PORTFOLIO

김지민

Frontend Developer

실무에서 마주친 문제를 배경 → 문제 → 해결 → 결과 순서로 정리한 사례와,
직접 진행한 개인&팀 프로젝트입니다.

PROJECTS

꿀잠닥터 Frontend

2025.08 ~ 재직중

슬립포레스트 · 수면 헬스케어 B2C 앱 신규 개발·고도화 · Web(Next.js) / Hybrid App(Flutter) / Admin(React)

App Store Google Play

01 꿀잠닥터 신규 개발 참여 — 디자인 시스템·테스트 체계 구축

문제

- 신규 프로젝트를 기획 단계부터 새로 만드는 과정이라, 반복되는 Figma 퍼블리싱 작업과 컴포넌트 재사용 기준·검증 체계 없이는 개발 속도 저하와 UI 불일치가 누적될 위험이 컸음

해결

- 기획 단계부터 참여해 Figma 디자인을 AI로 퍼블리싱 자동화하고, 재사용성을 기준으로 컴포넌트를 설계해 디자인 시스템을 구축, Storybook으로 전체 컴포넌트를 문서화하고 Vitest 기반 유닛 테스트를 붙여 검증 체계를 함께 마련

결과

- 디자인 변경 시 반복되는 퍼블리싱 공수를 줄이고, Storybook을 디자이너·개발자 간 공통 참조점으로 활용해 협업 효율을 높였으며, 유닛 테스트로 컴포넌트 회귀 안정성을 확보

02 RN → Flutter 마이그레이션 의사결정 참여 및 기술 검증

문제

- 조명 연동 등 Native 기능이 늘어나며 RN 앱이 무거워지고 성능·메모리 관리 이슈와 3rd party 의존성 리스크가 커져, 신뢰도 있는 전환 방향에 대한 결정이 필요했음

해결

- 제품팀 회의에서 RN 유지 대비 Flutter/Native 전환의 이점(경량화, 3rd party 의존성 제거)과 리스크(코드 푸시 불가, 로그인·퍼블리싱 등 핵심 기능 전면 재개발)를 비교해 'Flutter로 우선 검증하고 학습 곡선이 지나치게 가파르면 Native로 전환'하는 조건부 전략 수립에 참여하고, iOS 구현을 담당해 로그인·알람·백그라운드 재생·센서·블루투스·걸음수 등 핵심 기능별로 라이브러리 기술 검증을 진행

결과

- 재생 라이브러리의 유지보수 중단과 handleLifecycle 제약(백그라운드 재생·연속 재생 동시 지원 불가)을 검증 단계에서 발견해 대체 라이브러리 전환 및 '저녁 루틴은 오디오 전용 제공'으로 스코프를 조정했고, Android Health Connect의 raw 데이터 제약을 확인해 걸음수 로그는 Native 코드 연동이 필요하다는 결론을 실제 구현 전에 도출

기술 검증 항목

SNS 로그인 | 알람 (iOS) | 백그라운드 재생 | 센서 데이터 | 블루투스 연동 | 걸음수 (HealthKit)

03 Sentry 기반 프로덕션 에러 모니터링 체계 도입

문제

- 로컬에서 재현되지 않는 프로덕션 오류는 발생 경로와 원인 파악이 어려워, 사용자 리포트에 의존해 사후 대응하는 데 그침

해결

- React Native·Next.js 프로젝트에 Sentry를 연동해 프로젝트별 대시보드로 Crash Free Sessions·Apdex·오류 발생 추이를 파악하고, 개별 이슈는 Breadcrumbs(발생 과정)·Tags(Device/OS/Browser/URL)·HTTP 요청(Cookie/Header) 정보로 발생 경로를 재구성할 수 있도록 구축

결과

- 재현 없이도 오류 발생 시점의 정확한 경로와 컨텍스트를 파악할 수 있는 모니터링 기반을 마련

오류 추적 체계

Breadcrumbs (발생 과정) → Tags & Context (Device/OS/Browser/URL) →
HTTP Request (Cookie/Header)

04 AI Agent 개발 workflow 구조화

문제

- AI Agent가 프로젝트 컨벤션과 검증 절차 없이 작업해 결과물의 일관성과 추적성이 낮음

해결

- Cursor Rules/Skills와 AGENTS.md에 프로젝트 컨벤션, Jira 티켓 구현, E2E 시나리오, MR 생성, 작업 종료 DoD를 문서화하고, PRD → Plan → 구현 → E2E → Closeout 전 과정을 표준화

결과

- Cursor 기반 AI 워크플로우를 성공적으로 구축·정착시켜 MR 처리 속도(주간)가 1.2건 → 9~15건대로 오르며 생산성이 크게 증대되고, 스킬을 단계적으로 설계해 붙인 뒤로는 코드 품질도 함께 좋아지며 커밋 티켓 추적률 9% → 78%·AI 커밋 완결률 0% → 88%로 누락 사례가 뚜렷이 감소

05 Playwright 기반 주요 사용자 플로우 E2E 체계 구축

문제

- AI로 생성한 E2E 스펙만으로는 실제 사용자 플로우·API 스펙과의 정합성을 보장할 수 없음

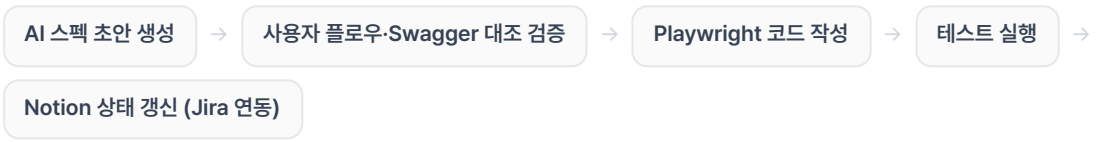
해결

- 온보딩·홈·리포트·MY·HFF·루틴·NPS 등 주요 플로우를 19개 spec·약 303개 케이스로 구성하고, 실패 케이스를 사용자 플로우·Swagger 스펙과 대조 검증, Notion에 시나리오 상태·수정 이력을 Jira와 연동해 관리

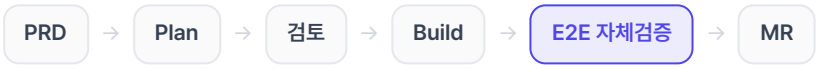
결과

- 남은 오류는 백엔드와 협업해 해결하고, 로컬 및 Mobile Chrome/Safari 환경에서 반복 실행 가능한 회귀 검증 체계로 운영

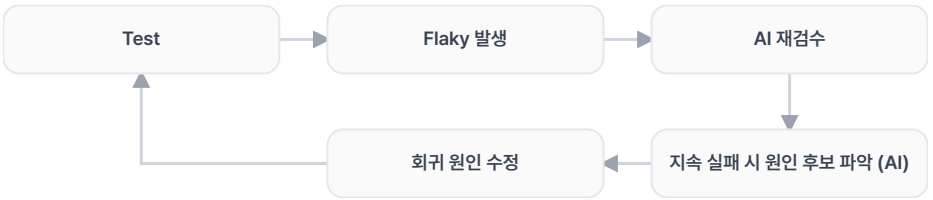
E2E 시나리오 구현



개발 워크플로우



회귀 피드백 루프



06 GA4 기반 사용자 행동 수집과 운영 지표 연결

문제

- WebView SPA 특성상 GA4 자동 pageview만으로는 실제 라우팅 이동을 정확히 추적하기 어려움

해결

- 자동 pageview를 끄고 route 변경 기반 page_view와 sign_up·banner_click·share·error_log 등 이벤트를 직접 설계해 수집하고, error_log는 front·report·api·rn 발생 소스별로 dimension 타입을 구조화(예: front는 error/unauthorized/middleware/not-found, api는 commFetch 단위)해 소스를 특정할 수 있도록 설계

결과

- Admin에서 GA Data API의 stream/event/custom dimension 필터로 DAU·이벤트·오류 로그 등 운영 리포트로 연결해 시각화. ECharts 표출을 위한 응답 DTO를 직접 설계해 백엔드와 공유하고 API 설계 논의에 참여

ROUTE 변경 추적

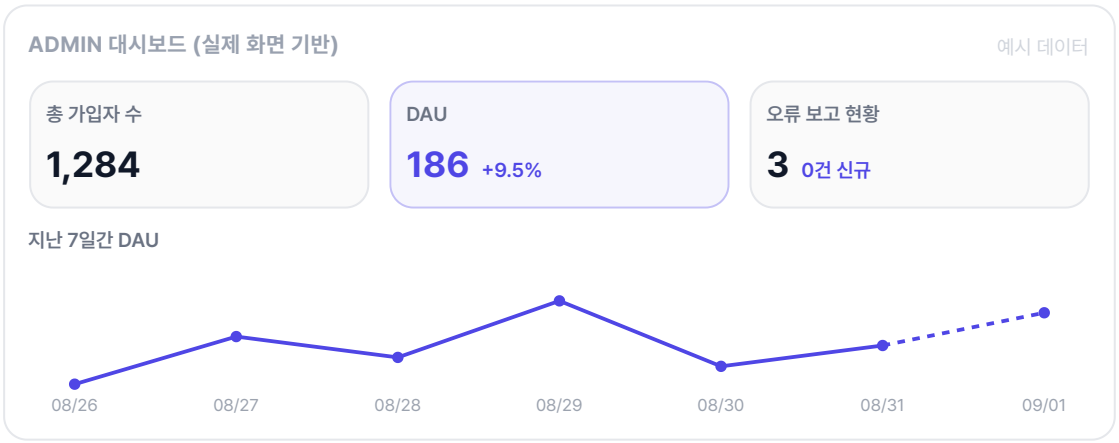
pathname·searchParams 감지 → pageview(url, prevPage) → gtag('page_view')

커스텀 이벤트

sign_up / banner_click / share / error_log → gtag('event', action, params)

ADMIN 조회

runReport 조회 → stream 필터 → DAU·오류 대시보드



07 Admin 재구축 및 API·서버 상태 관리 구조 표준화

문제

- 레거시 Admin은 기존 임시로 구축되어 있던 구조로 문서화가 없어 유지보수·재사용이 어려웠고, 도메인마다 fetch·인증·에러 처리와 query key 관리 방식도 제각각이라 확장이 힘들었음. 별도 디자인 리소스도 없어 화면 UI까지 직접 구성해야 하는 상황이었음

해결

- 입사 후 Admin 재구축을 맡아 폴더 구조와 API 통신 규약을 새로 정의하고, commFetch + TanStack Query + query-key-factory 구조를 단계적으로 도입해 10개 Query 도메인에 공통 적용, access token 갱신 시 refreshPromise를 공유. 디자이너 리소스 없이 AI를 활용해 화면 UI 디자인까지 직접 적용

결과

- 병렬 요청의 중복 refresh를 방지해 인증 처리를 안정화하고, 문서화되지 않았던 구조를 표준 패턴으로 재정립

08 Jira 스프린트 연동 대시보드로 팀 보고 체계 자동화

문제

- 전체 회의마다 프로젝트 진행 상황을 매번 별도로 정리해 보고해야 해서, 반복되는 보고 준비가 팀의 회의 부담으로 누적됨

해결

- Admin 재구축 과정에서 GA4 이벤트 대시보드와 함께 Jira Sprint API를 연동해, 개발팀 스프린트 티켓 진행 현황을 Admin에서 바로 조회할 수 있도록 구성

결과

- 전체 회의 시 별도 보고 자료 없이 대시보드 화면을 함께 보며 논의할 수 있는 체계를 만들어, 반복되는 보고 준비 부담을 줄임

9월 1주차 SPRINT (실제 화면 기반) 예시 데이터

완료	진행 중	백로그	전체 진행률
3	1	4	62%

타입	작업 내용	진행 상태	담당자
작업	카카오 SDK 업데이트	완료	김지민
작업	Flutter 오류 셀 동기화	진행 중	김지민
작업	오류 보고 유형 분류	완료	김지민
버그	모달·스낵바 오류 UX	리뷰	김지민
작업	전면 오류 네트워크 점검	완료	김지민

09 iOS 통합 오디오 세션 관리

문제

- RN 대비 오디오 재생 방식을 전환하는 과정에서 WebView 내 유튜브 등 웹 콘텐츠 오디오와 세션이 충돌해 제대로 정리되지 않는 문제 발생

해결

- 재생·녹음 상황별 우선순위를 정리해 오디오 세션을 전환·공유하는 로직을 설계하고, 웹 콘텐츠 재생 시작·종료 시 세션을 명시적으로 비활성화·재활성화하도록 처리

결과

- TestFlight 내부 테스터 리포트를 바탕으로 재현·수정해 오디오 충돌 없이 안정적으로 동작 (현재 내부 테스트 단계, 정식 배포 전)

10 iOS·Android 걸음 수 동기화 정합성 개선

문제

- iOS HealthKit 구간 조회 시 15분 격자 밖 마지막 구간이 버킷에서 누락되고, HealthKit이 권한 허용 여부를 앱에 공개하지 않아 상태 판별이 어려움

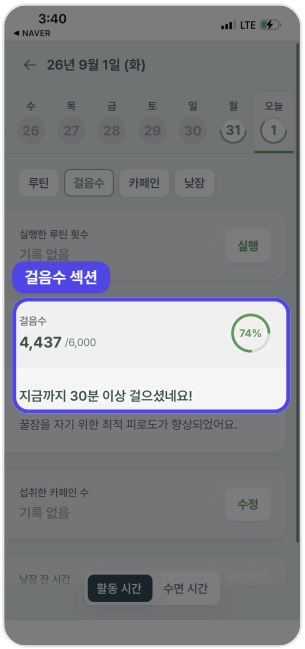
해결

- Android의 버킷 라벨링 로직을 iOS(Flutter)에도 동일하게 구현해 partial 구간 누락을 방지하고, 최근 7일 걸음 수 유무로 권한 상태를 추론. 표시(기기 총합)와 서버 저장(버킷 raw)의 책임을 분리해 초기 도입했던 diff 보정 로직을 제거

결과

- iOS·Android 걸음 수가 버킷 누락 없이 동기화되고, 기상 알람 시 최대 13일 백필까지 안정적으로 처리되는 동기화 체계로 운영

실제 UI (걸음수 강조)



버킷 로그 정합성 (예시 데이터)

버킷 (15분 단위)	걸음수
06:00–06:15	480
07:30–07:45	512
09:00–09:15	605
09:15–09:30	588
12:00–12:15	460
12:15–12:30	398
15:00–15:15	610
15:15–15:30	548
15:30–15:38 (8분, 격자 밖 partial)	**236**

BEFORE (partial 누락)

4,201 버킷 합계

UI 4,437와 불일치 (diff 236)

AFTER (partial 포함)

4,437 버킷 합계

UI 4,437와 일치 ✓

11 WebView ↔ Native navigation race condition 해결

문제

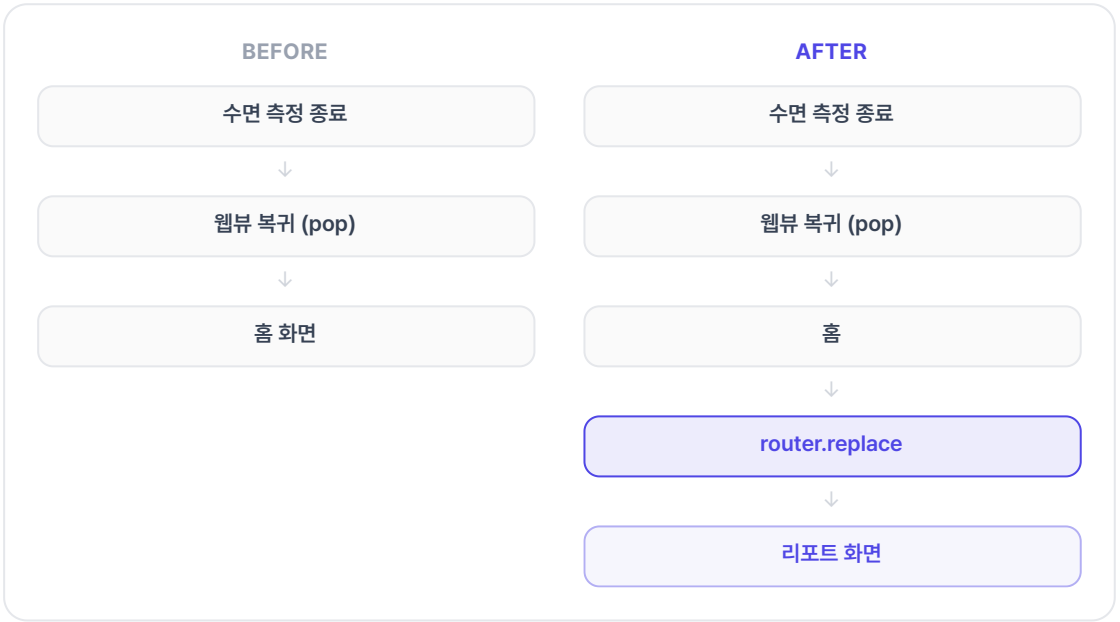
- 수면 측정 종료 후 Native의 postMessage가 React listener 등록 전에 도착해 화면 이동이 유실

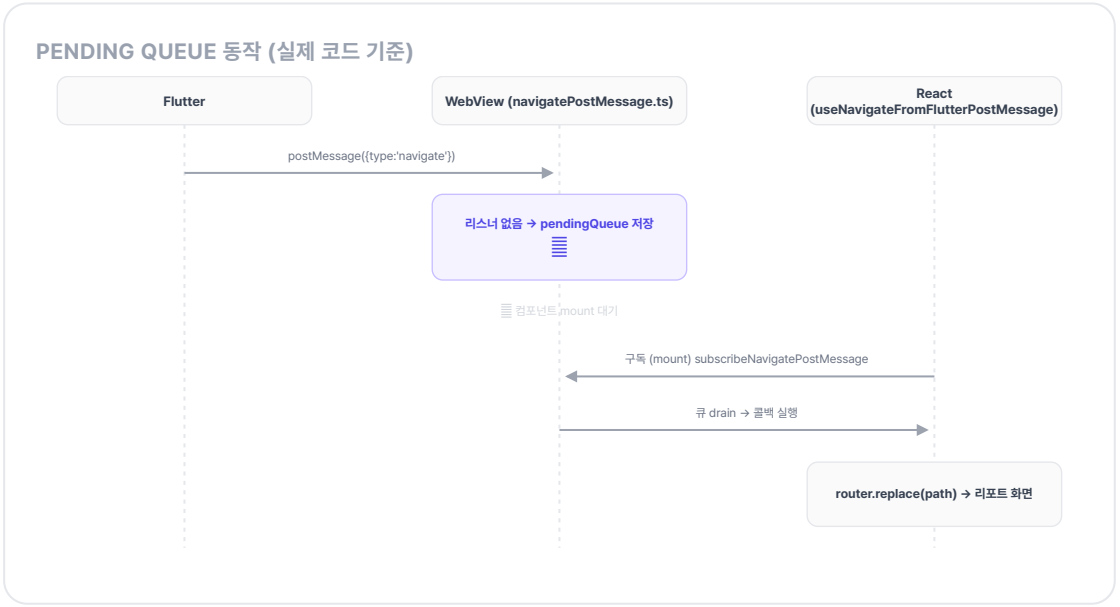
해결

- pending event queue와 early listener를 적용하고 Next.js App Router의 router.replace와 연결

결과

- full reload 없이 SPA navigation을 구현해 화면 이동 유실 없이 안정적으로 동작



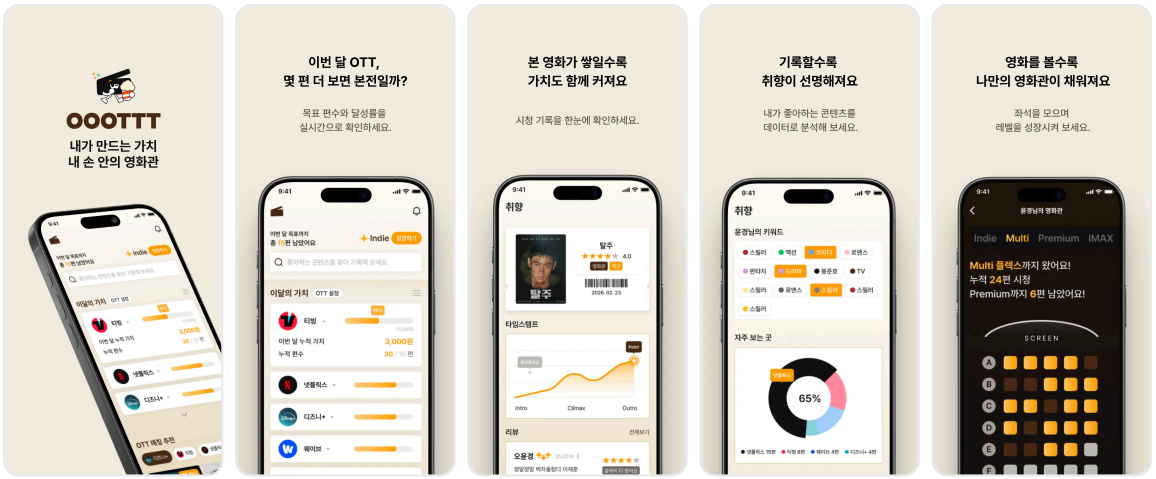


OOOTTT Frontend / Flutter

2026.03 ~ 2026.08

5인 팀 · 취향 기반 OTT 추천 및 구독 요금제 관리 플랫폼 · App Store / Google Play 배포

- iOS/Android 빌드 관리와 App Store·Google Play 심사 대응을 직접 진행해 실 사용자가 다운로드할 수 있는 앱으로 배포
- 라이브러리로 표현하기 어려운 인터랙션을 위해 CustomPainter 기반 커스텀 차트·애니메이션을 직접 설계
- 검색 데이터는 클라이언트 조회, 시청 기록·통계는 서버 누적으로 분리하는 데이터 책임 범위를 팀과 협의해 설계



GitHub | App Store | Google Play

MOMO Frontend

2024.12 ~ 2025.02

5인 팀 · 밥친구 매칭 서비스

- React·TypeScript 기반으로 UI와 핵심 기능을 개발하고, TanStack Query로 서버 상태를 관리
- STOMP 기반 실시간 양방향 채팅을 구현하고, Recoil로 클라이언트 상태를, Tailwind CSS로 UI 스타일링을 구성

GitHub | Demo

OlaOla Frontend

2024.11 ~ 2024.12 (약 1개월)

개인 프로젝트 · 클라이밍 커뮤니티 사진·영상 공유 플랫폼 · 백엔드 서버 미사용

- 별도 백엔드 없이 브라우저 IndexedDB를 저장소로 활용해 미디어 업로드·게시글 CRUD를 클라이언트에서 직접 설계
- 무한 스크롤·페이지네이션·이미지 캐러셀을 라이브러리 없이 순수 JavaScript로 직접 구현, Firebase Authentication으로 로그인 연동

기술 블로그

velog.io · 2024.12.18

[개인 프로젝트 - OlaOla] IndexedDB로 브라우저 환경에서 로컬 데이터베이스 구축하기

백엔드 구현보다 프론트엔드 역량 강화에 집중하기 위해, Firebase 같은 서버 환경 대신 IndexedDB로 브라우저에서 직접 로컬 데이터베이스를 구축한 과정을 정리했습니다.

GitHub Live 블로그