Building Forms

Terms

React Hook Form
Ref hook
Schema-based validation libraries
Zod

Summary

- To handle form submissions, we set the onSubmit attribute of the form element.

- We can use the ref hook to access elements in the DOM. This technique is often used to read the value of input fields upon submitting a form.

- We can also use the state hook to create state variables and update them as the user types into input fields. With this technique, every time the user types a character into an input field, the component containing the form gets re-rendered. While in theory this can cause a performance penalty, in practice this is often negligible.

- React Hook Form is a popular library that helps us build forms quickly with less code. With React Hook Form, we no longer have to worry about using the ref or state hooks to manage the form state.

- React Hook Form supports the standard HTML attributes for data validation such as required, minLength, etc.

- We can validate our forms using schema-based validation libraries such as joi, yup, zod, etc. With these libraries, we can define all our validation rules in a single place called a schema.

HANDLING FORM SUBMISSION

```
const App = () => {
  const handleSubmit = (event: FormEvent) => {
    event.preventDefault();
    console.log('Submitted');
  };

  return (
    <form onSubmit={handleSubmit}>
    </form>
  );
};
```

ACCESSING INPUT FIELDS USING THE REF HOOK

```
const App = () => {
  const nameRef = useRef<HTMLInputElement>(null);

  const handleSubmit = (event: FormEvent) => {
    event.preventDefault();

    if (nameRef.current)
      console.log(nameRef.current.value);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input ref={nameRef} type="text" />
    </form>
  );
};
```

MANAGING FORM STATE USING THE STATE HOOK

```
const App = () => {
  const [name, setName] = useState('');

  return (
    <form>
      <input
        type="text"
        value={name}
        onChange={(event) => setName(event.target.value)}
      />
    </form>
  );
};
```

MANAGING FORM STATE USING REACT HOOK FORM

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

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

  const onSubmit = (data: FieldValues) => {
    console.log('Submitting the form', data);
  }

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input {...register('name')} type="text" />
    </form>
  );
};
```

VALIDATION USING HTML5 ATTRIBUTES

```
const App = () => {
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm<FormData>();

  const onSubmit = (data: FieldValues) => {
    console.log('Submitting the form', data);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input {...register('name', { required: true })} type="text" />
      {errors.name?.type === 'required' && <p>Name is required.</p>}
    </form>
  );
};
```

DISABLING THE SUBMIT BUTTON

```
const App = () => {
  const {
    formState: { isValid },
  } = useForm<FormData>();

  return (
    <form>
      <button disabled={!isValid}>Submit</button>
    </form>
  );
};
```

SCHEMA-BASED VALIDATION WITH ZOD

```
import { FieldValues, useForm } from 'react-hook-form';
import { z } from 'zod';
import { zodResolver } from '@hookform/resolvers/zod';

const schema = z.object({
  name: z.string().min(3),
});

type FormData = z.infer<typeof schema>;

const App = () => {
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm<FormData>({ resolver: zodResolver(schema) });

  const onSubmit = (data: FieldValues) => {
    console.log('Submitting the form', data);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input {...register('name')} type="text" />
      {errors.name && <p>{errors.name.message}</p>}
    </form>
  );
};
```