Solution: Build a Radio Group Component

Here is the completed solution code for the `Radio/index.js` file:

```
export const RadioGroup = ({ onChange, selected, children }) => {
  const RadioOptions = React.Children.map(children, (child) => {
    return React.cloneElement(child, {
      onChange,
      checked: child.props.value === selected,
    });
  });
  return <div className="RadioGroup">{RadioOptions}</div>;
};

export const RadioOption = ({ value, checked, onChange, children }) => {
  return (
    <div className="RadioOption">
      <input
        id={value}
        type="radio"
        name={value}
        value={value}
        checked={checked}
        onChange={(e) => {
          onChange(e.target.value);
        }}
      />
      <label htmlFor={value}>{children}</label>
    </div>
  );
};
```

Step 1

The API for the `RadioGroup` component is defined as two props: `selected`, which is a string that matches one of the `RadioOption` values and `onChange`, which is the event that gets called whenever a selection changes, providing the new value as an argument.

```
<RadioGroup onChange={setSelected} selected={selected}>
  <RadioOption value="social_media">Social Media</RadioOption>
  <RadioOption value="friends">Friends</RadioOption>
  <RadioOption value="advertising">Advertising</RadioOption>
  <RadioOption value="other">Other</RadioOption>
</RadioGroup>
```

Step 2

The `RadioOptions` variable should be assigned to the return value of `React.Children.map`, which will be a new React element. The first argument passed to the map function should be the `children` prop, and the second is a function that gets invoked in every `child` contained within the `children` property. Recall that a `children` prop is a special prop all React components have and that it presents a special data structure, similar to arrays, where you can perform iterations. However, they are not exactly instances of JavaScript arrays. That's why to iterate over all siblings you should use the special `React.children.map` API provided by React.

Inside the map projection function, you should first clone the element using `React.cloneElement`, passing as first argument the target `child` element and as a second argument a configuration with all new props. The resulting element will have the original element's props with the new props merged in.

`onChange` can be passed to each child (`RadioOption`) as it is and checked is the property the `RadioOption` uses to determine if the underlying `radio` input is selected. Since `RadioGroup` receives a `selected` property, which is a string pointing to the value of the option that has been selected, `checked` will be only true for one of the options at any point in time. This is guaranteed by performing an equality check, comparing the `RadioOption` value prop with the selected value.

Finally, the `RadioGroup` component returns the new `RadioOptions` elements by wrapping them in curly braces.

```
import * as React from "react";

export const RadioGroup = ({ onChange, selected, children }) => {
  const RadioOptions = React.Children.map(children, (child) => {
    return React.cloneElement(child, {
      onChange,
      checked: child.props.value === selected,
    });
  });
  return <div className="RadioGroup">{RadioOptions}</div>;
};
```

Step 3

The `RadioOption` component now receives two new props implicitly, `onChange` and checked, that `RadioGroup` is injecting via children manipulation, as seen in the previous section.

The `value` prop is already provided explicitly inside the `App.js` component and `children` represents the label text for the radio input.

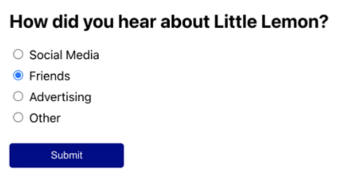
You have to connect the props `value`, `checked` and `onChange` correctly. First, both `value` and `checked` props should be passed to the `radio` input as is. Then, you should use the `onChange` event from the `radio` input, retrieve the `value` property from the event target object and pass it to the `onChange` prop as the argument as seen below. That completes the implementation of the `RadioOption` component.

```
export const RadioOption = ({ value, checked, onChange, children }) => {
  return (
    <div className="RadioOption">
      <input
        id={value}
        type="radio"
        name={value}
        value={value}
        checked={checked}
        onChange={(e) => {
          onChange(e.target.value);
        }}
      />
      <label htmlFor={value}>{children}</label>
    </div>
  );
};
```

Step 4

Once you run the application in the browser, you should see something similar to the screenshot below.

The important thing is that the button should be enabled as soon as a selection is made. Don't worry if nothing happens when you click it, it's intended. In this exercise, the button click event has no action bound to it.



Mark as completed

Like Dislike Report an issue