REACT

LESSON 8 – setState

CO2214 - PRACTICAL WORK ON CO2224

- Before we start let me point that I will be using the ES7 React JS snippets Extensions.
- To understand the dos and don’ts with State and set State let us create counter component.
- Start by creating a new component called "Counter.js"
- So, within the components folder create a new file named as "Counter.js"

In Counter.js

- since we are learning about state in class components this is going to be class component as well and with react snippets, I can type **rce** and tab out and we have its class component.
- Now Counter.js show this code

```
import React, {Component} from 'react'

export class Counter extends Component {
  render () {
    return (
      <div>Counter</div>
    )
  }
}

export default Counter
```

- I only want default export So, I will remove the name export.

```
import React, {Component} from 'react'

class Counter extends Component {
  render () {
    return (
      <div>Counter</div>
    )
  }
}

export default Counter
```

- Within the div tag I am going to have the text count and let me have Counter component in the app component.

```
import React, {Component} from 'react'

class Counter extends Component {
  render () {
    return (
      <div>Count</div>
    )
  }
}

export default Counter
```

in App.js

```
import './App.css';
import Counter from ‘./Components/Counter';
function App () {
  return (
    <div className="App">
      <Counter />
    </div>
  );
}

export default App;
```

- If you quickly check and look at the browser you should be able to see the text "Count".
- Go back to Counter Component
- The first thing we need is count state to keep track of the counter value and we initialize the state in the constructor.
- The snippet is rconst

```
import React, {Component} from 'react'

class Counter extends Component {

    constructor(props) {
      super(props)

      this.state = {
         first
      }
    }

  render () {
    return (
      <div>Count</div>
    )
  }
}

export default Counter
```

- So constructor a called to super and this.state which is an object and the state object has its property called count initialized to 0.
- Now we can bind this state value with curly braces so within the random method Count - {this.state.count}
- next let us add the button to increment the count value.
- So, I am going to add parameters to return statement at enclosing div tag
- The div tag within the return statements and then add the button element So button tag the text is going to be increment and we

listen to the click event very on the opening button tag onClick attribute.

- This is going to be equal to curly braces and an arrow function within the curly braces.

```
import React, {Component} from 'react'

class Counter extends Component {

  constructor(props) {
      super(props)

      this.state = {
         count: 0
      }
    }

  render () {
    return (
    <div>

       <div>Count - {this.state.count} </div>
       <button onClick={() => this.increment()}>Increment</button>

    </div>
    )
  }
}

export default Counter
```

- Now let us define the increment method write after the constructor increment and if we can recollect when we have to change state of the component, we need to use the setState method but you might be curios what will happen if you don't use it and try to change the state directly.
- let us try out it.

within the increment method

this.state.count = this.state.count +1

you are basically incrementing the count value by 1

and also going to simply log the updated count value in the console.

console.log(this.state.count)

- so, we are basically trying to catch the current value of count incremented by 1 and we are assigning that value
- let us see this and take look at the browser.
- I am going to open the console now I will click on increment button when I click you can see the value not incremented in the UI in the console you can see that it has changed from 0 to 1.

```
import React, {Component} from 'react'

class Counter extends Component {

    constructor(props) {
      super(props)

      this.state = {
         count: 0
      }
    }

    increment ()
    {
        this.state.count = this.state.count + 1
        console.log(this.state.count)
    }
  render () {
    return (
    <div>

       <div>Count - {this.state.count} </div>
       <button onClick={() => this.increment()}>Increment</button>

    </div>
    )
  }
}

export default Counter
```

- What this means is that the UI is not rendering when the state is changing and this is the main reason, we should never modify the State directly.
- The only place where we can assign this.state is the constructor.
- so let us make the change

```
import React, {Component} from 'react'

class Counter extends Component {

    constructor(props) {
      super(props)

      this.state = {
         count: 0
      }
    }

    increment ()
    {
      this.setState({
        count: this.state.count + 1
        })
        console.log(this.state.count)
    }
  render () {
    return (
    <div>

       <div>Count - {this.state.count} </div>
       <button onClick={() => this.increment()}>Increment</button>

    </div>
    )
  }
}

export default Counter
```

- Now save this and take the look at the browser click on increment you can see that count value in the UI increments to 1.

Note: First dos and don’ts with state is never modify the state directly.

- In State make use of setState when you modify the state directly react will not re-render the component, setState on the other hand will let react know it has to render the component.

- Otherwise, one detail to observe with the setState in the browser you can see that when you click that on increment value is one but we take the look at console the value is zero. So, the console value is one less than the rendered value. in this because called to **setState are asynchronous.**

- So what is happening is console.log(this.state.count) where here is being called before the state is actually set. Many times, in your application you might want to execute some codes only after the state has been updated.

- To handle the such a situation you can call back to function as the second parameter to the setState method.

- The setState method has 2 parameters.

1. State object
2. callback function

- The callback function will be arrow function and within the function body let us log to the console call back value it is going to be this.state.count

- Now the code is look like,

```
increment ()
    {
      this.setState(
        {
        count: this.state.count + 1
        },
        () => {
          console.log("callback value",this.state.count)
        }
        )
        console.log(this.state.count)
    }
```

- So, we have one console log value outside the setState method and one within the call back function within the setState method.

- Now save this and take the look at browser, click on the button and in the console, you can see that we have value of zero and then callback value of 1.

- zero is from this Asynchronous console.log statement and one is from callback function console.log statement.

Note: So, this brings to the second dos and don'ts

Whenever you need to execute some code after the state has been changed do not place that code write after the setState method. In state place that code within the callback function that is passed as the second parameter to the setState method.

- let us make the scenario complicated.

- make a new method called incrementFive() within the body I simple called the increment method five times. also now on click off the button we going to call this.incrementFive()

```
import React, {Component} from 'react'
class Counter extends Component {
    constructor(props) {
      super(props)

      this.state = {
         count: 0
      }
    }
    increment ()
    {
      this.setState(
        {
        count: this.state.count + 1
        },
        () => {
          console.log("callback value",this.state.count)
        }
        )
        console.log(this.state.count)
    }
    incrementFive()
    {
      this.increment()
      this.increment()
      this.increment()
      this.increment()
      this.increment()
    }
  render() {
    return (
    <div>

       <div>Count - {this.state.count} </div>
       <button onClick={() => this.incrementFive()}>Increment</button>

    </div>
    )}}

export default Counter
```

- so, the count should increment from 0 to five when we click on the button.
- let us take the look at the browser, when I click on the button you can see that value changes to 1 instead of change into five. at in the console 0 is load five times even the callback value 1 is load five times.

  Why is this?

  This behavior is because react make group multiple setState calls in to single update for better performance.

  - what happens in this scenario is that all the five states call I done in one single go and the updated value does not carry over between the different calls. whenever you have to update the state these on previous State, we need to pass a function are given to setState method instead of passing in an object.

  - Change the code like this

```
import React, {Component} from 'react'

class Counter extends Component {

    constructor(props) {
      super(props)

      this.state = {
         count: 0
      }
    }

    increment ()
    {

      this.setState(prevState =>({
        count: prevState.count+1
      }))
      console.log(this.state.count)
    }
    incrementFive()
```

```
    {
      this.increment()
      this.increment()
      this.increment()
      this.increment()
      this.increment()
    }
 render() {
    return (
    <div>

       <div>Count - {this.state.count} </div>
       <button onClick={() => this.incrementFive()}>Increment</button>

    </div>
    )
  }
}

export default Counter
```

- So very important you make note of this difference you are not using the current state, you are always using previous state.
- Save this and back to the browser, click on increment the count is displayed 5.
- You are able to correctly render the UI based on the State, the console value is locked are from the asynchronous console log statement and can be ignore for now.

Note: So, this the last dos and don'ts

when you have to update the state used on the previous state make sure to passing function as an argument instead of the regular object.

- The function has access to the previous state which can be used to calculate the new state and turn start the second parameter this function is the props object.
- so new state is depended on the props as well props.addValue may be you can go with the function parameter approach and make use of props.

Summary

setState

1. Always make use of setState and never modify the state directly.

2. Code has to be executed after the state has been updated?Place that code in the call back function which is the second argument to the setState method.

3. When you have to update state based on the previous state value, pass in a function as an argument instead of the regular object.