전하성, Frontend Developer

Introduction

끊임없이 배움을 추구하며 **“어제보다 오늘 더 나은 개발자”**가 되고자 하는 4년차 프론트엔드 개발자 전하성입니다.

대학에 기술경영으로 입학했으나, 우연히 수강하게 된 객체지향 프로그래밍 수업에서 문제를 논리적으로 분석하고 코드로 구현하는 과정에 큰 성취감과 즐거움을 느껴 컴퓨터공학을 복수전공으로 선택하게 됐습니다. 본격적으로 개발자의 길에 들어선 이후 React, Next.js, TypeScript 기반으로 다수의 웹 서비스를 구현하고 배포하며 실무 경험을 쌓아왔습니다.

WorkOnward에서는 **지도 기반 채용 검색 서비스**의 핵심 기능을 주도적으로 개발하여 **MAU 20% 증가**를 달성했으며, **GraphQL과 MongoDB 쿼리 최적화**로 쿼리 속도를 **30~50%** 개선했습니다. 또한 Next.js 12에서 14로의 점진적 마이그레이션을 설계하고 주도하여 서비스 중단 없이 최신 아키텍처로 전환했습니다.

현재는 Aicy에서 **AI 기반 재무제표 분석 챗봇**을 개발하여 SSE 기반 스트리밍으로 즉각적인 응답 경험을 제공했고, **빌드 사이즈 최적화**로 전체 번들 사이즈를 **65% 감소**시켜 로드 속도를 대폭 개선했습니다.

저는 **"작동하는 코드"**와 **"좋은 코드"**의 차이를 고민합니다. 당장의 기능 구현도 중요하지만, 사용자 경험을 해치는 요소는 없는지, 이 코드를 6개월 후에도 쉽게 이해하고 수정할 수 있을지를 함께 생각하며 개발합니다.

Skills

- Frontend: **React.js, Next.js, Typescript**
- State Management: **React Query, Zustand, MobX**
- Styling: **Tailwind CSS, Ant Design, Mantine**

Experience

Aicy

2025.06 - 재직중 | 풀스택 개발자 | Seoul, South Korea

빌드 사이즈 최적화 및 로딩 성능 개선

프로젝트 설명: Next.js 기반 웹앱의 초기 번들 사이즈 및 로드 속도를 최적화하기 위한 빌드 구조 개선 프로젝트. Tree-shaking 버그를 해결하고 코드 스플리팅을 통해 초기 로딩 성능 개선

기술 스택: Next.js, Lighthouse, Bundle Analyzer, Webpack

핵심 역할

- Next.js 빌드 구조 분석 및 번들 사이즈 최적화 전략 설계
- 코드 스플리팅과 Lazy Loading 전략을 통한 번들 경량화

- 리소스 최적화를 통한 로드 타임 개선 전략 수립

이슈

- 빌드 분석 결과 최대 **1.1MB** 페이지 발견, 간단한 login 페이지도 **500KB** 초과
- Tree-shaking 미작동, 대용량 라이브러리 무분별한 포함, 폰트/이미지 최적화 미적용으로 페이지 로드 속도 저하

해결

- **Bundle Analyzer**로 병목 구간을 식별하고 **Webpack sideEffects** 설정과 import 구조 개선을 통해 Tree-shaking이 정상 작동하도록 수정
- **next/dynamic**을 활용한 코드 스플리팅 전략을 도입해 echarts 등 대용량 라이브러리와 모달/팝업 등 초기 렌더링에 불필요한 컴포넌트를 동적 로드하여 첫 로드 번들 사이즈 최소화
- **next/font**를 사용하여 로컬 및 웹 폰트의 preload를 적용하고 CLS 문제를 해결
- **next/image**를 통해 이미지 리사이징 및 WebP 포맷 변환을 자동화하여 **LCP 성능**을 개선

성과

- 전체 번들 사이즈 **65% 감소**
- 최대 초기 번들 크기를 **930KB → 250KB로 감소**
- Lighthouse Performance 점수 **12점 증가 (68 → 80)**
- **LCP 3초 단축**

AI 재무제표 분석 챗봇 서비스

프로젝트 설명: Gemini API 기반으로 사용자가 업로드하는 다양한 포맷의 재무제표를 분석해 대화형으로 제공하는 웹 챗봇 서비스

기술 스택: Next.js, Typescript, React, NestJS, Gemini API

핵심 역할

- 대용량 PDF 파싱을 위한 비동기 처리 아키텍처 구축 및 안정성 확보
- Gemini API 연동을 통한 AI 응답 스트리밍 구조 설계
- AI 스트리밍 중 즉각적 피드백 및 직관적인 채팅 인터페이스 개발

이슈

- 100장 이상의 대용량 PDF 파싱 시 **proxy timeout** 발생으로 업로드 실패율 증가
- 실시간 채팅을 위한 스트리밍 구조 설계 및 구현 전략 부재
- 여러 페이지에서 채팅 로직 중복 구현으로 유지보수성 저하

해결

- **BullMQ** 기반 **비동기 작업 큐**를 도입하여 파싱 작업을 **백그라운드 잡으로 분리**하고 API 아키텍처를 동기식에서 작업 상태 **Polling 패턴**으로 변경하여 proxy timeout 문제를 해결

- NestJS의 **SSE 엔드포인트**를 구현하여 Gemini API 응답을 토큰 단위로 실시간 스트리밍하고 첫 응답 대기 시간을 단축
- SSE 연결과, 메시지 상태 관리 및 에러 핸들링 로직을 **React Custom Hook으로 캡슐화**하여 관심사 분리를 통해 재사용성을 확보
- **React.memo**와 **리스트 가상화**를 적용하여 불필요한 리렌더링 방지 및 대화 히스토리 증가 시 렌더링 성능 최적화

성과

- BullMQ 기반 비동기 파싱 및 자동 재시도로 대용량 PDF 업로드 **성공률 90%** 이상 확보
- SSE 기반 스트리밍으로 첫 응답 대기 시간 평균 **10초 → 2초로 단축**
- 공통 채팅 훅으로 3개 이상 페이지에서 일관된 경험 제공 및 유지보수성 향상

WorkOnward

2022.05 - 2025.03 | 풀스택 개발자 | Remote, US

렌더링 성능 최적화

프로젝트 설명: 복잡한 UI 구조와 자주 업데이트되는 컴포넌트에서 발생하는 렌더링 성능 문제를 분석하고 최적화

기술 스택: React DevTools Profiler, React.memo, useMemo, useCallback, Virtualized List

핵심 역할

- React 렌더링 성능 분석 및 병목 현상 개선
- 메모이제이션 적용을 통한 불필요한 리렌더링 방지
- 대용량 리스트 가상 스크롤 구현을 통한 성능 최적화

이슈

- 복잡한 UI 구조에서 불필요한 리렌더링이 빈번하게 발생하여 전반적인 반응 속도 저하 및 렌더링 병목 초래
- 대용량 리스트 렌더링 시 과도한 DOM 노드 생성으로 인한 초기 렌더링 지연

해결

- **React DevTools Profiler**로 컴포넌트별 렌더링 시간 및 리렌더링의 빈도를 측정 하여 병목 컴포넌트 식별
- **Component Composition Pattern**을 사용하여 props drilling 제거로 리렌더링 최소화
- **상태 하양 이동** 패턴을 사용하여 상태 변경이 잦은 로직을 컴포넌트로 분리하고 **React.memo**로 불필요한 하위 컴포넌트 리렌더링 차단
- **useMemo**와 **useCallback**으로 props의 참조 동일성을 유지하여 React.memo의 얇은 비교 최적화 유지
- 큰 리스트 컴포넌트에 **react-virtuoso**를 도입하여 **viewport** 내 아이템만 렌더링하는 가상 스크롤 구현

성과

- React.memo/useMemo/useCallback 적용으로 컴포넌트 총 렌더링 시간 **57% 절감 (2530ms → 1090ms)**, 리렌더링 횟수 **20% 감소 (640회 → 510회)**
- react-virtuoso를 통한 가상 스크롤로 리스트 렌더링 시간 **96% 단축 (272ms → 11ms)**

Next.js 12 → 14 마이그레이션 및 App Router 전환

프로젝트 설명: Next.js 12에서 14로 점진적 마이그레이션을 설계 및 주도하여 App Router 및 서버 컴포넌트 도입

기술 스택: Next.js, App Router, Server Components, React 18

핵심 역할

- Next.js 12에서 14로의 점진적 마이그레이션 전략 설계 및 실행
- Pages Router와 App Router의 공존 구조 설계
- React Server Component 도입 및 데이터 패칭 구조 재설계

이슈

- 추후 Next.js 12의 지원 중단 예상 및 최신 기능(**서버 컴포넌트, Streaming, Metadata API**) 활용 불가
- Pages Router의 페이지 단위 데이터 로딩으로 **props drilling** 및 코드 복잡성 증가

해결

- **Pages Router**와 **App Router**가 공존하는 구조를 설계하여 **점진적 마이그레이션** 기반 구축
- API 호출 및 정적 컨텐츠는 서버 컴포넌트로, 상태 및 이벤트 핸들러 등 상호작용이 필요한 로직은 클라이언트 컴포넌트로 **관심사 분리** 달성
- 서버 컴포넌트 내에 데이터 패칭 로직을 배치하여 getServerSideProps의 **중앙화된 데이터 패칭 구조를 분산**시키고 props drilling 문제 해결
- **Parallel Routes**와 **Streaming**을 도입하여 컴포넌트 단위의 **독립적인 데이터 요청, 로딩 UI, 에러 바운더리** 구현
- Next.js 14의 주요 변경사항 (next/image, next/link, Metadata API 등)을 문서화하고 **마이그레이션 가이드**를 팀원에게 제공하여 팀의 **빠른 적응과 코드 일관성 확보**

성과

- 서비스 중단 없이 전체 페이지 **Next.js 14 App Router**로 전환 완료
- 서버 사이드 렌더링 기반 데이터 패칭으로 props drilling 제거 및 코드 유지보수성 향상
- **App Router** 전환으로 향후 Next.js 버전 업그레이드 용이성과 빠른 신규 기능 적용 가능성 확보

지도 기반 채용 검색 서비스

프로젝트 설명: 하이퍼로컬 채용 플랫폼의 지도 UI 및 채용 데이터 렌더링 구조를 프론트엔드에서 주도적으로 개발. GraphQL 기반 데이터 패칭 구조를 최적화하고, 실시간 인터랙션 UX를 강화

기술 스택: Next.js, GraphQL, MongoDB

핵심 역할

- 지도 기반 실시간 채용 검색 UI 설계 및 구현
- GraphQL과 MongoDB 기반 데이터 패칭 최적화
- URL 기반 모달 시스템으로 지도 컨텍스트 유지 및 지도 탐색 경험 개선

이슈

- 지도 이동 시 과도한 API 호출, 대량 마커 리렌더링, 중복 쿼리로 인한 성능 저하
- 채용 공고 상세 페이지 전환시 기존 지도 컨텍스트가 사라져 지역 탐색 경험 단절

해결

- **debounce** 기법을 적용하여 viewport 변경 이벤트가 완료된 시점에만 리패칭하여 과도한 네트워크 호출 제어
- **GraphQL** 쿼리의 요청 필드를 마커에 필요한 최소한의 필드만 선택적으로 요청하여 데이터 **오버패칭 방지**
- 대량의 마커에 **클러스터링 기법**을 적용하여 **DOM 생성을 최소화**하고 클러스터링 연산 결과를 메모이제이션하여 **불필요한 리렌더링 방지**
- **MongoDB aggregation pipeline과 facets** 오퍼레이터를 활용해 페이지네이션 데이터와 전체 카운트 조회를 단일 쿼리로 통합하여 DB 라운드트립 최소화
- **Next.js Intercepting/Parallel Routes**를 활용한 URL 기반 채용 공고 상세 모달 구현

성과

- 지도 이동 시 API 응답 속도 **1-3초 내 유지** 및 대량 마커 렌더링 성능 개선
- 페이지네이션 쿼리 시간 **30-50% 단축**
- URL 기반 모달로 공고 **딥링크 공유 가능 및 SEO 최적화 달성**
- 지도 검색 도입으로 **MAU 20% 증가**, 일일 사용자의 **50%**가 활용하는 핵심 기능으로 성장

협업 및 개발 프로세스 개선

이슈

- 디자이너/기획자, 개발팀, QA 팀으로 구성된 개발 프로세스에서, 개발 완료 후 QA 단계에서 기획자가 고려하지 못한 엣지 케이스와 예외 상황이 지속적으로 발견됨
- QA에서 발견된 이슈로 인한 재작업이 반복되어 출시 일정 지연 및 개발 리소스 낭비

해결

- 개발 착수 전 디자이너/기획자, 개발자, QA가 참여하는 **리뷰 세션을 도입**하여 유저 플로우와 **예외 상황을 사전**에 논의
- 작업 단계를 여러 **마일스톤**으로 나누고, 각 마일스톤마다 간단한 리뷰 세션을 통해 구현된 기능 검증 및 **예외 케이스 조기 발견**

성과

- 마지막 QA 단계에서 발생하는 **추가적인 예외 상황 대폭 감소**

- 엣지 케이스 사전 발견으로 급한 분기 처리 및 예외 코드가 최소화되어 **코드 품질 및 유지보수성이 향상**

Education

Stony Brook University

2022.05 졸업 | 컴퓨터공학 · 기술경영

Contact

LinkedIn: linkedin.com/in/hasungjun

GitHub: github.com/iianjun

Portfolio: hasungjun.com