QR로 만난 사람을 스탬프로 모으고, 서로 고른 사람만 매칭되는 웹앱

Live Demo **GitHub** **Instagram 행사 기록** **LinkedIn 행사 기록**



React · TypeScript · Vite

배경

2026 I/O Extended: Hack the Beat는 개발 시간이 3시간 남짓인 해커톤이었고, 주제는 "**파티를 더 잘 즐길 수 있는 서비스 만들기**"였습니다. 가장 큰 특징은 **심사위원이 사람이 아니었다는 점**입니다. 창업가·엔지니어·투자자 역할의 AI 에이전트가 세 번씩 채점하고, Playwright가 배포 링크에 접속해 실제로 작동하는지 확인했습니다.

그래서 팀 4인큐(서새찬·홍의민·백지훈·이예람)는 제품을 만들기 전에 채점 기준부터 분석했습니다. 짧은 시간 안에는 한 사람이 개발하는 편이 빠르다고 보고 구현은 서새찬 님이, 아이디어와 기획은 제가 맡았습니다.

맡은 역할

- **서비스 아이디어 제안** — 파티 패스포트 주제 제안 문서 작성, v2~v4 개정
- **심사관 페르소나로 기획 검증** — 세 페르소나를 구축해 제 기획안을 리뷰하고 v3에 반영
- **수익 모델 기획** — 정원 초과 유료 옵션, 다음 단계의 연락처 교환 옵션 명세
- **프론트엔드·백엔드 PRD 작성** — 구현 담당 팀원이 따를 화면 문구, API, 데이터 구조

과정

주최 측이 공개한 채점표부터 봤습니다. 심사에 그대로 쓸 채점표가 미리 공개됐습니다. 12개 항목을 0~10점으로 매기고, 세 영역을 가중 합산한 뒤 주제 적합성을 배율로 곱합니다.

- **작동 여부 (34%)** — 핵심 흐름 완주, 배포 안정성, 기획안과 구현의 일치, 완성도. 브라우저 검증 결과로만 채점합니다.
- **GTM 전략 (33%)** — 타깃 고객, 첫 유입 채널, 파티형 확산 구조, 재방문 이유
- **비즈니스 가능성 (33%)** — 수익 모델, 시장 규모, 차별점, 운영 비용
- **주제 적합성 (배율)** — 더하지 않고 곱합니다. 주제와 멀면 총점이 최대 0.35배까지 줄어듭니다.

채점표 본문에는 점수 상한도 숨어 있었습니다. "~할 계획"처럼 의도만 적으면 5점, 기획안에 쓴 기능이 브라우저에서 확인되지 않으면 그 기능에 기댄 항목은 6점을 넘을 수 없었습니다.

채점표를 거꾸로 따라가 심사관 페르소나부터 만들었습니다. 실제 심사 에이전트의 프롬프트는 비공개라, 채점표와 운영진 설명으로 창업가·엔지니어·투자자 3명을 설계했습니다. 세 페르소나 모두 12개 항목을 전부 채점하고, 무엇을 근거로 인정하는지만 다릅니다. 최종 점수는 세 심사관의 평균이라, 한 명에게만 잘 보이면 점수가 3분의 1만 오릅니다.

- **창업가** — 첫 사용자 100명을 어디서 누가 데려오는지, "단톡방·인스타로 이미 되는 거 아니야?"를 묻습니다. 기획안에 쓴 기능이 앱에 실제로 있는지도 확인합니다.
- **엔지니어** — 브라우저 검증 기록만 믿습니다. 제출한 3단계가 모두 성공했는지, 에러는 없는지, 기획안의 기능이 화면에서 확인되는지를 봅니다.
- **투자자** — 숫자 없는 문장은 인정하지 않습니다. 누가·언제·얼마를 내는지, 시장 규모를 직접 계산했는지, 대체재를 이름으로 댈 수 있는지를 봅니다.

파티에 모이는 이유를 새로운 인연으로 봤습니다. 사람들이 파티에 가는 가장 큰 목적은 새로운 사람을 만나는 데 있다고 생각했습니다. 그런데 막상 가면 처음 보는 사람에게 말 걸 계기가 없어, 아는 사람하고만 이야기하다 돌아옵니다. "인스타 좀 주실래요?"처럼 바로 연락처를 묻는 것도 부담스럽습니다. 그래서 말을 거는 일 자체를 게임으로 만든 **파티 패스포트**를 제안했습니다.

- **피크민의 수집 느낌** — 피크민이 걸음 수를 모으듯, 대화한 사람과 서로 QR을 태그해 "만난 사람 수"를 모읍니다. 많이 만날수록 내 캐릭터가 새싹에서 꽃으로 자랍니다.
- **포켓몬 GO의 획득 느낌** — 첫 만남, 3명, 참가자 절반처럼 기준을 넘길 때마다 증표를 얻고, 증표는 다음 파티에도 남습니다. "증표 받으려면 3명은 만나야 해"가 말 걸 핑계가 됩니다.
- **소개팅 앱의 장점** — 참여할 때 관심사를 고르면, 1:1 미션 상대와 겹치는 관심사가 대화 소재로 뜹니다. 파티가 끝나면 다시 만나고 싶은 사람을 몰래 고르고, 서로 고른 사이에만 결과를 보여 줍니다.

아이디어도 채점표로 골랐습니다. 팀원들이 낸 아이디어 7개를 완벽히 구현한다고 가정하고 하나씩 채점했고, 최고점은 파티 패스포트였습니다. 확산 구조 항목은 "초대·동반 참여 없이는 제품이 성립하지 않아야 10점"인데, 파티 패스포트는 만남이 곧 제품이라 혼자서는 쓸 수 없습니다. 모든 동작을 브라우저에서 확인할 수 있다는 점도 맞았습니다.

첫 채점에서 투자자 점수가 가장 낮았습니다. 기획안을 세 페르소나로 채점하니 10점 만점에 창업가 6.4, 엔지니어 7.2, **투자자 3.8**이었습니다. 투자자가 엄격히 보는 수익 모델과 차별점이 약했고, 엔지니어는 초대 링크로 들어오기만 해도 "만남"이 기록된다고 지적했습니다. 이 지적을 반영해 기획안을 고쳤습니다(v3).

기술적 결정

"만남"은 사용자가 직접 확인한 기록만 인정

상황 v2는 초대 링크로 참여하면 초대한 사람과 자동으로 "만남"이 기록됐습니다.

문제 원격으로 들어와도 만남이 쌓여 핵심 숫자를 믿을 수 없다는 것이 엔지니어 페르소나의 지적이었습니다.

결정 v3 기획안에서 자동 기록을 없애고, 상대의 4자리 코드를 받아 `지금 만났어요`를 눌러야 기록되게 했습니다.

결과 구현이 Spring REST 서버로 바뀌면서 제출본에는 자동 기록이 남았고, v4 문서에 결정 대기로 표시했습니다.

수익 모델: 호감 확인 결제 대신 정원 초과 유료 옵션

상황 처음 생각한 "파티 후 받은 호감을 확인할 때 결제"는 심사 에이전트가 테스트할 수 없어 점수 상한에 걸렸습니다.

결정 파티 인원이 20명을 넘으면 결제하는 유료 옵션으로 바꿨고, 결제 화면과 서버 쪽 차단은 구현 담당 팀원이 만들었습니다.

결과 다시 채점하니 점수가 유의미하게 올랐습니다.

이후 호스트가 켜는 연락처 교환 유료 옵션(9,900원, 서로 선택한 쌍에게만 공개)을 명세했고, 해커톤 시간 안에 구현하지는 않았습니다.

QR을 못 찍는 사용자를 위한 "코드로 태그"

상황 1차 사용자는 자동화 브라우저였고, 브라우저는 카메라로 QR을 찍을 수 없습니다.

결정 4자리 코드 입력을 기본 기능으로 PRD에 넣고, 화면 문구도 제출 시나리오와 글자 단위로 맞췄습니다.

결과

- 2026 I/O Extended: Hack the Beat에서 **최종 1위**를 했습니다.
- 해커톤 현장에서 참가 팀들이 서로의 서비스를 써 보는 시간에 **참가자 29명이 서비스를 실제로 사용**했습니다. 처음 보는 사람끼리 QR을 찍으며 증표를 모으는, 만들고 싶었던 장면이 그대로 일어났습니다.

배운 점

- **평가자가 시스템이면 채점 기준을 요구사항처럼 다뤄야 합니다.** "테스트할 수 없는 결제"를 "화면에서 지나갈 수 있는 결제"로 바꾸는 것만으로 점수가 움직였습니다.
- **제 아이디어도 검증받기 전까지는 가설로 봐야 합니다.** 제 기획안도 첫 리뷰에서 3.8점을 받았고, 지적을 받아들여 고친 버전이 최종안이 됐습니다.

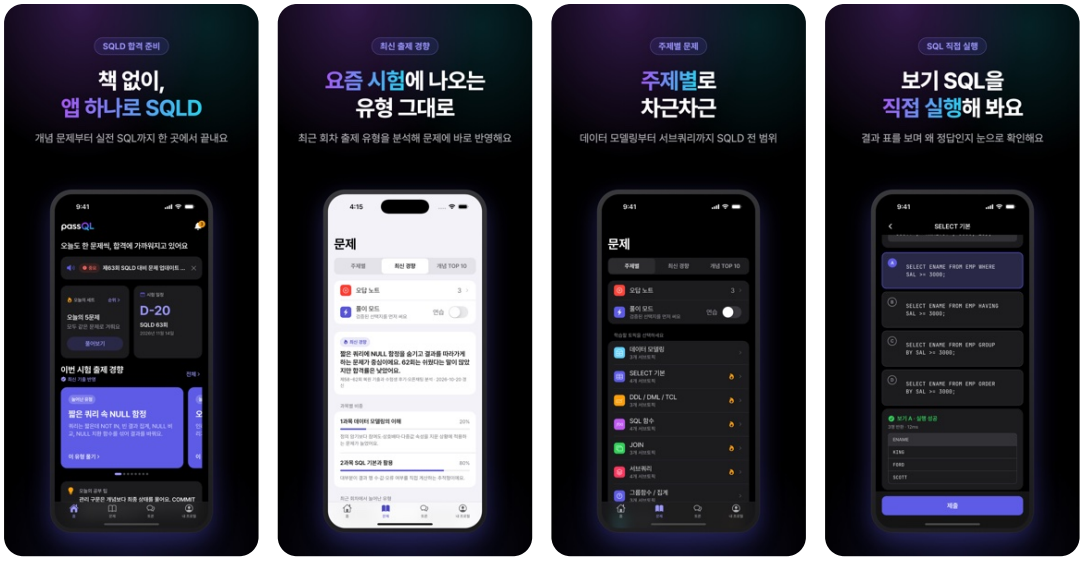
passQL

2026.04 – 2026 · Flutter · Riverpod · React · TypeScript

App Store 출시 · 웹 서비스

AI가 함정 오답을 만들고, 보기 SQL을 샌드박스 DB에서 실행해 확인하는 SQLD 학습 앱

App Store **Website** **GitHub**



Flutter · Riverpod · React · TypeScript

배경

passQL은 SQLD 문제를 풀면서 보기 SQL을 실제로 실행해 보고, 틀린 이유를 AI 해설로 확인하는 학습 서비스입니다. iOS 앱과 웹으로 출시했고, Google Play는 준비 중입니다.

Twin-Fang 팀 2명이 2026년 4월부터 만들고 있습니다. 서버와 AI는 팀원이 맡았고, 저는 웹·앱 화면과 세컨드브레인 도입을 맡았습니다.

문제는 이렇게 만들어집니다

- **AI가 함정 오답을 만듭니다.** 정답 SQL을 먼저 실행하고, 그 결과를 보고 수험생이 헷갈리기 쉬운 지점(잘못된 JOIN 조건, GROUP BY 누락, NULL 처리 실수)을 겨냥합니다.
- **샌드박스 DB가 검증합니다.** 문제마다 임시 PostgreSQL DB를 만들어 모든 선택지를 실행하고, 정답이 딱 하나일 때만 저장합니다.
- **Oracle 문법은 바꿔서 실행합니다.** `NVL`·`SYSDATE` 같은 함수는 PostgreSQL 문법으로 바꾸고, 바꿀 수 없는 문법은 개념 문제로 냅니다.

맡은 역할

- **웹 프론트엔드 (2026.04)** — React·TypeScript. 문제 풀이, 설정, 건의사항 화면과 daisyUI 디자인 토큰 정리
- **Flutter 앱 UI/UX (2026.10)** — 신규 가입자 온보딩, 홈·하단 탭·내 프로필·토론 탭 개편
- **세컨드브레인 도입** — 이슈와 결정 기록을 에이전트가 꺼내 쓰는 볼트로

구현은 AI 에이전트에게 맡기고, 저는 원인을 좁히고 결과를 검증하는 일을 맡았습니다.

홈이 2.5초 동안 비어 있던 문제

상황 앱을 켜면 홈이 3.5~4초 동안 스켈레톤으로만 보였습니다.

원인 홈은 인사말·일정, 학습 현황, 추천 문제 세 묶음이 모두 와야 그려졌습니다. 추천 문제는 등급을 확인한 뒤에 요청해서 가장 늦었습니다. 맨 아래 섹션 하나가 맨 위 카드까지 붙잡고 있었습니다.

한 일 추천 문제를 첫 표시 조건에서 빼고, 도착하면 맨 아래에 따로 채우게 했습니다. 다시 합쳐지지 않도록, 추천이 끝나지 않아도 홈이 뜨는지 보는 테스트를 넣었습니다. 이 테스트는 수정 전에 실패하고 수정 후에 통과합니다.

결과 홈 내용이 뜨기까지 평균 **2476ms에서 191ms**로 줄었습니다. 추천 문제는 그대로 약 2.5초 뒤에 붙습니다.

iPhone 16 Pro 시뮬레이터 디버그 빌드에서 운영 API로 10회 잰 값입니다. 핫 리스타트로 재서 앱 시작 단계(Firebase 초기화, 세션 복원)는 빠져 있습니다.

세컨드브레인 도입

코드만 봐서는 "왜 이렇게 됐는지" 알 수 없었습니다. 이유는 GitHub 이슈와 작업 리포트 댓글에 흩어져 있었습니다. 그래서 제가 만든 Second Brain Template을 모노레포에 설치했습니다.

- **넣은 기록**: 이슈 428개, 완료 리포트 347건, 결정 212개를 볼트에 넣었습니다.
- **바뀐 결정 잇기**: 결정이 바뀐 곳은 대체 관계로 이어, 지금 유효한 결정과 지난 결정을 나눴습니다. 하단 탭 결정은 네 번 바뀌었는데, 어느 결정을 찾아도 현재 결정으로 이어집니다. 범위가 겹쳐 판단이 필요한 4건은 main 코드를 보고 정했습니다.
- **지어낸 이유 막기**: 시험 삼아 5건을 넣고 원본과 대조하니, 원본에 없는 이유를 사실처럼 쓴 문장이 8곳 나왔습니다. 그래서 원본에 없는 이유는 "논의 기록 없음"으로 적게 했습니다. 배치마다 파일명·금액·버전을 원본과 대조했고, 마지막에 결정 19개를 다시 대조했습니다.
- **공개 저장소 정리**: 저장소가 공개라서 NAS 경로, 운영 도메인, 내부 포트를 노트에서 가렸습니다. 현재 파일만이 아니라 브랜치의 커밋 전체에서 지웠습니다.
- **계속 쌓이게**: 작업 리포트를 쓴 직후 지식화해 같은 PR에 담는 규칙을 `CLAUDE.md`에 넣었습니다.
- **꺼내 보기**: 볼트 검사 오류는 0입니다. "요금제"를 물으면 1·3개월 이용권 결정이, "결제 스위치"를 물으면 관련 이슈와 결정 두 개가 근거로 나옵니다.

이슈 노트 약 775개의 증상·원인 요약은 표본만 원본과 대조했습니다.

배운 점

- **첫 화면이 늦으면, 가장 느린 섹션이 전체를 붙잡고 있는지부터 봅니다.** 실제 처리 시간이 같아도 기다리는 시간은 줄일 수 있습니다.

- **AI가 정리한 기록도 원본과 대조합니다.** 5건만 넣었는데도 지어낸 이유가 8곳 나왔습니다.

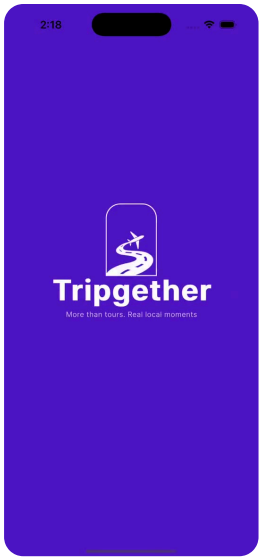
트립게더

2025.09 – 2025.11 · Flutter

세종대 창업캠프 대상 롯데 청년기업가대상 본선

SNS 게시물을 공유하면 장소 정보를 정리하는 AI 여행 가이드

GitHub



Flutter

배경

인스타그램 릴스나 유튜브에서 가 보고 싶은 장소를 봐도, 저장하려면 지도 앱에서 다시 검색해야 했습니다. **트립게더**는 SNS 앱의 공유 버튼으로 링크를 보내면 서버가 콘텐츠를 분석해 장소를 정리해 주는 AI 기반 여행 가이드 앱입니다.

팀 프로젝트였고, 저는 Flutter 앱을 맡았습니다. 분석 서버는 별도로 개발됐고, iOS·Android 배포 자동화는 팀원이 맡았습니다. 네이티브 공유 수신과 소셜 로그인은 제게 **처음 시도한 기술**이었습니다.

맡은 역할

- **공유 수신** — iOS Share Extension(Swift)과 Android MethodChannel(Kotlin)
- **로그인과 토큰 관리** — Google 로그인, Dio 인터셉터로 JWT 자동 주입·갱신
- **화면 구현** — 온보딩, 홈, 콘텐츠 상세, 지도, 코스 마켓, 마이페이지
- **지도** — 영업 상태를 보여 주는 커스텀 마커와 장소 바텀시트
- **푸시 알림과 다국어** — FCM 연동, 한국어·영어 지원
- **디자인 시스템** — 디자인 값을 테마로 통일, `textTheme` 사용처 76곳 이전

과정

처음 시도한 기술 ①은 네이티브 공유 수신이었습니다. 공유 수신 패키지는 iOS 빌드 문제로 빼고, Swift와 Kotlin 코드를 직접 작성했습니다. 이틀 동안 "체크포인트1~6" 커밋으로 되돌아갈 지점을 남기며 iOS부터 완성하고 Android까지 붙였습니다.

공유 화면은 여러 번 만들고 되돌렸습니다. 개발 환경에서 보이던 바텀 시트가 TestFlight 빌드에서는 바로 사라졌습니다. 진단용 빌드까지 올려 본 뒤, 바텀 시트 없이 즉시 저장하고 닫는 구조로 정리했습니다.

처음 시도한 기술 ②는 소셜 로그인이었습니다. 로그인을 취소하면 앱이 멈추는 오류와, 앱과 서버의 필드명이 달라 로그인이 실패하던 문제를 고쳤습니다. 코드에 있던 OAuth Client ID는 빌드 시점에 `.env`에서 주입하도록 바꿨습니다.

마지막 한 달은 백엔드 API를 붙이는 데 썼습니다. Mock 데이터로 화면을 먼저 만들고, 환경 변수로 실제 API와 전환하며 맞춰 갔습니다. 온보딩·관심사, 콘텐츠, 장소 저장, 지도 마커 API를 차례로 연결했습니다.

기술적 결정

Share Extension은 저장만 하고, 처리는 앱이 맡기

상황 Share Extension은 Flutter 밖의 별도 프로세스이고, 안에 UI를 넣으면 동작이 불안정했습니다.

결정 Extension은 공유 URL을 App Group 큐에 저장하고 바로 종료하며, 앱이 열릴 때 큐를 처리합니다.

결과 공유를 연달아 해도 큐에 쌓였다가 앱에서 한 번에 처리됩니다.

토큰 재발급은 인터셉터 한곳에서

상황 여러 API를 동시에 호출하다 토큰이 만료되면, 요청마다 재발급을 시도할 수 있었습니다.

결정 Dio 인터셉터에서 첫 요청만 재발급하고, 나머지는 Completer로 기다렸다가 새 토큰으로 재시도합니다.

생년월일 입력을 칸 8개에서 입력창 1개로

상황 숫자 한 칸씩 입력창 8개로 받았더니, 연속 백스페이스가 제대로 동작하지 않았습니다.

결정 입력창 하나에서 `YYYY / MM / DD`로 자동 포맷팅하고, 커서는 항상 끝으로 보내게 했습니다.

결과 입력창 8개·FocusNode 16개가 각각 1개로, 커서 로직은 40줄에서 5줄로 줄었습니다.

결과

- 세종대 창업캠프 대상
- 롯데 청년기업가대상 본선 진출
- iOS·Android 공유 수신부터 Google 로그인, 6단계 온보딩, 지도, FCM 푸시 알림, 한국어·영어 지원까지 앱 전반을 구현했습니다.

배운 점

- **개발 환경에서 되는 것과 배포 빌드에서 되는 것은 다릅니다.** 공유 화면 문제는 TestFlight 빌드에서만 드러나, 진단용 빌드를 따로 올려야 했습니다.
- **복잡한 로직을 다듬기보다 규칙을 단순하게 바꾸는 편이 나을 때가 있습니다.** 커서 계산과 공유 화면 모두, 마지막에 남은 것은 더 단순한 쪽이었습니다.

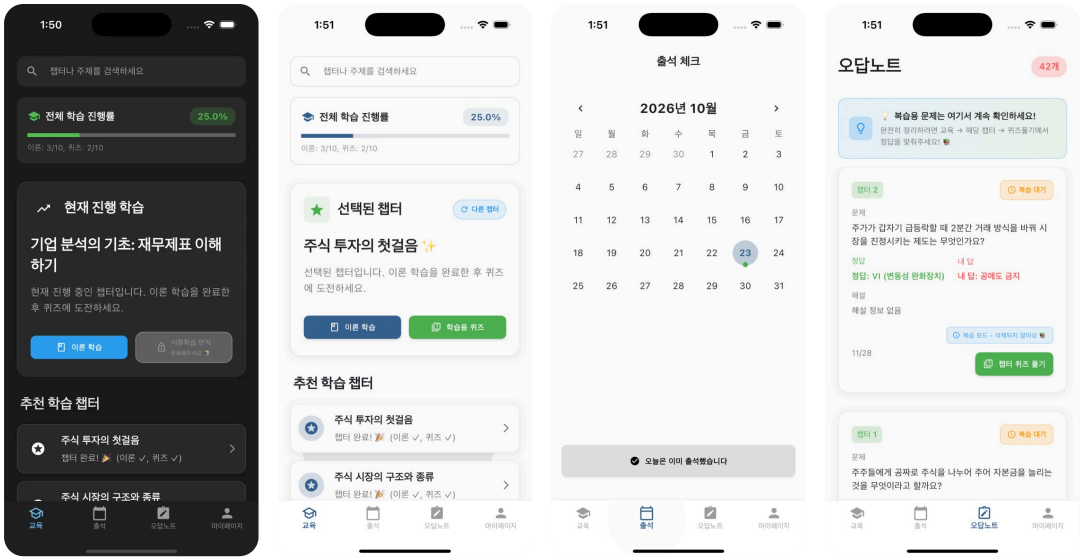
Stocker

2025.05 – 2025.11 · Flutter

Seeds 팀 프로젝트 우수상

주식투자를 배우는 핀테크 교육 앱 · 팀 돔황촤 코스피

GitHub



Flutter

배경

Stocker는 주식 투자를 처음 시작하는 사람을 위한 금융 교육 앱입니다. 이론 학습, O/X 퀴즈와 해설, 오답노트, 출석 체크, 투자 성향 분석을 한 앱에서 제공합니다.

팀 **돔황촤 코스피**로 진행했고, 저는 Flutter 앱 프론트엔드 리드를 맡았습니다. 팀원과 브랜치를 나눠 개발한 뒤 하나로 합쳤고, 백엔드는 Node.js REST API입니다.

맡은 역할

- **앱 프론트엔드 리드** — 앱 구조, 브랜치 통합, 상태관리 전환을 주도
- **앱 구조 설계** — 기능별 `data / domain / presentation` 구조와 라우팅 뼈대
- **학습 기능 화면** — 이론 학습, 퀴즈, 오답노트, 마이페이지, 다크/라이트 테마
- **브랜치 통합** — 팀원 브랜치(로그인, 성향 분석, 노트, 출석)와 제 브랜치를 통합
- **상태관리 전환** — Provider에서 Riverpod으로 단계적 마이그레이션
- **실서버 연동과 마감** — 퀴즈 API 호출 축소, 인증 가드, 릴리스 빌드 수정

과정

백엔드보다 먼저 화면을 만들었습니다. 2025년 7월 26일 저장소를 구성하고, 이론 학습 화면을 Mock 데이터로 먼저 만들었습니다. Repository를 두어 나중에 실제 API로 바꿀 수 있게 했습니다.

두 갈래로 개발한 기능을 하나로 합쳤습니다. 8월 26일 제 브랜치를 기준으로 팀원 브랜치를 통합했습니다. 통합 직후 홈 화면 `dispose()`에서 난 오류는 Provider 참조를 미리 저장해 고쳤습니다.

실서버에 붙이자 새로운 문제가 나왔습니다. 9월에는 통신·인증·퀴즈·무한 루프 문제를 차례로 고쳤습니다. 11월에는 API 요청·응답을 그대로 찍는 로깅부터 만들고 필드명과 라우팅을 맞췄습니다.

마지막 버그는 릴리스 빌드에서 나왔습니다. 릴리스 APK에서만 네트워크가 막혔고, Android 매니페스트에 INTERNET 권한이 빠져 있었습니다. 15초여야 할 Dio 타임아웃이 15000초로 계산되던 버그도 함께 고쳤습니다.

기술적 결정

Mock 우선 개발과 실행 설정으로 Mock·Real 전환

상황 백엔드 API가 완성되기 전에도 화면 개발을 진행해야 했습니다.

결정 데이터 소스를 Repository 뒤로 숨기고, `--dart-define`과 VS Code 실행 설정으로 Mock과 Real을 전환했습니다.

결과 코드를 고치지 않고 실행 환경을 바꿀 수 있어, 잘못된 설정을 커밋할 위험이 없어졌습니다.

Provider에서 Riverpod으로 단계적 전환

상황 처음 팀 가이드는 "상태 관리는 Provider만 사용"이었습니다.

문제 Provider는 BuildContext가 있어야 접근할 수 있고, 등록 누락은 실행해 봐야 에러가 났습니다.

결정 기존 Provider를 Riverpod과 공존시킨 뒤, 테마 → Repository → Notifier → UI 순서로 옮겼습니다.

결과 2025년 11월 8일 Provider 의존성을 완전히 제거했습니다.

퀴즈 API 호출을 시작·종료 두 번으로 축소

상황 시작·종료 외에 답을 고를 때마다 API를 호출해, N문제 퀴즈 한 번에 N + 2회를 호출했습니다.

문제 문제 수만큼 서버 부하가 늘고, 답을 고를 때마다 응답을 기다려야 했습니다.

결정 듀오링고 방식을 참고해, 시작할 때 문제·정답·해설을 받아 두고 답안은 끝날 때 한 번에 제출하게 했습니다.

결과 퀴즈 1회당 API 호출이 "문항 수 + 2"회에서 시작·종료 2회로 줄었습니다.

결과

- Seeds 팀 프로젝트 우수상

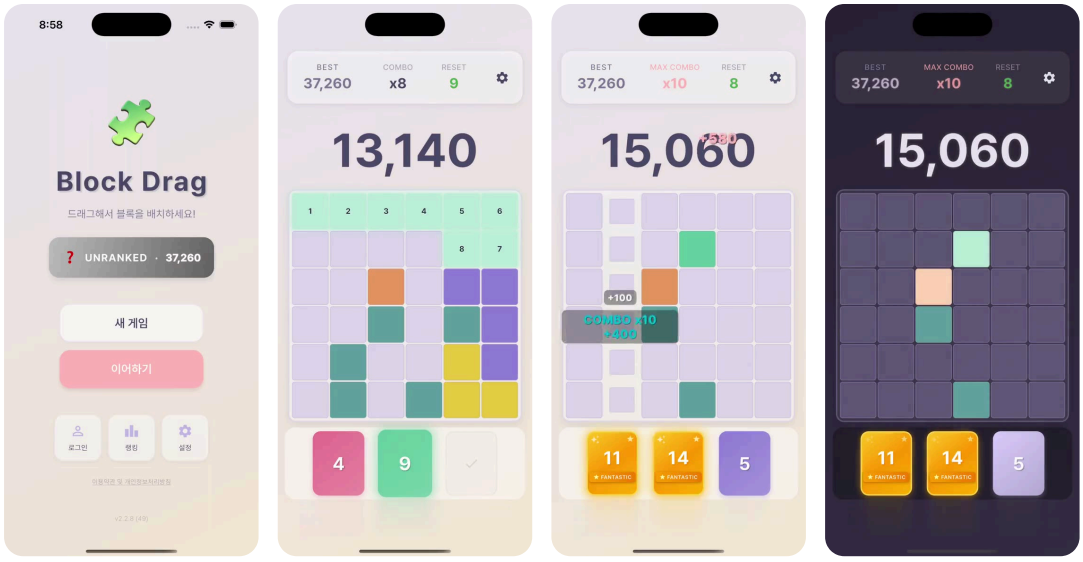
Block Drag

2025.05 – 2025.07 · Flutter · Supabase

Google Play · App Store 출시

공간 감각을 활용하는 퍼즐 게임

Google Play App Store



Flutter · Supabase

배경

Block Drag는 매 턴 주어지는 블록 3개를 드래그 앤 드롭으로 그리드에 놓아 가로·세로 라인을 완성하는 모바일 퍼즐 게임입니다. 그리드는 6x6과 8x8 중에서 고르고, 보드에는 장애물이 무작위로 놓입니다. 남은 블록을 하나도 놓을 수 없으면 게임이 끝나고, 보상형 광고를 보면 이어서 할 수 있습니다.

개발부터 스토어 배포, 출시 후 업데이트까지 혼자 진행한 1인 개발 프로젝트입니다.

맡은 역할

- **게임 로직** — 블록 생성, 라인 클리어, 게임 오버 판정, 점수·콤보 규칙
- **UI와 연출** — 드래그 경로 표시, 클리어·콤보 이펙트, 테마, 튜토리얼
- **사운드와 진동** — 배경음악·효과음, 진동 피드백
- **수익화와 배포** — AdMob 광고, Google Play·App Store 배포와 업데이트

과정

첫 주에 게임의 뼈대를 만들었습니다. 2025년 4월 3일 첫 커밋 당일에 장애물을 넣었고, 이튿날 게임 오버를 판정하는 DFS 알고리즘을 고쳤습니다. 4월 8일에는 광고를 붙였습니다.

재미는 숫자를 계속 고치며 찾았습니다. 장애물 생성 확률, 점수, 콤보 제한, 블록 가중치를 여러 번 조정했습니다. 4월 말에는 MAX 콤보에 도달하면 긴 블록을 주는 Fantastic 콤보를 넣었습니다.

구조 개선은 두 번에 걸쳐 했습니다. 9월에 게임 로직을 서비스 단위로 나눴지만, 1,200줄이 넘는 Provider 파일이 남았습니다. 그래서 12월에 Clean Architecture 기준으로 프로젝트 전체를 다시 세웠습니다.

다시 세운 뒤 기능을 하나씩 되살렸습니다. 사운드는 걷어냈다가 앱 생명주기와 연동되는 구조로 다시 넣었습니다. 이어서 튜토리얼, 크리티컬 라인 미션, 게임 통계와 티어 시스템을 추가했습니다.

기술적 결정

Clean Architecture로 전체 리빌드

상황 게임 상태와 규칙이 1,200줄이 넘는 Provider 파일 하나에 몰려 있었습니다.

결정 4계층으로 나눠 규칙은 domain 서비스로, 상태는 Riverpod 코드 생성과 Freezed 불변 엔티티로 옮겼습니다.

결과 레거시 코드를 지운 시점에 `flutter analyze` 결과가 59 errors에서 0 issues가 됐습니다.

로그인은 붙였다가 내려놓기

상황 2.1.0에서 Supabase 로그인과 랭킹을 붙였고, 12월 리빌드 때 걷어냈다가 Google OAuth로 다시 붙였습니다.

결정 2026년 1월 2일 Supabase 의존성을 제거하고, 로그인 버튼에는 "준비중입니다" 안내만 남겼습니다.

결과 게임 상태와 최고 점수는 서버 없이 기기에 저장하는 구조로 배포를 이어갔습니다.

업데이트 안내는 GitHub에 둔 appcast로

상황 버전 단위로 수정과 기능을 자주 배포했습니다.

결정 upgrader 패키지와 GitHub 저장소에 둔 appcast 파일로, 새 버전이 나오면 앱 안에서 알리게 했습니다.

결과

- Google Play와 App Store에 출시했습니다.
- 2.0.0(2025-05-10), 2.1.0(2025-07-25)을 거쳐 2.2.x 버전까지 업데이트를 배포했습니다.
- v2.2.9에서는 설정의 "점수표시" 옵션이 점수 애니메이션에 적용되지 않던 문제를 고쳤습니다.

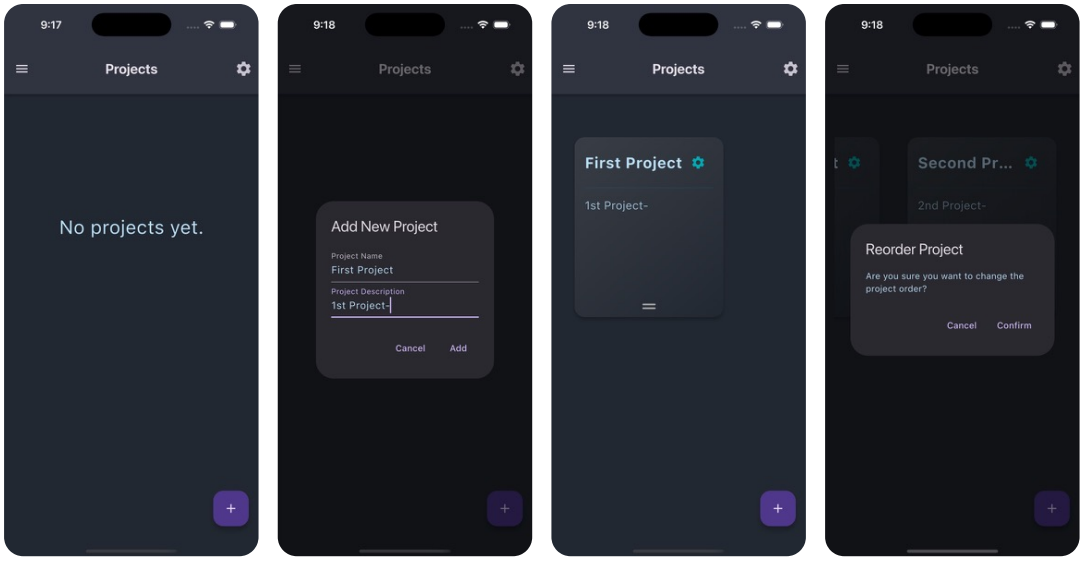
RuleBook

2025.03 – 2025.06 · Flutter

Google Play · App Store 출시

출석·벌칙·벌금을 관리하는 앱

Google Play **App Store**



Flutter

배경

교회 아이들의 출석과 벌점·상점처럼, 함께 정한 "우리들의 규칙"을 보기 편하게 정리하려고 만든 앱입니다.
RuleBook은 출석, 경고, 벌점, 벌금을 기록하고 관리하는 모바일 앱입니다. 학생을 그룹 단위로 묶고, 그룹마다 경고 몇 번이 벌점이 되는지와 벌점당 벌금을 따로 정할 수 있습니다. 데이터는 기기에 저장되고 Excel 파일로 내보낼 수 있습니다.

요구사항 문서(PRD) 작성부터 스토어 배포까지 혼자 진행한 1인 개발 프로젝트입니다.

맡은 역할

- **PRD 작성** — MVP 범위와 MVP 이후 기능 정리
- **핵심 로직** — 학생·그룹 관리, 경고의 벌점 자동 전환, 벌금 계산
- **출석 체크** — table_calendar 기반 날짜별 그룹 출석 화면
- **내보내기와 다국어** — Excel 내보내기·공유, 한국어/영어 전환
- **배포와 유지보수** — AdMob 광고, 반응형 UI, 스토어 버전 업데이트

과정

첫날 MVP를 만들었습니다. 2025년 2월 23일 첫 커밋 당일에 MVP 구현을 마쳤고, 이후 사흘 동안 UI를 다듬었습니다.

MVP 이후로 미뤄둔 기능을 출시 전에 당겨왔습니다. PRD에서 MVP 이후로 분류한 Excel 내보내기와 출석 체크를 3월에 먼저 만들었습니다. 그룹 출석은 여러 번 고친 끝에 완성했습니다.

출시 후에도 필드를 늘렸습니다. 3월 20일 정식출시 이틀 뒤 상점, 메모, 출석 일수 필드를 담아 1.1.1을 배포했습니다. 2026년 7월에는 Android 16 타깃 SDK 요구에 맞춰 1.1.7로 올렸습니다.

기술적 결정

MVP는 로컬 저장, 서버는 이후로

상황 PRD는 Firebase·Supabase 클라우드 저장과 사용자 간 동기화를 MVP 이후 기능으로 분류했습니다.

결정 서버 없이 shared_preferences와 JSON 직렬화로 기기에 저장하고, 상태는 Provider 서비스 계층으로 관리했습니다.

결과 출시 뒤 필드를 추가할 때 `fromJson`에 기본값 0을 두어, 기존 저장 데이터도 읽히게 했습니다.

경고 기준을 바꾸면 기존 기록도 새 기준으로 다시 계산

상황 경고가 그룹 기준에 도달하면 벌점 1점으로 바뀌는데, 이 기준은 언제든 바꿀 수 있습니다.

문제 기준만 바꾸면 이미 새 기준을 넘긴 학생의 경고가 벌점으로 바뀌지 않은 채 남습니다.

결정 설정을 저장할 때 학생 전원의 경고를 새 기준으로 나눠, 몫은 벌점에 더하고 나머지만 경고로 남겼습니다.

반응형 레이아웃은 flutter_screenutil로

상황 출시 두 달 뒤인 2025년 5월, 반응형 UI 작업을 진행했습니다.

결정 flutter_screenutil로 iPhone 12 화면(390×844)을 기준 삼아 크기를 비율에 맞춰 조정했습니다.

결과

- 2025년 3월 20일 Google Play와 App Store에 출시했습니다.
- 출시 후 1.1.1부터 1.1.7까지 업데이트를 배포했습니다.

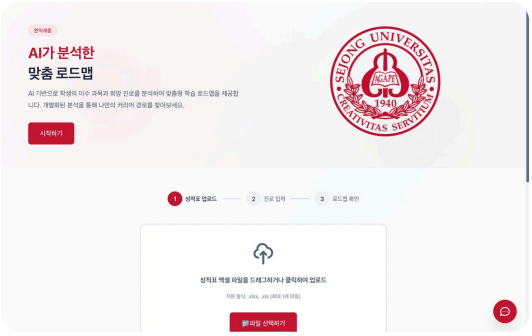
완익세종

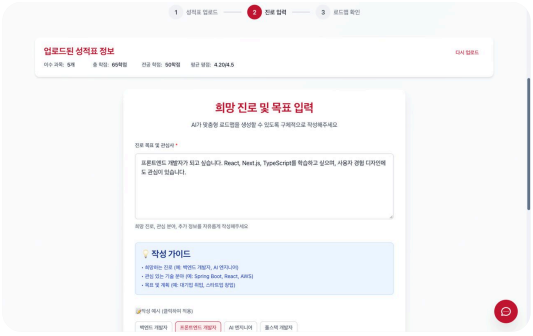
2025.12.23 – 2025.12.24 · React · Next.js

SW-AI 해커톤 장려상

AI 기반 진로-교과목 로드맵 추천 서비스 · 팀 완익세종

GitHub





React · Next.js

배경

제13회 세종대학교 SW-AI 해커톤에서 "**AI 기반 진로-교과목 로드맵 추천 시스템**" 주제를 골랐습니다. 어떤 과목을 듣고 방학에 무엇을 할지는 저희 팀도 하던 고민이었습니다.

완익세종은 성적표 Excel 파일과 희망 진로를 입력하면, Google Gemini가 학기별 추천 과목과 방학별 활동을 담은 로드맵을 만들어 주는 서비스입니다. 팀 완익세종(홍의민·이정연·박성열)으로 2025년 12월 23일 오후부터 24일 아침까지 개발했고, 저는 프론트엔드를 맡았습니다. 백엔드는 별도 저장소의 Spring Boot 서버였습니다.

맡은 역할

- **프론트엔드 전담** — Next.js 저장소의 모든 커밋
- **화면과 UI 컴포넌트** — 성적표 업로드, 역량 분석, 학기별 타임라인, 애니메이션
- **Next.js API Routes** — xlsx 기반 성적표 파싱, Gemini 기반 로드맵 생성
- **AI 챗봇 '세박사'** — 과목·교수 검색, Gemini 실패 시 로컬 검색으로 전환
- **백엔드 연동** — 데이터 구조 불일치 분석 문서, 응답 변환 어댑터

과정

오후에 시작해 저녁 전에 첫 로드맵을 띄웠습니다. 업로드부터 로드맵 생성까지 이어지는 첫 흐름을 2시간이 채 안 돼 완성했습니다.

밤 10시, 백엔드 문서와 프론트엔드 타입이 다르다는 것을 발견했습니다. 그대로 붙이면 학점과 과목 코드가 화면에서 빠지는 상황이었습니다. 바로 고치기 전에 필드 차이를 분석한 문서부터 썼습니다.

새벽 3시에는 원인을 알 수 없는 400 에러를 만났습니다. 백엔드는 상태 코드 외에 아무 정보도 주지 않았습니다. 추적해 보니 실패 시 로컬 모드로 폴백하는 경로가 여전히 백엔드 주소로 요청을 보내고 있었습니다.

고치고 나니 이번에는 로드맵이 비어 있었습니다. 응답 필드명이 명세와 달라 어댑터가 모든 과목을 걸러내고 있었습니다. 시간이 없어 두 형식을 모두 받는 임시 처리를 넣었습니다.

기술적 결정

Mock·Local·Backend 3가지 운영 모드

상황 백엔드가 별도 저장소에서 동시에 개발되고 있었습니다.

결정 UI용 Mock, Gemini를 직접 부르는 Local, Spring Boot를 쓰는 Backend 모드를 두고 환경 변수로 전환했습니다.

결과 백엔드가 준비되기 전에도 화면과 AI 흐름을 먼저 완성할 수 있었습니다.

Backend 모드에서는 폴백하지 않기

상황 백엔드 실패 시 로컬로 넘어가는 폴백이 백엔드 주소를 그대로 써서, 진짜 원인이 400 에러 뒤에 가려졌습니다.

대안 폴백 주소를 로컬로 바로잡거나, 폴백 경로에도 어댑터를 적용하는 방법을 검토했습니다.

결정 Backend 모드에서는 폴백을 없애고 백엔드 오류를 그대로 반환하게 했습니다.

백엔드 응답은 어댑터에서 프론트엔드 타입으로 변환

상황 백엔드 명세와 프론트엔드 구조가 달랐고, 실제 응답은 명세와도 달랐습니다.

결정 응답을 프론트엔드 타입으로 바꾸는 어댑터를 두고, 명세와 다른 형식은 받되 경고 로그를 남겼습니다.

결과 백엔드 수정을 기다리지 않고 로드맵을 표시했고, 호환 처리는 임시 조치로 문서에 남겼습니다.

결과

- 제13회 세종대학교 SW-AI 해커톤 장려상
- 성적표 업로드 → 역량 분석 → 학기별 로드맵 → Markdown 저장 → 챗봇 상담까지 이어지는 흐름을 해커톤 기간 안에 구현했습니다.
- 백엔드-프론트엔드 API 불일치, 400 에러 원인, 응답 형식 불일치 분석을 문서로 남겼습니다.

배운 점

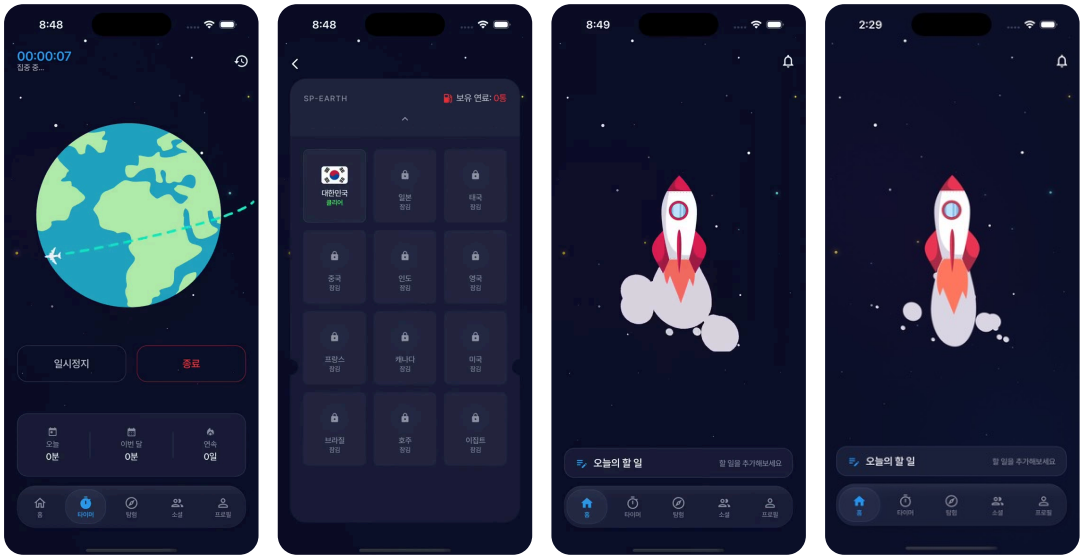
- **명세 문서만 믿고 연동하면 안 됩니다.** 문서와 코드, 실제 응답이 세 갈래로 달랐습니다.
- **폴백은 실패를 숨길 수 있습니다.** 안전장치로 넣은 폴백이 원래의 실패를 가리고 원인 모를 400 에러만 남겼습니다.

우주공부선

2025.12 – 2026 · Flutter

스터디 목표 설정·학습 기록·공유 플랫폼

GitHub



Flutter

배경

할 일 목록과 타이머는 기록은 남기지만, 쌓인 기록이 눈에 보이는 보상으로 이어지지는 않습니다.
우주공부선(Space Study Ship) 은 공부를 게임처럼 만든 학습 관리 앱입니다. 공부한 시간을 연료로 바꾸고, 그 연료로 행성과 지역을 해금하며 우주를 탐험합니다.

2026년 1학기 창의학기제 과제로 진행한 1인 과제였습니다. 서버의 초기 멀티모듈 구조와 배포 설정은 다른 기여자가 잡아 두었습니다.

맡은 역할

- **Flutter 앱 전체** — 할 일, 타이머, 연료, 탐험, 배지, 게스트 모드와 로그인
- **서버 도메인 기능** — Spring Boot·PostgreSQL로 할 일, 연료, 타이머 세션, 탐험
- **서버 인증 기능** — Google·Apple 로그인 토큰 검증, 기기별 세션, 회원 탈퇴
- **기획과 보고** — 주차별 개발 계획, 주간 보고서와 최종 발표

과정

로그인 없이 도는 앱부터 만들었습니다. 1~5주차에 타이머, 할 일, 연료·탐험·배지를 기기 안에서 동작하게 만들고, 7~12주차에 서버와 로그인을 붙였습니다. 덕분에 게스트 모드와 로그인 후 서버 저장을 함께 지원하게 됐습니다.

서버 연결 단계에서 시간을 가장 많이 쓴 건 정합성이었습니다. 기록 저장과 연료 충전 중 하나만 성공하면 공부 기록과 보상이 어긋났습니다. 그래서 기록 저장, 연료 충전, 학습시간 반영을 하나의 트랜잭션으로 묶어 중간에 실패하면 모두 취소되게 했습니다.

같은 오류를 세 번 만났습니다. 로그아웃과 게스트 전환에서 인증 코드와 저장소가 서로를 참조해 Riverpod 순환 의존성 오류가 났습니다. 처음 두 번은 문제가 난 곳만 고쳤고, 세 번째에 인증 코드가 데이터 계층을 바로 호출하도록 전체를 정리했습니다.

기술적 결정

공부를 우주 탐험 게임으로

상황 기존 Todo·타이머 앱은 기록에 그쳐, 꾸준히 쓸 동기와 성취감이 약했습니다.

결정 타이머 30분마다 연료 1을 주고 남은 분은 이월하며, 연료로 행성과 지역을 해금하고 학습 목표를 채우면 배지를 줍니다.

저장 요청마다 식별키를 붙여 중복 지급 방지

상황 모바일 앱은 통신이 끊기면 같은 공부 기록을 다시 보내, 연료가 두 번 지급될 수 있습니다.

결정 저장 요청에 `Idempotency-Key`를 붙여 이미 처리한 키는 기존 결과만 돌려주고, 동시 요청은 유니크 제약 위반을 잡아 다시 조회했습니다.

기기별 세션 관리를 위해 Refresh Token을 DB로 이동

상황 기존 서버는 Refresh Token을 Redis에 회원당 하나만 저장해, 한 기기의 세션만 유효했습니다.

결정 토큰을 기기 테이블에 저장하고 토큰 안에 기기 ID를 넣어, 로그아웃·재발급 때 기기를 구분했습니다.

결과 배포 직후 기존 사용자는 한 번 강제 로그아웃되는데, 설계 때 의도한 결과입니다.

결과

- 학기 말까지 앱과 서버가 연동된 베타 버전을 완성했고, 최종 발표 때 Google Play 비공개 테스트 버전을 게시했습니다.
- 탐험 콘텐츠는 행성 8개·세부 지역 30개, 배지는 6개 범주 25개입니다.
- 서버 API는 인증 6, 할 일·카테고리 8, 타이머 3, 연료 2, 탐험 4개로 모두 23개입니다.
- 학기 중 보고서 기준으로 프론트엔드·백엔드 자동 테스트 240개 이상이 통과했습니다.
- 소셜 화면은 샘플 데이터로 구성돼 있으며, 소셜과 배지의 서버 연동은 남은 과제입니다.

배운 점

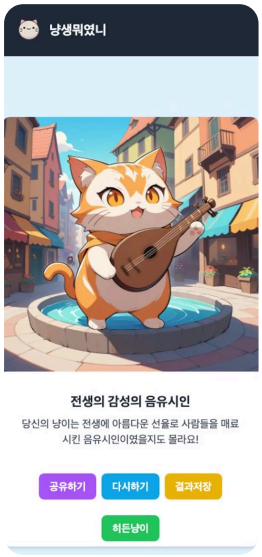
- **재요청이나 계정 전환 같은 상황은 처음부터 설계에 넣어야 합니다.** 연료 중복 지급 위험과 순환 의존성 오류를 겪고 나서 알게 된 점입니다.
- **기대효과는 사용자 데이터로 확인하기 전까지 가설로 둬야 합니다.** 공부를 더 오래 하게 되는지는 아직 확인하지 못했고, 재방문율과 연속 학습일로 확인하려 합니다.

냥생뭐였니?

2024.11 – 2024.12 · Next.js

고양이를 테마로 한 웹 콘텐츠

Website **GitHub**



Next.js

배경

MBTI나 "나의 광석" 같은 유형 테스트가 유행하는 걸 보고, 나도 하나 만들어 보자는 생각으로 시작했습니다. 반려묘의 성향을 묻는 질문에 답하면 그 고양이의 전생, 환생, 인간형 모습을 결과로 보여 주는 심리테스트형 웹 콘텐츠입니다. 반려동물을 키우거나 성격 유형 테스트와 결과 공유를 즐기는 사람을 대상으로 잡았습니다. 혼자 기획하고 개발했으며, Next.js(Pages Router), TypeScript, Tailwind CSS로 만들어 Vercel에 배포했습니다.

맡은 역할

- **1인 기획·개발** — PRD 작성, 질문·결과 데이터 설계, 화면 구현, 배포

과정

기획서부터 정리했습니다. 구현 전에 PRD를 커밋해 목표, 타겟 사용자, 화면별 기능, 데이터 구조, 2주 단위 개발 일정을 정리했습니다. 진행하면서 항목 옆에 "-완", "-하는 중"을 달아 체크리스트처럼 썼습니다.

만들면서 기획을 고쳤습니다. 질문은 5~7개에서 10개로, 선택지는 2~4개에서 3개로 정리했습니다. "다음" 버튼은 없애 고르면 바로 넘어가게 했고, 결과를 정하는 방식은 두 번 바꿨습니다(아래 기술적 결정).

MVP 이후로 미뤄 둔 기능도 넣었습니다. PRD의 "추가 개선사항"에 적어 둔 결과 이미지 저장과 광고를 앞당겼습니다. 결과 카드와 로고를 Canvas에 그려 PNG로 저장하게 했고, Google AdSense 스크립트를

추가했습니다.

기술적 결정

질문 분기 대신 유형 매칭으로 결과 판정

상황 PRD에서는 선택지마다 다음 질문이나 결과를 지정하는 분기 구조로 설계했고, 첫 구현도 그렇게 만들었습니다.

결정 모두가 같은 질문 10개를 거치고, 고른 유형 값을 모아 결과마다 둔 유형 목록과 비교하게 바꿨습니다.

"전부 일치"에서 "6개 이상 일치"로

상황 처음에는 답변 10개가 한 결과의 유형과 전부 맞아야 했고, 아니면 "냥생 미스터리!" 결과로 넘어갔습니다.

결정 6개 이상 겹치는 결과를 후보로 삼고, 많이 겹치는 순, 같으면 id가 작은 순으로 첫 번째를 보여 주게 바꿨습니다.

결과

- https://nyangseng.vercel.app 에 배포했습니다.
- 질문 10개, 결과 27종과 "냥생 미스터리!"를 갖췄습니다.
- 결과는 Web Share API로 공유하거나 PNG로 저장할 수 있고, "히든냥이" 페이지에서 모든 결과를 모아 볼 수 있습니다.
- 2024년 12월 말 Google AdSense 스크립트를 추가했습니다.

플래닝

2024.11 – 2025.01 · Python · Oracle Cloud

로스트아크 정보를 제공하는 Discord 봇

GitHub

Python · Oracle Cloud

배경

로스트아크 레이드 일정을 정하고 보스 공략을 찾아보는 일을 디스코드 안에서 할 수 있게 만든 봇 "플래닝"입니다. 날짜 투표, 보스 관문별 공략 이미지, Gemini AI 질의응답, 매주 공지사항 링크 전송 기능을 넣었습니다. Python과 discord.py로 혼자 만들었고, Oracle Cloud 서버에서 구동했습니다.

맡은 역할

- **1인 개발** — 기능 구현, 공략 데이터 관리, Oracle Cloud 서버 구동

과정

큰 파일 하나로 시작했습니다. 첫 커밋에는 1,400줄이 넘는 `main.py` 하나에 일정 투표, 보스 공략, Gemini 챗봇이 모두 들어 있었습니다. 같은 날 공략 코드의 변수 이름을 phase에서 gate(관문)로 통일했습니다.

쓰면서 기능을 붙였습니다. 12월에는 매주 수요일 오전 10시 5분 공지사항 링크 자동 전송과 보스 난이도 이모지 선택을 더했습니다. 봇 이름 "플래닝"과 README, 이용약관·개인정보처리방침 링크도 붙였고, 쓰지 않게 된 로스트아크 API 설정은 지웠습니다.

게임 업데이트를 따라갔습니다. 2025년 1월 신규 보스 모르둠을 추가하면서, 같은 날 보스 공략 코드 구조를 정리했습니다.

마지막에는 파일을 역할별로 나눴습니다. `main.py` 하나(약 1,100줄)에 명령어, 기능 클래스, 투표 화면이 섞여 있어 찾고 고치기 어려웠습니다. 기능마다 파일 하나(`cogs/`), 투표 화면(`ui/`), 날짜 처리(`utils/`), 봇 생성과 스케줄러(`bot/`)로 나누자 `main.py`는 실행만 하는 35줄이 됐습니다. README에 구조와 이유도 정리했습니다.

기술적 결정

보스별 함수 대신 데이터 파일 하나로

상황 보스 13종마다 비슷한 명령 함수가 반복돼, 모르둠을 넣으려면 같은 함수를 하나 더 써야 했습니다.

결정 보스 정보와 관문별 이미지 경로를 `boss_data.json`으로 옮기고, 보스별 명령은 공통 처리 함수를 부르게 바꿨습니다.

결과 `main.py`가 1,478줄에서 1,091줄로 줄었고, 이후 파일을 역할별로 나누면서 35줄이 됐습니다.

난이도는 다시 입력하는 대신 이모지 반응으로

상황 보스마다 노말과 하드 공략이 따로 있습니다. 그런데 난이도 없이 `/보스 발탄`만 입력하면 "`/보스 발탄 노말`처럼 다시 입력하라"는 안내만 보여 줬습니다.

결정 안내 메시지에 🇳·🇭 반응을 달아, 명령어를 입력한 사람이 10초 안에 누르면 그 난이도의 공략을 바로 보여 주게 했습니다.

결과 난이도를 함께 입력하는 기존 방식도 그대로 쓸 수 있습니다.

결과

- 군단장·에픽·어비스·카제로스 레이드 보스 13종의 관문별 공략을 제공합니다.
- 2025년 1월 신규 보스 모르둠까지 공략 데이터를 갱신했습니다.
- 1,478줄이던 `main.py`를 35줄 진입점으로 줄이고, 기능을 `cogs/`·`ui/`·`utils/`·`bot/`으로 나눴습니다.

은샘침례교회

2026.06 – 2026.09 · Next.js · TypeScript · Spring Boot · PostgreSQL · Docker · Oracle Cloud

교회 홈페이지와 API 서버 — 실제 운영 중

Website **GitHub**





Next.js · TypeScript · Spring Boot · PostgreSQL · Docker · Oracle Cloud

배경

충청남도 예산군 은샘교회의 홈페이지입니다. 교인은 설교·주보·소식을 보고, 성경 통독 챌린지와 차량 탑승 신청을 할 수 있습니다. 관리자는 콘텐츠와 회원, 역할·권한을 화면에서 직접 관리합니다.

처음부터 **다른 교회에도 다시 쓸 수 있는 템플릿**으로 설계했고, 새 교회는 `.env`만 채워 배포하는 "코드 수정 0"을 원칙으로 삼았습니다. 2026년 6월부터 백엔드와 프론트엔드를 혼자 만들었습니다.

맡은 역할

- **백엔드 설계·구현** — Spring Boot 4·Java 21·PostgreSQL·Redis, JWT 인증과 RBAC
- **프론트엔드 구현** — Next.js 16 App Router·TypeScript, 공개·회원·관리자 화면
- **디자인 시스템** — 디자인 토큰과 공통 컴포넌트, 고령 교인을 위한 큰 글자·UI
- **배포·운영** — 백엔드 OCI VM, 프론트엔드 Vercel, GitHub Actions CI/CD
- **문서화** — 설계 명세, 새 교회 배포 가이드, AI 에이전트용 `AGENTS.md`

과정

기능을 이슈 단위로 쪼개고, 설계 문서부터 썼습니다. 백엔드는 기반 4개와 도메인 13개로 나눴습니다. 기능마다 설계, 구현 계획, 구현, 코드리뷰 반영 순서로 PR을 만들어 6월 4일부터 7일까지 올렸습니다.

연혁 페이지는 설계만 네 번 했습니다. 풀스크린 히어로를 넣었다가 "모든 시대를 똑같이 다룬다"는 기준으로 걷어 냈습니다. 마지막에는 참고 사이트를 직접 스크롤하며 관찰해 풀폭 그리드로 바꾸고, 스크롤 엔진을 지워 코드를 줄였습니다.

실제 서버에 올리자 로컬에서는 안 보이던 문제가 나왔습니다. 백엔드가 멈추면 홈 전체가 에러 페이지가 되어, 소식 영역만 실패하도록 격리했습니다. 상세 페이지가 전부 500을 낸 날은 jsdom이 이전 Node 런타임에서 돌지 않는 것이 원인이라 Node 22로 고정했습니다.

기본 기능을 갖춘 뒤에도 기능을 이어서 붙였습니다. 성경 66권 1,189장 구조로 읽음 기록을 계산하는 통독 챌린지, 위치 좌표를 첨부하는 차량 탑승 신청, PWA, 검색엔진 최적화를 넣었습니다.

기술적 결정

멀티테넌시 대신 교회마다 별도 인스턴스

상황 여러 교회에 다시 쓸 수 있는 백엔드가 목표였습니다.

결정 테넌트 컬럼 없이 교회마다 DB와 배포 인스턴스를 따로 두어, 데이터 격리를 인프라에 맡겼습니다.

결과 새 교회 배포는 코드 복사, `.env` 작성, 배포 세 단계이고, 이 절차를 문서로 남겼습니다.

직분과 권한을 분리한 RBAC

상황 교회의 목사·장로·교사 같은 직분이 곧 사이트 권한은 아닙니다.

결정 직분과 권한을 별개 축으로 두고, 인가는 항상 `SERMON_WRITE` 같은 권한 단위로 검사합니다.

문제 처음에는 관리자가 다른 관리자를 임명하고 강등할 수 있었습니다.

결과 역할 부여·회수는 자기보다 엄격히 낮은 역할만 가능하게 바꿨습니다.

등록·수정 폼은 모달 대신 전용 페이지로

상황 공지·설교는 전용 페이지에서, 일정·갤러리·주보는 모달에서 등록·수정했습니다.

문제 모바일에서 태그 선택창이 모달 뒤에 깔리는 등 모달 문제가 반복됐습니다.

결정 모든 도메인을 전용 페이지 폼으로 통일하고, 삭제 확인 같은 작은 대화상자만 모달로 남겼습니다.

대안 모바일에서만 모달을 전체 화면으로 키우는 안은 비용은 가장 적지만, 같은 계열의 문제가 남아 기각했습니다.

결과

- `www.eunsaem.com` 도메인으로 배포했습니다.
- 설교·공지·일정·주보·갤러리, 성경 통독 챌린지, 차량 탑승 신청, 관리자 기능, PWA 설치를 지원합니다.
- 사이트맵·RSS·구조화 데이터를 갖추고 네이버 사이트 소유 확인을 적용했습니다.
- 부서 사진을 WebP로 변환·리사이즈해 46MB에서 5.5MB로 줄였습니다.
- 백엔드는 PR마다 실제 PostgreSQL로 전체 테스트를 돌리고, `deploy` 브랜치에 올리면 자동으로 배포됩니다.

배운 점

- **증상을 하나씩 막기보다 문제가 생기는 구조를 없애는 편이 낫습니다.** 폼을 페이지로 옮기자 같은 계열의 모바일 문제가 함께 사라졌습니다.
- **배포가 서버에 손으로 남긴 상태에 기대면, 그 상태가 사라지는 날 깨집니다.** 서버의 레지스트리 토큰이 지워져 배포가 실패했고, 이후 배포마다 일회용 토큰을 쓰게 바꿨습니다.

Second Brain Template

2026.07 – 2026 · AI 코딩 에이전트용 프로젝트 템플릿

회의·문서·이슈·결정을 지식으로 쌓고, 코드와의 충돌 감지·리포트·반복 이슈 탐지까지 한 흐름으로 묶는 프로젝트별 세컨드 브레인 템플릿

GitHub

Claude Code · Codex · Gemini · Obsidian · 의존성 0

배경

회의가 끝나면 노션이나 메모에 정리해 두었지만, 며칠만 지나도 헷갈렸습니다. 몇 주에서 몇 달이 지나면 "**그때 어떻게 하자고 했었지?**"를 다시 묻게 됐습니다. 코드에도 "**무엇을 하는지**"는 남지만 "**왜 그렇게 정했는지**"는 남지 않아, AI 에이전트도 코드만 보고는 결정의 이유를 답하지 못합니다.

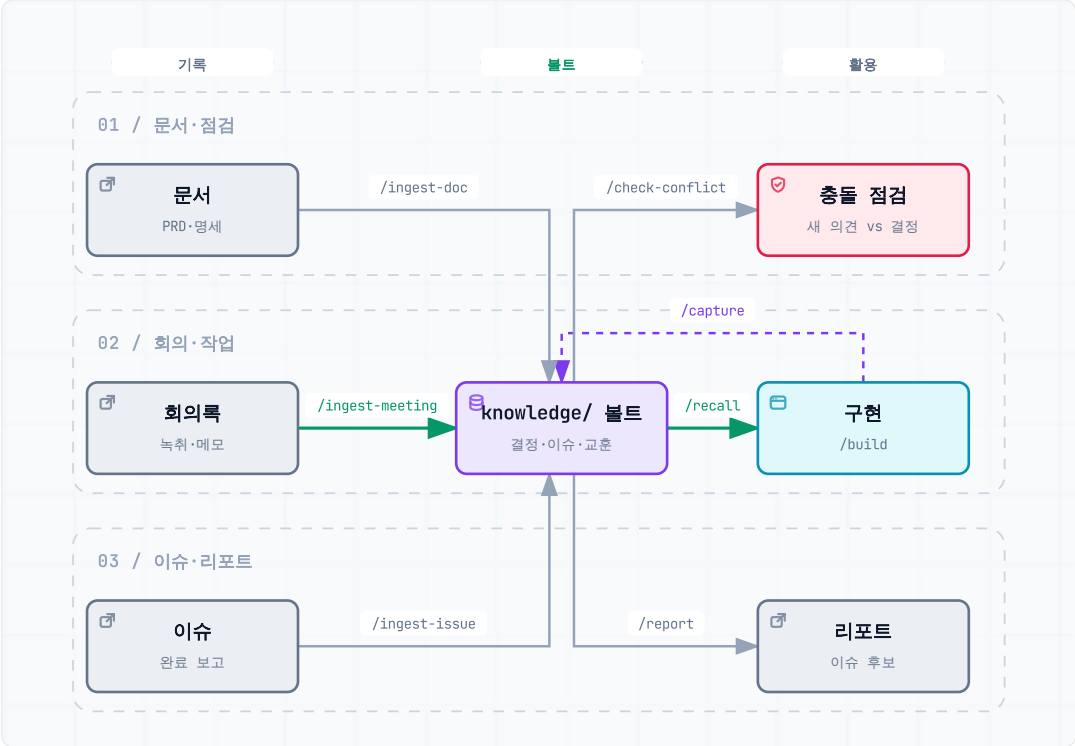
Second Brain Template은 회의·문서·이슈·결정을 프로젝트 안의 마크다운 볼트에 쌓고, 에이전트가 작업 전에 먼저 찾아보게 하는 템플릿입니다. 2026년 7월부터 혼자 만들고 있고, 세 프로젝트에 적용해 쓰고 있습니다.

만든 것

- **설치 한 줄** — `npx github:EM-H20/second-brain-template`. 다시 실행하면 노트는 두고 규칙·스킬만 갱신
- **스킬 15개** — 기록하기, 꺼내 쓰기, 점검·정리를 나눠 맡는 워크플로
- **의존성 0** — API 키·임베딩 없이 마크다운과 규칙만 사용. Claude Code·Codex·Gemini 지원

흐름

회의록·문서·이슈를 볼트에 쌓아 두고, 작업할 때 필요한 결정만 꺼내 쓰는 구조입니다.

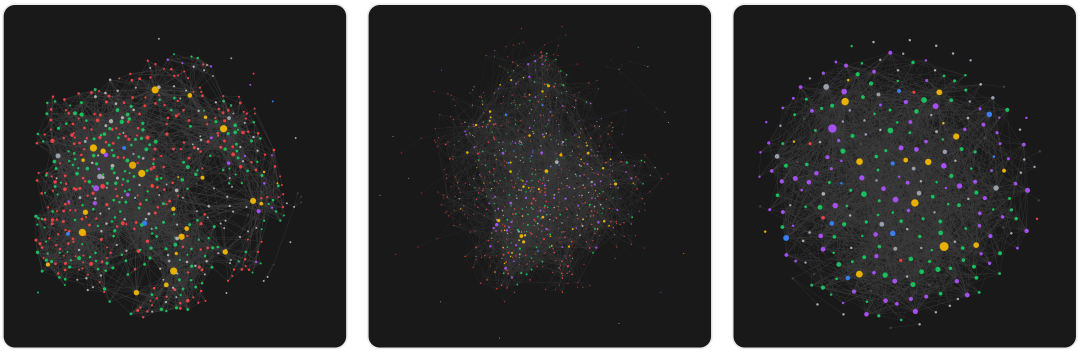


- **기록하기** — `/ingest-meeting`은 회의록에서 결정을 뽑아 정리하고, 기존 결정과 부딪히는지 확인합니다. 문서는 `/ingest-doc`, 이슈와 완료 보고는 `/ingest-issue`로 넣습니다.
- **꺼내 쓰기** — `/recall`이 주제와 관련된 결정·이슈·교훈을 브리프로 모으고, `/build`는 그 브리프를 바탕으로 구현을 시작합니다. 작업 중 생긴 규칙은 `/capture`로 다시 볼트에 넣습니다.
- **점검·정리** — `/check-conflict`는 새 의견을 기존 결정과 대조하고, `/maintain`은 볼트를 다시 묶고 교훈을 모읍니다. `/report`는 볼트를 근거로 보고서를 만듭니다.

명령어를 외우지 않아도 됩니다. 각 작업이 규칙 파일(`SECOND-BRAIN.md`)에 의도 단위로 정의돼 있어, "이 회의록 볼트에 넣어 줘"처럼 말해도 같은 흐름이 실행됩니다. Claude Code와 Gemini에서는 세션이 시작될 때 훅이 볼트의 주제 목록을 에이전트에게 자동으로 넣어 줍니다.

쌓인 볼트

적용한 프로젝트 세 곳의 볼트를 Obsidian 그래프 뷰로 열어 본 모습입니다. 점은 노트 하나, 선은 노트 사이의 링크입니다.



벤치마크

볼트가 실제로 답을 정확하게 만드는지 확인하려고, 2026년 9월 경찰과도둑 프로젝트로 1차 벤치마크를 만들었습니다.

세 조건을 비교했습니다. 모두 같은 코드 저장소를 갖고, 볼트와 도구만 다릅니다.

- **A** — 볼트 + 스킬 + 규칙 파일
- **B** — 볼트만
- **C** — 코드만

문항은 31개입니다. 결정 이유, 폐기된 결정 함정, 기록이 없어 "모름"이 정답인 문제, 코드 위치, 과거 이슈 재발, 볼트와 코드가 어긋난 경우, 작은 구현 과제로 구성했습니다. 문항마다 들어가야 할 정답 사실과 하면 안 되는 답을 미리 적었습니다.

채점은 두 가지로 했습니다. 인용한 파일 경로가 실제로 있는지는 스크립트로 확인하고, 정답을 맞혔는지는 다른 모델(Sonnet)이 판정했습니다. 조건마다 31회씩 모두 93회 실행했습니다.

결과

- **정확도는 볼트가 있으면 0.97, 코드만 있으면 0.71이었습니다.**
- **"왜 이렇게 정했나" 질문은 1.0 대 0.0이었습니다.** 코드만 있는 조건은 지어내지 않고 "기록이 없다"고 답했습니다.
- **없는 경로를 인용한 비율은 8.3%에서 0.8%로 줄었습니다.**
- **코드 위치 문항은 세 조건이 모두 1.0이었습니다.** 볼트가 필요 없는 대조군이 동률이라, 위의 차이는 볼트에서 나왔다고 볼 수 있습니다.

예상과 다른 결과도 있었습니다. 자연어 질문 31개에서 스킬은 한 번도 자동으로 실행되지 않았습니다. 에이전트가 볼트를 찾아본 것은 규칙 파일을 따랐기 때문이었고, 토큰은 볼트만 있는 B가 가장 적게 썼습니다.

한계

- 문항당 1회만 실행해, 한두 문항 차이는 우연일 수 있습니다.
- 정답표를 사람이 다시 검수하지 않았고, 구현 과제에는 컴파일 검사가 없습니다.
- 다음에는 3회 반복 실행과, 스킬을 직접 호출하는 조건을 추가할 계획입니다.

디스코드 브레인 봇



동심지키미-브레인

- 상황 경찰과도둑 팀에 볼트를 도입했습니다. 팀원 7명 중에는 개발자가 아닌 사람도 있었습니다.
- 문제 비개발자 팀원은 볼트를 쓰기 어려웠습니다. 볼트는 Claude Code와 Obsidian으로 여는 저장소라, 개발 도구를 쓰지 않으면 열어 볼 방법이 없었습니다. 행사 현장처럼 밖에 있을 때는 개발자도 바로 확인할 수 없었습니다.

- 결정 2026년 10월 17일, 팀 디스코드에서 말로 물으면 볼트를 찾아 답하는 봇을 만들었습니다. 예전에 게임 길드 디스코드에서 만들어 운영한 봇 플래닝의 경험을 참고했고, 특히 플래닝이 매주 공지사항 링크를 자동으로 보내던 기능이 아래 아침 소식의 바탕이 됐습니다. 답에는 근거 노트 id(예: DEC-0092)를 붙이고, 찾지 못하면 찾지 못했다고 말합니다. "사업자등록증 줘"처럼 말하면 원본 파일을 첨부합니다.
- 구조 discord.js 봇이 질문마다 Claude(Sonnet) 도구 루프를 한 번 돌려, 서버에 받아 둔 볼트를 검색하고 노트 본문을 읽은 뒤 답합니다. 도구 호출은 질문당 8회로 묶었고, 볼트는 5분마다 GitHub에서 받아 옵니다. 테스트 146개가 있습니다.
- **읽기만 하는 봇**: "지식화 남겨"라고 하면 볼트를 직접 고치지 않고 GitHub 이슈로 접수만 합니다. 반영은 담당자가 Claude Code에서 해서, 원본 보존·무결성 검사 같은 볼트 규칙을 사람이 확인한 경로로만 지킵니다.
- **아침 소식**: 월·수·금 아침마다 창업 지원사업·공모전·행사 공고를 웹에서 모아 카드 한 장으로 올립니다. 이번 실행에서 실제로 열어 본 링크와, 공고에서 읽은 마감일만 코드가 다시 걸러 통과시킵니다.
- 결과 만든 직후 시험 사용에서 질문 17건을 받아 11건에 답하고 4건을 이슈로 접수했고, 평균 응답은 9초였습니다. 월 $100 API 크레딧 안에서 운영합니다. 실제 팀 사용 지표는 아직 쌓이는 중입니다.

접수 이슈를 디스코드 없이 읽히게

문제 처음에는 스레드의 사용자 메시지만 이어 붙여 이슈를 만들어서, 봇의 답이 빠졌습니다. 담당자는 디스코드를 열어야 무엇을 남기라는 건지 알 수 있었습니다. 이슈 생성이 실패하면 요청이 어디에도 남지 않았는데, 봇은 저장했다고 안내했습니다.

해결 실패하면 원문을 서버에 남기고 담당자에게 알리게 했습니다. 이슈에는 디스코드 원문 대신, 봇이 대화 맥락을 정리한 "지식화할 내용"을 꼭 넣게 했습니다. 담당자가 Claude Code의 `/capture`에 바로 넣는 형태입니다.

결과 바뀐 형식으로 처음 들어온 접수는 배경·관련 노트·확인할 점이 정리된 채 왔고, 그대로 볼트 노트(DOC-0083)가 됐습니다.

아침 소식은 시험 실행으로 프롬프트 조정

문제 마감일은 직접 열어 본 공고 페이지에서 읽은 것만 카드에 실리는데, 시험 실행에서 봇은 페이지 열기 한도 10번 중 4번만 쓰고 끝냈습니다. 또 연구성과·보유기술이 있어야 하는 공고, 특정 학교·기관 소속만 지원할 수 있는 공고처럼 우리 팀이 지원할 수 있는지 알 수 없는 공고가 "확인 필요"라는 표시를 달고 올라왔습니다.

해결 남은 열기 한도는 맞아 보이는 공고의 상세 페이지를 여는 데 끝까지 쓰게 했습니다. 사이트 첫 화면이나 목록 페이지는 열지 않게 했습니다. "확인 필요"라고 적어야 할 공고는 아예 빼도록 프롬프트를 고쳤습니다.

결과 다시 돌리니 열람이 8번으로 늘었고, "확인 필요" 공고는 더 올라오지 않았습니다.

배운 점

- **기대와 측정은 다릅니다.** 볼트의 효과는 숫자로 확인됐지만, 스킬이 알아서 실행된다는 전제는 틀렸습니다.
- **대조군이 있어야 차이를 믿을 수 있습니다.** 코드 위치 문항이 동률이었기 때문에 다른 차이를 볼트의 효과로 읽을 수 있었습니다.

에브리페스타

2026.08 – 2026 · Next.js · TanStack Query

대학 축제를 축제·아티스트·학교별로 찾아보는 웹 서비스 — 프론트엔드 2명, 백엔드 2명 팀

Website **GitHub**

Next.js · TypeScript · TanStack Query · Playwright

배경

에브리페스타는 대학 축제를 축제·아티스트·학교별로 찾아보는 웹 서비스입니다. 프론트엔드 2명, 백엔드 2명이 함께 만들고 있고, 저는 프론트엔드를 맡았습니다.

맡은 역할

- **프론트엔드** — Next.js 화면 구현, 동료 PR 리뷰
- **지식 볼트 도입** — 회의·결정·이슈를 Second Brain Template으로 쌓아, 팀원과 AI 에이전트가 같은 기록을 보고 일하게 함

일한 방식

> 팀워크 · HOW I WORK ↑
>
> ### Disable 말고 Delete.
>
> 안 되는 버튼은 흐리게 두지 말고 지우자고 했습니다.
>
> 상황 헤더에는 프로필·알림·북마크 버튼이, 푸터에는 SNS 아이콘이 있었습니다. 하지만 로그인도 SNS 계정도 없어서 눌러도 아무 일이 없었습니다. 데모 때마다 "왜 안 눌리지?"라는 말이 나왔습니다.
>
> 한 일 회색으로 남기지 말고 자리째 지우자는 이슈를 올렸습니다. 기능이 생기면 다시 넣을 수 있게, 돌려놓을 자리는 코드에 적어 두었습니다.
>
> 결과 함께 프론트를 맡은 팀원이 리뷰하고 머지했습니다. 이후 그 팀원도 자기 이슈에서 "기능이 없는 자리는 지운다"를 이미 정한 규칙으로 인용했습니다.

배운 점

- **화면에 보이는 것은 전부 동작해야 합니다.** 그래야 데모에서 따로 설명할 예외가 없습니다.

Big-Brother

2026.09 – 2026 · React · Spring Boot

여러 CCTV의 위험 상황을 찾고 VLM이 자연어로 설명하는 관제 시스템 — 4명 캡스톤 프로젝트

GitHub

React · TypeScript · Vite · TanStack Query · zustand · Spring Boot

배경

Big-Brother는 여러 대의 CCTV 영상에서 위험 상황을 찾고, VLM이 그 상황을 말로 설명해 주는 관제 시스템입니다. 세종대학교 캡스톤 프로젝트로 4명이 만들고 있습니다.

역할은 풀스택, 클라우드·문서, VLM, YOLO 탐지로 나눴습니다. 다만 애자일로 움직여서, 손이 빈 사람이 다음 일을 맡습니다. 저는 풀스택을 맡아 프론트엔드와 백엔드를 연결했고, 여러 카메라의 결과를 합치는 융합 벤치도 만들었습니다.

맡은 역할

- **프론트엔드** — React·Vite, TanStack Query, 관제 화면
- **백엔드 연동** — Spring Boot API, AI 결과를 WebSocket으로 받는 흐름
- **다중 시점 융합 벤치** — 카메라 여러 대의 결과를 합치는 설정을 비교·평가하는 도구
- **지식 볼트 도입** — 회의 녹취로 세컨드브레인을 세우고, 과제물 초안을 뽑아 팀이 검토

일한 방식

> 디테일 · HOW I WORK ↑
>
> ### 늦게 오는 Alert는 Alert가 아니니까.
>
> 경고가 늦게 뜨던 창에, 경고 카드를 바로 띄웠습니다.
>
> 상황 — 관제원 창의 사건 목록은 AI 설명이 만들어진 뒤에야 사건을 보여 줬습니다. 메인 화면은 경고 순간 바로 깜빡이는데, 관제원 창만 보는 사람은 첫 경고를 늦게 알았습니다.
>
> 한 일 — 관제원 창 위쪽에 지금 경고 중인 카메라를 실제 화면 카드로 띄웠습니다. 처음엔 경고가 있을 때만 카드 줄이 나타났는데, 그 순간 영상이 아래로 약 200px 밀렸습니다. 그래서 카드 자리를 늘 비워 두고, 경고가 없으면 "경고 중인 카메라 없음"을 보여 주게 했습니다.
>
> 결과 — 카드가 생기거나 사라져도 영상 위치가 그대로입니다. 이 흐름은 요구사항 문서에 넣어 팀 제출본에 담겼습니다.

백엔드 구현은 AI 서브에이전트들이 나눠 맡았습니다.

한 일 — AI 결과를 실시간으로 받는 백엔드 작업을 여섯 단계로 나누고, 단계마다 구현 에이전트와 리뷰 에이전트를 따로 두었습니다.

제 몫 에이전트가 정할 수 없는 제품 판단을 내렸습니다. VLM 담당 팀원의 샘플 데이터를 쓰고, 추정값은 화면에 추정이라고 표시하기로 했습니다.

배운 점

- **생겼다 사라지는 화면 요소는 자리를 미리 잡아 둡니다.** 판단하는 중에 화면이 튀면, 누르려던 버튼이 옮겨가 잘못 누를 수 있습니다.