React Forms Submit

[(https://github.com/learn-co-curriculum/react-hooks-forms-submit)](https://github.com/learn-co-curriculum/react-hooks-forms-submit) [(https://github.com/learn-co-curriculum/react-hooks-forms-submit/issues/new)](https://github.com/learn-co-curriculum/react-hooks-forms-submit/issues/new)

Learning Goals

- Handle a form's submit event in React
- Use controlled inputs to validate values

Introduction

In this lesson, we'll discuss how to handle form submission in React.

If you want to code along there is starter code in the `src` folder. Make sure to run `npm install && npm start` to see the code in the browser.

Submitting a Controlled Form

Now that we've learned how to control a form with state, we want to set up a way to submit our form. For this, we add the `onSubmit` event listener to our `form` element:

```
// src/components/Form.js
return (
  <form onSubmit={handleSubmit}>
    <input type="text" onChange={handleFirstNameChange} value={firstName} />
    <input type="text" onChange={handleLastNameChange} value={lastName} />
    <button type="submit">Submit</button>
  </form>
);
```

Now, whenever the form is submitted (by pressing the Enter or Return key in an input field, or by clicking a Submit button), the `handleSubmit` callback function will be called. We don't have the `handleSubmit` function yet, so let's write it out:

```
function handleSubmit(event) {
  event.preventDefault();
  const formData = {
    firstName: firstName,
    lastName: lastName,
  };
  props.sendFormDataSomewhere(formData);
  setFirstName("");
  setLastName("");
}
```

Let's look at each line of code in this function:

- `event.preventDefault()` : The default behavior of a form is to **try and submit the form data based on a defined action (https://www.w3schools.com/html/html_forms.asp)** , which effectively causes the browser to refresh the page. We didn't (and don't need to) define an action. The result, however, is that the form makes a new request to the current page, causing a refresh. By using `event.preventDefault()` , we stop this behavior from happening.
- `const formData = { firstName: firstName, lastName: lastName }` : Here, we are putting together the current form data into an object using the values stored in state.
- `props.sendFormDataSomewhere(formData)` : A form, when submitted, should send the form data somewhere. As mentioned a moment ago, the traditional HTML way was to send data to a server or another page using the `action` attribute. In React, we handle requests with asynchronous JavaScript. We won't go into the details of how this works just yet, but we can think of `sendFormDataSomewhere()` as the code that handles sending our data off. This function might be defined in the same form component, or can be passed down as a prop.
- `setFirstName("")` : if we want to clear the input fields, all we need to do is set state! In a traditional JavaScript form, you might do something like `event.target.reset()` to clear out the form fields. Here, because we are using controlled inputs, setting state to an empty string clears out the values from the input fields once the data has been submitted.

You can contrast this to handling an *uncontrolled* form being submitted, in which case you would need to access the input fields from the DOM instead of accessing the values from state:

```
function handleSubmit(event) {
  event.preventDefault();
  // in an uncontrolled form, you need to access the input fields from the DOM
  const formData = {
    firstName: e.target[0].value,
    lastName: e.target[1].value,
  };
  props.sendFormDataSomewhere(formData);
}
```

Since we don't have a server to send our data to, let's remove our `sendFormDataSomewhere()` function. Instead, we'll demonstrate submission by modifying our `Form` component to access submitted values from state and list them in the DOM:

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

function Form() {
  const [firstName, setFirstName] = useState("Sylvia");
  const [lastName, setLastName] = useState("Woods");
  const [submittedData, setSubmittedData] = useState([]);

  function handleFirstNameChange(event) {
    setFirstName(event.target.value);
  }

  function handleLastNameChange(event) {
    setLastName(event.target.value);
  }

  function handleSubmit(event) {
    event.preventDefault();
    const formData = { firstName: firstName, lastName: lastName };
    const dataArray = [...submittedData, formData];
    setSubmittedData(dataArray);
    setFirstName("");
    setLastName("");
  }

  const listOfSubmissions = submittedData.map((data, index) => {
    return (
      <div key={index}>
        {data.firstName} {data.lastName}
      </div>
    );
  });

  return (
    <div>
      <form onSubmit={handleSubmit}>
        <input type="text" onChange={handleFirstNameChange} value={firstName} />
        <input type="text" onChange={handleLastNameChange} value={lastName} />
        <button type="submit">Submit</button>
      </form>
      <h3>Submissions</h3>
      {listOfSubmissions}
    </div>
  );
}

export default Form;
```

The above component will render previous form submissions on the page! We have a fully functioning controlled form.

Validating Inputs

One benefit we get from having our form's input values held in state is an easy way to perform validations when the form is submitted. For example, let's say we want to require that a user enter some data into our form fields before they can submit the form successfully.

In our `handleSubmit` function, we can add some validation logic to check if the form inputs have the required data, and hold some error messages in state:

```
// add state for holding error messages
const [errors, setErrors] = useState([]);

function handleSubmit(event) {
  event.preventDefault();
  // first name is required
  if (firstName.length > 0) {
    const formData = { firstName: firstName, lastName: lastName };
    const dataArray = [...submittedData, formData];
    setSubmittedData(dataArray);
    setFirstName("");
    setLastName("");
    setErrors([]);
  } else {
    setErrors(["First name is required!"]);
  }
}
```

Then, we can display an error message to our user in the JSX:

```
return (
  <div>
    <form onSubmit={handleSubmit}>
      <input type="text" onChange={handleFirstNameChange} value={firstName} />
```

```
        <input type="text" onChange={handleLastNameChange} value={lastName} />
        <button type="submit">Submit</button>
      </form>
      {/* conditionally render error messages */}
      {errors.length > 0
        ? errors.map((error, index) => (
            <p key={index} style={{ color: "red" }}>
              {error}
            </p>
          ))
        : null}
      <h3>Submissions</h3>
      {listOfSubmissions}
    </div>
  );
```

Conclusion

By setting up our form components using controlled inputs, we give React state control over the data being displayed in the DOM. As a benefit of having the form data in state, we can more easily access it once a form is submitted and either pass it along to another component or use it to make a fetch request. We can also more easily perform some validation logic when the form data is submitted.

Resources

- **React Forms (https://reactjs.org/docs/forms.html)**