React Anti-patterns

1. **Do not use anonymous functions or .bind() in component props**

If we have a Parent component which has a handleClick function which we need to pass it to the Child component with .bind() and arrow function.

***.bind()***

When we are using .bind() while passing handler function to **Child component**

**Concept:** Whenever you use .bind() it’ll return a new function which will be bound to the object that we pass to the .bind.

class Parent extends React.Component {  
  
 **handleClick() {  
 console.log("clicked");  
 }**  
   
 render() {  
 return (  
 <div>  
 <Child handler={**() => this.handleClick()**} />  
 // or if we use .bind()  
 <Child handler={**this.handleClick.bind(this)**} />  
 </div>  
 );  
 }  
}class Child extends React.Component { shouldComponentUpdate(nextProps) {  
 return nextProps.handler !== this.props.handler  
 } render() {  
 console.log("child component rendered");  
 return (  
 <button onClick={this.props.handler}>Child Component</button>  
 );  
 }  
}

**What happens?** Whenever Parent component re-renders, the Child component will also be rendered unnecessarily.

**Why?** **Since .bind() creates a new function each time it is run, this method would lead to a new function being created every time the render function executes.**

Therefore Parent component will return a new function as prop and for React it means that props for the Child component has changed. That’ll trigger a re-render on the Child component which is un-necessary.

1. **Anonymous function**

When we passing**anonymous functions as props to Child component.**

**What happens?** Whenever Parent component re-renders, the Child component will also be rendered unnecessarily.

**Why?**

Because we are passing anonymous functions, React will always re-render Child component since it receives a new anonymous function as a prop **which it is unable to compare to the previous anonymous function (since they are both anonymous)** which will cause the Child component to re-render because it’ll think it’s props has changed, and it’s the same case with the .bind() method also.

On the other hand, passing in a reference to the method like onClick={this.handleClick} lets React know when nothing has changed so it does not unnecessarily re-render. However, the anonymous function is sometimes unavoidable as when we need to pass in an argument available only in the context: onClick={() => {this.handleClick(argHereOnly)}} .

<https://medium.com/@muralikv/do-not-use-anonymous-functions-or-bind-in-component-props-8e835e1cddec>

<https://medium.com/@User3141592/react-gotchas-and-best-practices-2d47fd67dd22>

1. **Alternate to .bind(): Anonymous Function**

An alternative to binding (using .bind()) is to pass an anonymous function to wrap around the method. This, however has performance issues around re-rendering.

For example:

class ButtonComponent extends React.Component {

constructor(props) {

super(props)

this.state = {

message: 'Invoked from inside anonymous function!',

}

}

handleClick() {

console.log(this.state.message)

}

render() {

return

<button onClick**={() => this.handleClick()}** />

}

}

**How this works?**

Arrow function here with event handler is passing down execution context referring to class. That is why handleClick() is able to access this. But this has disadvantage that it impacts the performance by causing unnecessary re-rendering.

1. **Component Name**

Component name should start with capital letter otherwise React throws error.

## **Demo**

<MyComponent>  
 <app /> // Will not work :(  
</MyComponent><MyComponent>  
 <App /> // Will work!  
</MyComponent>

## **Problems**

If you create a component app and render it using JSX as <app label="Save" />, React will throw an error.

https://miro.medium.com/max/1152/1*xCB4cI255tVV41NvIozL7g.png

1. **Props in initial State**

Assigning props to this.state

import React, { Component } from 'react'

class MyComponent extends Component {

constructor(props){

super(props);

this.state = {

**someValue: props.someValue,**

};

}

}

**Problems**

The constructor or (getInitialState) is called **only at the time of component creation**. That is, constructor is invoked only once. Hence, when you change the props next time, the state won’t be updated and will retain its previous value.

**Solutions**

You can use this pattern if you want a specific behaviour. That is,**you want the state to be *seeded*by the values of props only once**. The state will be managed internally by the component.

In other cases, you can use componentWillReceiveProps lifecycle method to keep the state and props in sync, as shown here.

import React, { Component } from 'react'

class MyComponent extends Component {

constructor(props){

super(props);

this.state = {

someValue: props.someValue,

};

}

componentWillReceiveProps(nextProps){

if (nextProps.inputValue !== this.props.inputValue) {

this.setState({ inputVal: nextProps.inputValue })

}

}

}