React-Hook-Form

> 공식 문서
> https://react-hook-form.com/

서론

우리 동아리에서는 로그인과 회원가입과 관련해서 대부분 React-Hook-Form을 활용한다. 그래서 그런지 뭔가 이번에 리뷰하면서 내가 모르는 훅들이 많아서 제대로 리뷰하기 어려웠다. 그러는 김에 한번 쭉 정리하고 이미 들어간 코드이지만, 이번의 실수를 반복하지 않기 위해 다시 복귀하고 내용을 정리해보겠다.

React-Hook-Form 이란

React-Hook-Form 라이브러리는 사용하기 쉬운 검증 기능을 갖춘 성능이 뛰어나고 유연하며 확장 가능한 Form 라이브러리라고 소개하고 있습니다.

React-Hook-Form의 기본적인 동작방식은 비제어 컴포넌트로 동작합니다. 여기서 제어와 비제어에 대한 내용을 정리하고 시작해보겠습니다.

> 우리는 React state를 “신뢰 가능한 단일 출처 (single source of truth)“로 만들어 두 요소를 결합할 수 있습니다. 그러면 폼을 렌더링하는 React 컴포넌트는 폼에 발생하는 사용자 입력값을 제어합니다. 이러한 방식으로 React에 의해 값이 제어되는 입력 폼 엘리먼트를 “제어 컴포넌트 (controlled component)“라고 합니다.
> 출처 : 리액트 공식문서 제어 컴포넌트

한마디로 제어 컴포넌트란 **React에 의해 값이 제어되는 입력 폼 엘리먼트를 뜻한다**.
제어 컴포넌트는 **실시간으로 값이 동기화** 되는 특징이 있다. 또 리액트는 `state` 의 값이 변경될때마다 리렌더링 된다는 특징이 있는데, 예를 들어 사용자 입력 폼이 더 늘어날 경우에는 관리해야 하는 `state` 도 늘어나고 입력할때마다 컴포넌트 전체가 리렌더링 되기 때문에 불필요한 연산이 발생하게 된다.

거기에 입력 폼의 **유효성 검사** 까지 필요할 경우 에러 상태를 관리할 state와 검증 함수가 늘어나는 등 코드는 더욱 길어지고, 유지보수가 어려워지게 된다.

그렇다면 여기서 비제어 컴포넌트란 제어 컴포넌트의 반대가 된다. React에 의해 값이 제어되지 않는 입력 폼 엘리멘트를 뜻한다.

비제어 컴포넌트 방식을 사용하면 같은 메모리 주소를 가지게 되기 때문에 제어 컴포넌트처럼 실시간으로 값을 동기화 시키지 않고, `state` 로 값을 관리할 필요가 없이 **submit** 버튼을 누르면 그 값을 받아오게 된다.

입력 폼이 늘어나도 state를 늘릴 필요도 없으며, 값이 변경되어도 리렌더링 되지 않는다. 리액트에서 불필요한 리렌더링을 막는건 성능 개선에 필수적인 요소이다.

비제어 vs 제어 컴포넌트

feature	uncontrolled	controlled
one-time value retrieval (e.g. on submit)	✅	✅
validating on submit	✅	✅
instant field validation	❌	✅
conditionally disabling submit button	❌	✅
enforcing input format	❌	✅
several inputs for one piece of data	❌	✅
dynamic inputs	❌	✅

제출과 제출시 값 검색을 하는것은 제어 컴포넌트와 비제어 컴포넌트 모두 가능하지만, 그 외 실시간 유효성 검사, 조건부 제출 버튼 비활성화 등은 모두 제어 컴포넌트를 사용해야 가능하다고 나와있다.

React-Hook-Form의 장점

react-hook-form은 비제어 컴포넌트의 장점은 그대로 살리면서 제어 컴포넌트에서만 다룰 수 있는 실시간 유효성 검사, 실시간 동기화 등의 API를 제공하여 실시간 유효성 검사 및 동기화를 가능하게 해주는 유용한 라이브러리이다.

한마디로 제어 컴포넌트를 사용할 때 보다 훨씬 적은 코드로 훨씬 더 나은 성능을 경험할 수 있게 해주는 라이브러리이다.

- 리렌더링을 최소화시켜 마운팅 속도를 높여준다.
- 라이브러리를 선택할 땐 패키지 크기도 중요하다. 공식 홈페이지에 super light라고 표기된 걸 확인할 수 있을 정도로 종속성이 없는 작은 사이즈의 라이브러리이다.
- 타입스크립트를 기본으로 제공한다.

- 공식문서가 잘되어 있다.
- 지속적인 업데이트

우리들의 문제

이렇게 장점이 많은 코드인데 우리는 어떻게 해서 이런 회고를 하게 되었는가를 돌이켜보면 useWatch의 남용이었습니다.

useWatch 함수

useWatch 함수는 폼의 입력 값이 변경될 때마다 특정 작업을 수행할 수 있는 함수입니다. 이 함수는 다음과 같은 방식으로 사용됩니다.

```
import React, { useState, useEffect, useWatch } from "react";
import { Controller, useForm } from "react-hook-form";

const MyForm = () => {
 const { control, register, handleSubmit } = useForm();

 const watchedUsername = useWatch({
  name: 'username',
  control
 });

 return (
  <form onSubmit={handleSubmit(onSubmit)}>
   <Controller
    control={control}
    name="username"
    render={({ field }) => (
     <input
      value={field.value}
      onChange={e => field.onChange(e.target.value)}
     />
    )}
   />
  </form>
 );
};
```

위에 말했던 설계 방식에서는 우리가 했었던 방식에서는 모든 값에 useWatch를 감싸게 되었습니다. 과연 우리가 변경시에만 동작하도록 잘 설계를 했었는지를 고려하지 못하였고, 이를 통해 불필요한 렌더링이 굉장히

늘어났다고 할 수 있습니다.

> 참고자료: velog 블로그 https://toby2009.tistory.com/50 [코딩을 끄적끄적:티스토리]