About Me

[사용자 경험을 함께 고민하며, 팀과 함께 성장하는 프론트엔드 개발자]

저는 “사용자 경험을 함께 고민하며, 팀과 함께 성장하는 프론트엔드 개발자” 김권희입니다.

5년간의 경험을 통해, 단순히 화면 구현만 하는 개발자가 아닌 서비스 성과와 직결되는 개발들을 수행해 왔습니다.특히 React 기반 SPA에서 Next.js SSR로 전환하며 성능과 SEO를 동시에 개선했고, UI/UX 또한 디자이너와 함께 고민하며 웹페이지의 가독성을 높였으며 이를 통해 MAU를 20배 성장시키는 경험을 했습니다.

그리고 수동적으로 서비스를 배포 했던 방식을 GitHub Actions를 활용한 CI/CD 파이프라인 구축으로 배포 시간을 80% 단축하며 개발 문화 개선에도 이바지했습니다.

문제 해결 과정에서 저는 항상 사용자 불편을 줄이는 기술적 접근을 고민합니다. 프로필 이미지 비정상 노출 문제를 react-cropper로 해결하여 프로필 등록 완료율을 3배 높였고, 대용량 미디어 파일을 WebP/WebM으로 최적화하여 사용자 체감 속도와 비용 절감을 동시에 이끌어냈습니다. 또한 페이지 전환 속도 및 안정성을 개선 시키기 위해 Skeloton UI를 이용하여 안정적인 사용자 경험을 개선해 보기도 하였습니다.

앞으로도 저는 비즈니스 성과와 직결되는 프론트엔드 개발자라는 브랜드로 성장해 나가고자 합니다. 기술 트렌드에 대한 빠른 학습과 적용, 팀과의 협업을 통한 시너지 창출, 그리고 사용자가 만족하는 결과물을 만들어내는 집요한 실행력을 강점으로, 새로운 조직에서도 가치 있는 변화를 주도하겠습니다.

Skill

- **Frontend :** React.js, Next.js, JavaScript, TypeScript, React Native
- **State Management :** Recoil, React Query
- **Styling :** Styled-components, Emotion,
- **Tools:**
  - 협업 : Notion, Slack, GitHub, 네이버웍스, Confluence
  - 개발 : VSC (Visual Studio Code)
  - 디자인 : Figma, Zeplin, Adobe XD

Education

- 2013.03 - 2016.08 **세종대학교 컴퓨터공학 전공 (편입, 졸업)**
- 2009.03 - 2011.10 **수원과학대학교 (3년제 중퇴)**
- 2006.03 - 2009.02 **평촌고등학교 (인문계 졸업)**

Career Summary

- **Front end Developer 직무 관련 (총 5년차)**
  - 2022.12 - 2025.08 (주)플필 : Front end Developer 담당
  - 2020.09 - 2022.11 (주)딜리셔스랑고 온라인 개발팀 : Front end Developer 담당
- **Engineer 직무 관련 (3년)**
  - 2017.01 - 2020.09 (주)주안솔루션 : 개발 본부 엔지니어 담당 (대리)
    - SK Hynix, SK 이노베이션, 한국지역난방공사, 아모레퍼시픽, 한국수력원자력 공장 자동화 및 데이터 고도화 작업

Project

(주) 플필

2022.12 - 2025.09 (2년 10개월) | 정규직 | Front end 개발 | 대리

슬랙 온보딩 AI 봇(MVP) – LangChain + Claude Code 워크플로우

- **기간 :** 2025.03 ~ 2026.08 *(PoC → 파일럿 운영)*
- **기술 스택 :** Slack Bolt, OpenAI SDK, Claude Code, LangChain, GitHub Actions
- **이슈:**

  스타트업인 회사에서 온보딩 문서가 여러 툴에 분산되어 찾기가 힘들었고, 동료들에게 반복 Q&A가 발생해 팀 생산성이 저하. AI/RAG를 현업에 빠르게 적용해 효과를 검증하기 위해 슬랙 기반 MVP를 기획.
- **주요 역할 :**
  - 직무별(기획/디자인/개발/CS) 필수 문서, FAQ 선별, 설정
  - 정책/가이드/위키 정제 → 임베딩 → LangChain 체인 구성
  - **S**lack Bolt로 /onboard 커맨드, 모달 체크리스트, 스레드 Q&A 구현
  - 첫 Claude Code로 바이브 코딩하며 서브 에이전트/스킬/커맨드 활용
- **해결 및 성과 :**
  - 바이브 코딩으로 인한 3일 이내 MVP 구축
  - 온보딩 탐색 시간 40%↓, 반복 질의 감소
  - 에이전트·스킬·커맨드 모듈화로 기능 확장/교체 용이, 서로 다른 문서 구조에도 재사용성 확보

고객 정보 관리 및 대시보드 개발 운영 (BackOffice)

- **기간 :** 2024.04 ~ 2024.08
- **기술 스택 :** React, TypeScript, React Query, Recoil, Axios, Recharts(차트)
- **이슈 :**
  플랫폼 운영 과정에서 고객 정보 수정, 결제 수단, 환불 처리 등 CS 업무가 발생하면, 매번 개발자에게 직접 수정 요청을 전달해야 했습니다. 전용 백오피스가 없어 운영팀이 스스로 처리할 수 없었고, 그로 인해 응대 실수,위험, 개발 리소스 소모가 커졌습니다. 이에 즉시 처리할 수 있도록, 고객 관리·결제/정산·대시보드 등 자체 처리 가능한 백오피스 환경이 필요했습니다.
- **주요 역할:**
  - **요구 수집·정의 :** 운영·CS 직원 인터뷰를 통해 실제 업무 플로우와 불편 지점을 파악, 요구사항 목록/우선순위 수립
  - **관리자 UX 설계 :** SaaS형 관리자 페이지 패턴을 기준으로 정보 구조·화면 흐름 설계
  - **우선순위 기반 개발:** 긴급 이슈부터 기능 구현 및 배포
  - **핵심 기능 구현:**
    - 대시보드: 일/주/월 주요 KPI 시각화
    - 유저 관리: 검색/필터/정렬, 프로필/연락처 수정, 상태 변경, 엑셀 추출
    - 멤버십 관리: 멤버십 등록/해지/만료 상태 확인 및 변경, 이력 추적
    - 결제 관리: 결제 내역 조회·상세, 승인/취소/환불 처리, 정산 리포트 다운로드
- **결과 및 성과 :**
  - **CS 처리 속도 개선 :** 고객정보 수정·결제 확인/환불 등 이슈를 백오피스에서 즉시 처리하도록 전환 → 개발자 전달 재작업 감소
  - **운영 자율성 강화 :** 권한·유효성 검증·활동 로그를 적용해 비개발자도 안전하게 처리, 변경 이력 추적 가능
  - **핵심 기능 1차 오픈 :** 대시보드·유저관리·멤버십·결제관리 등 4개 핵심 영역을 우선 출시(MVP)하여 반복 수작업을 제품화
  - **데이터 가시성 확보 :** 대시보드로 일/주/월 KPI를 실시간 모니터링해 이슈 감지와 의사결정 리드타임 단축
  - **확장·유지보수 용이성 :** SaaS형 관리자 UX 패턴 + 공통 테이블/폼/모달을 도입해 동일 유형 화면의 재사용이 가능, 후속 요구(쿠폰/프로모션 등)도 신속 대응 가능

MobX → Recoil, React Query 상태관리 마이그레이션

- **기간 :** 2025.01 ~ 2025.04
- **기술 스택 :** React, TypeScript, Recoil, React Query
- **이슈 :**
  서비스 규모 확대로 MobX 전역 상태와 비동기 로직이 복잡해 졌으며 오류 추적과 신규 개발 속도가 저하되고, 상태 흐름 파악 난이도로 협업 힘들었습니다. 이를 해소하기 위해 로컬/서버 상태 분리 및 비동기 표준화를 위한 Recoil + React Query 마이그레이션을 추진했습니다.
- **주요 역할 :**
  - MobX 구조의 현재 상태를 분석하고, 로컬/서버 상태 분리 방향을 정의
  - Recoil + React Query 조합 선정 및 마이그레이션 로드맵 설계
  - Recoil atom/selector 설계, 서버 상태는 React Query 중심으로 재구성
  - 기존 MobX 스토어를 점진적으로 대체할 수 있도록 모듈 단위 전환 구조 설계
  - 공통 query, hooks, 상태 관리 패턴 직접 구현
  - 중복된 비동기 로직을 공통화하고, 신규 기능 개발에 재사용 가능하도록 리팩토링
- **해결 및 성과 :**
  - **상태 구조 단순화 :** MobX 전역 스토어를 Recoil atom/selector로 재구성해 화면별 필요한 상태만 참조하도록 하고, 상태 흐름 이해 시간을 단축
  - **비동기 로직 표준화 :** React Query 기반 서버 상태 일원화로 캐싱·리패칭·로딩/에러 처리를 공통 패턴으로 묶어 중복 코드와 실수 감소
  - **디버깅 및 개발 속도 향상 :** 상태 변경 지점을 명확히 해 상태/요청 관련 버그 재현, 원인 분석 시간을 줄이고 신규 기능 개발 개선
  - **협업 및 온보딩 효율 증대 :** 상태 관리 컨벤션,디렉터리 구조,atom/쿼리 가이드를 문서화해 신규 개발자의 러닝 커브를 낮추고 코드 리뷰 기준을 통일

WordPress → React SPA 마이그레이션

- **기간 : 2022.12 ~ 2023.04**
- **기술 스택 :** React, JavaScript, styled-components, Axios
- **이슈 :**

  기존 워드프레스 환경의 사이트가 페이지 단위 기능 확장이 어렵고, 플러그인, 스크립트가 누적되면서 DOM 조작 위주의 산발적인 코드가 계속 쌓이는 상태였습니다. 반응형 대응도 화면마다 따로 처리하면서 QA 이슈와 개발 피로도가 높아졌습니다. 이를 근본적으로 해결하기 위해 React 기반 SPA로 전면 리뉴얼을 진행하게 되었습니다.
- **주요 역할 :**
  - ESLint + Prettier를 통합해 팀 공통 코드 스타일·품질 기준 정립
  - src/{assets, components, hooks, pages, services, utils, styles}로 레이어 명확화
  - jsconfig 절대 경로 설정을 적용해 모듈 간 결합도 감소 및 가독성 향상
  - Atomic Design패턴으로 공통 컴포넌트를 체계적으로 분리
  - styled-components + ThemeProvider로 색상/타이포그래피/반응형 규칙을 디자인 토큰으로 관리
  - services/api 하위에 Axios 인스턴스 구성(공통 헤더, 인터셉터, 에러 처리)
  - Context API + Custom Hook으로 전역 UI 상태를 최소화해 예측 가능한 상태 흐름 구성
  - 스켈레톤 UI 적용으로 체감 성능 개선
- **해결 및 성과 :**
  - **개발 생산성 및 유지보수 효율 향상 :** Atomic Design과 공통 컴포넌트/디자인 토큰 도입으로 화면 개발 시 재사용성이 크게 늘어나 기능 추가, 수정 소요 시간이 단축, 코드 중복률 감소
  - **페이지 로딩 속도 및 체감 성능 개선 :** React SPA 구조 정리, 불필요한 DOM 조작 제거, 가상 스크롤/스켈레톤 UI 도입으로 페이지 로딩 시간이 평균 2.5초 → 1초 이하로 개선, 사용자 이탈률 감소에 기여

    - **서비스 확장성과 안정성 강화 :** API 통신 , 폴더 구조, 코드 스타일을 표준화해 새로운 기능/페이지가 추가될 때도 일관된 방식으로 개발, 리뷰가 가능해졌고, 버그 대응 속도도 향상
    - **비즈니스 임팩트 창출 :** 구조적인 리뉴얼 이후 서비스 신뢰도와 사용성이 향상되며 MAU 500명 → 10,000명(약 20배 성장) 달성

React SPA → Next.js SSR 마이그레이션

- **기간 :** 2023.12 ~ 2024.04
- **기술 스택 :** Next.js, TypeScript, React
- **이슈 :**
  1차 리뉴얼 후 트래픽, 캠페인이 확대되며 검색, 광고 유입의 중요성이 커졌지만 CSR 구조로 초기 로딩, 크롤러 인식이 지연되고 JS 단독 개발로 타입 기준이 흔들려 협업 비용이 증가해, 이를 해소하고자 Next.js 기반 SSR 전환, SEO 템플릿 도입, TypeScript 적용의 2차 리뉴얼을 추진했습니다.
- **주요 역할 :**
    - 기존 React CSR 구조를 분석하고, 주요 페이지를 Next.js 기반 SSR/SSG로 재구성
    - Dynamic Import + Code Splitting 전략을 설계해 초기 번들 크기 최소화
    - 페이지 타입별 title/description, OG 태그, canonical URL, robots.txt, sitemap 구조 정의
    - TypeScript 도입 및 공용 타입 정의 설계로 도메인 모델을 일관되게 사용
    - ESLint + Prettier 규칙을 통합하고, 빌드 단계에서 타입·린트 체크를 자동 수행하도록 구성
    - 백엔드 API 호출을 별도 모듈로 분리해 UI와 비즈니스 로직을 분리
- **해결 및 성과 :**
    - **초기 로딩 및 체감 속도 개선 :** Next.js SSR 전환과 코드 스플리팅으로 TTFB 약 45% 개선, LCP 1.2초 단축 → 첫 화면 로딩 체감 속도 향상으로 이탈률 감소에 기여
    - **검색 노출 및 유입 향상 :** SEO 메타 태그·sitemap·robots·구조화 데이터 적용으로검색 노출률 약 30% 증가, 주요 키워드 자연 유입 개선
    - **타입 기반 협업 및 품질 향상 :** TypeScript + 공용 타입 정의로 런타임 타입 오류 감소, QA 단계에서 발견되던 타입 불일치 이슈 감소, 코드 리뷰 효율 향상
    - **확장성과 유지보수성 강화 :** ESLint/Prettier, API 레이어 분리로 일관된 코드 스타일과 구조를 확보하고,
      신규 페이지·기능 추가 시에도 동일한 패턴으로 빠르게 확장 가능해짐

캐스팅 지원자 HR 시스템 개발

- **기간 :** 2023.04 ~ 2023.12
- **기술 스택 :** React, TypeScript,
- **이슈 :**

  엑셀·이메일 기반의 수작업 관리로 지원자 데이터 누락/중복이 발생하고, 검색·분류 기능 부재로 업무 시간이 과다 소요됨. 상태 추적이 어려워 협업 지연과 커뮤니케이션 오류가 반복되어 전용 웹 시스템이 필요했음.
- **주요 역할 :**
    - 디렉터·운영진 인터뷰로 AS-IS 업무 분석 → 기능 우선순위 수립(MVP 먼저)
    - 목록-상세-상태 변경이 빠른 관리자 UX(검색·필터·테이블·일괄 처리) 정의, 반응형 화면 흐름 설계
    - 공고 등록/관리, 지원자 CRUD 및 상태 파이프라인(지원→서류합격→오디션 예정→캐스팅)
    - 고급 검색/필터/태그, 연기 영상 업로드/미리보기, 알림(이메일/내부 알림)
    - 백엔드와 스키마·에러 규격 합의, Axios 인스턴스/인터셉터로 에러·인증·재시도 표준화
    - 권한, 변경 이력/활동 로그, CSV 내보내기, 운영 가이드/체인지로그 문서화
- **해결 및 성과 :**

- 처리 시간 60% 단축(지원자 조회·분류·상태 변경 등 반복 업무)
- 데이터 누락 0건 유지(상태 파이프라인·이력 추적 정착)
- 신규 디렉터 일평균 10명+ 가입 유도, 현장 피드백 긍정적
- 기존 수동 프로세스 전면 대체, 운영팀의 상시 업무를 제품화하여 협업 효율과 서비스 경쟁력 동시 강화

(주) 딜리셔스랑고 (광고 에이전시)

2020.09 - 2022.11 (2년 2개월) | 정규직 | Front end 개발 | 매니저

AWS + GitHub Actions 기반 CI/CD 구축

- **기간 :** 2021.04 ~ 2022.11
- **기술 스택 :** AWS S3, CloudFront, Route 53, GitHub Actions, Slack Webhook
- **이슈:**

  외주 프로젝트에서 수동 FTP 배포로 시간이 오래 걸리고, 캐시 무효화 누락·롤백 어려움 등으로 가용성·일관성이 떨어짐. 도메인/SSL 세팅도 매번 반복되어 운영 피로가 높았음.
- **주요 역할 :**
  - 브랜치 전략(main/dev) 기준 빌드→업로드→캐시 무효화 자동화 파이프라인 설계
  - S3 정적 호스팅 + CloudFront CDN, Route53 도메인, ACM SSL 적용(HTTP→HTTPS 리다이렉트)
  - dev/staging/prod 버킷·배포 분리, CloudFront 태그 기반 무효화
  - Actions에서 lint·type-check·build 선행, 실패 시 즉시 중단
  - 배포 로그·접속 로그를 CloudWatch/S3로 적재, Slack 알림 연동(성공/실패·변경 내역)
- **해결 및 성과 :**
  - 배포 시간 80% 단축 *(15분 → 3분)*, 야간/긴급 수정 리드타임 대폭 감소
  - 무중단 배포율 100%(CloudFront 오리진 교체/버전 롤백로 다운타임 없이 전환)
  - 캐시 이슈 감소: 경로/태그 기반 정밀 무효화로 잘못된 리소스 노출 건 0건 유지
  - 배포 실패/재작업 50%↓: 빌드 전 품질 게이트와 자동 롤백 스크립트 도입 효과
  - 운영 표준 정립: 신규 프로젝트 도메인·SSL·배포 구성을 템플릿화하여 온보딩 시간 50%↓, 팀 커뮤니케이션 Slack 자동 공지로 투명성↑

반응형 개발 표준화 시스템 구축 및 공통 UI 가이드 수립

- **기간 :** 2020.09 ~ 2022.11
- **기술 스택 :** Styled-components, Media Query, Flex/Grid Layout
- **이슈 :** 프로젝트별로 반응형 기준이 제각각이라 UI 파편화·레이아웃 깨짐이 반복되고, 유사 레이아웃을 매번 새로 구현하며 중복/유지보수 비용이 증가.
- **주요 역할 :**
  - GA로 주요 해상도 구간(PC/Tablet/Mobile) 분석 → 표준 브레이크포인트 합의
  - 색·폰트·간격을 ThemeProvider로 디자인 토큰화, 일관된 스케일 적용
  - Grid/Flex 기반 그리드 시스템과 반응형 Mixin 제작, 헤더/카드/리스트 등 공통 컴포넌트화
  - 사용 예시·금지 패턴·접근성 체크리스트를 정리한 반응형 가이드 배포
  - 실제 디바이스 크로스 테스트로 터치 영역/픽셀 오차 사전 검출 프로세스 운영
- **성과 :**
  - 개발 속도 50% 향상(신규 화면 평균 4일 → 2일)
  - 모바일/태블릿 레이아웃 이슈 다수 감소, 접근성(탭/포커스/명도 대비) 준수율 개선

    - 코드 중복 체감 감소 및 컴포넌트 재사용률 증가로 유지보수 비용 절감
    - 기획·디자인·개발 간 공통 언어 정립으로 협업 마찰 감소, 리뷰 효율 향상

LG OLED Display 웹 접근성 개선

- **기간 :** 2021.06 ~ 2021.08
- **기술 스택 :** React, Lighthouse, Axe DevTools, WCAG 2.1/KWCAG
- **이슈 :** React 마이그레이션 후 시맨틱 구조·대체 텍스트·포커스 처리 미비로 접근성 진단에서 다수 오류가 발생했고, 클라이언트 요청에 따라 표준(WCAG/KWCAG) 수준까지 개선이 필요했습니다.
- **주요 역할 :**
    - Lighthouse/Axe로 이슈 유형 분류 & 우선순위 수립
    - header/nav/main/section/footer 등 시맨틱 구조 재정비, 적절한 ARIA(role/aria-label) 적용
    - 키보드 접근성(탭 순서, 포커스 인디케이터, 숨김 요소) 개선
    - 디자이너·기획자와 디자인 훼손 없는 대체 패턴 합의
    - 재발 방지를 위한 접근성 체크리스트/코딩 가이드 문서화 및 전달
- **해결 및 성과 :**
    - Lighthouse 접근성 58 → 95점 달성, 핵심 페이지 전반 품질 상향
    - 스크린 리더 기준 핵심 콘텐츠 인식률 100% 확보(대체 텍스트/읽기 순서 정합)
    - 키보드 전용 사용자 전 구간 탐색 가능 상태로 개선(포커스 링·탭 흐름 표준화)
    - 체크리스트 정착으로 신규 페이지 사전 검수 체계화, 반복 이슈 대폭 감소 및 유지보수 효율 향상