Using React Hook Form to Create Uncontrolled Forms

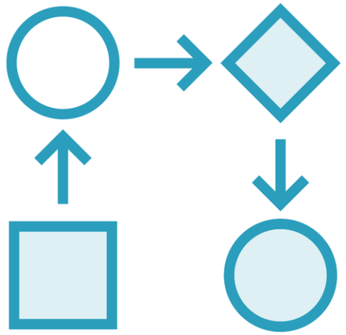
Nitin Singh
Full-stack Developer

Overview

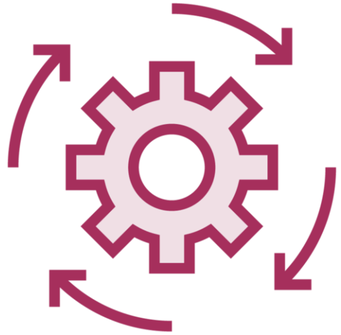
- Understand key advantages of using React Hook Form library
- How to add React Hook Form to your app
- Lifecycle of a form built with React Hook Form
- Adding validations to uncontrolled forms
- Monitoring values for instant feedback

Controlled components
re-render the entire form on
every change in the form.

Salient Features of React Hook Form



Uncontrolled elements



Minimize re-renders

Works with Controlled components & UI libraries

Validation helpers & support for schema-based validators

Integrating React Hook Form in Your App

To install run: npm install react-hook-form

The useForm hook is used to access register, handleSubmit methods and the errors object

useForm hook accepts configs to alter the behavior of the form

Configs: mode, defaultValues, resolver

```
import React from "react";
import { useForm } from "react-hook-form";

export default function App() {

  const {register,handleSubmit,errors}=
useForm();

  const mySubmit = data => console.log(data);

  return (

    <form onSubmit={handleSubmit(mySubmit)}>

      <input {...register(”email")}/>

      {errors.email && <span>
             This field is required</span>}

      <input type="submit" />

    </form>
  );
}
```

- Import useForm hook
- Get register, handleSubmit and errors
- This callback is invoked if validation doesn’t fail after form submission
- Set the form’s onSubmit to handleSubmit
- Register the input tag with name
- If errors are present render them

Steps in the Form's Lifecycle

1

Form configuration

Pass config to useForm

2

Field registration

Call register on the fields

3

Value updation

Values stored in map with field name as key

4

Run validations

On form submission. Errors stored in map with field name as key and re-render triggered

5

Successful submission

Invoke onSubmit configured in <form>

Data Validation in React Hook Form

Adding Validations

1

HTML standard

register function accepts standard HTML validation rules with messages

2

Schema validation

Register resolver for schema in useFrom hook

```
<input {...register("name", {required: "error message"})} />

<input type="number" {...register("age", {min: { value: 18, message: "minor"}})} />

<input type="submit" />
```

HTML standard based validation

required, min, max, minLength, maxLength, pattern, validate

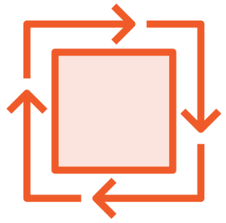
```
// npm install @hookform/resolvers yup

import { useForm } from "react-hook-form";
import { yupResolver } from
'@hookform/resolvers/yup';
import * as yup from "yup";

const schema = yup.object().shape({
   name: yup.string().required()
});

export default function App() {
   const { register, handleSubmit, errors } =
      useForm({ resolver: yupResolver(schema)});

   return (
     <form onSubmit={handleSubmit(onSubmit)}>
     <input {...register("name")}/>

     <p>{errors.name?.message}</p>

     <input type="submit" />
   </form>
  );
}
```

- Install Yup resolver
- Import Yup and Yup resolvers
- Define Yup schema for form
- Register schema using Yup resolver
- If error for field is present render it

Monitoring Values for Instant Feedback

Use the watch function to specify name of field(s) to watch

It returns the latest values of the field(s)



Watched fields trigger re-renders on every change in the field

Summary

- Salient features of React Hook Form
- How to integrate React Hook Form in an app
- Lifecycle of forms created with React Hook Form
- Adding validations and instant feedback to uncontrolled forms

Thank You