React: Forms

Phase 2 | Week 1, Lesson 4

Sakib Rasul | Updated January 31, 2024 | Created June 15, 2023

Today's Objectives

Today, we'll answer the following questions:

1. *How do we write forms in React?*
2. *How do we handle form submissions?*
3. *What does it mean for a form to be **(un)controlled**?*
4. *Why should we write controlled forms?*

Uncontrolled Forms

```
export default function App() {
  return (
    <>
      <h1>Fill this out, please!</h1>
      <Form />
    </>
  )
}
```

```
export default function Form() {
  return (
    <form>
      <label>Name<input /></label>
      <input type="submit" />
    <form>
  )
}
```

Controlled Forms

- Earlier this week, we learned that a user event is often a sign we have something dynamic and independent that we can hold in state.
- When a user fills out a form, they fire countless events.
- Most of these events affect a form’s input values.
- Ergo, it makes sense to hold and update input values in state!

```
export default function App() {
  return (
    <>
      <h1>Fill this out, please!</h1>
      <Form />
    </>
  )
}
```

```
export default function Form() {
  return (
    <form>
      <label>Name<input /></label>
      <input type="submit" />
    <form>
  )
}
```

```
export default function Form() {
  const [name, setName] = useState("");
  function handleChange(event) console.log(event.target.value)
  return (
    <form>
      <label>Name
        <input value={name} onChange={handleChange} />
      </label>
      <input type="submit" />
    <form>
  )
}
```

What’s your favorite movie?

(Un)controlled Components

- An **uncontrolled component** has state that cannot be altered by its parent.
  - In other words, an **uncontrolled component** is driven by state.
- A **controlled component** renders something managed by a parent.
  - In other words, a **controlled component** is driven by props.
- An `<input>` element is *uncontrolled* when its value isn't held in state.
- An `<input>` element is *controlled* when its value is held in state.
- The act of "controlling" a component is often known as *lifting state up*.

```
export default function Form() {
  const [name, setName] = useState("");
  function handleChange(event) setName(event.target.value);
  return (
    <form>
      <label>Name
        <input value={name} onChange={handleChange} />
      </label>
      <input type="submit" />
    <form>
  )
}
```

Why “control” a form?

- Controlling a component (i.e., lifting its state) requires setup.
  - It’d be tedious and unwise to lift all state.
  - Often, state belongs right where its rendered.
- However, controlling a component grants us maximal flexibility.
  - For forms, we get cool stuff like input validation and synchronization.

*I put “control” in quotes because when we talk about a controlled form, we really mean a form that controls its children.

I’m feeling lucky.

Questions? // Thanks!