최기환

rlghks3004@gmail.com
Website · GitHub · LinkedIn · Obsidian

성능·아키텍처 Frontend Engineer

- 300대 인스턴스 실시간 폴링 환경에서 렌더 총량을 96.7% 줄이고 INP를 400ms대에서 100ms대로 안정화했습니다. 폴링·리사이즈가 겹치던 상황에서 10 FPS에서 60 FPS로 개선한 것도 같은 맥락입니다.
- 팀 전체가 사용하는 공용 데이터 그리드를 처음부터 설계·운영했습니다. table 태그 구조의 레이아웃 한계를 div 기반 헤드리스 구조로 재설계해 DOM 노드 90% 감소, 리사이즈 처리 22ms→0.5ms를 달성하고, 20+ 기능을 안정적으로 조합·운영하고 있습니다.
- 레거시 ExtJS 코드베이스에서 비동기 타이밍 오류, this 바인딩 깨짐 등 코어 JavaScript 레벨 이슈를 직접 파고들어 해결한 디버깅 역량을 갖추고 있습니다.

Experience

프론트엔드 개발자

2024.11 ~ 현재

주식회사엑셈

인스턴스 통합 모니터링 대시보드 개발

300대 인스턴스를 실시간 폴링하며 리사이즈·드래그가 동시에 발생하는 환경에서 렌더 총량 96.7%(150회→5회) 감소, INP 400ms대→100-ms대 안정화를 달성했습니다. 카드형 2-depth 구조를 고밀도 그리드 1-depth 허브로 재설계해 한 화면 비교와 즉시 RTM/PA 이동 흐름을 확보했습니다.

- **useFrozenData**로 폴링 중 데이터 스냅샷을 고정하고, **useDeferredValue**로 리사이즈 렌더를 저우선 처리하며, 핵심 셀에 memo를 적용해 폴링·리사이즈가 겹치는 구간의 렌더 총량을 150회에서 5회로 줄였습니다.
- **restart/start/stop** 인터페이스의 PollingManager로 화면별 타이머를 묶어 선언적으로 폴링 주기를 제어하도록 일원화했습니다.
- 서로 영향을 주는 필터·조회 상태와 반복되던 저장/복원 로직을 중앙 저장소와 버전 기반 마이그레이션으로 정리해 수정 범위를 국소화했습니다.
- API 전면 교체 시점에 Vue 연장 대신 React와 사내 디자인 시스템 기반으로 재구축해 구조 부채 누적을 줄였습니다.
- Feature Flag 기반으로 대시보드 변경을 고객사별로 격리해 배포 영향 범위를 제어했습니다.

대규모 데이터 화면용 공용 데이터 그리드 개발

팀 내 기여 1위(95건 이슈 처리)로 주도한 공용 데이터 그리드입니다. table 태그의 레이아웃 한계로 고정 컬럼·가상화·리사이즈 조합이 불가능해지자 div 기반 헤드리스 구조로 전면 재설계했고, descriptor 하나만 등록하면 기능이 확장되는 파이프라인으로 20+ 기능을 안정적으로 운영 중입니다.

- 통합 테스트로 기존 동작을 고정한 뒤 table 태그에서 div 기반 렌더링으로 마이그레이션했습니다. 셀 배치를 absolute에서 Row absolute + Cell flex 구조로 재설계해 열 조작과 레이아웃이 일치하도록 맞췄습니다.
- 재설계로 DOM 노드 90% 감소, 리사이즈 처리 22ms→0.5ms를 달성했습니다.
- 행 가상화로 실제 DOM 마운트를 뷰포트 범위로 제한하고, React.memo에 값 기반 비교 함수를 적용해 스크롤 중 불필요한 리렌더를 억제했습니다.
- 셀·행마다 개별 핸들러를 붙이는 대신 테이블 레벨 이벤트 위임을 적용해 핸들러 수와 상호작용 처리 비용을 낮췄습니다.
- 검색·정렬·페이지네이션 옵션을 query 상태 모델로 정규화하는 기반을 만들고, 컬럼 가시성 변경·고정 컬럼 재마운트·행 DnD 모드 조합에서 상태가 어긋나지 않도록 단위 테스트와 브라우저 테스트로 회귀 조건을 고정했습니다.
- UI와 로직을 헤드리스 구조로 분리하고, 기능별 descriptor와 columnContract 컴파일 단계로 데이터·그룹·유틸리티 컬럼의 검색·필터·정렬·-DnD 대상 여부를 표준화했습니다. Storybook 시나리오, 단위 테스트, 브라우저 통합 테스트로 컬럼 가시성·복사·Excel export·DnD 조합 회귀를 관리했습니다.
- PR마다 복잡도·코드 중복·함수 길이 등 7개 품질 지표를 CI에서 자동 검사해 리뷰 전 품질 기준선을 보장하는 게이트를 구축했습니다.
- TailwindCSS v4, Vite 8, Storybook 10.3 전환과 npm 스코프 마이그레이션을 병행했습니다. 이후 Atlaskit DnD 의존성을 제거하고 자체 Pointer Events 기반 DnD 엔진으로 열·행 재정렬 흐름을 통합했습니다.

차세대 데이터베이스 성능 모니터링 제품 개발

차세대 DB 모니터링 제품에서 위젯 빌더, 상태 관리, 저장소 추상화, SQL 분석 UX를 담당했습니다. 전역 스토어 20개를 지역 스토어로 전환해 초기화 리렌더 50% 감소·보일러플레이트 70% 감소를 달성하고, 저장 엔진 추상화로 레이아웃 저장 방식을 DB 영구 저장으로 하루 만에 마이그레이션했습니다.

- 전역 Zustand 스토어 20개를 Context API 기반 지역 스토어로 전환하고, 스토어 생성을 팩토리 함수로 추상화했습니다. 초기화 리렌더 50% 감소, 전역 상태 사이드 이펙트 제거, 보일러플레이트 70% 감소 효과를 이끌었습니다.
- Zustand persist 미들웨어의 저장 엔진을 인터페이스로 추상화해, 기존 로직을 유지한 채 로컬 스토리지 기반 레이아웃 저장을 DB 영구 저장으로 하루 만에 마이그레이션했습니다.
- line·area·bar·scatter·table 5개 차트 타입을 어댑터/레지스트리 구조로 묶어, 신규 위젯 추가 시 공통 빌더를 건드리지 않고 해당 타입 모듈만 확장하도록 재설계했습니다.
- **One SQL Detail**, **Plan Change**, **SQL Scatter**, Monaco 기반 SQL 에디터 등 SQL 분석 핵심 화면을 구현해 실행 계획 비교, diff 시트, 바인드 변수 하이라이팅·값 치환, 드릴다운 같은 분석 UX를 제품 기능으로 연결했습니다.
- SQL Detail·Text/Plan/Bind·Object Detail·Search SQL·Parallel Session 등 SQL Analysis 전체 화면군을 레거시에서 차세대 FSD 구조로 전담 마이그레이션했습니다.
- 보안 취약점 자동 분석(Secret Detection), 테스트 커버리지 리포트, MR별 Storybook 프리뷰 환경을 포함한 GitLab CI/CD 파이프라인을 구축했습니다.
- MSW 기반 API 모킹 환경을 도입해 백엔드 의존 없는 개발 테스트 흐름을 체계화하고, Sentry 에러 모니터링을 통합했습니다.

프론트엔드 개발 생산성 및 진단 구조 개선

레거시 ExtJS 코드베이스와 차세대 제품 개발 흐름에서 수동 추적·환경 준비·페이지 제작 검증을 정리하고, 진단 도구와 하네스를 공용화해 초기 환경 셋업 3시간을 1분 이내로 단축했습니다.

- **mfo_v5_starter**에 레포·서브모듈·환경 감지·**config.json** 생성·실행 절차를 중앙화해, 초기 셋업 3시간이 걸리던 레거시 개발 환경을 1분 이내에 시작 가능한 흐름으로 줄였습니다.
- 팀 공용 브라우저 진단 도구를 구축해 **console.log** 없이 SQL 응답을 실시간으로 확인하고 컴포넌트를 IDE로 바로 추적할 수 있게 해, 레거시 ExtJS 이슈의 원인 추적 비용을 낮췄습니다.
- 레거시 SQL 참조·실행 payload를 Oracle/PostgreSQL 기준으로 점검하는 **sql:report**와 **sql:lint-runtime** 흐름을 만들고, 오류·경고·DB 분기·소스 경로별로 필터링 가능한 브라우저 리포트 UI까지 연결했습니다.
- Mock DB 기반 E2E를 기능 영역별 스위트와 MR 전용 scenario/resolver 레이어로 재구성하고, 250개 MR 후보를 브라우저 경로·REST/WS ledger clean gate·리포트 아티팩트로 추적하는 회귀 하네스를 정비했습니다.
- **@exem/design-studio** 코어 패키지와 Design Studio 템플릿에 page/i18n/screen-spec/policy 문서 스캐폴딩·검증, pageOnlyCommit 가드, iframe 미리보기·SNB 라우팅을 묶어 실제 프로젝트에서 dependency로 재사용 가능한 페이지 단위 검증 하네스로 공용화했습니다.
- Vitest Browser와 Playwright 기반 통합 테스트로 preview shell bridge, same-origin iframe navigation, page 파라미터 정규화, iframe src encoding을 고정해 Design Studio 미리보기 회귀를 코드 레벨에서 추적했습니다.
- WebSocket 공통 모듈에 JSDoc과 사용 예시를 추가해 호출 규칙과 응답 해석 기준을 문서화하고, 화면별 추측에 의존하던 통신 로직 분석 부담을 줄였습니다.
- 브라우저 기반 검증 중 발견된 RTM 프레임 초기화 순서, SQL Elapsed Time 필터 값 shape, 설정 화면의 빈 선택 상태 예외를 추적해 초기화 순서 보정과 입력 정규화로 레거시 ExtJS 런타임 오류를 줄였습니다.

Frontend Product Engineering

2024.11 ~ 현재

개인 프로젝트

Resume Design System 구축

React 19와 Vite 기반으로 A4 이력서를 블록 단위로 편집하는 개인용 디자인 시스템을 구축했습니다. 블록 registry, IndexedDB 저장·스냅샷, **@media print**와 **window.print()** 기반 PDF 출력, JSON 복사·붙여넣기와 URL 공유용 읽기 전용 프리뷰를 연결해 작성·검증·공유 흐름을 하나의 도구로 묶었습니다.

- Header, Positioning, Core Impact, Career, Case Study, Skills 등 12개 이력서 블록을 자가 등록 registry 패턴으로 구성해 새 블록 추가가 독립적인 폴더 단위 변경으로 끝나도록 설계했습니다.
- A4 캔버스와 PDF 출력이 같은 디자인 토큰과 print CSS를 공유하도록 정리하고, 페이지 크기 고정·여러 페이지 출력·오버플로우 경고를 붙여 화면 미리보기와 출력 결과의 차이를 줄였습니다.

- Zustand와 Immer 기반 상태 모델에 IndexedDB 저장, 스냅샷, 자동 백업, 저장 상태 표시를 연결해 로컬 작성 도구에서도 데이터 손실과 편집 흐름 끊김을 줄였습니다.
- @dnd-kit 기반 블록 드롭·재정렬, 다중 블록 선택·이동, bulk removal을 구현해 이력서 내용을 코드 수정 없이 화면에서 재구성할 수 있게 했습니다.
- 링크 alias·favicon 칩, 헤더 연락처 자동 감지, JSON copy/paste, 압축 URL 기반 read-only A4 preview를 추가해 이력서 버전 재사용과 공유 흐름을 확장했습니다.

Technical Writing

LLM이 인용 번호를 세지 못하는 문제를 마커 시스템으로 해결한 이야기 2026.05
Read post

검색 API가 80만 토큰을 뱉길래, 도구를 쪼갰다 2026.04
Read post

LangGraph interrupt() 디버깅: checkpoint_ns가 조용히 삼킨 resume 경로 2026.04
Read post

리액트로 바라보는 정책과 메커니즘의 분리 2026.01
Read post

Zustand 스토어를 지역화하기 2025.11
Read post

Awards

항해 플러스 프론트엔드 1기 2024.01
팀 스파르타
프론트엔드 심화 과정 (최우수 수료)

Certificates

TOEIC: 925점 2022.03
YBM

Skills

LANGUAGES
TypeScript JavaScript HTML/CSS

FRAMEWORKS & LIBRARIES
React Astro Zustand TanStack Table TanStack Query TanStack Virtual ECharts TanStack Router

TESTING
Vitest Playwright MSW Storybook

TOOLS
Vite Git Biome Turborepo Docker GitLab CI