

프리온보딩 프론트엔드 과제 Week 1 TEAM 16

wanted 프리온보딩 1주차 과제입니다

배포 페이지

[Team16 배포 링크](#)

사용 라이브러리

- formatter : Eslint, Prettier
- API : Axios
- Style : Emotion

디렉토리 구조

```
src
├── apis
├── components
├── constants
├── context
├── hooks
├── pages
├── styles
├── types
├── utils
├── App.tsx
├── index.tsx
└── Router.tsx
```

- **component**와 **api**의 기능을 분리하여 각각의 **역할**과 **책임**을 명확하게 할당하고 재사용성을 향상시키려 했습니다.
- **constants** : 협업시 필요한 **일관된 규칙**과 **구조**를 유지하기 용이하게 하였습니다.
- **hooks & util** : 로직을 추상화하여 **재사용 가능한 한형태**로 분리하여 개발 **생산성** 향상시켰습니다.

Best Practice 선정 과정

Axios vs Fetch

창수

- 주관적으로 axios가 fetch보다 사용하기 편리했음.
- fetch의 경우 라이브러리를 설치하지 않고 바로 사용할 수 있다는 장점이 있음.

진호

- axios의 장점은 fetch에 비해 간편함, 가독성이 좋음, 에러처리가 간편하다는 점이 있음.
- fetch는 JavaScript 기능으로만 구현 가능하다는 장점이 있으나, json데이터를 받아오려면 자동 변환이 안되서 따로 `await response.json()` 처리를 해야한다는 점과 에러 처리시 따로 조건문을 걸어 처리해야한다는 단

점이 있음.

성민

- axios는 instance를 생성하여 header를 설정하거나 여러가지 기본 옵션을 설정해줄 수 있고, interceptor를 사용하여 api 호출 시 token 검사를 용이하게 할 수 있음. 다만, 별도로 설치해줘야 하는 번거로움 및 프로젝트 사이즈가 커진다는 단점이 있음
- fetch는 별도의 라이브러리 설치가 필요 없음. fetch의 경우 instance를 별도로 생성하여 header를 설정해줄 수 있는지 모르겠음.

진혜

- axios는 라이브러리를 따로 설치해야한다는 단점이 있음. 개인적으로 자주 사용하지 않아 익숙치 않음.
- fetch는 구현이 간단한 편이나, json 변환 등의 이유로 코드가 복잡해지는 단점이 있음.

승호

- axios의 장점은 fetch 사용보다 코드 사용이 쉬움. 개인적으로 fetch를 사용해 본 경험이 적음. 단점은 라이브러리를 따로 설치해야한다는 단점이 있음.
- fetch는 javascript 내장 기능이어서 바로 사용할 수 있다는 장점이 있는 반면, json 데이터를 직접 변환 시켜야한다는 단점이 있음.

지원

- axios는 json 타입으로 바로 사용 가능해서 코드가 간결해진다는 장점이 있고, 개인적으로 사용 경험이 많아 익숙함.
- fetch json 변환을 따로 해주어야한다는 번거로움이 있음.

2. 팀원들의 모범사례를 각출하여 각자 가동되도록 코드 구현을 한 소감은?

창수

- 어려웠던 점 : 수정 중 완료를 누르지 않고 취소를 누른다음에 체크 박스를 누르면 수정 중에서 작성 중이었던 글 들이 update 됨.
- 신경 쓴 부분 : 에러 처리에 대한 생각을 하지 못했는데, 에러 처리를 제대로 해주어야 함.
- 팀원 코드리뷰 후기 : if문으로 수정 모드와 일반 모드를 나눴는데, 삼항 연산자를 사용하니 훨씬 보기 좋았음

진호

- 어려웠던 점
 - todo CRUD에 대한 처리 : 컴포넌트안에 함수 로직을 몰아 넣는것보다 분리해서 구현했으면 하는 아쉬움이 있었음
 - todo 구현시 컴포넌트 분리 방법 : React.memo, useMemo를 사용하여 렌더링 최적화를 하면 어떨까 하는 생각
 - local데이터를 활용한 화면 렌더링 : 사용자에게 더 좋은 경험을 주기 위해 노력
- 신경 쓴 부분 :
 - API (에러처리 포함)와 component간의 분리 : api와 컴포넌트를 분리하는게 유지보수 측면에서 더 좋다고 판단
 - 컴포넌트의 단순화 (edit 버튼) : HTML 파일을 최대한 읽기 편하게 만들

- 팀원 코드리뷰 후기 : CRUD 함수 따로 구현, 에러 처리 및 토큰 처리, 재사용 가능한 컴포넌트와 파일 분리 등에 대한 점이 긍정적으로 평가 됨.

성민

- 신경 쓴 부분
 - 페이지와 컴포넌트의 역할에 대한 고민 : CRUD 함수를 pages가 담당하는 것이 좋을까, 아니면 연관 있는 컴포넌트에서 처리하는 것이 좋을까?
 - props drilling : 이거 때문에 CRUD 함수를 담당하는 위치가 pages와 각각의 컴포넌트로 분리되었는데, props drilling을 최소화하기 위해 신경을 좀 많이 썼던 것 같음. useReducer를 사용하려는 시도를 해봤지만 아직 사용이 미숙하여 해결하지 못하고 원래 상태로 제출했음
 - 컴포넌트의 분할 : 강의 때 멘토님께서 컴포넌트를 무작정 물리적으로 분할하는 것이 아니라, 논리적으로 분할해야 한다고 하셨음. 이와 비슷한 고민을 계속해서 했고, 논리적으로 컴포넌트를 분리하려는 노력을 더 해서 컴포넌트 간 결합을 최소화 하려는 노력이 필요하다고 생각함

진혜

- 어려웠던 점 : 클린코드를 작성하기 위한 코드 재사용에 대한 고민과 api 에러처리에서 어려움을 겪었음.
- 신경 쓴 부분 : 수정 기능을 사용할 때 서버 통신 시간 때문에 한 템포 늦게 업데이트 되는 점을 수정 싶었으나 잘 안 됨.
- 팀원 코드리뷰 후기 : 컴포넌트 및 폴더 분리가 잘 되었고, 토큰 처리 및 관리를 효율적으로 하였음. 유지보수나 코드 재사용이 용이한 코드가 좋아 보임.

승호



- 어려웠던 점 : 서버 데이터와 동기화 하는 점이 어려웠음.
- 신경 쓴 부분 : 각각의 에러 처리에 관하여 생각보다 많은 생각을 해야했음. api를 최대한 밖으로 빼둔점, 컴포넌트 분할 이 생각보다 초반에 시작하기에 가장 어려워 보였음.
- 팀원 코드리뷰 후기 : 개인적으로 함수를 스위치 케이스 문을 써서 사용하는게 인상 깊고 todo에 관한 handler를 그렇게 만들면 유지 보수할때 좋아 보인다는 생각이 들었음. 조금더 개선적인 생각을 하자면 switch 문 보다는 추후 todo의 데이터가 많거나, input이 많은 것을 다루는 handler 라면 각각의 함수를 handler로 변환해서 export 해두는것이 어떤가 하는 개선적인 의견이 있음. 하지만 간단한 동기화에 관해서 switch 문이 많은 효율성을 돋보임.

지원

- 어려웠던 점 : 컴포넌트를 분리하는 기준, 즉 한 컴포넌트에 코드가 있어서 가독성이 좋지 않음. 체크아웃 상태 보존에 어려움 겪음.
- 신경 쓴 부분 : 기본 구현을 완벽히 하는 것.
- 팀원 코드리뷰 후기 : api, 알림창 등 컴포넌트를 분리하여 유지보수가 편하도록 해야할 것. 불필요한 렌더링 방지한 것이 좋아 보였음.

Team 16 팀원 소개

 FE 팀원 : 창수  FE 팀원 : 진호  FE 팀원 : 성민

 FE 팀원 : 승호  FE 팀원 : 지원



 FE 팀장 : 진혜