11. react-hook-form + zod 가이드

LGES Web Application Platform 구축

Exported on 2024-04-02 13:29:01

Table of Contents

1 1. 개요 4

2 2. 참고 가이드 5

3 3. react-hook-form 6

3.1 1) refetch 시 params를 같이 넘겨 호출하기 위해 6

4 4. zod 8

4.1 1) form schema 정의 8

4.2 2) Primitive type 9

4.3 3) rule 정의 10

4.4 4) error message 11

4.5 5) @hookform/resolvers/zod  사용 예제 12

4.6 6) 사용자 정의 Validation Rule ( refine  ) 13

5 5. 사용 예제 16

5.1 1) zod를 사용하여 form에 사용될 schema를 정의합니다. 16

5.2 2) form에서 사용될 속성 type을 정의하기 위해 schema를 이용하여 infer합니다. 17

5.3 3) resolver를 통해 useForm 설정합니다. 17

5.4 4) form 선언 및 input 선언 18

5.5 5) submit 진행 20

5.6 6) error 처리 (helper text, toast, modal 등) 20

5.7 7) login-page.tsx 참고 21

* [1. 개요](#scroll-bookmark-3)
* [2. 참고 가이드](#scroll-bookmark-4)
* [3. react-hook-form](#scroll-bookmark-5)
  + [1) refetch 시 params를 같이 넘겨 호출하기 위해](#scroll-bookmark-6)
* [4. zod](#scroll-bookmark-7)
  + [1) form schema 정의](#scroll-bookmark-8)
  + [2) Primitive type](#scroll-bookmark-9)
  + [3) rule 정의](#scroll-bookmark-10)
  + [4) error message](#scroll-bookmark-11)
  + [5) @hookform/resolvers/zod 사용 예제](#scroll-bookmark-12)
  + [6) 사용자 정의 Validation Rule ( refine )](#scroll-bookmark-13)
* [5. 사용 예제](#scroll-bookmark-14)
  + [1) zod를 사용하여 form에 사용될 schema를 정의합니다.](#scroll-bookmark-15)
  + [2) form에서 사용될 속성 type을 정의하기 위해 schema를 이용하여 infer합니다.](#scroll-bookmark-16)
  + [3) resolver를 통해 useForm 설정합니다.](#scroll-bookmark-17)
  + [4) form 선언 및 input 선언](#scroll-bookmark-18)
  + [5) submit 진행](#scroll-bookmark-19)
  + [6) error 처리 (helper text, toast, modal 등)](#scroll-bookmark-20)
  + [7) login-page.tsx 참고](#scroll-bookmark-21)

# 1. 개요

표준 F/W FE 에서 Form을 관리하기 위해 react-hook-form과 더불어 schema validation을 위해 zod를 사용합니다.

react-hook-form을 사용하는 이유는 까다로운 form과 validation 편의성을 위함입니다.

* 참고: (번역) 여러분이 React Query를 필요한 이유: <https://velog.io/@cnsrn1874/why-you-need-react-query>

zod를 사용하는 이유는 schema validation을 사용하여 form validation 편의성과 type 안정성을 위해 사용합니다.

# 2. 참고 가이드

* react-hook-form UI라이브러리 통합 시 가이드: <https://react-hook-form.com/get-started#IntegratingwithUIlibraries>
* react-hook-form 스키마 validation 가이드: <https://react-hook-form.com/get-started#SchemaValidation>
* react-hook-form 타입스크립트 가이드: <https://react-hook-form.com/get-started#TypeScript>
* zod resolvers 가이드: <https://github.com/react-hook-form/resolvers?tab=readme-ov-file#zod>
* zod 가이드: <https://github.com/colinhacks/zod?tab=readme-ov-file#basic-usage>

위 가이드를 바탕으로 react-hook-form과 zod를 사용합니다.

# 3. react-hook-form

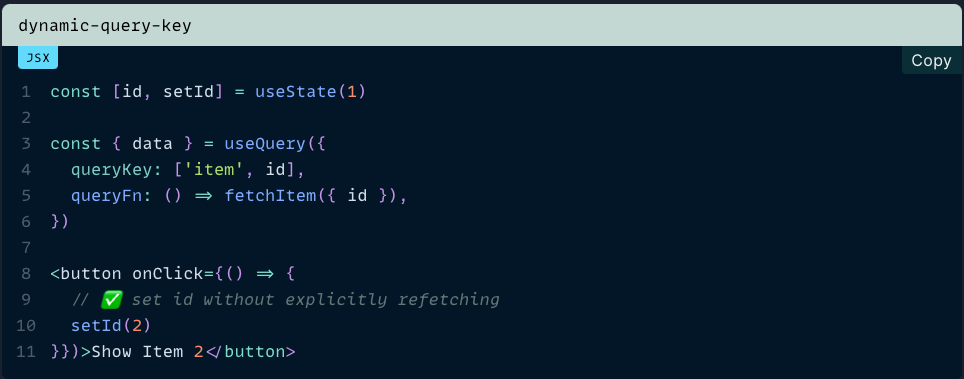
* <https://react-hook-form.com/get-started#Quickstart>

form 사용, 관리 편리성을 위한 여러 hook과 component를 제공해줍니다.

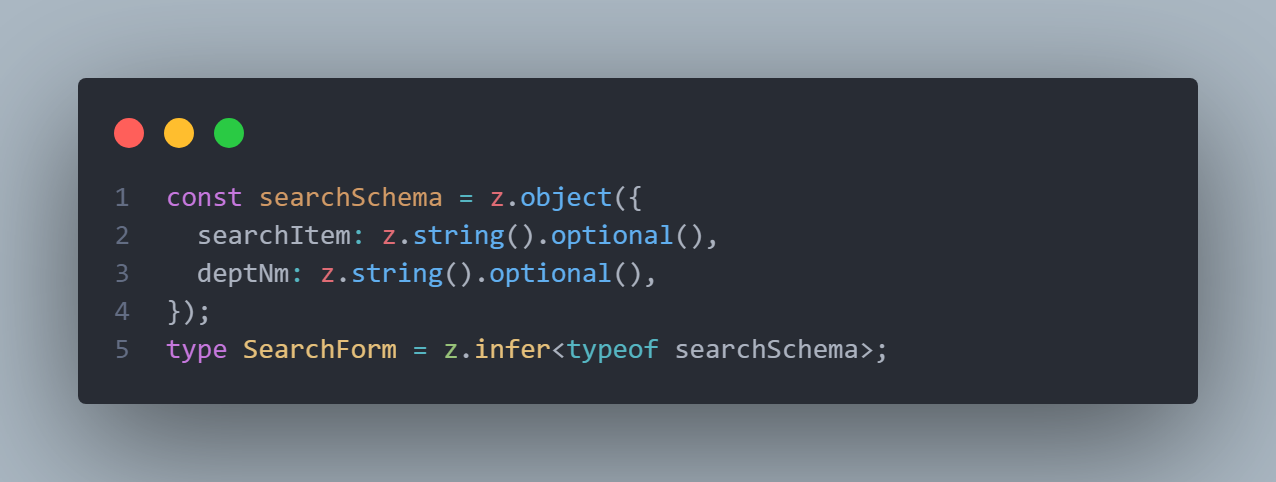
## 1) refetch 시 params를 같이 넘겨 호출하기 위해

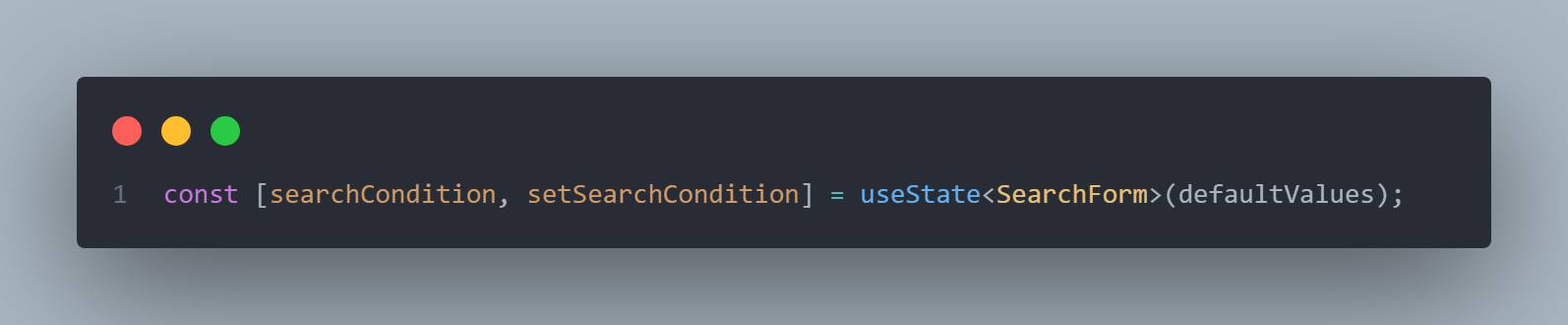
* 참고: <https://tkdodo.eu/blog/react-query-fa-qs#how-can-i-pass-parameters-to-refetch>
* 결론: 상태 사용

여러가지 방법이 있습니다만 주로 사용되는 방법은 useState를 활용하여 상태값을 이용한 refetch를 사용합니다.

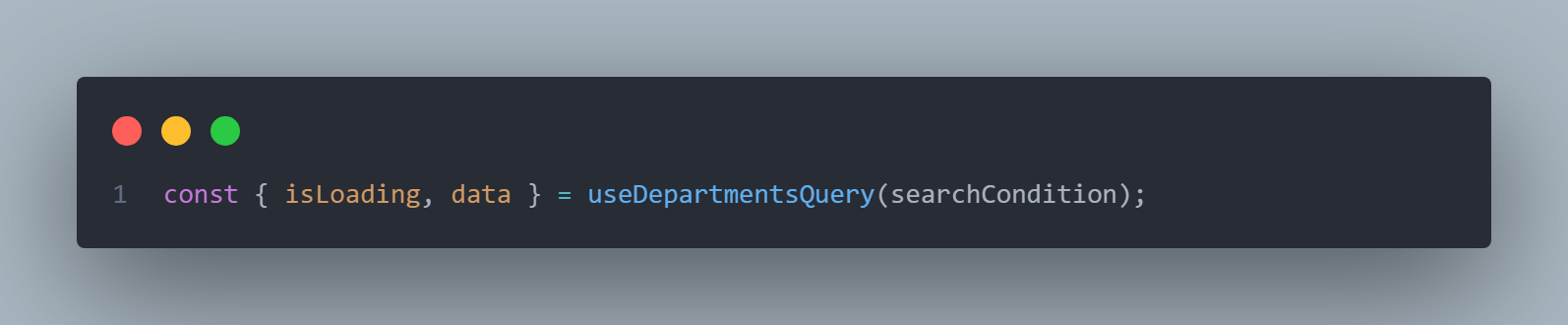


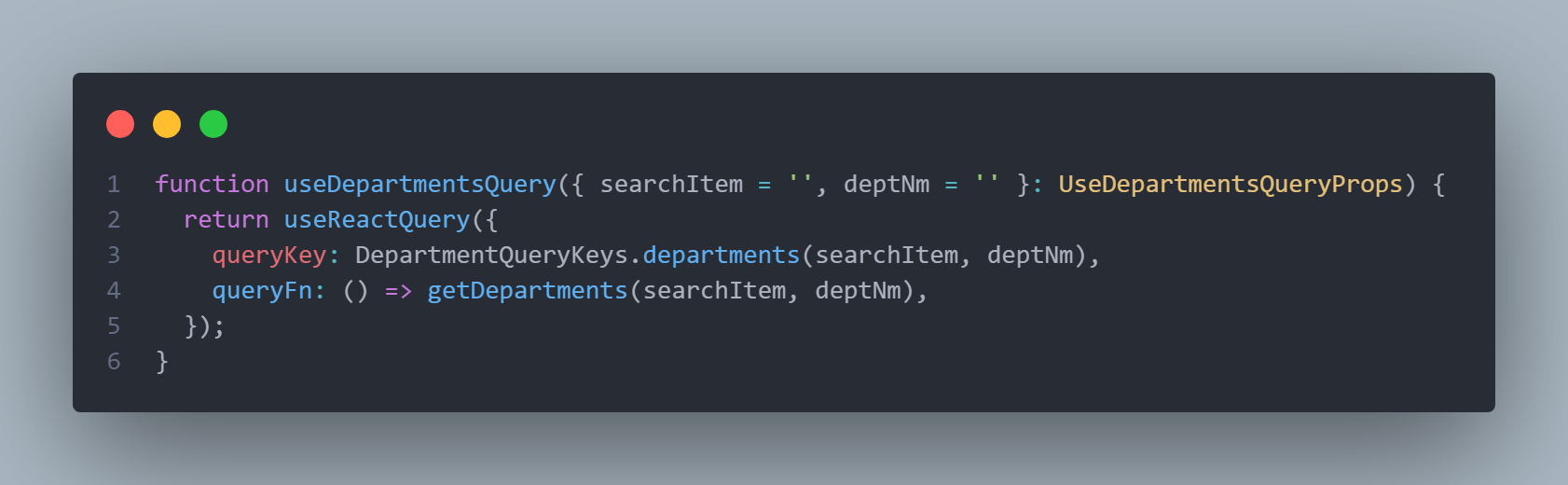
사용하는 예는 다음과 같습니다.



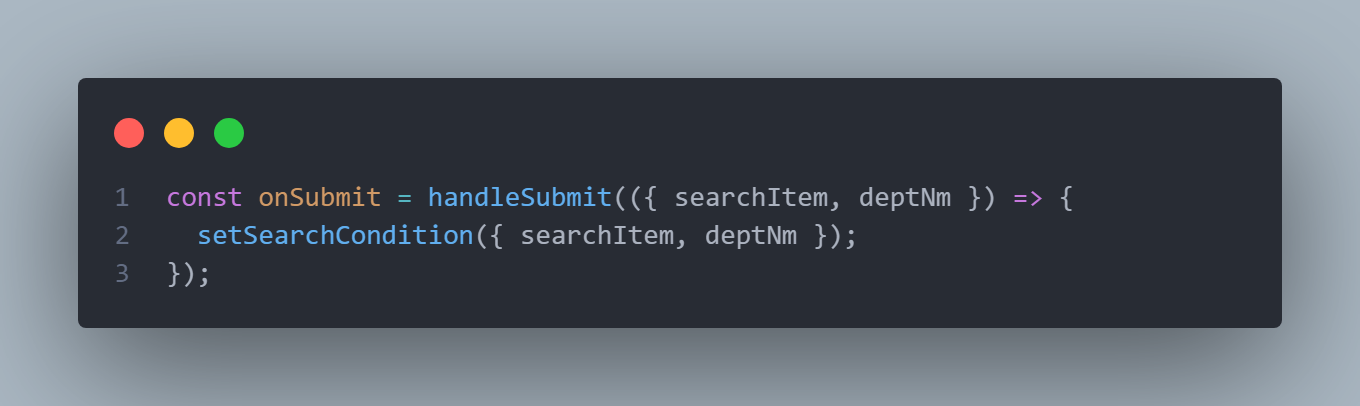


스키마를 지정하고 infer 타입을 활용합니다.



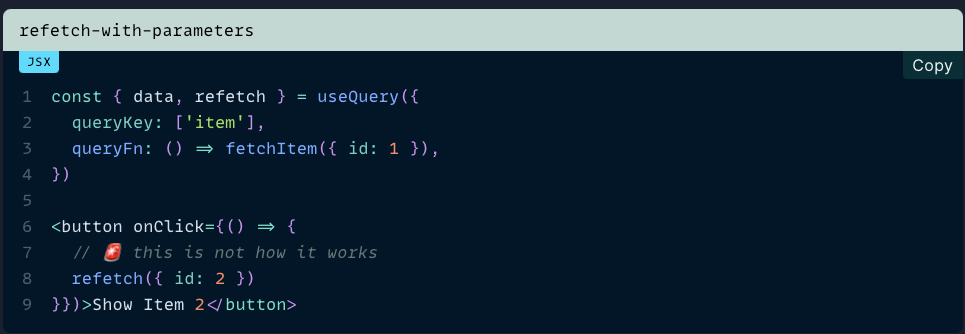


useQuery hook 만들 시에 enabled 조건 하에 queryKey가 바뀌는 순간 자동 refetch가 됩니다.



이를 이용해서 react-hook-form submit 시에 해당 상태에 set을 하고 자동 refetch하도록 합니다.

주의: refetch 함수에는 params를 넘기더라도 요청된 params로 fetch가 되지 않습니다.

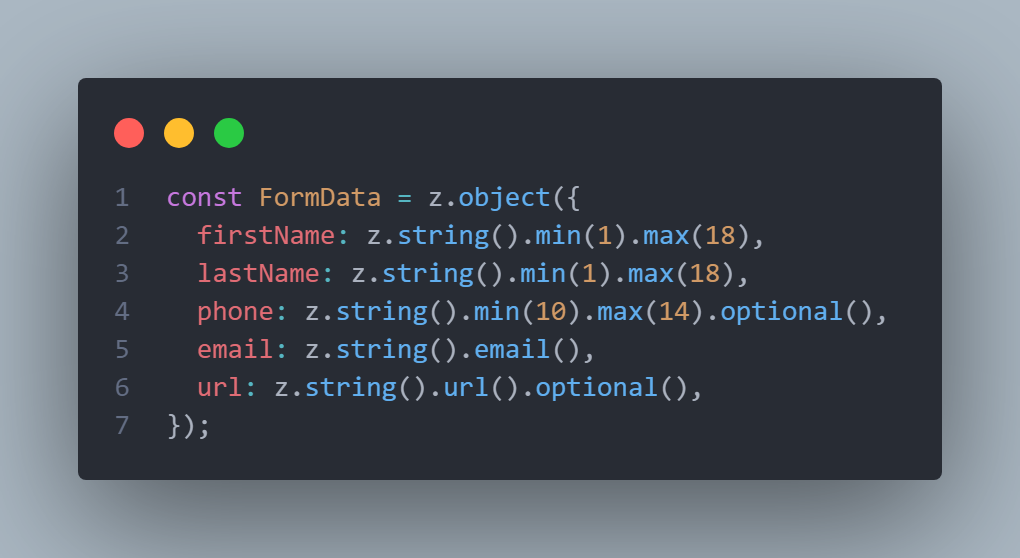


# 4. zod

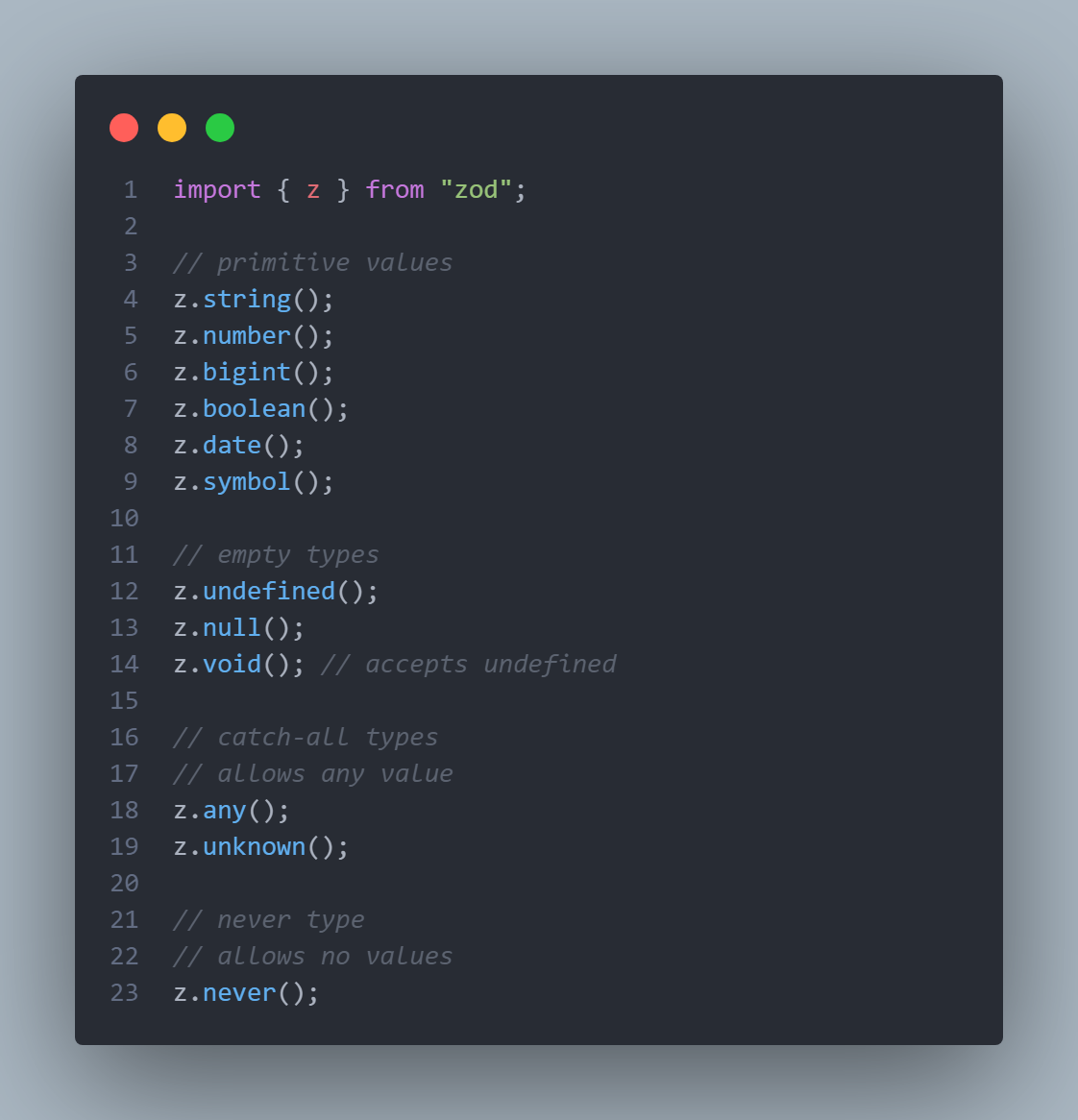
* <https://zod.dev/?id=basic-usage>

zod는 schema validation + typescript 타입 안정성 을 위해 사용되는 툴입니다.

## 1) form schema 정의



## 2) Primitive type



primitive type으로 정의된 rule이 있고, 그 외에 많은 rule을 제공합니다. 또한 편의성을 위한 많은 유틸을 제공합니다.

* Literal
* NaN
* enum
* optional
* nullable
* object
  + .shape
  + .keyof
  + .extend
  + ...
* array
  + .element
  + .nonempty
  + .min
  + .max
  + .length
* tuple
* union
* record
* ... (더 자세한 내용은 공식 가이드 페이지(<https://zod.dev/>)를 참고하시면 좋습니다.)

## 3) rule 정의

모든 type에 대해 룰을 지정할 수 있고 미리 제공되는 편의성 rule도 있습니다.

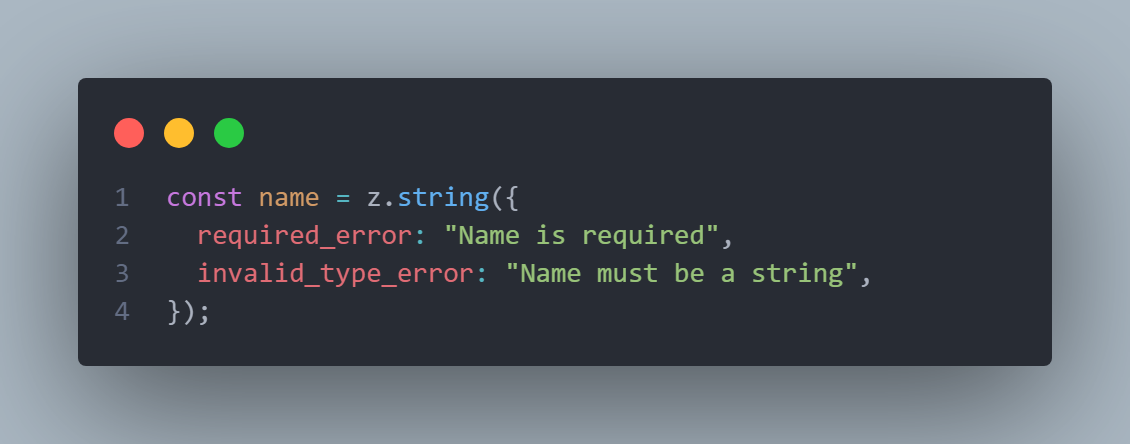


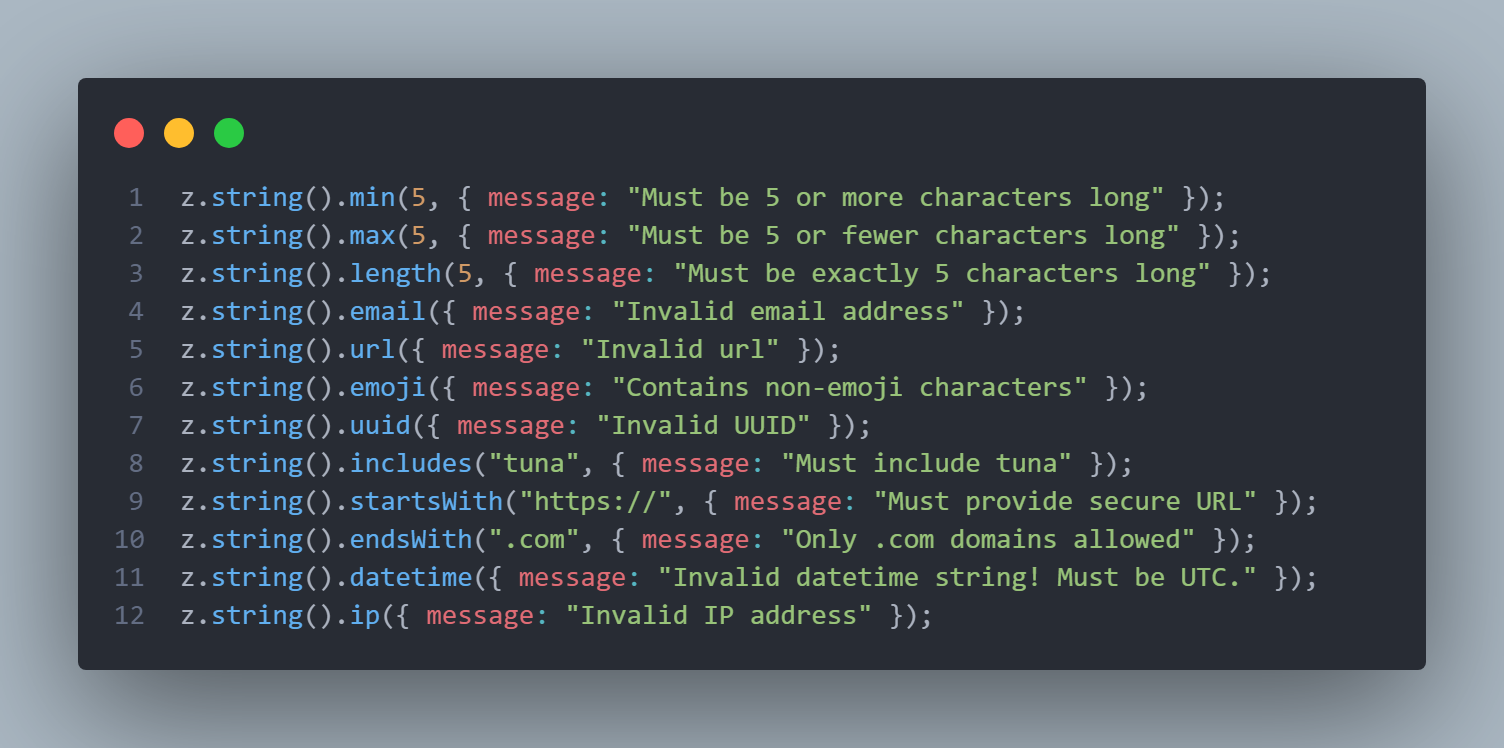




## 4) error message

validation이 통과되지 않을 경우에 사용될 메시지를 rule 정의와 함께 지정할 수 있습니다.





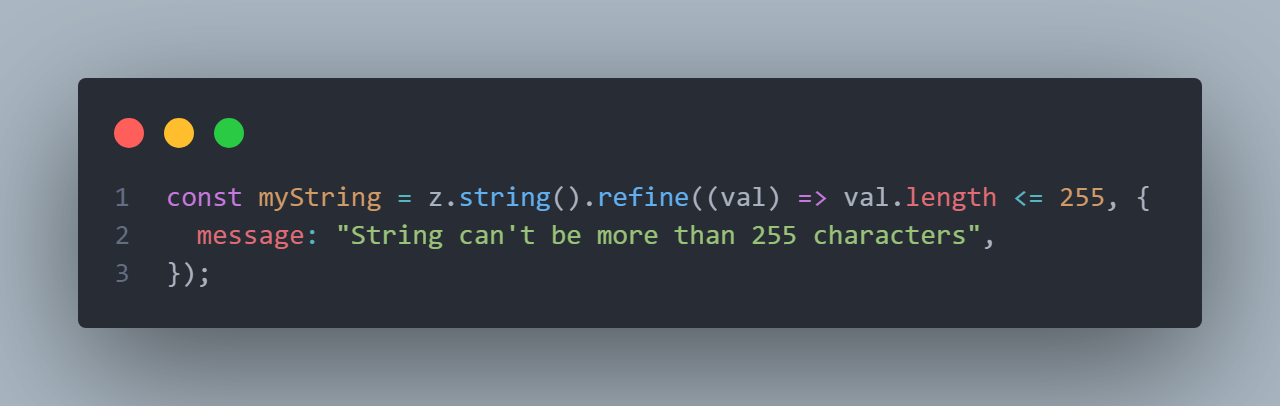
## 5) @hookform/resolvers/zod  사용 예제

react-hook-form과 같이 사용하기 위한 resolver 라이브러리를 사용합니다.

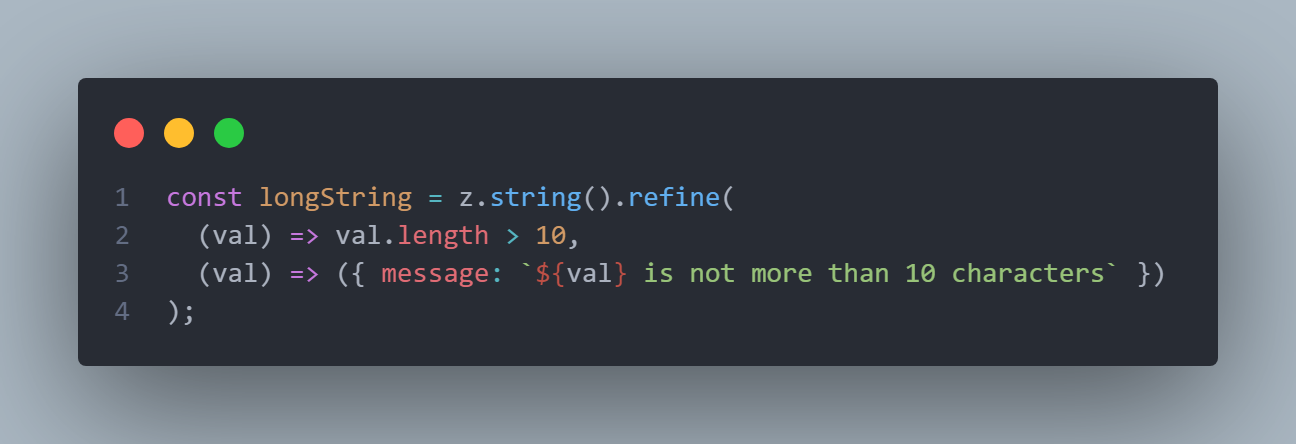


## 6) 사용자 정의 Validation Rule ( refine  )

zod에서는 custom validate를 위한 refine  함수를 제공합니다.



에러 메시지를 값에 따른 동적 메시지로 사용할 수도 있습니다.



해당 field가 아닌 다른 field의 에러 메시지로도 선언 가능합니다.



조금 더 커스텀하게 validate 룰, 메시지를 처리하기 위해서 superRefine  함수를 사용할 수도 있습니다.

(사실 refine 함수는 superRefine 의 syntactic sugar 입니다.)



# 5. 사용 예제

## 1) zod를 사용하여 form에 사용될 schema를 정의합니다.

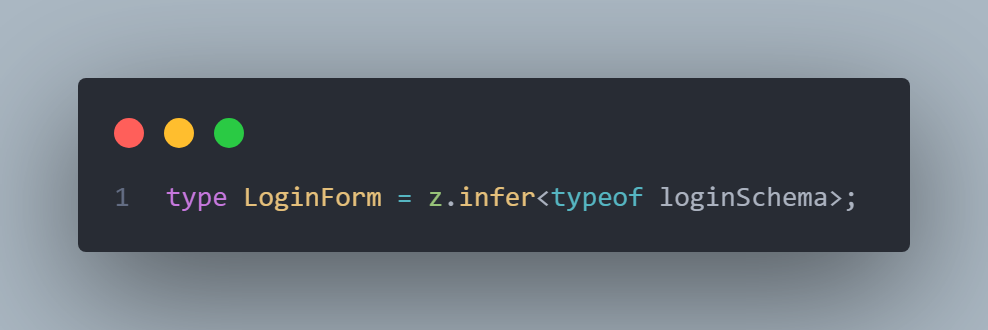




primitive한 zod의 타입을 이용해서 validation rule과 메시지 등을 정의합니다.

자세한 내용은 위에 작성한 문서 사이트를 참고하세요.

## 2) form에서 사용될 속성 type을 정의하기 위해 schema를 이용하여 infer합니다.



useForm 등 react-hook-form에 사용될 기능에 form 타입을 위해 infer 합니다.

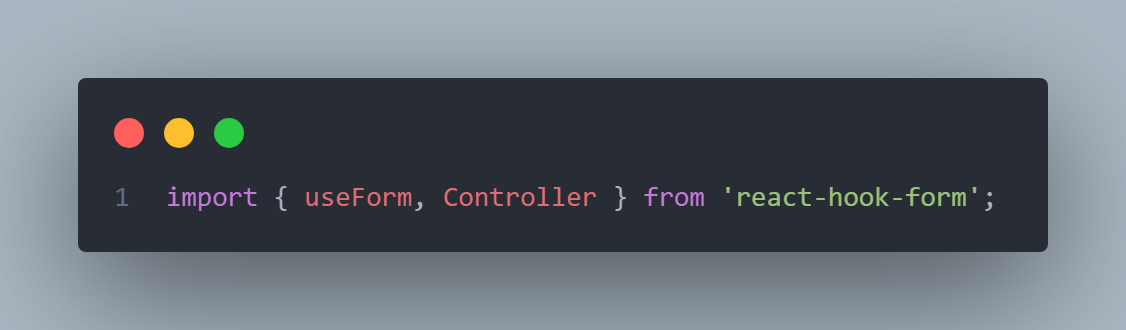
## 3) resolver를 통해 useForm 설정합니다.





제네릭타입으로 infer한 타입을 넣음으로 인해 타입(value 등)이 자동으로 infer 됩니다.

## 4) form 선언 및 input 선언





form 과 form에 사용될 여러 input 등을 선언하여 구성합니다.

저희는 주로 디자인 시스템 UI라이브러리 컴포넌트(MUI기반이 많음)를 사용하기에 <Controller>  컴포넌트를 통해 value등을  렌더링을 사용하게 됩니다.

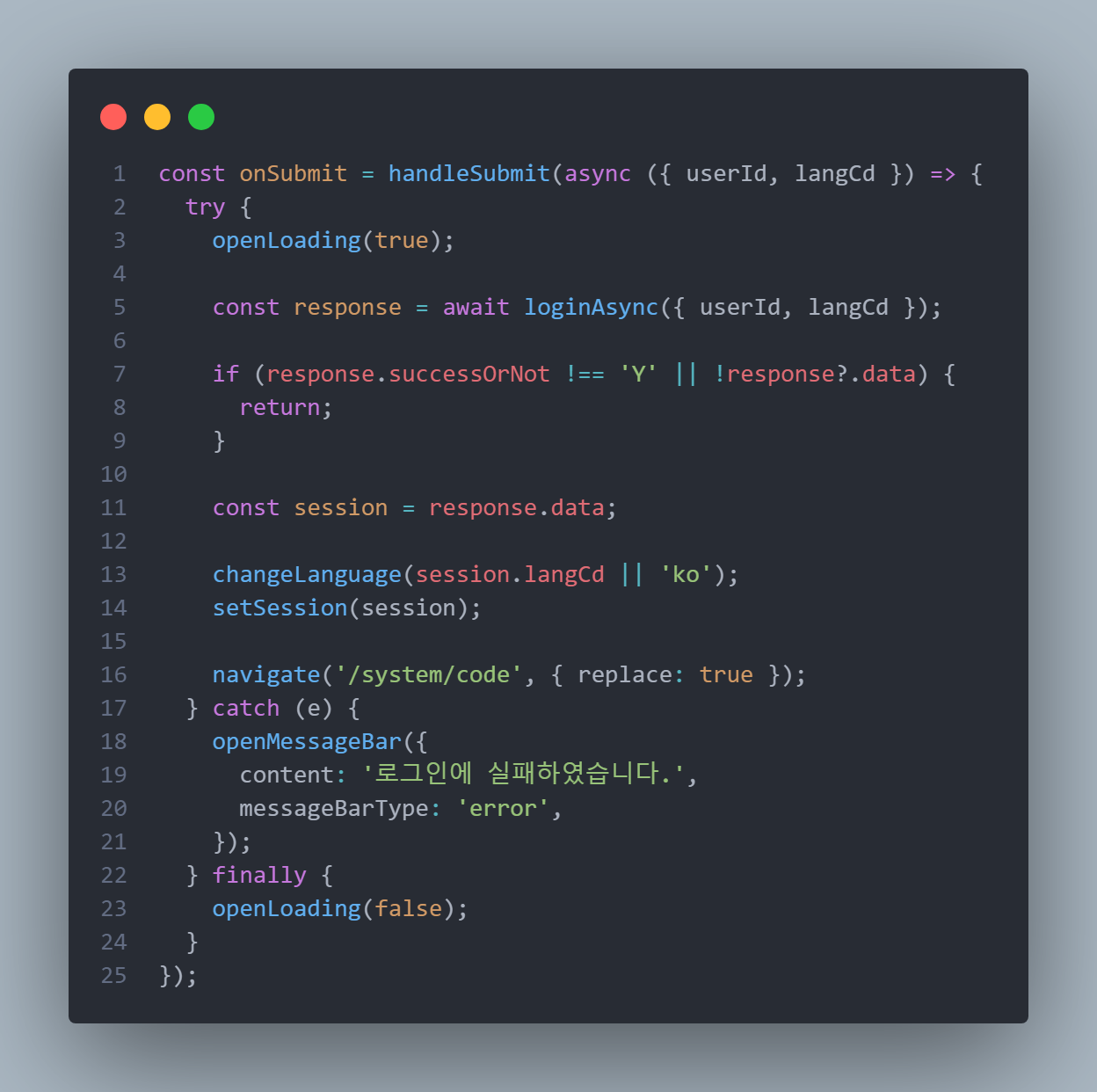
(다만, register를 이용한 방식으로 사용해도 됩니다만 유연한 value 변경이나 값에 따른 UI 렌더링 변경등을 위해 가이드 대로 controller를 주로 사용합니다.)

## 5) submit 진행

우리가 정의한 schema에 따라 valid한 form data라면 react-hook-form에서 제공하는 handleSubmit을 통해 data를 쉽게 받아 처리할 수 있습니다.

handleSubmit의 첫번째 파라미터는 form이 submit 시에 validation을 통과하면 호출되는 callback 함수 입니다.

두번째 파라미터는 반대로 통과하지 못하면 호출되는 callback 함수입니다. 주로 첫번째 파라미터인 valid success callback 함수만 정의해서 사용합니다.



## 6) error 처리 (helper text, toast, modal 등)

schema validation을 통과하지 못할 시 mode(all, onChange, onBlur 등이 있습니다. 문서를 참고해주세요) 시점에 맞춰 validation 통과 못할 시 errors에 상태값을 확인할 수 있습니다.





formState의 errors를 확인하여 타겟 form data의 에러를 확인할 수 있습니다. 여기서 message는 schema 정의 시 작성한 에러 메시지 입니다.

## 7) login-page.tsx 참고

프로젝트 내 사용 예시를 보려면 login-page.tsx를 참고하시면 됩니다.