

# [사용자] 회원가입 + zod로 유효성검사 처리

☰ 구현 어려움 정도	☆☆
☑ 구현 완료	☑
📅 기간	@December 9, 2024
☑ 정리 완료	☑

Input, Label 컴포넌트를 사용해 회원가입 폼을 만들어줍니다. 저는 샤드 cn의 input, label, card, button ui를 사용해 제작했습니다.

Card

Displays a card with header, content, and footer.

<https://ui.shadcn.com/docs/components/card>

Shadcn UI components

built using Radix UI and Tailwind CSS

Accessible and customizable components that you can copy and paste into your apps. Free. Open Source.

Create project

Name

Enter your project name

Create

Cancel

Calendar

January 2022

February 2022

1 2 3 4 5 6 7 8 9 10 11 12

13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

▼ ui 보기(click!)

```
// SignUpForm.tsx

'use client'; // 클라이언트 컴포넌트로 변경

import { ChangeEvent } from "react";
import { Input } from "../ui/input";
import { Label } from "../ui/label";
import FormCard from "../FormCard";
import Submit from "../Submit";
const SignUpForm = () => {

  const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
    const { name, value } = e?.target;
    console.log(name, value); // 콘솔 확인
  }

  return (
    <FormCard
      title="회원가입"
      footer={{ label: "이미 계정이 있으신가요?", href: "/login" }}
    >
      <form className="space-y-6">
        <div className="space-y-1">
          <Label htmlFor="name">이름</Label>
          <Input id="name" name="name" placeholder="이름을 입력해주세요" onChange={handleChange} />
        </div>
        <div className="space-y-1">
          <Label htmlFor="email">이메일</Label>
          <Input
            id="email"
            name="email"
            type="email"
            placeholder="example@example.com"
            onChange={handleChange}
          />
        </div>
        <div className="space-y-1">
          <Label htmlFor="password">비밀번호</Label>
          <Input
            id="password"

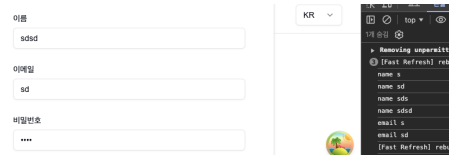
```

```

        name="password"
        type="password"
        placeholder="*****"
      />
    </div>
    <Submit className="w-full">가입하기</Submit>
  </form>
</FormCard>
)
}
export default SignUpForm;

```

콘솔 name, value 값 확인



```

// FormCard.tsx

import { ReactNode } from "react";
import { Card, CardContent, CardFooter, CardHeader, CardTitle } from "../ui/card";
import Link from "next/link";
type FormCardProps = {
  title: string;
  footer: {
    label: string;
    href: string;
  };
  children: ReactNode;
}
const FormCard = ({ title, footer, children }: FormCardProps) => {
  return (
    <Card className="w-[500px] flex flex-col items-center border">
      <CardHeader>
        <CardTitle>{title}</CardTitle>
      </CardHeader>
      <CardContent className="w-[90%]">{children}</CardContent>
      <CardFooter>
        <Link className="text-sm text-sky-700" href={footer.href}>
          {footer.label}
        </Link>
      </CardFooter>
    </Card>
  )
}
export default FormCard;

```

```

// Submit.tsx
import { Button, ButtonProps } from "../ui/button";

const Submit = ({ children, ...others }: ButtonProps) => {
  return (
    <Button type="submit" {...others}>
      {children}
    </Button>
  )
}

```

```

    });
  }
  export default Submit;

```

## 유효성 검사 로직 구현하기

zod 라이브러리 사용해 유효성 검사 로직 구현

zod: 타입스크립트를 지원하는 스키마 검증 라이브러리

- 스키마 검증(schema validation)은 데이터 구조와 내용을 미리 정의된 규칙(스키마)에 따라 확인하는 과정을 의미
- 형식 검사:** 데이터가 특정 데이터 타입(string, number, boolean 등)을 따르는지 확인
    - 예시: 이름이 문자열이어야 하고, 나이가 숫자여야 한다.
  - 구조 검사:** 데이터가 정의된 필드와 계층 구조를 올바르게 포함하고 있는지 확인
    - 예시: 객체 형태의 데이터가 name, age, email 등의 필드를 포함해야 함
  - 제약 조건 검사:** 값의 범위, 길이 제한, 필수/선택 여부 등 추가 조건을 확인
    - 예시: 나이는 0 이상 120 이하이어야 하고, 이메일 형식이어야 한다.

```

import { z } from "zod";


export const SignUpSchema = z.object({
  name: z
    .string()
    .min(1, { message: "이름을 입력해주세요." })
    .regex(/^[a-zA-Zㄱ-힣]+$/, {
      message: "이름은 문자만 입력할 수 있습니다.",
    }),
  email: z.string().email({ message: "올바른 이메일 형식을 입력해주세요." }),
  password: z
    .string()
    .min(8, { message: "패스워드는 최소 8자 이상이어야 합니다." })
    .regex(/[A-Z]/, {
      message: "패스워드는 최소 1개 이상의 대문자를 포함해야 합니다.",
    })
    .regex(/[a-z]/, {
      message: "패스워드는 최소 1개 이상의 소문자를 포함해야 합니다.",
    })
    .regex(/[0-9]/, {
      message: "패스워드는 최소 1개 이상의 숫자를 포함해야 합니다.",
    })
    .regex(/[\w_]/, {
      message: "패스워드는 최소 1개 이상의 특수문자를 포함해야 합니다.",
    }),
});

export const LoginSchema = z.object({
  email: z.string().email({
    message: "올바른 이메일 형식을 입력해주세요.",
  }),
  password: z.string().min(1, {
    message: "패스워드를 입력해주세요.",
  }),
});

```

TypeScript-first schema validation with static type inference

colinhacks/zod

TypeScript-first schema validation with static type inference


<https://zod.dev/>


319
Contributors

2m
Used by

937
Discussions

35k
Stars


1k
Forks



## useFormValidate 만들고 에러 핸들링하기

TypeScript-first schema validation with static type inference

colinhacks/zod

TypeScript-first schema validation with static type inference


<https://zod.dev/?id=error-handling>


319
Contributors

2m
Used by

937
Discussions

35k
Stars

1k
Forks



```
import { useState } from "react"
import { ZodObject, ZodRawShape } from "zod";

export const useFormValidate = <T>(schema: ZodObject<ZodRawShape>) => { // 회원가입, 로그인 Error 메시지 다름 ->
  // fieldErrors 객체 키는 에러메세지 있을 경우에만 fieldErrors가 존재하므로 Partial 추가
  const [errors, setErrors] = useState<Partial<T>>>();

  const validateField = (name: string, value: string) => {
    setErrors({
      ...errors,
      [name]: undefined,
    })

    const parseValue = schema.pick({ [name]: true }).safeParse({
      [name]: value
    });

    if (!parseValue.success) {
      setErrors({
        ...errors, // 기존 에러 객체 복사
        ...parseValue.error.flatten().fieldErrors // fieldErrors 객체 복사
      })
    }
  }

  return { errors, validateField }
}
```