김종민

Frontend Developer

Email. yung3152@gmail.com
Phone. 010-4023-6518
GitHub. github.com/Guksu
Blog. guksulog.com

자기소개

5년 차 프론트엔드 개발자입니다.
Next.js·React·TypeScript 기반으로 **성능 최적화와 레거시 개선, 장애 대응**에 강점을 가지고 있으며,
화면 구현을 넘어 운영 제품에서 반복되는 문제를 구조적으로 개선하는 데 관심이 많습니다.

테스트 코드를 기반으로 한 **TDD 방식의 개발**을 선호합니다.
좋은 테스트 코드는 추상화가 잘 되어 있어 구현이 바뀌어도 흔들리지 않는 테스트라고 생각하며,
핵심 로직을 배포 전에 테스트로 검증하는 개발 습관을 유지하고 있습니다.

기획·디자인 단계 참여부터 동료 개발자와의 기술 리딩, 고객 커뮤니케이션까지
개발 밖의 역할도 주도적으로 맡아왔습니다.

AI를 활용한 자동화에 관심이 많아 AI 코딩 도구를 실무 워크플로에 맞게 설계·도입해 왔으며,
더 나은 AI 활용 방식을 꾸준히 고민하고 있습니다.

경력

(주)클라이머스

Frontend Developer · 2025.03 - 2026.07
이커머스 플랫폼 · 셀러센터 · 내부 어드민 개발
Next.js · React · Flutter · GraphQL · TypeScript · Tailwind · TanStack Query · Recoil

MAU 15만 이커머스 플랫폼 '찹스틱스'의 프론트엔드 개발을 담당했습니다.
기획·디자인 단계부터 참여해 팀 의사결정을 함께했습니다.
합류 초기에는 레거시 오류를 안정화했고, 이후 성능 개선·개발 환경 개선·AI 활용 자동화로 집중 영역을 넓혀왔습니다.

상품 상세 페이지 전환 속도 개선

문제 상황 상품 목록에서 상세로 이동할 때 화면 전환이 눈에 띄게 느렸습니다. 원인은 getServerSideProps였습니다. 옵션·리뷰·배송 정보까지 수십 개 필드를 전부 내려줬고, 불필요한 데이터까지 포함된 쿼리 때문에 서버 응답 자체가 느려져 렌더링 시작이 늦어지는 구조였습니다.

해결 과정 검색 엔진에 실제로 필요한 건 상품명·설명·썸네일·가격뿐이었습니다. SEO 전용 쿼리를 분리해 SSR은 이 데이터만 처리하게 했습니다. 옵션·리뷰·배송 정보는 클라이언트에서 가져오도록 전환했습니다.

결과 SSR 페이로드 **74%** 감소(3.4kB → 0.9kB), 응답 시간 **68%** 단축(283ms → 90ms). 페이지 전환 체감 속도가 크게 개선됐고, SEO 품질은 그대로 유지했습니다.

상품 등록 시 반려 사유를 미리 알려주는 이미지 자동 검수

문제 상황 셀러가 상품을 등록하면 커머스팀이 썸네일 정책 위반 여부(검은 배경, 썸네일·대표사진 중복 등)를 수동으로 검수했습니다. 하루 20건 이상이 쌓이며 검수 구간이 병목이 되어 커머스팀 업무가 지연되고 있었고, 별도 요청이 없었지만 이를 해결할 문제로 판단해 개선을 주도했습니다.

해결 과정 검수 자체를 없애기보다, 등록 단계에서 셀러가 스스로 반려 가능성을 확인하게 하는 접근을 택했습니다. AI 모델이나 별도 서버 없이 브라우저에서 Canvas로 이미지 픽셀을 직접 분석했습니다. 이미지 가장자리 픽셀의 밝기를 읽어 반려 사유 1순위였던 검은 배경을 감지하고, 두 이미지를 작은 지문 형태로 요약해 비교하는 방식(Average Hash)으로 썸네일·대표사진 중복 여부를 판별해 업로드 시점에 경고를 띄웠습니다.

결과 셀러가 등록 전에 스스로 검수하게 되면서 커머스팀의 수동 검수 병목이 줄었습니다. 다만 비용 문제로 완전 자동화가 아닌 사전 검수에 그친 점은 아쉬움으로 남습니다.

매일 반복되던 Sentry 에러 확인 업무 자동화

문제 상황 Sentry 알림이 와도 원인 파악은 수작업이었습니다. 매번 Sentry에 접속해 스택트레이스와 코드를 대조해야 했고, 외부 SDK 노이즈에 묻혀 반복 이슈와 신규 이슈를 구분하는 데도 시간이 들었습니다.

해결 과정 Sentry API로 전날 주요 이슈를 수집하고, Codex가 코드베이스를 기반으로 원인 후보·영향 범위·확인할 코드 경로를 정리하는 파이프라인을 만들었습니다. 노이즈를 걸러낸 종합 리포트를 매일 오전 Slack 채널로 발송해 팀 전체가 같은 화면에서 이슈를 확인하게 했습니다. OpenAI API 종량 과금 대신 이미 구독 중인 Codex를 활용해 추가 비용 없이 운영했습니다.

결과 매일 반복되던 확인 비용이 줄었고, 노이즈에 묻혀 놓치고 있던 에러를 리포트를 통해 발견·해결했습니다. 이슈의 원인 후보와 우선순위를 팀이 함께 파악하는 체계가 됐습니다.

화면 녹화 기반 QA 이슈 재현 도구 개발

문제 상황 내부 어드민은 비용 정책상 Sentry를 쓸 수 없었습니다. QA 이슈가 텍스트와 스크린샷으로만 전달돼 입력 순서·네트워크 상태 같은 맥락이 자주 누락됐고, 버그가 나면 개발자가 담당자 자리로 직접 가서 화면을 봐야 하는 일이 반복됐습니다.

해결 과정 사용자의 화면 세션을 그대로 기록·재생하고, 같은 타임라인에 네트워크 요청과 콘솔 에러를 함께 수집해 하나의 리포트로 묶는 도구를 개인 사이드 프로젝트(qa-recorder)로 직접 개발해 실무에 적용했습니다. 별도 서버나 브라우저 확장 없이 HTML 파일 하나로 공유되도록 설계했고, 민감정보는 기록 시점에 마스킹했습니다. 상시 녹화로 인한 메모리·용량 문제는 20분 슬라이딩 윈도우로 최근 기록만 유지해 해결했습니다.

결과 담당자 자리로 찾아가 화면을 확인하던 과정이 사라지고, 파일 하나로 이슈 상황을 그대로 재생해 확인할 수 있게 됐습니다. QA와 개발 간 이슈 전달 비용이 줄었습니다.

외부 결제 후 돌아왔을 때 결제 정보가 사라지는 문제 해결

문제 상황 기프트 카드 결제가 정상 처리되지 않는다는 CS가 접수됐습니다. Flutter 앱이 웹뷰로 넘겨준 Base64 카드 이미지를 결제 완료 후 서버로 다시 전송해야 하는 구조였는데, 이 이미지를 Recoil로만 들고 있어 외부 결제 페이지로 리다이렉트되는 순간 값이 사라지고 있었습니다. localStorage는 Base64 이미지 용량 한계로 쓸 수 없었습니다.

해결 과정 결제 플로우에서 반드시 보존해야 하는 데이터와 일시적인 UI 상태를 먼저 구분했습니다. 이미지를 서버에 먼저 업로드하고 URL로 주고받는 방법도 검토했지만, 결제 프로세스 수정 범위와 리스크를 고려해 클라이언트에서 해결하는 쪽을 택했습니다. 용량 제한이 넉넉한 IndexedDB에 카드 이미지를 저장하고, 외부 결제로 나가기 전에 기록하고, 돌아온 직후 복원하는 흐름으로 구조를 바꿨습니다.

결과 결제 플로우의 데이터 유실 버그를 제거했습니다. 외부 페이지 이동이 있는 플로우의 상태 보존 기준도 마련했습니다.

컴포넌트마다 달랐던 스타일 기준을 디자인 토큰으로 통일

문제 상황 디자이너가 없는 환경에서 styled-components 스타일이 파편화됐습니다. 색상·타이포그래피·간격 기준이 컴포넌트마다 달랐고, Storybook도 방치돼 스펙 확인이 어려웠습니다.

해결 과정 색상·타이포그래피·간격 기준을 단일 디자인 토큰으로 정리해 Tailwind 설정에 반영했습니다. 운영 중인 서비스라 한 번에 갈아엎지 않고 수정 빈도가 높은 영역부터 점진 전환했으며, AI를 활용해 주요 화면 80% 이상과 공통 컴포넌트 50여 개를 전환했습니다. 방치된 Storybook을 재구성해 컴포넌트 스펙을 코드 레벨에서 확인하게 했습니다.

결과 스타일 기준이 일원화되어 화면 개발·수정 시 판단 비용이 줄었습니다. 운영 리스크를 낮추며 구조 개선을 진행했습니다.

페이지 JS 번들 56.8% 감축

문제 상황 앱 규모에 비해 빌드 번들 크기가 컸습니다. 원인을 분석해 보니 lodash는 함수 하나만 써도 전체가 번들에 포함됐고, tsx 파일에서 export한 상수를 다른 파일에서 가져다 쓰면서 해당 컴포넌트 파일 전체가 딸려 들어왔으며, 모달 등도 코드 스플리팅 없이 초기 번들에 포함돼 있었습니다.

해결 과정 lodash를 ESM 기반 lodash-es로 전환해 사용하는 함수만 번들에 포함되게 했습니다. 공용 상수는 순수 상수 모듈로 분리해 불필요한 코드 유입 경로를 차단하고, 모달 등 초기 화면에 필요 없는 컴포넌트는 코드 스플리팅으로 분리했습니다.

결과 공통 JS 번들 **37%**, 페이지 평균 **56.8%** 감축. 초기 로딩 비용을 줄이고 이후 성능 최적화의 기준을 마련했습니다.

(주)버츄어라이브 / (주)엘케이벤쳐스

Frontend Developer · 2023.05 - 2024.12

관광객 뷰티 예약 플랫폼 · 인생네컷 점주앱 개발 · 유지보수

Next.js · React · TypeScript · Canvas API · GitLab CI/CD

버츄어라이브에서 관광객 대상 다국어 뷰티 예약 앱을 담당했습니다.
고용승계로 엘케이벤쳐스에 합류해 동료 개발자와 코드 리뷰를 하며 기술 선택과 작업 분배를 리딩했습니다.
테스트 코드와 코드 컨벤션을 주도적으로 정립했습니다.

촬영 사진·템플릿 합성 기능 구현과 모바일 Safari 크래시 해결

문제 상황 촬영 사진과 템플릿 합성을 Canvas API로 구현했지만, iOS Safari에서 합성이 실패하거나 화면이 멈추는 오류가 발생했습니다. 고해상도 촬영 이미지 크기 그대로 Canvas를 생성하면서 iOS의 Canvas 메모리 한계를 넘는 게 원인이었습니다.

해결 과정 Canvas 최대 크기 기준을 정해 이미지를 기기 환경에 맞게 스케일링해 합성하고, 사용이 끝난 Canvas와 이미지 참조는 즉시 해제했습니다. 인쇄 품질을 유지하는 선에서 해상도와 메모리 사용량의 균형점을 찾아가며 조정했습니다.

결과 모바일 Safari의 합성 오류를 줄였습니다. 다양한 기기에서 안정적으로 동작하도록 개선했습니다.

수동 배포를 자동 배포 파이프라인으로 전환

문제 상황 배포가 수동이라 휴먼 에러 가능성이 높았고, 실패도 늦게 인지했습니다. 스테이징·운영 전환에도 안정적인 검증 절차가 필요했습니다.

해결 과정 GitLab push 시 자동 빌드되는 파이프라인을 구성하고, 결과를 Slack으로 받았습니다. 스테이징·운영은 Blue-Green으로 분리하고, 최종 전환은 수동 검증 후 진행하도록 설계했습니다.

결과 휴먼 에러 가능성을 줄이고 배포 상태를 빠르게 인지하게 됐습니다. 자동화와 수동 검증을 함께 둔 구조로 안정성을 확보했습니다.

아이폰 사진(HEIC) 업로드 실패 문제 해결

문제 상황 아이폰 기본 포맷인 HEIC 파일을 업로드하면 서버에서 처리 오류가 발생했습니다. Android도 버전별로 HEIC 처리 방식이 달라 서버만으로는 일관된 대응이 어려웠습니다.

해결 과정 서버 수정 대신 업로드 전에 클라이언트에서 해결하는 방향을 택했습니다. heic2any라는 라이브러리를 사용해 브라우저에서 HEIC를 JPEG로 변환한 뒤 업로드하게 했고, Android 버전별 처리 차이는 분기 로직으로 대응했습니다.

결과 업로드 실패 케이스를 줄였습니다. 기기별 파일 처리 차이에 안정적으로 대응하게 됐습니다.

(주)헬로비즈

Frontend Developer · 2022.02 - 2023.02
MVP 제품 개발 · 클라이언트 커뮤니케이션
React · TypeScript · TanStack Query · Zustand

7개 MVP 제품을 고객과 직접 소통하며 요구사항 정의부터 프론트 개발까지 수행했습니다.
백엔드 리소스가 부족할 땐 DB 구조를 파악해 API 명세서를 직접 설계했습니다.
Redux에 얽혀 있던 상태 관리는 TanStack Query와 Zustand로 분리해 유지보수성을 높였고, React 컴포넌트 로직을 순수 JavaScript 모듈로 변환해 Android 앱에서도 쓸 수 있게 했습니다.
코드 컨벤션과 라이브러리 도입 기준을 정리해 팀 협업의 기반을 마련했습니다.

프로젝트

guksu-harness GitHub ↗

- **AI 코딩 도구가 프로젝트마다 일관된 방식으로 일하게 만드는 도구**입니다. 누가·어떻게·어떤 순서로 작업할지를 미리 정의해, 매번 설명하지 않아도 같은 절차가 반복되게 했습니다(작업 절차 10종).
- AI를 여러 개 동시에 돌리면 비용이 **약 15배**로 뛰기 때문에, 가장 저렴한 방식부터 쓰고 그걸로 안 되는 근거가 있을 때만 규모를 키우도록 설계했습니다.
- 지켜야 할 규칙은 문서로 적는 대신 **도구가 직접 막습니다.** AI가 임의로 커밋하거나 비밀 키 파일을 열거나 운영 브랜치를 고치는 것을 자동 차단했고, 배포 전 점검은 눈으로 훑는 대신 실제 브라우저에서 측정해 통과 여부를 판정하게 했습니다.

먹로그 App Store ↗ GitHub ↗

- 맛집 기록 iOS 앱을 기획·디자인·번역·개발·QA·App Store 출시까지 단독 수행한 풀사이클 개인 프로젝트입니다.
- React Native(Expo) · Supabase(Postgres RLS·Edge Functions) · Kakao Map(WebView JS SDK 브리지) 기반입니다.
- 기획→퍼블리싱→구현→비주얼·로직 QA로 역할을 나눈 5-에이전트 AI 하네스를 직접 설계해 50개 이상의 스프린트를 운영했고, 전 과정 산출물(plan/ui-spec/qa-report)을 저장소에 남겼습니다.
- 계측으로 지도 콜드 로드 병목(WebView 부팅 88%)을 특정해 프리워밍으로 63% 단축했고, TDD로 1,565개 테스트를 유지합니다.

wvkit GitHub ↗

- 웹뷰 환경에 특화된 헤드리스 UI 컴포넌트 라이브러리입니다.
- CSS3DRenderer 카메라 모델 기반의 가로·세로 페이저와 핀치 줌을 제공하는 ScrollContainer를 구현했습니다.
- visualViewport API로 iOS 키보드에 의한 레이아웃 밀림을 방지하는 StableInput과, 네이티브 바운스와 충돌하지 않는 상태 머신(PullToRefresh)을 포함합니다.
- 소프트 키보드와 세이프 에어리어 변화를 감지하는 훅을 제공합니다.

qa-recorder GitHub ↗

- rrweb 기반 웹 앱 QA 세션 재현 도구입니다.
- DOM replay, 네트워크 요청, 콘솔 에러, 민감정보 마스킹을 하나의 리포트로 묶어 QA와 개발 간 이슈 전달 비용을 줄이는 것을 목표로 했습니다.
- 별도 서버나 브라우저 확장 없이 단일 파일로 재현 환경을 공유할 수 있도록 설계했습니다.

학력

강원대학교

무역학과 졸업

[2013 - 2020]