Audit Access Expires Apr 20, 2020
You lose all access to this course, including your progress, on Apr 20, 2020.

Forms

Controlled Components

HTML form elements such as inputs, text areas, and select fields naturally keep some internal state. When we use HTML form elements in React, we tie that natural state to the React Component state so that all of the state can be maintained by a single source.

We accomplish this by doing the following two steps:

1. Whenever the input value is changed, call an event handler to update the component state to the new input value
2. Re render the the React Element with its value attribute set to the updated state input value

Form elements that have their state's controlled by React in his manner are called Controlled Components.

Controlling Input fields

To turn an input field into a Controlled Component, we must first declare an event handler that will update the state input value whenever the form input value is changed.

The event.target.value attribute can be used to obtain the form input value:

```
    handleChange(event){
        this.setState({value: event.target.value})
    }
```

We then must attach the event handler to the <input> element and set the input value equal to the state input value:

```
    render(){
        return (
            <input type = "text" value = {this.state.value} onChange
= {this.handleChange}/>
        )
    }
```

Lastly, we must not forget to bind the event handler to the component instance and also declare the initial state value:

```
    constructor(props){
        super(props)
        this.state = {value: ''}
        this.handleChange = this.handleChange.bind(this)
    }
```

Putting it all together:

```
class ControlledInput extends React.Component{

    constructor(props){
        super(props)
        this.state = {value: ''}
        this.handleChange = this.handleChange.bind(this)
    }
    handleChange(event){
        this.setState({value: event.target.value})
    }
    render(){
        return (
            <input type = "text" value = {this.state.value} onChange
= {this.handleChange}/>
        )
    }
}
```

Controlling Checkboxes

Checkboxes use a checked attribute instead of a value attribute.

Example:

```
class ControlledInput extends React.Component{

    constructor(props){
        super(props)
        this.state = {checked: false}
        this.handleChange = this.handleChange.bind(this)
    }
    handleChange(event){
        this.setState({checked: event.target.checked})
    }
    render(){
        return (
            <input type = "checkbox" checked = {this.state.checked}
onChange = {this.handleChange}/>
        )
    }
}
```

Controlling TextArea fields

Controlling TextAreas is similar to controlling Input Fields in React:

```
class ControlledTextArea extends React.Component{

    constructor(props){
        super(props)
        this.state = {value: ''}
        this.handleChange = this.handleChange.bind(this)
    }
    handleChange(event){
        this.setState({value: event.target.value})
    }
    render(){
        return (
            <textarea type = "text" value = {this.state.value}
onChange = {this.handleChange}/>
        )
    }
}
```

Controlling Select Tags

Controlling Select Tags is similar to controlling Input Fields in React:

```
class ControlledSelect extends React.Component{

    constructor(props){
        super(props)
        this.state = {value: 'apple'}
        this.handleChange = this.handleChange.bind(this)
    }
    handleChange(event){
        this.setState({value: event.target.value})
    }
    render(){
        return (
          <select value={this.state.value} onChange=
{this.handleChange}>
            <option value="apple">apple</option>
            <option value="banana">banana</option>
            <option value="carrot">carrot</option>
            <option value="donuts">donuts</option>
          </select>
        )
    }
}
```

Select Components can also have their options dynamically generated using the map() method. Example:

```
class ControlledSelect extends React.Component{

    constructor(props){
        super(props)
        this.state = {value: 'apple'}
        this.handleChange = this.handleChange.bind(this)
    }
    handleChange(event){
        this.setState({value: event.target.value})
    }
    render(){
        var array = ["apple","banana","carrot","donuts"]
        var options = array.map( (item) =>
            <option value = {item}>{item}</option>
        )
        return (
          <select value={this.state.value} onChange=
{this.handleChange}>
            {options}
          </select>
        )
    }
}
```

Handling Multiple Inputs

If your form has multiple inputs, you can set each of their values to a different attribute on the component state. It is useful to use ES6's computed property name feature to accomplish this.

Example:

```
class ControlledMultiple extends React.Component{

    constructor(props){
        super(props)
        this.state = {value: 'apple'}
        this.handleChange = this.handleChange.bind(this)
    }
    handleChange(event){


        this.setState({[event.target.name]: event.target.value})
    }
    render(){
        var array = ["apple","banana","carrot","donuts"]
        var options = array.map( (item) =>
            <option value = {item}>{item}</option>
        )
        return (
            <form>
                <input name="inputName" type = "input" value =
{this.state.inputName} onChange = {this.handleChange}/>
                <textarea name="textAreaName" type = "text" value =
{this.state.textAreaName} onChange = {this.handleChange}/>

                <select name = "selectName" value=
{this.state.selectName} onChange={this.handleChange}>
                    {options}
                </select>
            </form>
        )
    }
}
```