경력기술서 · 포트폴리오

안승찬 개발자

Email omnipoo@naver.com | **Phone** +82 10-4035-0672 | **GitHub** github.com/Ahnseungc
Blog blog.omnipo58-e94.workers.dev | **LinkedIn** linkedin.com/in/seungchan-ahn

Contents

(주)골드앤컴퍼니 · 금골디 iOS Android 2026.01 – 재직 중

01 **Streaming SSR 전환으로 WebView 진입 속도 개선**
2026.11 · Suspense 스트리밍으로 로그인 첫 바이트를 백엔드 응답에서 분리

02 **리워드 시스템 및 보상형 광고 연동 구조 개선**
2026.08 · 3단 광고 폴백과 SSV 검증 연동

03 **금 상품 주문·결제 프론트엔드 흐름 개선**
2026.06 – 2026.07 · 시세 고정 정책 연동·결제 진입 공통 함수

04 **WebView 성능 최적화 및 서비스 트래픽 대응**
2026.04 – 2026.05 · 번들 예산·state 위치·PM2·CDN

05 **금팔기 판매 퍼널 프론트엔드 재설계 및 전환율 개선**
2026.02 – 2026.03 · 가격을 먼저 보여주는 퍼널로 통합

(주)케어마인더 창업팀 2024.09 – 2025.02 · 정규직 2025.03 – 2026.01

06 **병동 간호 환자 요청 관리 플랫폼 아키텍처 개선 및 프론트엔드 개발 리드**
2025.03 – 2025.08 · WebView 하이브리드 전환·모노레포 통합

일하는 방식 (주)골드앤컴퍼니 · 2026

07 **코딩 에이전트 하네스 설계**
2026 · 규칙·스킬·서브에이전트로 고정한 에이전트 작업 기준

Projects

Streaming SSR 전환으로 WebView 진입 속도 개선

SSR은 유지한 채 스트리밍 전환, 로그인 첫 바이트 77~239ms → 17~19ms

(주)골드앤컴퍼니

2026.11 · 운영 반영

프론트엔드 · 실측·스펙·구현
Flutter 앱 WebView 설정 조사 포함

RENDERING
Next.js App Router, Suspense Streaming SSR

SERVER STATE
TanStack Query v5 (pending dehydrate)

MEASURE
Playwright · CDP 스로틀, Navigation·Resource Timing

APP
Flutter InAppWebView, iOS WebKit 디버그 프록시

문제 발견

푸시로 들어오면 앱이 매번 새 WebView로 문서를 처음부터 받았습니다. 로그인 화면은 서버가 백엔드 API를 모두 기다린 뒤에야 첫 바이트를 보내, 백엔드가 느려지면 흰 화면 시간이 그대로 늘어났습니다(dev 실측 최대 1.25s).

원인 분석

- 망 조건 4단계(Wi-Fi·4G·Slow 4G·3G)로 콜드 진입을 실측해, 로그인 첫 바이트가 서버의 prefetch 대기에 묶여 있음을 확인했습니다.
- 진입 직후 요청 18개를 추적해, 세션 조회 5회는 진입 공지의 재확인 로직이, 현재 화면 재렌더는 하단 내비가 리다이렉트되는 경로를 prefetch하는 것이 원인임을 코드에서 찾았습니다.

대안 검토와 선택

- **CSR로 바꾸지 않고** SSR을 유지한 채, 서버가 기다리는 지점만 옮겨 pending 쿼리를 스트리밍했습니다.
- **서비스 워커는 보류**했습니다. 앱 설정과 시뮬레이터 실측으로 푸시 WebView끼리 JS 디스크 캐시를 이미 공유함(JS 63개 중 53개)을 확인했습니다.
- 첫 화면 히어로에 서버 데이터가 필요한 **금 상점은 기다리는 구조를 유지**했습니다.

해결

- 4개 화면의 서버 prefetch를 비대기로 바꾸고, 401 재로그인 판정을 Suspense 안으로 옮겨 셸이 먼저 보이게 했습니다.
- 로컬에 저장된 자산과 스트리밍 데이터가 어느 순서로 와도 0원으로 깜빡이지 않도록 테스트로 고정했습니다.
- 같은 스크립트로 전후를 비교하는 측정 도구를 만들어 팀이 재사용하게 했습니다.

검증

- 로그인 첫 바이트 중앙값 **77~239ms → 17~19ms**(같은 조건에서 전후 비교). 운영 반영 후 **43~51ms**로 백엔드 응답 시간과 무관해졌습니다.
- /my-safe 진입 요청 **18 → 13개**, 세션 조회 **5 → 2회**, 서버 데이터 재호출·현재 화면 재렌더 0건.
- 데이터 도착 시각은 그대로이고(1.21 → 1.08s) 그 전에 화면이 먼저 그려지게 됐습니다. 남은 병목인 인증 체인(약 1초)은 다음 단계로 정리했습니다.

리워드 시스템 및 보상형 광고 연동 구조 개선

3단 광고 폴백과 SSV 검증 연동으로 보상이 끊기지 않는 리워드 구조

문제 발견

보상형 광고가 AdMob 하나에만 의존해서, no-fill이 나면 유저는 보상을 못 받고 회사는 수익을 놓쳤습니다. 보상을 클라이언트 콜백만으로 확정하면 광고를 끝까지 보지 않아도 보상을 받는 경로가 생길 수 있었습니다.

원인 분석

- 광고 소스가 하나라 no-fill이 곧 보상 실패로 이어졌습니다.
- 기기에서 오는 콜백만으로 보상을 확정하는 구조로는 위조를 막을 수 없었습니다.

대안 검토와 선택

- 콜백을 받는 즉시 지급하는 방식은 단순하지만 위조를 막을 수 없었습니다. 백엔드가 설계한 **서버 검증(SSV) 구조에 맞춰**, 클라이언트는 콜백으로 보상을 확정하지 않고 서버 응답을 기다리도록 바꿨습니다.

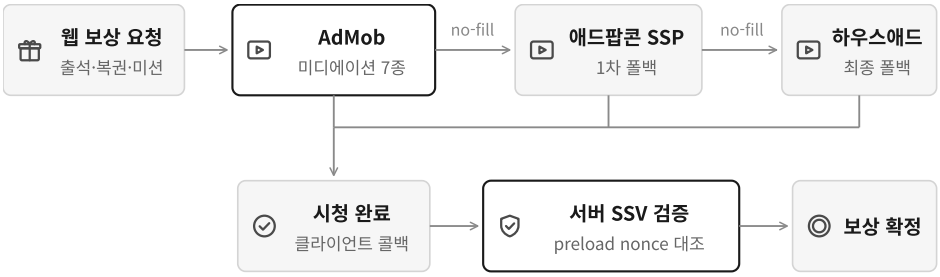
해결

- **장애 모니터링**: no-fill과 광고 만료는 Sentry, Crashlytics 장애 집계에서 빼서 **실제 SDK 장애만 보이게** 했습니다.
- **광고 폴백 체인**: AdMob(미디에이션 7종) → 애드팝콘 SSP → 하우스애드 순으로 넘어가게 해, 광고가 없어서 보상을 못 받는 경험을 없앴습니다.
- **SSV 연동**: 광고 preload 시점에 백엔드가 발급한 nonce를 연동하고, 보상 확정 대기·성공·실패 상태를 화면에서 처리했습니다.

검증

- 보상형 광고 완료 **751만 회**.
- 데일리 미션 완료 월 44만 → 104만 건, 복권 응모 100만 건 이상. 제품·마케팅 전체 성과이며, 저는 웹·앱 클라이언트 구현과 계측을 맡았습니다.

구조



(주)골드앤컴퍼니

2026.08

프론트엔드 · Flutter 앱 개발
백엔드·기획·디자인과 협업

FRAMEWORK
Next.js 15, React 19, TypeScript, Flutter

SERVER STATE
TanStack Query v5

CLIENT STATE
Zustand

ADS
AdMob 미디에이션, 애드팝콘

ANALYTICS
Airbridge, Sentry, Crashlytics

Projects

금 상품 주문·결제 프론트엔드 흐름 개선

시세 고정 연동과 주문 복원, 결제의 36.9%가 리워드로 모은 금 사용

문제 발견

금은 결제하는 동안에도 가격이 바뀝니다. 그리고 앱은 상품 상세, 결제, 입금을 서로 다른 WebView 스택으로 띄워서, 결제를 마치고 돌아오면 이전 화면이 옛 데이터를 보여줄 수 있었습니다.

원인 분석

- 시세를 프론트에서 다시 계산하면 계산 시점마다 금액이 달라집니다.
- WebView마다 JS 메모리와 React Query 캐시가 따로라, 한 화면의 변경이 다른 화면에 전달되지 않습니다.

대안 검토와 선택

- 금액을 프론트에서 다시 계산하지 않고, **백엔드가 주문 시점에 Redis로 고정한 시세**를 그대로 쓰면서 만료 시각을 카운트다운으로 보여줬습니다.
- 캐시를 통째로 비우면 상품 상세 조회가 대기 주문을 새로 만들 수 있어서, **대기 주문과 결제에 관련된 키만** 갱신했습니다.
- 주문 ID를 URL에 싣지 않고 **사용자, 상품, 수량을 복원 기준**으로 삼아, 같은 상품을 다시 열면 이어서 결제할 수 있게 했습니다.

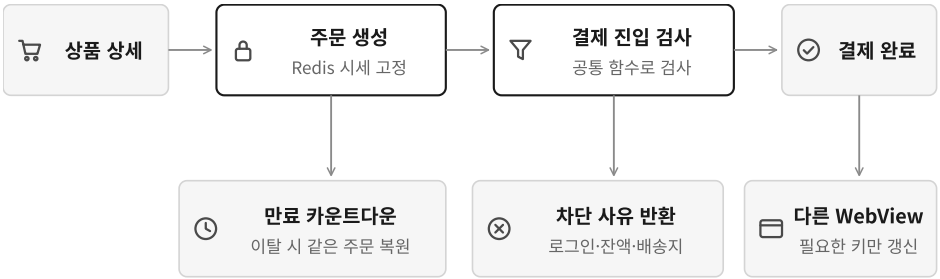
해결

- 화면 곳곳에 흩어지기 쉬운 결제 진입 조건(로그인, 본인인증, 잔액, 배송지)을 “다음 스텝”과 “차단 사유”를 반환하는 **공통 함수**로 모았습니다.

검증

- 진입 조건별 6가지 경우를 테스트로 고정해, 이후 시세 프리징으로 주문 API가 바뀔 때 안전망으로 썼습니다.
- **결제의 36.9%가 리워드로 모은 금을 사용**해 리워드가 구매로 이어지는 구조를 확인했습니다.

구조



(주)골드앤컴퍼니

2026.06 – 2026.07

프론트엔드 · Flutter 앱 개발
백엔드·기획·디자인과 협업

FRAMEWORK
Next.js 15, React 19, TypeScript

SERVER STATE
TanStack Query v5

STATE SYNC
BroadcastChannel

TESTING
Vitest

BACKEND 연동
Redis 시세 프리징 주문

Projects

WebView 성능 최적화 및 서비스 트래픽 대응

번들 예산·state 위치 조정으로 홈 JS -29%, 스크롤 렌더 -93%, 피크 CPU 절반으로

문제 발견

모바일 WebView에서 초기 로딩이 무거웠습니다. 홈 First Load JS는 530kB였고, 타이머가 많은 화면은 입력이 없어도 계속 리렌더됐습니다. 푸시 직후엔 서버 CPU가 70~80%까지 올랐습니다.

원인 분석

- 금모으기 SVG 아이콘 컴포넌트 원본만 약 1.9MB였고, 이게 들어간 청크가 5.5MB까지 커져 있었습니다.
- 렌더 측정 스크립트로 104개 페이지를 실측하니, 화면 루트의 1초 타이머·롤링 state 때문에 화면 전체가 매초 다시 그려졌습니다.

대안 검토와 선택

- 무거워 보이는 곳부터 손대지 않고, **First Load JS 상한(공통 200kB, 화면별 350kB)**을 먼저 정한 뒤 이를 넘는 코드만 나눴습니다.
- memo를 곳곳에 붙이기보다 **state를 실제로 쓰는 하위 컴포넌트로 내리는 것**을 먼저 했습니다.
- 짧은 피크 때문에 서버를 늘리는 건 비용 낭비라고 보고, **같은 사양에서 PM2 클러스터로 멀티코어**를 쓰게 했습니다.

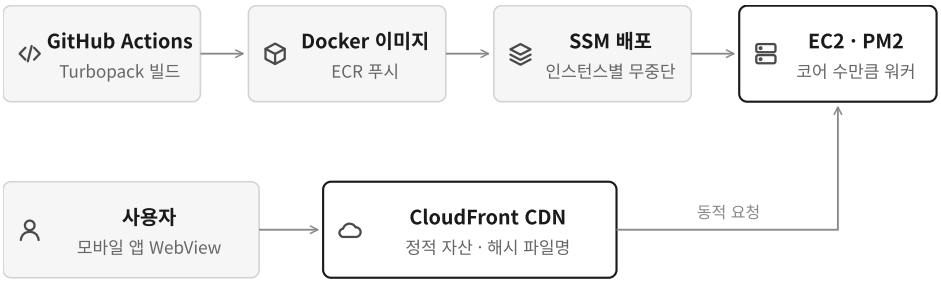
해결

- 차트, 채팅, PC 사이드는 dynamic import로, inline SVG는 정적 파일로, GIF는 팝업이 열릴 때만 로드하는 MP4로 바꿨습니다. 인프라는 CDN·PM2 클러스터·무중단 배포로 정리했습니다(아래 구조).

검증

- 홈 First Load JS **530 → 378kB(-29%)**, 공개 미디어 21MB → 약 200KB, 스크롤 렌더 **1,276 → 83회**.
- k6로 3,000 VU를 검증했고, 피크 CPU는 **70~80% → 30~40%대**, 운영 빌드는 11.0분 → 7.5분이 됐습니다.

구조



(주)골드앤컴퍼니

2026.04 – 2026.05

프론트엔드 · Flutter 앱 개발
백엔드·기획·디자인과 협업

FRAMEWORK
Next.js 15, React 19

INFRA
AWS ECR·SSM, CloudFront, Docker

CI·CD
GitHub Actions, Turbopack

LOAD TEST
k6, PM2

금팔기 판매 퍼널 프론트엔드 재설계 및 전환율 개선

가격을 먼저 보여주도록 퍼널 순서를 뒤집고, 방식별 퍼널 3개를 하나로 통합

문제 발견

개편 전에는 직접 택배, 안심 수거, 금은방 입찰이 진입부터 서로 다른 퍼널 3개였습니다. 사진 업로드, 본인인증, 견적 확인 화면이 방식마다 따로 있었고, 사용자는 가격을 보기도 전에 거래 방식부터 골라야 했습니다.

원인 분석

- 가장 중요한 정보인 **가격이 퍼널 끝**에 있고, 판단 근거가 없는 거래 방식 선택이 맨 앞에 있었습니다. 화면도 세 벌이라 정책 하나를 바꾸려면 세 군데를 고쳐야 했습니다.

대안 검토와 선택

- 퍼널 3개를 각각 고치는 대신 기획·디자인·백엔드와 함께 **하나로 합치고 순서를 뒤집었습니다**. 판매 정보, 예상 견적, 입찰, 업체 선택을 거친 뒤 마지막에 거래 방식을 고릅니다.
- 거래 방식 화면을 **입찰과 업체 아래 경로**로 두어, 푸시나 딥링크로 들어와도 같은 화면이 열리게 했습니다.
- 거래 이후 화면은 방식별로 나누지 않고 **상태 코드 기준**으로 만들었습니다. 모든 조합을 목데이터로 먼저 만들어 카드가 상태값만 보고 그려지게 했습니다.

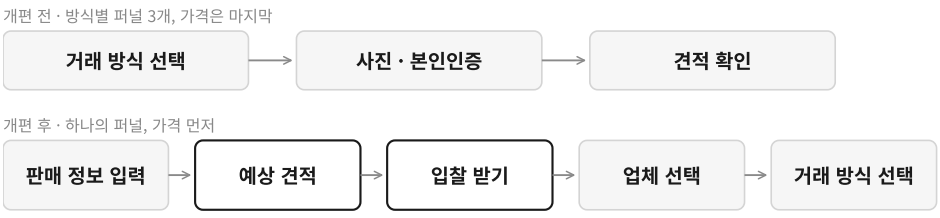
해결

- **이탈 복귀**: 백엔드의 단계별 임시저장(Redis)과 CRM 푸시에 연동해, 함량 선택에서 이탈한 사용자가 견적이 복원된 화면으로 돌아오게 했습니다.
- **측정과 개선**: 판매 시작부터 거래 완료·철회까지 GTM 이벤트를 심고, 이탈이 큰 1→2스텝의 버튼 위치·이탈 방지 CTA·입력 항목을 고쳤습니다.
- **WebView 메모리**: 고해상도 사진 동시 리사이즈가 WebView 메모리를 넘겨 순차 처리로 바꾸고, 신분증 사진도 업로드 전에 줄였습니다.
- **결제·파트너 화면**: 토스페이먼츠 2단계 결제를 붙이고, 파트너 웹 거래 상세·매입 장부까지 만들어 상태 정의를 맞췄습니다.

검증

- 1→2스텝 **이탈률 50% → 40%**, **완료율 12% → 16%**(MMP Airbridge 기준). 입찰 요청의 **34%가 거래로 이어졌고**(취소 3%), 판매액은 수억 원대입니다.

구조



(주)골드앤컴퍼니

2026.02 – 2026.03

프론트엔드 · Flutter 앱 개발
백엔드·기획·디자인과 협업

FRAMEWORK
Next.js 15, React 19, TypeScript

FUNNEL
@use-funnel, URL 하위 경로

STATE
Redis 임시저장, CRM 푸시

PAYMENT
Toss Payments 결제위젯

TESTING
Playwright E2E

ANALYTICS
GTM

Projects

병동 간호 환자 요청 관리 플랫폼 아키텍처 개선 및 프론트엔드 개발 리드

WebView 하이브리드 전환으로 당일 핫픽스 배포

문제 발견

RN 중심 구조에서는 사소한 기능 수정도 앱 재배포와 심사 과정을 거쳐야 해서 24시간 돌아가는 병동에 빠르게 대응하기 어려웠습니다. 서비스별로 코드가 분산돼 공통 기능을 반복 개발했고, 24시간 켜진 병동 태블릿의 네트워크와 앱 상태를 원격에서 파악할 방법도 없었습니다.

원인 분석

- 화면 로직이 앱 바이너리 안에 있어 수정마다 스토어 심사를 거쳐야 했고, 4개의 B2C/B2B 도메인이 각자 같은 UI·로직을 따로 구현하고 있었습니다.

대안 검토와 선택

- WebView는 네트워크에 더 의존하고 네이티브만큼 부드럽지 않습니다. 그래도 병동에서는 **심사 없이 바로 고쳐 배포하는 것**이 더 중요하다고 판단했습니다.

해결

- **WebView + Next.js 전환 주도**: 웹과 앱이 공통 코드를 공유하고 네이티브는 브릿지만 맡게 나눴습니다. CSR이던 WebView는 Next.js ISR+SSR로 옮기고, 프론트 3인의 학습 범위와 컨벤션을 정리했습니다.
- **모노레포 통합**: 4개 도메인을 pnpm 모노레포로 합치고 공통 UI·비즈니스 로직·타입을 패키지로 분리했습니다. Atomic Design 기반 디자인 시스템을 Storybook으로 문서화했습니다.
- **모니터링**: 네트워크 유실, 백그라운드 전환, 앱 종료를 추적하는 태블릿 상태 페이지를 만들고, FCM 원격 기능으로 장애에 바로 대응하게 했습니다.
- **배포**: 변경된 도메인만 빌드·배포하게 하고, S3 APK 배포와 FCM으로 앱 업데이트를 자동화했습니다.

검증

- 핫픽스를 **앱 심사 없이 당일 배포**하게 됐고, **전체 코드량 약 40% 절감**, 초기 로드 **약 30% 향상**, 디자인 시스템 도입 후 팀 UI 작업 속도 **50% 향상**을 확인했습니다.
- 모니터링으로 **현장 연락 전에 원격에서 먼저 장애를 인지하고 대응**하게 됐습니다.

그 밖의 프로젝트

- **강남 베드로병원 도입 V1.1.0** · 2025.01 – 2025.02 · 창업팀: 웹 알림을 Electron 앱으로 전환(OS 알림·트레이 상주·자동 업데이트), 태블릿 시니어·수면 모드
- **구미 강남병원 도입 V1.0.0** · 2024.09 – 2024.12 · 창업팀: 첫 도입 버전. SSE 실시간 요청, STOMP 채팅, 로그인 공통화, Jenkins CI/CD

(주)케어마인더

2025.03 – 2025.08

프론트엔드 3인 리드
백엔드 2인, 디자이너, 기획과 협업
창업팀 2024.09 – 2025.02

병동 간호 인력의 환자 요청을 관리하는 병원용 플랫폼

WebView · B2B·B2C

FRAMEWORK
Next.js, React Native

SERVER STATE
React Query

CLIENT STATE
Zustand

VALIDATION
Zod

STYLING
Tailwind CSS, Storybook

TESTING
Jest, Cypress

INFRA / CI·CD
Jenkins, AWS S3, FCM

Projects

코딩 에이전트 하네스 설계

팀이 반복해서 잡던 리뷰 지적을 규칙으로 옮겨 에이전트 결과물의 편차를 줄임

문제 발견

에이전트가 임의 색상과 클래스를 만들어 쓰거나, 화면마다 웹뷰 대응(키보드, safe area)이 빠져 결과물 편차가 컸습니다.

해결

- **리뷰를 규칙으로**: 팀이 반복해서 잡던 리뷰 지적을 에이전트 규칙으로 옮겼습니다.
- **규칙과 스킬**: 웹뷰 노하우, 디자인 토큰 검증, 예외 처리 기준을 규칙과 스킬로 정리해 팀 레포에서 운영했습니다.
- **역할별 서브에이전트**: UI 구현과 QA를 나눈 파이프라인을 만들고, Figma 값 대조 → 구현 → 시각 QA → 테스트 → 커밋을 에이전트의 작업 완료 기준으로 고정했습니다.

일하는 방식

스펙과 측정 계획을 먼저 문서로 쓰고, 에이전트로 구현한 뒤 테스트와 시각 QA로 검증합니다.

(주)골드앤컴퍼니

2026

직접 설계 및 운영
팀 레포 공용 에이전트 하네스

TOOLS
Cursor, Claude Code

HARNESS
규칙 · 스킬 · 역할별 서브에이전트

PIPELINE
Figma 대조 → 구현 → 시각 QA → 테스트 → 커밋