React Controlled Components

Prior Knowledge

- Be able to create and update state.
- Be able to attach event listeners in React.
- Understand the default behaviour of html forms.

Learning Objectives

- Understand how to control an input's value in React.
- Understand the difference between `controlled` and `uncontrolled` components.
- Be able to build a functional form in React using controlled components.

Controlled Components

In html we can use several types of tags such as `input`, `textarea` and `select` that allow our users to enter data. If we render one of these tags in React the input will work just as it does in html and allow the user to interact with it.

For example, consider the snippets from this Shopping List App:

```
const App = () => {
  const [list, setList] = useState(['Bread', 'Strawberries', 'Chocolate']);
  return (
    <div>
      // rest of App's render
      <ItemAdder setList={setList} />
      <ShoppingList items={list} />
    </div>
  );
};

const ShoppingList = (props) => {
  return (
    <ul>
      {props.items.map((item) => {
        return (
          <li key={item}>
            <p>{item}</p>
          </li>
        );
      })}
    </ul>
  );
};
```

```
const ItemAdder = () => {
  return (
    <form>
      <label>
        Add a new item:
        <input />
      </label>
      <button type="submit">Add item</button>
    </form>
  );
};
```

This input will allow the user to type into it, but we have no knowledge of that input's current value in React. The input keeps its own internal state and if we wanted to access it we would have to go back to the DOM to try and read its value. This is what's referred to as an uncontrolled component. These can be useful if you are integrating React into a non-react project but in general we want our state to be our source of truth and to be in control of what value the input has at all times.

To do this we will use a controlled component. See Controlling an input with a state variable for more info.

A component is `controlled` if:

1. Its value is set with a prop.
2. Changes to that value are handled by React.

An input's value can be passed to it from React. So in this case we will keep the `newItem` in state and pass the value to the component. When the user attempts to change the value we will update our state to reflect that change as below.

```
const ItemAdder = () => {
  const [newItem, setNewItem] = useState('');
  return (
    <form>
      <label>
        Add a new item:
        <input
          // 1. set the value of the input
          value={newItem}
          // 2. Update the state whenever a change is made to that value
          onChange={(event) => setNewItem(event.target.value)}
        />
      </label>
      <button type="submit">Add item</button>
    </form>
  );
};
```

Every controlled component must have **both** of these 2 parts. If one is missing, the input will either have a fixed value or we will be unable to change it from within React.

onChange

A React `onChange` event is triggered by any change to the inputs value, unlike the DOM `onchange` event which only triggers when the input box loses focus.

Inside the `onChange` function we can use the `event.target.value` to read the incoming change and update our React state accordingly.

This updates the input as the user types and allows us to reset the form by changing our state.

Forms

Forms can be submitted in a number of ways, either by clicking on buttons with a type of `submit` (the default button type when placed within a form) or by pressing enter on a focused input. This triggers the `submit` event of the form and we can handle that submission using `onSubmit`. As we are building a Single Page Application in React, it is common to prevent the default form behaviour of navigating away from the page and handle the submission logic in JavaScript.

In our shopping list example, we want to add an item to the parent's state and reset our form.

```
const ItemAdder = ({ setList }) => {
  const [newItem, setNewItem] = useState('');

  const handleSubmit = (event) => {
    // prevent the form's default submission behaviour
    event.preventDefault();
    // add the newItem to our list in App
    setList((currList) => {
      return [newItem, ...currList];
    });
    // reset the input to be empty
    setNewItem('');
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Add a new item:
        <input
          value={newItem}
          onChange={(event) => setNewItem(event.target.value)}
        />
      </label>
      <button type="submit">Add item</button>
    </form>
  );
};
```

The React docs go into more detail on how to handle different types of input following the principles of controlled components. See select and textarea for some examples.