

포트폴리오

프로젝트별 담당 업무, 문제 해결 과정과 구현 자료입니다.

다운소프트랩 · 업무 프로젝트

Bitria

멀티 거래소 연동 암호화폐 거래 플랫폼

2025.11 – 현재 | 프론트엔드·백엔드 API 개발 담당

여러 암호화폐 거래소를 연동해 현물·선물·마진 거래를 제공하는 플랫폼입니다.

웹·모바일 앱의 프론트엔드와 백엔드 API 개발을 담당하고, 인프라 운영에 참여했습니다.

주요 업무

거래 화면	TradingView 차트, 실시간 시세·호가·체결, 현물·선물·마진 주문 및 포지션·체결 내역 구현
백엔드 API	주문 생성·조회·취소·수정 API를 개발하고 요청 검증·권한 확인·주문 처리 서비스와 연동했습니다. 계좌별 자산·USDT 환산·스냅샷·일별 손익 조회도 구현했습니다.
인증·보안	회원가입·소셜 로그인, 2단계 인증(2FA)·패스키·신뢰 기기·세션·계정 복구 기능을 개발했습니다. 고객확인(KYC)·입출금 연동과 API 키 관리도 구현했습니다.
멀티 플랫폼	하나의 코드베이스로 웹·iOS·Android·Electron 앱 빌드, 모바일 앱 스토어 출시, 네이티브 푸시·소셜 로그인·생체인증 연동
공통 구조	FSD(Feature-Sliced Design) 구조와 시맨틱 디자인 토큰을 설계했습니다. 다크·라이트 테마와 브랜드 2종·6개 로케일을 지원했습니다. 웹·앱별 코드를 분리하고 컴포넌트 API 기준을 문서화했습니다.
서비스 운영	가격 조건별 알람·클라이언트 오류 수집·앱 버전 정책·고객 문의(CS) API 개발, k3s 프로덕션 배포·운영 및 장애 대응

기술 TypeScript · Next.js 16 · React 19 · Tailwind CSS · Zustand · TanStack Query · Capacitor · Electron · Node.js · Express · Prisma · MariaDB · Redis · Docker · k3s

대표 문제 해결

교차 마진 주문 가능액 계산 오류 수정

레버리지 모달과 주문 폼이 최대 주문 가능액을 따로 계산해 같은 조건에서도 표시 금액이 최대 56배 차이 났습니다.

거래소 문서와 API 응답을 비교해 두 원인을 찾았습니다.

교차 마진 계산에 격리 마진용 리스크 한도를 적용했고, 잔고에 레버리지를 단순 곱하는 계산도 문제였습니다.

표시값만 고치지 않고 계산 로직을 공통 모듈로 통합했습니다.

수수료·안전 여유분·리스크 한도·레버리지 상한·최소 주문 단위를 반영했습니다.

주문 수량의 부동소수점 오차는 BigNumber로 처리했습니다.

결과

두 화면의 계산값을 거래 단위까지 일치시켰습니다.

경계 사례 20건을 검증하고, 운영 환경에서 발견한 양방향 포지션의 조건부 주문 누락도 수정했습니다.

모바일 홈 JavaScript 실행 시간 개선

프로덕션 모바일 홈에서 Lighthouse 성능 점수 38점, 총 차단 시간(TBT) 1,610ms를 측정했습니다.

JavaScript 청크 실행 시간을 분석해 shared/ui의 일괄 재수출 구조가 불필요한 모듈까지 같은 청크에 포함하는 것을 확인했습니다.

optimizePackageImports를 적용했지만 청크 수와 전체 번들이 늘어 이를 되돌렸습니다.

대신 컴포넌트를 직접 import하도록 바꾸고 three.js 배경은 늦게 불러오도록 했습니다.

결과

로컬 프로덕션 빌드 비교에서 TBT가 443ms에서 226ms로 약 49% 줄었습니다.

일부 청크 크기는 늘었지만 초기 로드에서 불러오는 JavaScript는 22KB 줄었습니다.

Lighthouse 모바일 설정(CPU 4배 스로틀·느린 4G)에서 워밍업 후 3~5회 측정했습니다.

프로덕션 진단값 1,610ms와 로컬 비교값은 측정 환경이 다릅니다.

실시간 갱신 범위를 표시 컴포넌트로 제한

페이지 최상위에서 실시간 스냅샷을 구독해 시계가 갱신될 때 주문 폼·하단 패널·닫힌 UI까지 다시 렌더링됐습니다.

웹에서는 같은 현상이 재현되지 않았습니다.

앱 화면에서는 데이터 구독을 실제 시계 표시 컴포넌트로 옮기고, props와 구독 인자의 참조를 안정화했습니다.

결과

12초 프로파일링에서 누적 렌더 시간 1,028 → 416ms(약 60% 감소), 렌더 횟수 2,802 → 1,323회(약 53% 감소)를 확인했습니다.

앱 현물 트레이딩 화면에서 조작 없이 실시간 데이터만 갱신한 구간입니다.

해당 구간에서 주문 폼·하단 패널의 불필요한 렌더링도 줄었습니다.

구현 구조와 설계

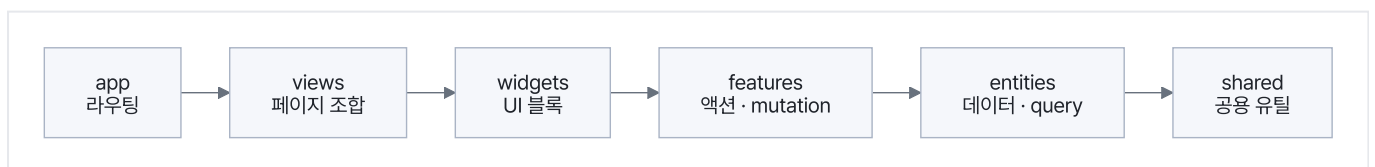
FSD 아키텍처와 화면 분기 기준

화면과 도메인이 늘어나도 의존성이 뒤섞이지 않도록 app → views → widgets → features → entities → shared 순의 단방향 규칙을 정했습니다.

조회(query)는 entity에, 변경(mutation)은 feature에 두어 각 레이어의 책임을 분리했습니다.

웹과 앱의 라우트가 다르다면 폴더를 분리했습니다.

화면 크기 차이가 작으면 prop으로 분기하고, 마크업 차이가 크면 디바이스별 파일을 두도록 기준을 정했습니다.

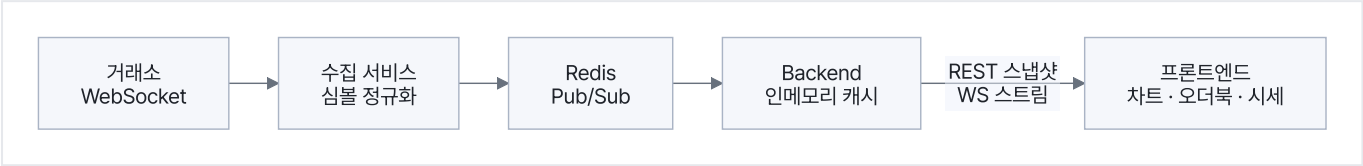


FSD 레이어 간 의존 방향과 각 레이어의 책임

실시간 데이터 전달과 구독 구조

거래소 WebSocket 데이터는 수집 서비스의 심볼 정규화와 Redis Pub/Sub을 거쳐 백엔드에 전달됩니다.
프론트엔드는 REST 스냅샷과 WebSocket 스트림으로 차트·호가·시세를 표시합니다.

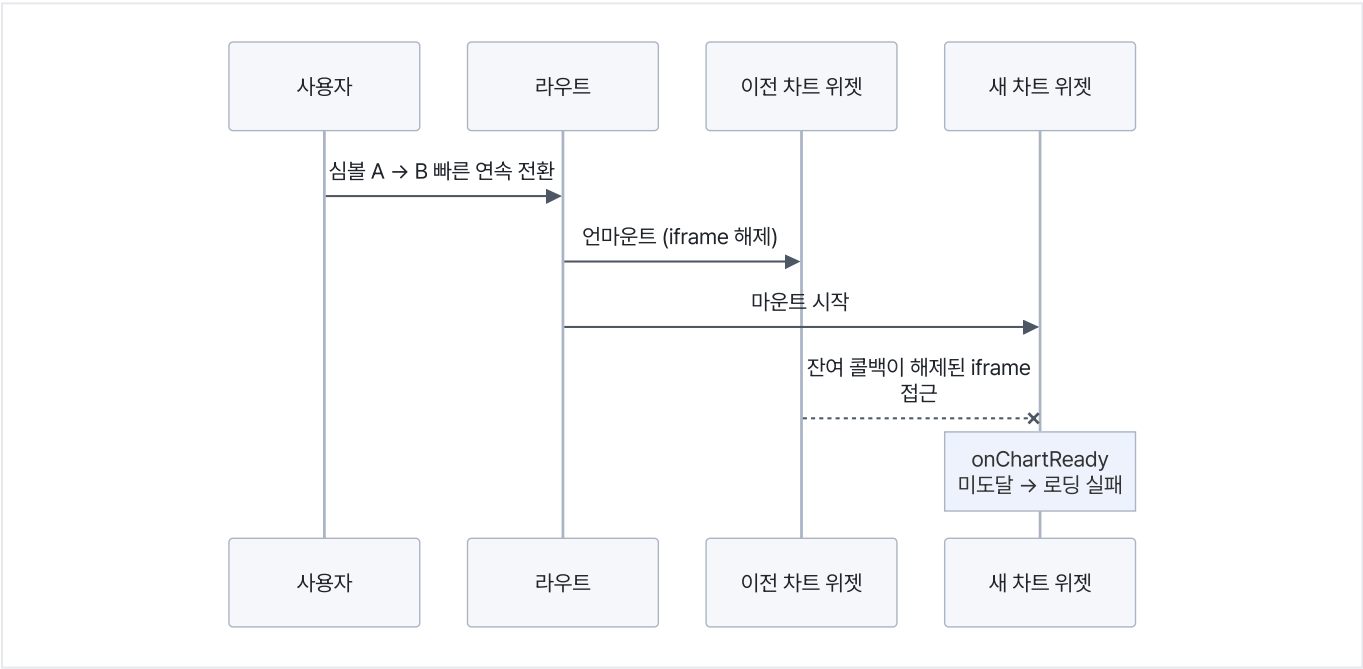
실시간 구독을 값을 표시하는 컴포넌트로 옮기고, 티커 데이터도 항목별로 구독하도록 분리했습니다.
보이지 않는 오더북과 닫힌 드로어는 구독을 중지하고, 체결 내역 행에는 memo를 적용해 갱신 범위를 줄였습니다.



실시간 데이터가 프론트엔드에 전달되는 흐름

차트 로딩 실패의 원인 분석과 해결

간헐적인 차트 로딩 실패를 클라이언트 오류 로그와 서버·CDN 응답 시간으로 추적했습니다.
심볼을 빠르게 바꾸면 이전 위젯의 콜백이 이미 해제된 iframe에 접근해 새 차트가 초기화되지 않는 상황을 재현했습니다.
라우트가 바뀔 때 페이지 전체가 다시 마운트되는 구조가 원인이었습니다.
차트 뷰를 레이아웃에 두어 불필요한 재마운트를 막고, 위젯 초기화 중 심볼이 바뀌어도 최신 심볼을 적용하도록 수정했습니다.

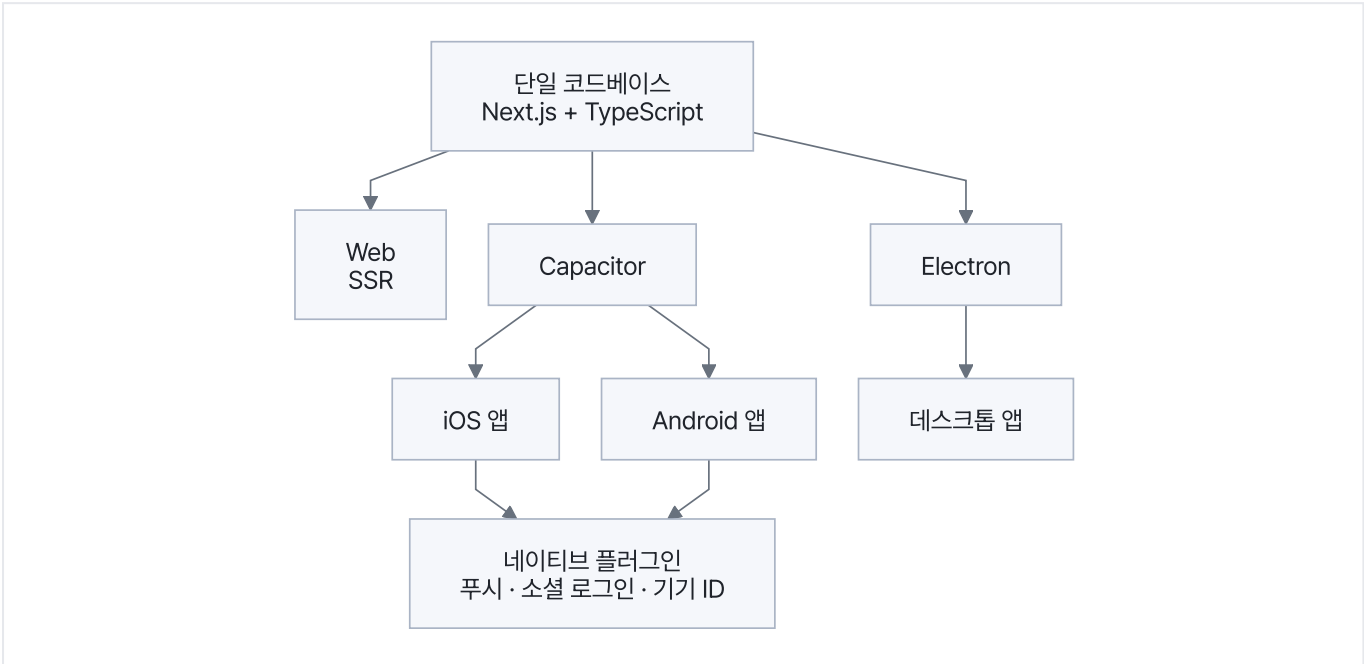


수정 전: 빠른 심볼 전환에서 발생한 위젯 생명주기 충돌

단일 코드베이스의 멀티 플랫폼 대응

Next.js·TypeScript 코드베이스를 웹, Capacitor 기반 iOS·Android 앱, Electron 데스크톱 앱으로 빌드했습니다.
모바일 앱을 스토어에 출시하고 푸시(APNs·FCM)·소셜 로그인·기기 식별 기능을 연결했습니다.

앱 강제 업데이트 정책을 연동하고, Electron 앱의 API 요청을 식별하는 헤더를 추가했습니다.



웹·모바일·데스크톱 빌드와 네이티브 기능 연결 구조

다운소프트랩 · 업무 프로젝트

Gobit

코인 정보·커뮤니티 플랫폼

2024.10 – 2025.12 | 프론트엔드·백엔드 개발 담당 · 2인 팀

암호화폐 시세·차트·뉴스·커뮤니티와 모의투자 기능을 제공하는 코인 정보 플랫폼입니다.
2인 팀에서 Next.js 웹 클라이언트를 새로 구축하고 Node.js·Express API를 개발했습니다.
Binance 시세를 수집해 Redis에 캐시하고 Socket.IO로 전달했으며, 회원·게시글 기능과 운영 도구를 구현했습니다.

주요 업무

웹 클라이언트	Next.js 15·React 19로 마켓·코인 상세 등 주요 화면을 구현하고 TradingView 차트와 tiptap 에디터를 연동했습니다.
실시간 데이터	시세는 Socket.IO로, 나머지 조회 데이터는 SWR로 관리했습니다. Binance 시세를 Redis에 캐시하고 소켓 재연결·헬스체크 기능을 개선했습니다.
서비스 기능	회원 인증·출석·레퍼럴·게시글·댓글·북마크 기능과 관련 API 개발
운영 개선	SSR·이미지·폰트를 최적화하고 SEO와 코인 상세 페이지 LCP를 개선했습니다. coinIns에서 Gobit으로 리브랜딩하고 점검 모드·회원 탈퇴 기능을 구현했습니다.

결과

웹 클라이언트 전 페이지와 API를 구축하고 coinIns에서 Gobit으로 리브랜딩했습니다.
2025년 12월 주요 기능 개발을 마친 뒤 유지보수했습니다.

기술 TypeScript · Next.js 15 · React 19 · SWR · Zustand · Radix UI · TradingView · tiptap · Node.js · Express · Prisma · Redis · Socket.IO

레거시 플랫폼 통합과 Kubernetes 이전

2026.07 - 2026.08 · 이전 후 유지보수

Gobit·모의투자 시스템·코인 매매 시그널 서비스가 Windows·Linux 서버 5대에 분산돼 있었습니다.

핵심 시세 수집기는 소스 코드 없이 .NET 바이너리만 남아 있었습니다.

유지보수와 배포를 이어갈 수 있도록 서비스를 모노레포로 통합하고 Kubernetes 환경으로 이전했습니다.

- 기존 바이너리의 동작과 데이터 형식을 분석해 수집기를 Node.js로 재작성했습니다.
실시간 시세·캔들·과거 데이터 수집 모드를 구현하고, 연결 분산 처리와 요청 속도 제한으로 WebSocket 연결 끊김에 대응했습니다.
- 중계 계층을 정리해 데이터 흐름을 수집기 → Redis → 서버로 단순화했습니다.
- 13개 서비스의 Dockerfile과 Kubernetes 매니페스트를 작성하고, Ingress 설정 및 DB 백업·캔들 보관 CronJob을 구성했습니다.
- GitHub Actions 빌드·배포를 자동화하고, 설정값을 Kubernetes Secret에서 불러오도록 변경했습니다.

결과

서버 5대에 분산된 서비스를 모노레포로 통합하고 Kubernetes 환경으로 이전했습니다.

기존 바이너리와 결과를 대조해 새 수집기를 검증하고, 13개 서비스의 빌드·배포를 자동화했습니다.

[웹 클라이언트 코드](#) ↗ [백엔드 코드](#) ↗

두잇 · 업무 프로젝트

doit-homepage

회사 홈페이지 React 마이그레이션

2024.05 - 2024.07 | 프론트엔드 개발 담당

jQuery 기반 회사 홈페이지를 React·TypeScript·Vite SPA로 재작성해 한국어·영어와 반응형 화면을 지원했습니다.

주요 업무

마이그레이션	메인·회사소개·서비스·인증·파트너 5개 페이지를 컴포넌트 구조로 재작성
UI 재사용	사내 UI 라이브러리 doit-uikit을 적용해 색상·타이포·레이아웃 등 공통 요소 재사용
다국어	i18next로 한국어·영어를 지원하고 번역 리소스를 페이지별 네임스페이스로 분리
반응형	CSS와 JavaScript가 같은 브레이크포인트를 사용하도록 구성하고 화면 크기 변화는 CSS로 처리했습니다. 모바일에서 불필요한 영역은 렌더링하지 않도록 했습니다.

결과

주요 페이지를 React로 전환하고 한국어·영어 및 반응형 화면을 구현했습니다.

공통 라이브러리를 적용하고 사용 중 발견한 개선점을 라이브러리에 반영했습니다.

기술 TypeScript · React 18 · Vite · styled-components · doit-uikit · i18next · react-responsive · Zustand · React Router 6 · Kakao Maps SDK

doit-uikit

사내 공통 UI 라이브러리

2024.02 – 2024.07 | 프론트엔드 개발 담당

프로젝트마다 반복 구현하던 UI를 공통 라이브러리로 만들고, React 개발 규칙을 팀에 공유했습니다.

주요 업무

컴포넌트	Atomic Design 기반 UI 컴포넌트 24종 개발: 기본 컴포넌트 12종, 차트 5종을 포함한 복합 컴포넌트 12종
패키징	React-React DOM을 peer dependency로 분리하고 vite-plugin-dts로 타입 선언을 자동 생성해 npm 배포
문서화	Storybook·Chromatic으로 사용법·Props·상태별 UI 문서화
팀 표준	디렉토리 구조·디자인 패턴·라이브러리 선정 기준 정립, ESLint·Prettier 설정 및 팀 온보딩 교육

결과

D.A.X와 회사 홈페이지에서 사용하며 요구사항에 맞춰 라이브러리를 개선하고 배포했습니다.
컴포넌트·사용 문서·런터 설정을 함께 제공해 공통 UI와 개발 규칙을 적용했습니다.

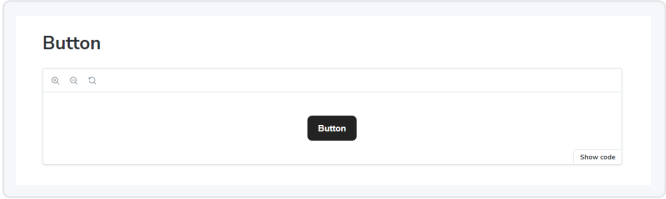
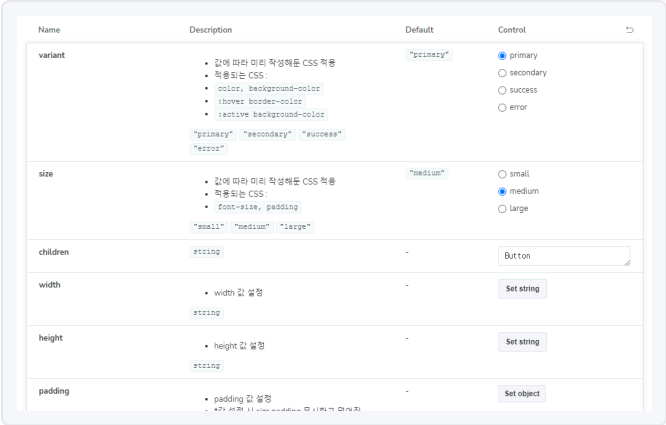
기술 TypeScript · React 18 · Vite · styled-components · Storybook 8 · Chromatic · vite-plugin-dts

구현 구조와 설계

컴포넌트 조합과 라이브러리 배포

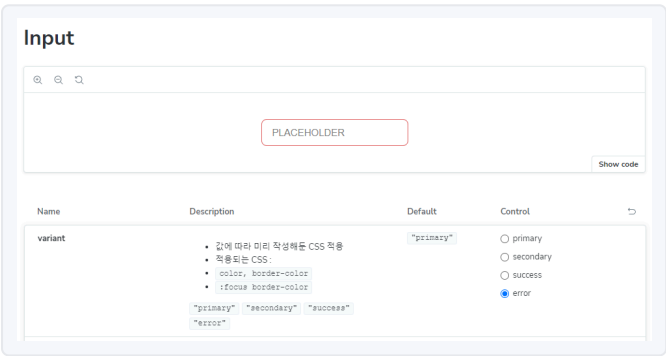
Button·Input 같은 Atom을 조합해 FileInput 등의 Molecule을 구성했습니다.
Storybook에서 Props·상태·사용 예를 확인하고 Playground로 직접 값을 바꿔볼 수 있도록 문서화했습니다.
vite-plugin-dts로 타입 선언을 함께 배포하고 React-React DOM은 peer dependency로 분리했습니다.
D.A.X와 회사 홈페이지에 적용하면서 발견한 요구사항을 라이브러리에 반영했습니다.

doit-uikit · 프로젝트 화면

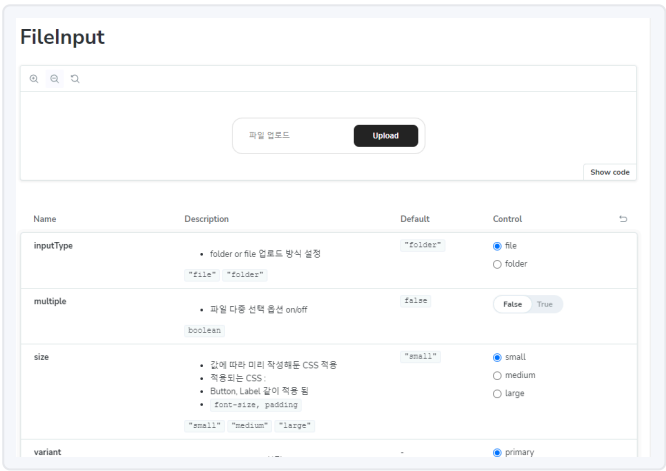


Button 컴포넌트의 Storybook 예시

doit-uikit 컴포넌트의 Storybook 문서와 Props 설정 화면



Input 컴포넌트의 상태-Props 문서



Input-Button-Label을 조합한 FileInput 컴포넌트

[npm 패키지](#)

두잇 · 업무 프로젝트

D.A.X

AI 모델 학습·운영 플랫폼

2024.01 – 2024.06 | 프론트엔드 개발 담당 · 백엔드 1명과 협업

데이터 수집·라벨링부터 모델 학습·추론까지 지원하는 사내 플랫폼입니다.
백엔드 개발자 1명과 협업해 프론트엔드 전 화면을 개발했습니다.

주요 업무

- 플랫폼 화면 프로젝트·대시보드·클래스·컬렉션·라벨링·모델 학습·추론 화면 구현
- 데이터 도구 캔버스 기반 바운딩 박스 라벨링, 이미지·폴더 업로드, 컬렉션 ZIP 다운로드 기능 개발
- 공통 구조 검색 조건·페이지네이션·로딩 상태 처리, 사내 표준 구조와 doit-uikit 적용

라벨링 이미지는 화면 크기에 맞춰 축소되지만 학습용 좌표는 원본 해상도로 저장해야 했습니다.
원본↔화면 좌표의 양방향 변환을 전용 축으로 분리해 그리기·저장·재표시가 같은 규칙을 사용하도록 구현했습니다.

기술 TypeScript · React 18 · TanStack Query 5 · Zustand · React Router 6 · Axios · doit-uikit

구현 구조와 설계

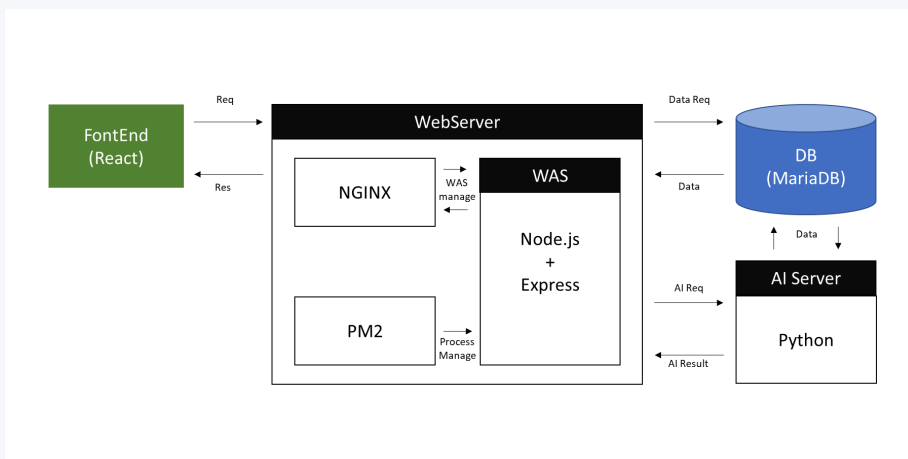
서비스 구성과 프론트엔드 담당 범위

React 프론트엔드는 웹 서버 API를 통해 프로젝트·라벨·모델 데이터를 조회하고 변경합니다.

백엔드와 AI 서버가 연결된 전체 구성에서 프론트엔드 화면과 API 연동을 담당했습니다.

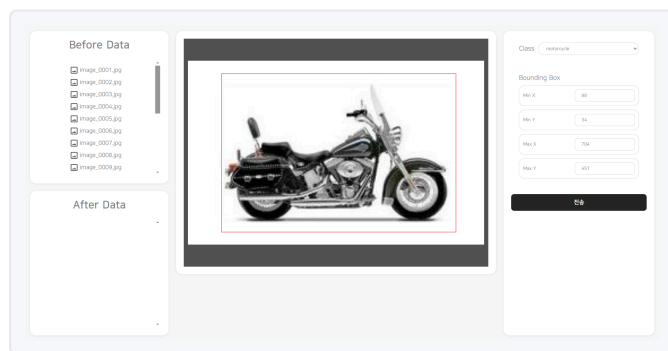
사내 표준 디렉토리 구조를 적용하고 API 호출을 `auth·project·class·datamanage·aiModel·operation` 등 도메인별 서비스 모듈로 나눴습니다.

원본·렌더 좌표 변환은 `useCoordinateTransform` 혹으로 분리해 그리기·저장·재표시에 같은 규칙을 적용했습니다.

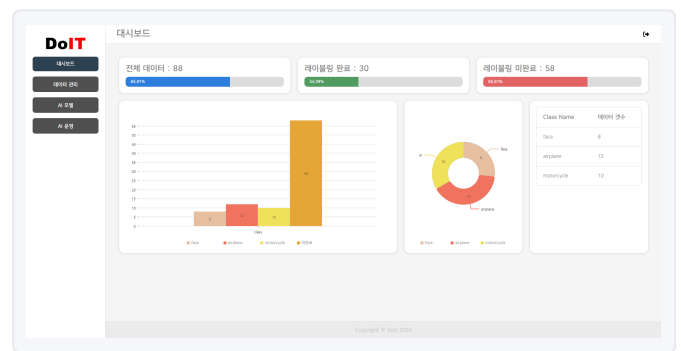


D.A.X 전체 서비스 구성 — 담당 범위는 React 프론트엔드와 API 연동

D.A.X · 프로젝트 화면



D.A.X에서 이미지 위에 바운딩 박스를 그려 라벨링하는 화면



D.A.X의 프로젝트 데이터와 학습 현황 대시보드

XRGIS BackOffice

XR·GIS 원격 협업 시스템 백오피스

2023.08 – 2023.12 | 프론트엔드·백엔드 개발 담당

가상·증강·혼합현실(XR)과 지도·공간정보(GIS)를 결합한 원격 협업 서비스입니다.

사용자·팀 관리와 OBJ 모델·영상 파일 처리를 위한 백오피스를 개발했습니다.

React 화면과 Express API를 구현하고 GridFS 기반 파일 저장 및 MariaDB 기반 세션 인증을 연동했습니다.

주요 업무

풀스택 개발 React 기반 사용자·팀·데이터 관리 화면과 Express API 개발

파일 처리 MongoDB GridFS 기반 대용량 파일 업로드, archiver 기반 ZIP 일괄 다운로드 구현

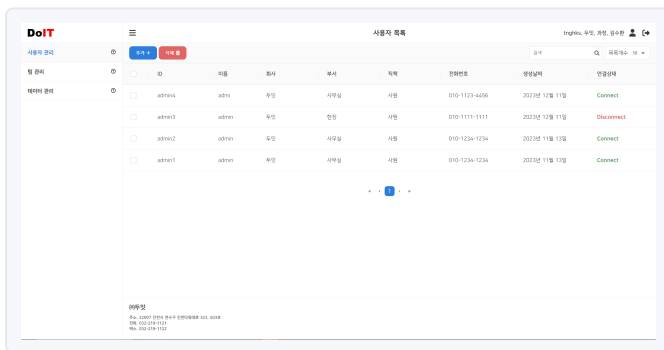
인증·권한 사내 공통 관리자 계정의 세션 인증과 팀 소속에 따른 파일 다운로드 권한 적용

상태·피드백 서버 상태는 React Query, UI 상태는 Recoil로 분리하고 API 결과를 Toast로 안내

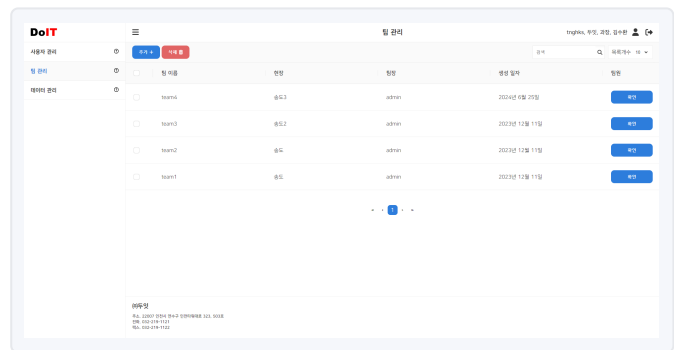
MongoDB에 파일을 저장해야 하는 요구사항에 맞춰, 일반 문서의 16MB 제한에 걸리는 OBJ·영상 등을 GridFS의 청크 분할 저장으로 처리했습니다.
같은 관리자 계정을 사용하는 사내 서비스와는 MySQL 기반 세션 저장소를 통해 인증 체계를 맞췄습니다.

기술 TypeScript · React 18 · React Query · Recoil · styled-components · Express · MongoDB(GridFS) · MariaDB

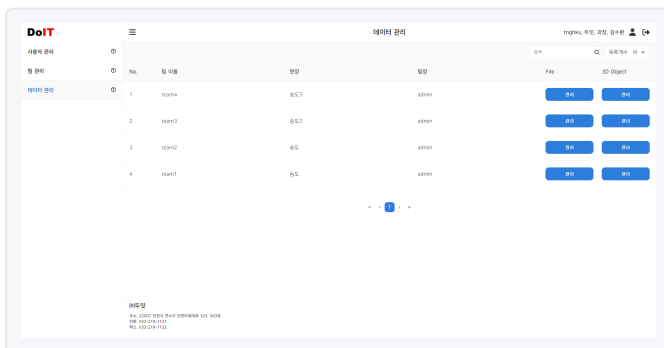
XRGIS BackOffice · 프로젝트 화면



XRGIS 사용자 관리 화면



XRGIS 팀 관리 화면



XRGIS 팀별 데이터·파일 관리 화면