김봄 | 주니어 개발자

Channel

Email | Portfolio | GitHub | LinkedIn | Blog

Summary

React·Next.js·TypeScript를 기반으로 **사용자 경험과 데이터 흐름을 함께 설계**하고, 운영 피드백과 **성능·행동 데이터로 개선을 반복**하는 **프론트엔드 개발자**입니다.

- 약 280명이 사용하는 교내 서비스의 관리자 웹을 구축·유지보수하며, 데이터 변경 주기에 맞춘 캐시 기준을 재설계해 최신 정보는 유지하면서, 동일 탐색 시나리오의 API 호출 횟수를 45회 → 7회, 84% 줄였습니다.
- 이미지 로딩 병목을 개선해 모바일 Lighthouse 성능을 86점 → 99점으로 높이고, LCP를 4.1초 → 1.0초로 단축했습니다.
- Amplitude로 누적 방문자 653명의 행동 데이터를 분석해 주요 이탈 구간을 식별하고 콘텐츠 구조를 재설계하여, Contact 도달률을 39.6% → 48.3%로 개선했습니다.

Main Projects

Washer_Client v2

2025.10 - 2026.04 · 유지보수 중

Next.js, React, TypeScript, Tailwind CSS, TanStack Query, Zustand, Zod, FSD

배포 링크 →

GitHub →

기수 간 인수인계로 운영되는 광주소프트웨어마이스터고 세탁기 관리 시스템의 v2 클라이언트입니다.

200명 이상의 전교생을 대상으로 운영되는 서비스로, 사용자가 세탁기 상태를 확인하고 예약 및 신고 흐름을 사용할 수 있도록 구성했습니다. 실제 운영 환경에서 유지보수까지 이어가고 있습니다.

FrontEnd: 2명, Android: 2명, BackEnd: 2명, Design: 1명

핵심 성과

- 운영 대상이 약 210명에서 280명으로 33% 확대되는 시점에, 페**이지 이동·재진입마다 같은 데이터를 반복 요청**하는 문제 발생. 데이터 변경 주기에 맞춰 캐시 기준을 재설계해 동일 탐색 시나리오의 **API 요청 45회에서 7회로 84%** 감소
- API 호출·필터·모달·상태 관리·UI가 집중된 **1,603줄의 단일 관리자 페이지를 5개 화면·5개 도메인 구조로 분리**해 기능별 책임과 변경 범위를 명확화
- 5개 도메인의 7개 주요 API 응답·70개 DTO 필드에 런타임 검증을 적용하고, nullable 불일치 **5건 수정** 및 조회·변경 API 12개에서 발생하는 오류를 **9개 유형으로 정규화**

주요 기여

관리자 운영 정보 통합 및 조회 흐름 구축

[상황]
- 운영자가 기기 고장 여부와 사용 가능 상태를 확인하려면 세탁실에 직접 방문해야 했음
- 예약·고장 신고·사용자 패널티 정보가 기능별로 분산되어 문제 원인과 관련 정보를 한 흐름에서 파악하기 어려웠음
- 신입생 유입으로 서비스 운영 대상이 약 **210명에서 280명 규모**로 확대되면서 효율적인 관리자 운영 체계가 필요

[수행]
- 대시보드·기기·예약·사용자·고장 신고를 관리하는 **5개의 관리자 화면** 구축
- 기기 상태와 예약 이력·고장 신고 정보를 `machineId` 기준으로 연결하여 기기 단위의 운영 정보를 함께 조회하도록 구성
- 층·기기 종류·처리 상태를 기준으로 검색 및 필터 기능 구현
- 예약 취소, 기기 상태 변경, 고장 신고 처리, 사용자 패널티 해제 등 주요 관리자 액션을 화면별로 연결
- Discord를 통해 운영 관리자와 상시 피드백을 주고받으며 실제 운영 과정에서 발견된 불편 사항을 지속적으로 개선

[결과]
- 현장 방문 없이 관리자 웹에서 기기 상태와 예약·고장 신고 정보를 원격으로 확인할 수 있는 운영 흐름 구축
- 분산된 운영 정보를 기기와 층 기준으로 연결해 문제 확인부터 후속 조치까지의 탐색 흐름을 단순화
- 약 **280명 규모의 운영 대상**에 대한 기기·예약·신고·패널티 관리를 하나의 관리자 서비스에서 처리할 수 있는 환경 구축
- Discord 기반 운영자 피드백을 유지보수 과정에 반영하는 반복 개선 구조 마련

API 응답 검증 및 데이터 경계 재설계

[상황]
- TypeScript 제네릭으로 서버 응답 타입을 지정했지만 실제 응답은 런타임에서 검증되지 않음
- nullable 필드와 선언 타입이 일치하지 않았고, 정의되지 않은 상태값이 정상적인 업무 상태로 처리될 가능성이 존재

[수행]
- API 응답을 unknown으로 수신한 뒤 Zod Schema를 적용해 5개 도메인의 7개 주요 API 응답·70개 DTO 필드를 검증
- Schema에서 TypeScript 타입을 추론해 타입 선언과 검증 규칙을 단일화
- 검증된 DTO만 Mapper와 UI에 전달하고, API 계약 위반과 관리자의 판단이 필요한 정상 업무 상태를 구분
- 모든 응답을 일괄 변환하지 않고, 화면 요구사항에 따라 Raw Response 유지와 Mapper 적용 기준을 분리

[결과]
- 필드 누락·타입 불일치·정의되지 않은 상태값을 API 진입 단계에서 감지하고, nullable 불일치 5건 수정
- 검증되지 않은 외부 데이터가 Mapper와 UI로 전달되는 흐름 차단
- 서버 응답 구조와 화면 표현의 변경 범위를 Schema와 Mapper 내부로 제한

서버 상태 캐시 및 요청 구조 최적화

[상황]
- 페이지 이동과 재진입 과정에서 동일 데이터를 반복 요청해 관리자 화면의 API 호출이 과도하게 발생
- 변경 빈도가 다른 기기·예약·사용자 데이터를 동일한 캐시 기준으로 처리

[수행]
- 페이지의 직접 API 호출을 제거하고 `HTTP Client → 도메인 API 함수 → TanStack Query Hook`으로 데이터 흐름 분리
- 도메인별 Query Key를 중앙화하고 데이터 변경 빈도에 따라 `staleTime`을 구분
- 예약 삭제·기기 상태 변경·패널티 해제 이후 관련 도메인 캐시만 명시적으로 무효화

● 고장 신고와 기기 데이터를 `machineId` 기준으로 조합해 층별 고장 현황을 조회하도록 구성

[결과]
● 고정된 관리자 화면 탐색 시나리오 기준 API 요청 **45회 → 7회, 84% 감소**
● 실시간성이 높은 기기·예약 상태의 최신성을 유지하면서 반복 요청 최소화
● 관리자가 층별 고장 현황과 기기별 예약 이력을 하나의 화면 흐름에서 확인할 수 있도록 개선

관리자 화면 구조 및 컴포넌트 경계 재설계

[상황]
● V1 관리자 페이지가 1,603줄까지 증가하며 API 호출·데이터 변환·모달 상태·사용자 액션·UI 렌더링이 한 파일에 집중
● 유사한 화면을 일괄 공통화할 경우 도메인별 API와 업무 흐름까지 범용 컴포넌트에 결합될 위험이 존재

[수행]
● 세탁기·건조기처럼 데이터만 다르고 업무 흐름이 같은 화면은 동일 컴포넌트로 재사용
● 예약 이력·기기 관리처럼 외형은 비슷하지만 API와 사용자 행동이 다른 기능은 **별도 도메인 컴포넌트로 유지**
● 공통 패널 외형은 `StatusPanelShell`, 공통 액션 배치는 `StatusRowActions`로 추출

[결과]
● 1,603줄의 단일 관리자 페이지를 5개 화면·5개 도메인 구조로 분리
● 화면 형태가 아닌 **변경 이유와 업무 흐름을 기준으로 컴포넌트 경계 설정**
● 반복 UI의 수정 지점을 공통 컴포넌트로 모으면서 도메인별 기능 변경의 전파 범위 축소

인증 오류 및 세션 복구 흐름 개선

[상황]
● 내 정보 조회 실패 원인을 구분하지 않고 모든 오류를 로그인 만료로 처리
● 서버·네트워크·응답 검증 오류에서도 토큰과 Query Cache가 삭제되어 강제 로그아웃될 가능성이 존재

[수행]
● 인증·권한·서버·네트워크·응답 검증 오류를 **공통 오류 모델로 분류**
● 최초 `401` 응답은 토큰 재발급으로 복구하고, 재발급까지 실패한 경우에만 세션 정리
● 복구 가능한 오류에서는 토큰과 Query Cache를 유지하고 오류 유형별 안내와 재시도 흐름 제공
● 세션 정리 책임을 `AdminLayout`에서 공통 인증 로직으로 분리

[결과]
● 12개 API 흐름에서 발생하는 오류를 인증·권한·서버·네트워크·응답 검증 등 9개 유형으로 정규화
● 인증과 무관한 오류로 인한 불필요한 강제 로그아웃 방지
● 실제 인증 만료에서만 토큰과 캐시를 정리하도록 복구 기준 명확화
● API 계약 오류와 인증 오류를 분리해 원인 추적과 사용자 안내 개선

Web Portfolio

2025.08 - 2026.06 · 유지보수 중

Next.js · React · TypeScript · GSAP · ScrollTrigger · Lenis · SCSS · Amplitude

배포 링크 →

GitHub →

스크롤 인터랙션을 활용해 개발자로서의 관점과 프로젝트 경험을 단계적으로 전달하는 개인 웹 포트폴리오입니다.

단순히 프로젝트를 나열하는 데 그치지 않고, **사용자의 시선과 정보 이해 순서를 고려한 스토리형 구조**를 직접 기획·디자인·개발했습니다. 배포 이후에는 사용자 행동 데이터와 성능 지표를 기반으로 콘텐츠 흐름과 웹 성능을 지속적으로 개선하고 있습니다.

Design,Development : 1명

핵심 성과

- 되감기·상태 유지·pin 전환처럼 서로 다른 ScrollTrigger 동작을 **Controller 계층으로 통합**해 애니메이션 생성·초기화·정리 절차 표준화
- 초기 화면에 필요하지 않은 이미지까지 동시에 요청되던 병목을 추적해 **이미지 로딩 우선순위 재설계**, 모바일 **Lighthouse 86점 → 99점·LCP 4.1초 → 1.0초 개선**
- Amplitude에서 **섹션 도달·외부 링크 클릭 이벤트와 40% 노출 기준을 직접 설계**하고, 방문자 653명의 행동 데이터를 기반으로 콘텐츠를 개편해 Contact 도달률 39.6% → 48.3% 개선

주요 기여

서로 다른 스크롤 동작을 통합한 실행 아키텍처 재설계

[상황]

- 각 Stage에는 스크롤 방향에 따라 되감기는 장면, 한 번 노출된 상태를 유지하는 장면, 특정 구간을 화면에 고정해 콘텐츠를 전환하는 장면 등 **서로 다른 방식의 인터랙션이 존재**
- 여러 Stage와 Scene의 DOM 탐색, Animation 생성, ScrollTrigger 연결과 cleanup이 각 Hook에 혼재해 파일 간 **데이터 전달과 제어 흐름을 파악**하기 어려웠음

[수행]

- 전체 스크롤 실행을 조율하는 Stage Hook, Stage 내 Animation의 생성·초기화·정리를 관리하는 Controller 집합, 개별 Scene의 Timeline을 제어하는 Animation Controller로 **관리 계층 설계**
- Stage는 각 장면의 큰 DOM 구간만 수집하고, Scene에서는 내부의 세부 요소만 **탐색하도록 범위를 분리**한 뒤, Stage Controller가 수집된 요소와 Animation을 연결하도록 구성
- Stage Hook이 `DOM 수집 → Controller 생성·초기화 → ScrollTrigger 연결 → cleanup` 순서로 장면별 실행을 조율하도록 구성
- 스크롤과 함께 역재생되는 장면, 한 번 노출된 뒤 유지되는 장면, `pin`·상태 전환·progress 분할이 필요한 특수 장면으로 구분해 **ScrollTrigger 등록 기준**을 설계

[결과]

- 여러 파일이 연결된 구조에서도 DOM 수집부터 Timeline 실행까지 **데이터와 제어 흐름을 한 방향으로 추적**할 수 있도록 개선
- 새로운 Animation을 정해진 순서에 따라 연결·초기화·Trigger 등록·정리할 수 있도록 **확장 절차를 표준화**
- Trigger와 Timeline을 각각 생성한 계층에서 정리하도록 구성해 생성과 cleanup의 책임을 명확히 함

과도한 이미지 로딩 우선순위로 발생한 LCP 병목 개선

[상황]
- 모바일 Lighthouse 측정에서 LCP가 4.1초로 나타나 초기 화면의 주요 콘텐츠 노출이 지연됨
- Chrome Performance 패널로 확인한 결과, 초기 화면의 LifeMotion 이미지가 LCP 요소로 측정됨
- 첫 8개 이미지에 모두 높은 로딩 우선순위를 적용해, 실제로 필요한 이미지와 나중에 필요한 이미지가 동시에 요청되고 있었음

[수행]
- Chrome Performance 패널의 LCP 마커와 실제 DOM 요소를 확인해 이미지 로딩 병목 추적
- 반복 갤러리 이미지에 적용된 `priority`를 제거하고 Next.js Image의 기본 지연 로딩 적용
- 이미지 품질을 75에서 60으로 조정하고, 화면 크기에 맞는 반응형 `sizes` 설정 유지

[결과]
- 모바일 Lighthouse 성능 점수 86점 → 99점
- LCP 4.1초 → 1.0초, 약 76% 단축

콘텐츠 도달 기준을 직접 설계한 사용자 행동 측정 환경 구축

[상황]
- 긴 스크롤형 포트폴리오 특성상 방문자가 어떤 콘텐츠까지 확인하고 어느 구간에서 이탈하는지 파악하기 어려웠음
- 정성적인 피드백과 단순 방문자 수만으로는 콘텐츠 구조 변경의 필요성과 개선 효과를 판단하기 어려웠음

[수행]
- Amplitude Browser SDK를 적용하고 Autocapture를 비활성화한 뒤, 섹션 도달·외부 링크 클릭 등 분석에 필요한 행동을 커스텀 이벤트로 정의
- 섹션이 일정 비율 이상 노출된 경우에만 도달 이벤트를 기록하고, 섹션명·순서·화면 너비를 함께 수집
- 누적 방문자 653명의 행동 데이터를 분석해 주요 이탈 구간을 식별하고, 이를 기준으로 콘텐츠 구조를 재설계
- 개편 전·후 기간의 Contact 도달률을 동일한 기준으로 비교해 구조 변경 효과를 검증

[결과]
- 누적 방문자 653명의 섹션 도달 및 링크 클릭 데이터를 기반으로 콘텐츠 탐색 흐름을 정량화
- 이탈 구간을 기준으로 콘텐츠 구조를 개편한 뒤 **Contact 도달률 39.6% → 48.3%로 개선**
- 정성적 피드백에 의존하던 개선 방식을 **사용자 도달률과 행동 데이터를 기준으로 판단하는 반복 개선 구조**로 전환

Other Projects

효잇

2025.09 – 현재

React Native · Expo · TypeScript · TanStack Query · Zustand · pnpm Workspace

가족 안부 확인 모바일 서비스 개발 (React Native)
- 부모·자녀 역할을 분리하고 **역할 선택 → 가족 연결 → 역할별 홈**으로 이어지는 UX 흐름 설계
- Expo Router와 Zustand를 연계해 인증·역할·온보딩 상태에 따른 라우팅 및 접근 제어 구현
- TanStack Query로 사용자 정보·알림·가족 연결 상태의 서버 데이터 조회 및 갱신 흐름 관리

- pnpm Workspace 기반 **모노레포를 설계**해 모바일 앱과 공통 패키지를 분리하고, 역할별 기능과 공통 기반 코드의 책임 경계 구성
- **팀 리드**로 제품 방향과 기능 우선순위를 정하고 디자인·개발 일정 및 작업 범위 조율

Nova

2024.03 - 2024.12

React Native · Expo · JavaScript · React Navigation · Axios · OpenAI API

캐릭터 성장형 진로 탐색 앱 개발 (React Native)

- 전공별 미션 → 경험치 획득 → 캐릭터 성장으로 이어지는 진로 탐색 흐름 기획
- 홈·미션·성장 현황·프로필 화면과 미션 완료에 따른 경험치·레벨 상태 흐름 구현
- 디자이너 없이 컬러·간격·카드·타이포그래피 기준과 전공별 캐릭터 5종 직접 제작
- 팀 리드로 서비스 방향과 기능 우선순위를 조율하고 교내 아이디어 페스티벌 1위 수상

Awards

과학기술정보통신부 장관상 · 전국 SW마이스터고 연합 해커톤 1위

2025.11 | 과학기술정보통신부·정보통신기획평가원(IITP) 주관

- 다문화 가정 학부모의 학교 공문서 이해를 돕는 AI 플랫폼 **ClipCat** 개발
- AI 서비스 UX 설계 및 Next.js 기반 문서 업로드·변환 결과·챗봇 인터페이스 구현

ClipCat 프로젝트 자세히 보기 →

AppJam 27th 우수상

2024.08 | K-디지털 플랫폼 주관 · 생활정보 부문

- 목표 달성 과정을 퀘스트와 성장 경험으로 구성한 목표 관리 서비스 **Cando** 개발
- 목표 진행률 시각화와 AI 추천 콘텐츠 흐름 설계 및 클라이언트 구현

Cando 프로젝트 자세히 보기 →

광주SW마이스터고 아이디어 페스티벌 최우수상

2025.01 | 교내 아이디어 페스티벌 1위

- 전공별 미션과 캐릭터 성장으로 관심 분야를 탐색하는 진로 탐색 앱 **NOVA** 개발
- 팀 리드로 서비스 기획·React Native 클라이언트 개발·UI 시스템 및 캐릭터 디자인 담당

NOVA 프로젝트 자세히 보기 →