To Use React Hook Form, We Can Import It:

```
import { useForm } from 'react-hook-form'
```

**********

Then We Destruct What We Want:

```
const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting },
    reset,
    getValues
  } = useForm();
```

**********

The FormState Contain Multiple Properties That We Can Use For Validation.

**********

To Make The Field is Required We Can Use:

```
<input { ...register('email', { required: 'The Email Is Required'}) }
type='email' placeholder='Entre Your Email'
          className='px-4 py-2 rounded outline-none' />
```

**********

To Check More Validation Like Min-Length:

```
<input
          { ...register('confirm',
            {
              required: 'The Confirm Password is Required',
              minLength: {
                value: 5,
                message: "The Confirm Must Be 5-Chars At Least"
              }
            })

          }
        type='password' placeholder='Confirm Your Password'
          className='px-4 py-2 rounded outline-none' />
```

**********

The Basic For Handling Form Submitting Using React Hook Form is:

```
const onSubmit = (data: FieldValues) => {
    console.log('Data is: ', data);

    console.log('The Error is: ', errors);

    console.log('The isSubmitting is: ', isSubmitting);

    console.log('The getValues is: ', getValues());

    reset();
  }


<form onSubmit={handleSubmit(onSubmit)} className='flex flex-col gap-y-2 w-
[450px]'>
```

**********

To Handle The Form Error Without Any Problem:

Note: Here We Set The Message Between `` To Avoid Any Compilation Error.

```
{errors.confirm && <p className='text-red-700'>{`${errors.confirm.message}`}</p>}
```

**********

To Define Custom Error Rule We Set:

```
 <input
      { ...register('confirm', {
              required: 'The Confirm Password is Required',
              minLength: {
                value: 5,
                message: "The Confirm Must Be 5-Chars At Least"
              },
              validate: (value) => value !== getValues('password') && 'Password
And Confirm Must Match'
            })

          }
        type='password' placeholder='Confirm Your Password'
          className='px-4 py-2 rounded outline-none' />
```

**********

To Use Zod With React-Hook-Form:

```
import { z } from 'zod';
```

Then We Define The Shape Of Data That We Want To Use:

```
const registerSchema = z.object({
  email: z.string()
    .min(5, 'Email Must Be At Least 5-Characters')
    .email('Enter Valid Email Address Please'),

  password: z.string().min(5, 'Password Must Be At Least 5-characters'),

  confirm: z.string().min(5, 'Confirm Password Must Be At Least 5-Characters'),
});
```

We Can Extract The Type Of Our Schema using z.infer:

```
type registerType = z.infer<typeof registerSchema>;
```

Then We Can Use Our Resolver With useForm Like That:

```
const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting },
    reset,
    getValues
  } = useForm({ resolver: zodResolver(registerSchema)});
```

**********

To Validate The Submitted Data We Use Refine-Method:

Note 1: The Message is The Msg That We Want To Display To User.

Note 2: The Path is The Element That We Want To Bind The Message To Its Properties, It Can Be Multiple Elements.

Note 3: By This Way We Don't Need **getValues()-Method**.

```
const registerSchema = z.object({
  email: z.string()
    .min(5, 'Email Must Be At Least 5-Characters')
    .email('Enter Valid Email Address Please'),

  password: z.string().min(5, 'Password Must Be At Least 5-characters'),

  confirm: z.string().min(5, 'Confirm Password Must Be At Least 5-Characters'),
}).refine(data => data.password === data.confirm, {
  message: 'Confirm Password Must Match Password',
  path: ['confirm']
});
```

**********

Note: We Can Use Multiple Schema Instead Of One Schema, Where the zodResolver Accept Async Function.

**********

To Select The Type Of Our Form:

```
type RegisterType = z.infer<typeof registerSchema>;
```

```
const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting },
    reset,
  } = useForm<RegisterType>({ resolver: zodResolver(registerSchema)});
```

**********

Note 1: By Setting The Type Of Our Form, We Don't Need import FieldValues From react-hook-form

```
import { useForm } from 'react-hook-form';

const onSubmit = async (data: RegisterType) => {
    await new Promise(resolve => setTimeout(resolve, 2000));
    console.log('Data is: ', data);
    console.log('The Error is: ', errors);
    console.log('The isSubmitting is: ', isSubmitting);
    reset();
  }
```

**********

Note 1: With Zod And React Hook Form, We Can Handle The Server Errors Like:

Note 2: We Can use setError From React-Hook-Form:

Note 3: The responseData Here is From fetch-JS-API:

```
if (responseData.errors) {
  const errors = responseData.errors;
  if (errors.email) {
    setError("email", {
      type: "server",
      message: errors.email,
    });
  } else if (errors.password) {
    setError("password", {
      type: "server",
      message: errors.password,
    });
  } else if (errors.confirmPassword) {
    setError("confirmPassword", {
      type: "server",
      message: errors.confirmPassword,
    });
  } else {
    alert("Something went wrong!");
  }
}
```

**********

Note 1: In The Server Side We Can Use Our Schema By Re-Define It, Our Export It

```
registerSchema.safeParse(data);
```

**********

```
import { signUpSchema } from "@/lib/types";
import { NextResponse } from "next/server";

export async function POST(request: Request) {
  const body: unknown = await request.json();

  const result = signUpSchema.safeParse(body);
  if (!result.success) {

  }

  return NextResponse.json({});
}
```

**********

```
import { signUpSchema } from "@/lib/types";
import { NextResponse } from "next/server";

export async function POST(request: Request) {
  const body: unknown = await request.json();

  const result = signUpSchema.safeParse(body);
  let zodErrors = {};
  if (!result.success) {
    result.error.issues.forEach((issue) => {
      zodErrors = { ...zodErrors, [issue.path[0]]: issue.message };
    });
  }

  return NextResponse.json(
    Object.keys(zodErrors).length > 0
      ? { errors: zodErrors }
      : { success: true }
  );
}
```

**********