Mini Project: Managing States

setState: Managing State of Cart Item

What is setState?
setState enqueues changes to the component state. It tells React that this component and its children need to be re-rendered. setState function will accept an Object that will be eventually merged into the component's current state.
setState function can be applied in two forms:

Form 1:

```
increaseQuantity() {
    console.log('this',this.state)
    // setState form 1
    this.setState({
        qty: this.state.qty+1
    });
}
```

Here setState merges with the original state object and performs **shallow merging** *(i.e. if we want to change quantity only, it will use quantity from state object only rather than touching all other properties)*. After setState, React will **render** a newly updated quantity.

Form 2 :

```
increaseQuantity() {
    console.log('this',this.state)
    // setState form 2
    this.setState((prevState) => {
        return{
            qty:prevState.qty+1
        }
    });
}
```

Here setState is written in function form where we pass a parameter called **prevState,** which provides the previous count of quantity to the current state and then setState performs shallow merging and renders the new data.

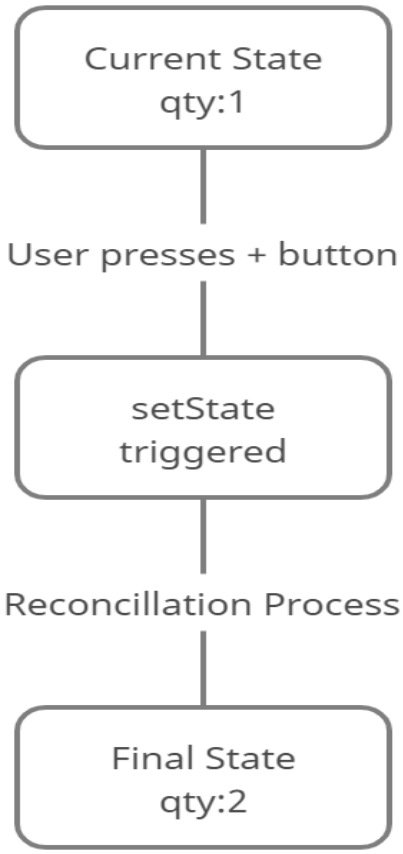
Which form should be chosen: When and Why?

Form 1: This should be chosen when we don't change a property with respect to the **previous state**. For example, if we want to change the title from Mobile-Phone to Phone so we can directly use form 1 in the following way:

```
console.log('this',this.state)
// // setState form 1
this.setState({
    title:'Phone'
});
```

Form 2: This form should be used when changing a property; we need its previous state.

How does setState work?



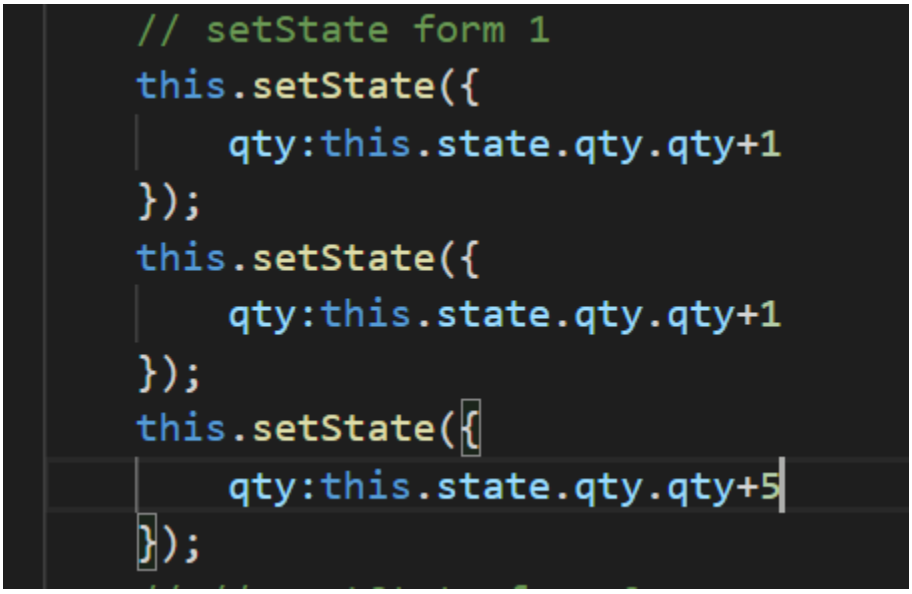
This is actually kicking off a process that React calls **reconciliation**. The reconciliation process is the way React regenerates the DOM by making changes to the object in terms of state change. When a **setState ()** request is triggered, React creates a new tree that contains reactive elements in the component (along with the updated state). This tree is used to determine how the material of the quantity should change in response to the transformation of the state by comparing it with the elements of the previous tree. React knows what changes will be **implemented** and will only update parts of the DOM where necessary.

Batching

What is Batching?

Batching is a feature in React that **combines** state updates. No matter how many setState calls you perform, they will be merged into a single call rendering a **single result**. The single result rendered depends upon the setState form.

For Form 1:



```
// setState form 1
this.setState({
    qty:this.state.qty.qty+1
});
this.setState({
    qty:this.state.qty.qty+1
});
this.setState({
    qty:this.state.qty.qty+5
});
```

In this form, all setState forms are merged and React uses the last object into consideration and changes according to it. Like in the above example, React will merge all three calls and take the last object as the final one and increase quantity by five as in the previous object quantity is increased by 5.

For Form 2:

```
this.setState((prevState) => {
    return{
        qty:prevState.qty+1
    }
});
this.setState((prevState) => {
    return{
        qty:prevState.qty+1
    }
});
this.setState((prevState) => {
    return{
        qty:prevState.qty+1
    }
});
```

React will create a queue in this form and will enqueue setState calls one by one in that queue and render a single output while dequeuing. Like In the above example, let's assume that:
Current State =1
Now first setState call will be enqueued and will update the current state to 2
Current State =2
Now second setState call will be enqueued and will update the current state to 3
Current State =3
Now third setState call will be enqueued and will update the current state to 4
Current State =4
So while dequeuing, the calls will be merged and the last current state, that is, 4, will be rendered.

Why is Batching important?

- To reduce the number of calls.
- For optimised performance

Callback Function

What is CallBack Function?

A **callback** function is a function passed into **another function** as an argument, which is then invoked inside the outer function to complete some **routine or action**. For example:

As the setState function is an **asynchronous** call for **event handlers,** we are unaware of when this function completes its call. To know when the function call is completed, we use this callback function which returns the correct value of the property of the state object after the function completes its call.

Here is a code demonstration for syntax:

```
//  setState form 2
this.setState((prevState) => {
    return{
        qty:prevState.qty+1
    }
},() =>{
    console.log('this.state',this.state)
} );
```

Condition Where Batching Fails

setState function is not an **asynchronou**s call every time. It is a **synchronous** call for AJAX and Promises. So, when we call the setState function for AJAX or promises, batching fails and calls are rendered the number of times they are called. Let's illustrate with an example:

```
testing () {
  const promise = new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('done');
    }, 5000);
  })

  promise.then(() => {
    // setState acts like a synchronus call
    this.setState({ qty: this.state.qty + 10 });

    this.setState({ qty: this.state.qty + 10 });

    this.setState({ qty: this.state.qty + 10 });

    console.log('state', this.state);
  });
}
```

Here in the code above, we created a testing function that makes a promise call with a **setState** function called thrice, which updates the qty value by 10. The output for this code comes out to be 31 because:
For first call qty: 1+10 =11
For second call qty: 11+10 =21
For third call qty:21+10=31
Here we can see that the setState function is called thrice and it is a synchronous call. Hence, the concept of batching fails here.

Summarising It

Let's summarise what we have learnt in this module:

- Learned about setState function and its two forms.
- Learned about batching and how it works.
- Learned about callback function.
- Why and when batching fails.

Some References:

- More information about setState function : https://reactjs.org/docs/state-and-lifecycle.html
- More information about Batching: https://react-redux.js.org/api/batch