민준홍

010-2317-3750 @pathas@naver.com

안녕하세요. 성장형 프론트엔드 개발자 민준홍입니다.
Javascript, Typescript, React.js 기반의 개발을 주로 하고 있으며 어제보다 나은 개발자가 되기 위해 노력중입니다.

경력 4년 6개월

딥노이드(Deepnoid)

2025.01 - 재직 중 (7개월) | 정규직 | 프런트엔드 개발 | 주임

B/O 서비스 개발

2025.05 - 2025.07

[Client] react with vite
[Tools] @tanstack/react-query, msw, zod, husky
[Architecture] FSD

기존에는 프로젝트 구조로 atomic 패턴을 주로 사용했었는데 프로젝트가 커지다 보면 컴포넌트 구조가 너무 깊어지거나 여러 명이 동시에 작업하는 경우 작업 영역이 겹치거나 하는 등의 문제가 발생하곤 했습니다. 그래서 새로운 구조를 찾던 중 FSD 를 알게 되었는데 프로젝트의 단위를 기능별로 나누고 우선 기능 개발을 한 뒤에 재사용 가능한 부분을 shared 나 widget 등의 영역으로 분리하는 개발 방식이 매력적으로 다가와 도입하게 되었습니다. 실제로 사용을 해보니 프로젝트의 레이어로 이루어진 위계 구조 자체가 주는 설계의 편리성도 있었고, 작업 영역이 겹칠 일도 거의 없었고, 재사용 가능한 컴포넌트나 기능들을 기능 개발 후에 분리하다 보니 개발 편의성도 높아진 것 같았습니다. 다만, 프로젝트 규모가 작은 경우에는 관리 측면에서 오버헤드가 있을 수도 있겠다는 생각이 들었습니다.

커밋 메세지 컨벤션을 정하긴 했지만 각자 사용법이 조금씩 다르고 린트나 포맷팅을 하지 않고 코드를 저장소에 올리거나 하는 경우가 잦아서 husky를 도입하게 되었습니다. husky 를 사용해서 pre-commit 단계에서 린팅 및 포맷팅을 진행하고, commit-msg 단계에서 커밋 메세지를 검증하고 pre-push 단계에서 타입 체크 및 빌드를 진행하므로써 저장소의 코드는 항상 온전한 상태를 유지할수 있게 되었습니다. 매 커밋마다 프로젝트 전체를 린팅하거나 포맷팅하기에는 부담이 돼서 그 부분은 lint-staged 를 통해 staged 된 파일들에만 적용할 수 있게 했고, 커밋 시에는 git-cz 를 이용해서 일련의 프롬프트를 따라 구조화된 커밋 메세지를 작성할 수 있게 했습니다.

zod는 개인적으로만 써보고 프로젝트에는 처음 적용해봤는데 form 데이터 검증을 선언적으로 할 수 있으면서 유틸 함수들을 따로 작성하지 않아도 되는 점들이 유용했습니다. @tanstack/react-form 라이브러리의 유효성 검사 스키마로도 사용할 수 있어서 적용하기도 편리했습니다. 이전에 ajv 로 서버 데이터를 스키마로 검증해본 적이 있었는데 서버 데이터 검증이 필요한 경우에도 사용해 볼 수 있을 것 같습니다.

백엔드 API 가 프런트엔드 작업 후에 나오는 일정이 잡혀서 msw 를 도입하게 되었습니다. mocking 함수를 두거나 샘플 데이터를 활용할 수도 있지만 실제 API 를 호출하는 것과 동일하게 작업할 수 있는 점이 큰 장점으로 작용했습니다. 물론 API 의 controller 부분을 거의 같은 정도로 작성해야 해서 작업량이 늘어나긴 하지만 프런트엔드 개발이 먼저 이루어진다면 충분히 사용 가

치는 있다고 생각합니다. 추가로 퍼블리셔가 같이 작업을 하는데 개발 서버가 아직 갖춰지지 않아 로컬 서버로 API 테스트를 해야 하는 상황에서 API 테스트 시에는 MSW를 껐다가, 테스트 후에는 다시 인터페이스를 맞춘 MSW를 켜는 식으로 코드를 관리해서 퍼블리셔 측에도 영향이 가지 않게 작업을 할 수 있었습니다.

이전까지의 프로젝트들에서는 프로젝트 이슈를 관리하지 않고 있었어서 gitlab 의 마일스톤과 이슈, 라벨을 활용해서 이슈 관리를 시작했습니다. 마일스톤은 릴리즈 버전으로 정하고, 이슈는 해당 마일스톤에 포함되는 기능들에 대응하도록 했고, scoped 라벨을 활용해서 해당 이슈가 기능 개발인지 버그 픽스인지, 어느 작업 영역에 관여하는지, 어떤 상태인지를 한 눈에 파악할 수 있게 했습니다.

dicom viewer 사내 라이브러리 개발

2025.02 - 2025.05

[Tools] @cornerstonejs, tsup, verdaccio, storybook

기존에 의료용 dicom 파일을 웹 화면에 렌더링 하기 위해서 OHIF 뷰어 코드를 클론해서 사용 중이었는데 프로젝트가 그 자체만으로도 크고 코드도 수정하기가 쉽지 않은 구조로 되어 있어서 사내 자체 라이브러리를 개발하게 되었습니다.

의료 도메인에 대한 지식이 전무한 상태였어서 dicom 파일 포맷이나 orthanc 서버에 대한 기초적인 학습을 병행하면서 @cornerstonejs 공식 문서와 소스 코드를 분석했습니다. 전체 OHIF 뷰어 기능 중 stack, volume 뷰포트를 활용할 수 있는 뷰어, 툴바, 툴 히스토리, 시리즈 조회 등의 기능을 컴포넌트 및 HOC 로 개발해서 라이브러리로 활용할 수 있게 했습니다.

개발 후에는 전체 기능을 한 눈에 볼 수 있게 storybook 에 정리해서 올려두었습니다. 스토리에서 dicom 이미지 렌더링 테스트를 바로 할 수도 있었고 스토리 파일에서 전체 기능에 필요한 코드를 모두 볼 수도 있어서 타 프로젝트에서 라이브러리를 사용하는 데에도 무리가 없게 했습니다.

개발한 라이브러리는 사내 서버 내 verdaccio 에서 관리할 수 있게 해두어 다른 프로젝트들에서 편하게 설치해서 사용할 수 있게 했습니다.

프런트엔드 개발 컨벤션 정의

2025.01 - 2025.02

[Tools] eslint, prettier, typescript

온보딩 기간에 프로젝트들을 인계 받다 보니 프로젝트 간 팀 공통 설정이나 컨벤션이 정의되어 있지 않았습니다.
그래서 가장 먼저 균일한 코드 품질을 위한 도구들을 도입했습니다.
eslint, prettier 를 먼저 도입했는데 기존 코드들과 최대한 충돌이 나지 않는 범위 내에서 설정을 진행했습니다.
typescript 의 경우 설정은 되어 있었으나 타입을 거의 활용하지 않는 상태였어서 팀 회의를 통해 컨벤션을 정하고 타입을 이전보다 적극적으로 사용할 수 있게 했습니다.

로지올(LOGiALL)

2021.01 - 2024.04 (3년 4개월) | 정규직

주문관리 웹앱

2022.02 - 2022.12

[Client] React.js with Typescript

프로젝트 신규 개발 및 유지보수, 성능 개선

여러가지 시도를 해본 프로젝트였는데 우선 사내 최초로 프로젝트에 yarn berry 를 도입했습니다.
npm 의 여러 가지 단점(node_modules 용량, 모듈 검색 속도, 유령 의존성 등)에 불만을 느끼던 차에
yarn berry 에 대해 알게 되었고 pnp 모듈 관리 방식의 장점
(모듈 용량 감소, 모듈 검색 속도 개선, 유령 의존성 제거, zero-install 등) 이 너무 매력적으로 느껴져서
yarn berry를 도입하게 되었습니다.

또 프로젝트 생성 시에는 모듈 번들러로 webpack 을 사용했지만
vite 의 esbuild 를 사용한 신속한 HMR 과 간단한 설정, 빠른 빌드 속도가 매력적으로 느껴져 교체 테스트 진행 후
webpack 설정을 모두 걷어내고 vite 로 번들러를 대체했습니다.
빌드 속도는 이미 빠른 편이었던 터라 교체 후 빌드 속도에서 큰 차이를 보이지는 않았지만
개발 환경에서 HMR 의 속도가 크게 향상된 점은 체감할 수 있었습니다.

리액트 프로젝트이다 보니 리렌더링을 최소화하기 위해
중간중간 지역으로 옮길 수 있는 상태는 모두 전역에서 지역으로 옮기고
코드 스플리팅을 적용해서 자주 사용하지 않거나 처음에 사용자가 볼 필요가 없는 페이지는
초기 로딩에 포함되지 않도록 했습니다.
또 크롬 lighthouse 를 이용해서 중간중간 성능 점검을 하여 크롬의 성능 권장사항을 충족시키려고 노력했습니다.

마지막으로 api 응답 실제 데이터와 프런트에서 정의한 타입 간 발생하는 차이를 즉각적으로 확인하기 위해 ajv 와 typescript-json-schema 를 도입했습니다.
api 응답을 사전에 정의한 json schema 로 검증해서 실제 응답 데이터에 대한 신뢰도를 높였습니다.
실제로 사전 통보 없이 api 응답 데이터가 변경된 적이 몇 번 있었는데 그 때마다 즉각 발견해서 신속하게 대처할 수 있었습니다.

사내 백오피스 어드민

2021.01 - 2022.03

[Client] Next.js with Typescript
[Server] Nodejs/Graphql
[DB] Oracle

유지보수 및 공통 컴포넌트/함수 작성,
Oracle 쿼리부터 Graphql 리졸버 및 Next.js 페이지 작업

기존 Next.js 소스에 유사하게 반복되는 코드들이 있어서
최대한 재사용 가능한 함수로 분리하려고 노력을 많이 했고,
스트링으로 직접 작성되어 있는 코드들을 변수나 enum, 객체 등을 이용해서
관리가 용이하도록 수정했습니다.

next.js 프로젝트 빌드 시 너무 오랜 시간이 걸려서 찾아보니
next.js 가 자체적으로 사용하는 타입 체크 플러그인이 원인인 것을 알게 되었고
해당 플러그인을 webpack 의 다른 플러그인으로 대체하여 빌드 시간을 감소시켰습니다.
추가로 원래는 next 9 버전으로 구성된 프로젝트였으나 next 12 버전부터 도입된 swc 를 사용하기 위해

버전을 12 로 올렸고 그에 따라 개발 시 HMR 의 속도가 눈에 띄게 향상되는 효과를 볼 수 있었습니다.

ag-grid(그리드 라이브러리) 에 너무 많은 row 가 출력되는 경우 화면이 버벅이는 현상이 있었는데
관련 설정을 찾아내서 버벅임 현상을 제거했습니다.

어드민 사이트이다 보니 차트 관련 작업이 꽤 있었는데 highcharts 를 이용해서 데이터를 시각화했습니다. 차트 렌더링 시 버벅임 현상을 없애기 위해 기존에 프런트에 작성 되어있던 일부 데이터 가공 로직을 서버 사이드로 이전하는 등의 처리를 추가로 진행했습니다.

채팅 웹앱
2023.02 - 2023.02

[Client] React.js with Typescript

프로젝트 신규 개발

웹소켓을 이용한 채팅의 기본적인 기능, 메세지 송수신, 안 읽은 메세지 표현, 읽음 처리 등의 기능들을 구현했습니다.

해당 프로젝트에서는 처음부터 전역 상태를 최대한 만들지 않는 방향으로 작업을 했습니다. 기본적인 프로젝트 구조는 주문관리 웹앱을 따랐으나 recoil 의 버전이 아직 메이저 버전까지 올라오지 않았다는 점이 아무래도 조금 걸려서 recoil 과 유사한 jotai 와 zustand 를 비교해보고 리렌더링을 최대한 감소하는 것을 목표로 하는 라이브러리인 jotai 로 전역 상태 관리 라이브러리를 대체하는 테스트를 진행했습니다. 하지만 recoil을 사용해오면서 특별히 문제가 된 적은 없었기 때문에 프로젝트 간 구조의 통일성을 위해 팀원들과 상의 후 최종적으로는 recoil을 사용하게 되었습니다.

컴포넌트에서 로직과는 상관이 없는 loading, disabled 등으로 주로 선언되는 상태를 제거하고 상태의 흐름을 보다 선언적으로 관리할 수 있는 방법을 찾던 와중 자바스크립트로 구현된 유한 상태 기계 라이브러리인 xstate 를 알게 되었고, 기존의 로그인 처리나 데이터 페칭 등의 프로세스를 별도의 브랜치에서 xstate 를 이용해 리팩터링 해보고 기존 방식과의 장단점을 비교해 봤습니다. 프로젝트에 채택되지는 않았지만 팀원들과 여러 가지 상태 관리 라이브러리를 탐색하고 테스트해볼 수 있는 기회가 되었습니다.

B2B 제공용 크롬 익스테션
2022.03 - 2022.05

[Client] React.js with Typescript

B2B 업체에 제공하기 위한 크롬 익스텐션을 개발했습니다. 기본적으로는 react 와 typescript 를 사용했으며 webpack 을 이용해서 빌드 시 크롬 익스텐션에서 요구하는 형식을 맞출 수 있게 작업했습니다. 프로젝트 작업 중 까다로웠던 점은 B2B 업체에서 필요한 데이터를 받기 위한 API 를 제공해 줄 수 없다고 해서 익스텐션에서 브라우저의 document 를 파싱해서 데이터를 추출해야 했던 작업인데 마침 크롤링에 대해 개인적으로 공부를 하고 있던 차라 puppeteer, cheerio 등의 라이브러리를 사용해보기도 했고 기본적으로는 document 내부의 element 와 해당 element 의 text 나 value 를 찾는 일이었기 때문에 다행히 필요한 데이터를 추출할 수 있었습니다.

PC 용 웹앱 제작
2022.02 - 2022.04

[Client] React.js with Typescript

처음부터 전체 프로젝트 구성과 개발에 참여했습니다.
api fetch 용 커스텀 hook 을 제작하여 개발에 편의성을 더했습니다.

상태 관리 라이브러리로 recoil 을 처음 도입해 보았는데
redux 에 비해 획기적으로 코드의 양을 줄일 수 있었고
기본적으로 리액트의 상태와 동일하게 사용할 수 있다 보니
너무 편하게 상태 관련 작업을 할 수 있었습니다.

c# winform 을 사용해서 웹뷰를 띄우고 윈도우 메세지를 처리하는 등의 작업이 필요했는데
c# 에 대한 지식이 전혀 없었지만 검색과 개발자 선배 분들의 조언을 통해서 요구 기능들을 구현할 수 있었습니다.

사내 물류 관리 서비스

2021.06 - 2021.06

[Client] Next.js with Typescript
[Server] Nodejs/Graphql
[DB] Oracle

소규모의 프로젝트였지만 기간이 얼마 되지 않아
팀원과 역할 분담을 확실히 했으며 맡은 업무를 다 하는 경우
서로 잔업을 도와주며 신속하게 작업을 이어나갔고
예정된 기간에 프로젝트를 완료할 수 있었습니다.

사내 React.js Boilerplate 제작

2022.01 - 2022.02

[Language] Typescript
[Library] React, Material UI, Emotion
[Bundler] Webpack

사내에서 주로 사용되는 기술 스택으로 React.js Boilerplate 제작

신규 프로젝트에 사용하면서 부족한 점이 있으면 업데이트하고
브랜치에 따라 pwa, electron 프로젝트로도 사용할 수 있게 작업했습니다.

데일리버튼

2020.06 - 2020.12 (7개월) | 정규직

세종시 두루타 버스 관제 사이트 유지보수

2020.06 - 2020.12

[Client] Next.js

기존 Next.js 소스 리팩토링,
네이버맵 렌더링 성능 향상,
신규 페이지 작업

T map API 호출 시 화면이 멈추는 현상이 있었는데 API 호출 설정을 바꾼 뒤
비동기로 호출해서 해당 현상을 제거했습니다.

네이버 지도가 화면에 여러 개 겹쳐서 렌더링되는 현상이 있었는데
next.js 소스의 useEffect 내부의 상당히 긴 코드를 분석하고 분리해서
지도가 한 번만 렌더링될 수 있도록 수정했습니다.

외부 데이터 제공용 API

2020.11 - 2020.11

[Server] Spring Boot
[DB] Mysql
[Cloud] Naver Cloud Platform

Spring Boot 를 활용한 외부 데이터 제공용 API 서버

외부 업체에 데이터를 제공하기 위해
Spring Boot 를 활용해서 CSV 파일을 다운로드할 수 있는 API 서버를 구축했습니다.

학력

가톨릭대학교

2012.03 - 2019.02 | 졸업 | 심리학과, 철학과 학사

스킬

React | JavaScript | CSS3 | Git | HTML | SQL | TypeScript | Node.js | GraphQL

링크

- 넥스트를 활용하여 후지필름 카메라의 설정 정보를 웹에서 관리할 수 있도록 만든 사이트입니다. 외국인 유저도 사용할 수 있게 다국어 처리를 해두었습니다.
  https://fujifilm-x-recipes.vercel.app/ko

- 스벨트를 이용해서 만든 windows10 컨셉의 포트폴리오입니다.

https://windows-svelte.vercel.app

개발 공부한 내용을 정리해둔 개인 블로그입니다.

https://pathas.tistory.com/