How Do Forms Work in React?

Forms are fundamental to every web application because they let you handle user input, collect data, and trigger actions.

In React, forms are managed using state or refs, giving you full control over their behavior and validation. These two ways to manage forms are called "controlled" and "uncontrolled" input.

Let's look at what controlled and uncontrolled inputs are.

Controlled input is the most "React-like" way to handle form inputs. With controlled inputs, you store the input field value in state and update it through `onChange` events. This gives you complete control over the form data and allows instant validation and conditional rendering.

The process works like this: React maintains the form state with the `useState` hook, and you update it on every change. When a user types in an input field, the `onChange` event fires, updates the state, and React re-renders the component with the new value.

```
import { useState } from "react";

function App() {
  const [name, setName] = useState("");

  const handleChange = (e) => {
    setName(e.target.value);
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log(name);
  };

  return (
    <>
      <form onSubmit={handleSubmit}>
        <label htmlFor="name">Your name</label> <br />
        <input value={name} id="name" onChange={handleChange} type="text" />
        <button type="submit">Submit</button>
      </form>
    </>
  );
}

export default App;
```

The benefits of controlled inputs include the following:

- Immediate access to the form data.
- You can implement instant validation.
- You can conditionally disable the submit button.
- You can control the input value programmatically.

Uncontrolled inputs on the other hand are seen more in traditional HTML forms. So, instead of handling the inputs through the `useState` hook, uncontrolled inputs in HTML maintain their own internal state with the help of the DOM.

Since the DOM controls the input values, what you need is to pull in the values of the input fields with ref. This approach requires less code and performs better because refs do not make React re-render.

Here's an example of uncontrolled inputs:

```
import { useRef } from "react";

function App() {
  const nameRef = useRef();

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log(nameRef.current.value);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="name">Your</label>{" "}
      <input type="text" ref={nameRef} id="name" />
      <button type="submit">Submit</button>
    </form>
  );
}

export default App;
```

One very noticeable advantage of uncontrolled inputs is that they require less code. They also perform better and feel more natural to React beginners who are familiar with HTML.

So, which should you use between controlled and uncontrolled inputs?

Use controlled inputs when you need dynamic form updates, real-time validation, or when you want to sync input values with state. They provide better control but require more re-renders.

Use uncontrolled inputs when you need simpler forms, want to access values only on submission, or when you're working with non-React code.

Regardless of which you use between controlled and uncontrolled inputs, here are some best practices you should adhere to while making forms in React:

- Always prevent the default form submission.
- Ensure you validate inputs before submission.
- Always provide clear feedback to users with loading, validation errors or other related states.

Questions

How do you manage input field values in a controlled input?

By storing the value in state and updating it through `onChange` events.

By directly modifying the DOM input value and using JavaScript to get the values.

By using refs to track changes and assigning the ref attribute to each input.

By setting the input value to `null` for a start and writing a function to get them later.

Which of these is a benefit of controlled inputs?

They update the DOM directly without state management.

They allow more control over the form data.

They prevent form validation.

They disable the `onChange` event handler.

How do uncontrolled inputs handle form data in React?

They store the input value in the component state.

They use the `useReducer` hook for state management.

React updates the input value through `onChange` events.

The DOM manages the form data internally.