**8 Mistakes every junior React developer makes -** [Kushan Madhusanka](https://kusham1998.medium.com/?source=post_page-----7ceace725d36--------------------------------) Aug 7, 2022

Every programmer who learns React makes a huge amount of mistakes in the learning process. Sometimes they don’t even know that they are making those mistakes. If you become fluent in React you need to avoid these mistakes and code according to the best practices. So, I thought to show the errors that you guys doing and help to make them right. Before that, we need to know what we are gonna deal with.

*What is React JS?*

React is a javascript library developed by Facebook in 2011. Most people misunderstand it as a framework. It’s an extremely components base library. It allows developers to build fast user interfaces for websites and applications. Let’s see some key reasons why we use React.

1. High performance reinforced by virtual DOM.
2. Reusability of components.
3. Vast community.
4. The power of Flux and Redux.
5. Since it uses JSX, anyone can learn it easily who knows HTML.
6. Unique react Hooks

Then, let’s talk about what are the mistakes most developers make and how to correct these mistakes.

**Mistakes related to updating a state**

I’m gonna tell you about two scenarios in which you guys make mistakes. Think you have a state called number in your component.

const [number, setNumber] = useState(0);

Now, you are going to update that state and access it in the next line.

setNumber(5);  
console.log(number);

You will think that this will console 5. But it doesn’t. It will console the previous value as 0 .

The reason?

You should know thatsetState function (here setNumber) is an asynchronous function. Since that function is the asynchronous controller will not hold until it updates. It will go to the next line and console the value of number (It will console 0 since the number is not updated 5 yet). So, If you are writing another statement using the updated value of numberbelow that setNumner statement, your logic will be completely wrong.

In class components, we have a way to access the updated value right after updating it. But for functional components, we can’t use this method.

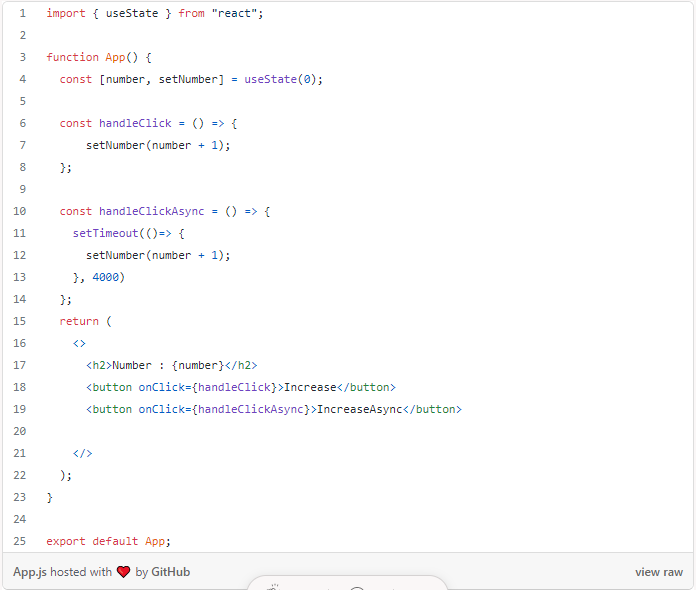
you can call a callback function with setState to access the updated value.

this.setState({ number: 5 }, () => console.log(this.state.number));

Now, it will console 5 . This is the full code segment.



Let’s see another mistake you are doing when updating a state. Think about a scenario like this. You have a variable called numberand two buttons to increase the value of this variable. One button is a normal button that increases the number by 1. Other button is increasing the number asynchronously. The error occurs with that button. This is the code for the scenario most developers writes.



The Problem?

There is nothing wrong with that Increase button. But If we click IncreaseAsync button once and then click Increase Button 5 times before 4 seconds number will increase to 5. But after 4 seconds number will be 1. The reason for that is in handleClickAsync function when the number is updating, setNumber function get the value of the number when the IncreaseAsync button is clicked. At that moment, value of number was 0. So, After 4 seconds number became 1.

We can overcome this problem setting setNumber function like this. Then, it will get the current value of the number to update.

setNumber((prevSate)=> prevSate + 1);

After correction full code will be like this.



You can put setNumber in handleClickas it was if you want. That will not make a difference.

**How to initialize a state correctly [For Objects]**

This is also a common mistake that most junior developers make. They initialize a state without a default value. For a single integer or string, this is not much of a problem. But if you are initializing an object you must mention the default values. Otherwise, you will get an error when accessing the property of that object. Let’s get an object called user .

If you are initializing it like this.

const [user, setUser] = useState();

After that somewhere else you assign a value to it.

setUser({  
 name: "kushan",  
 age: 24,  
 friends: ["john", "Anne", "will"]  
});

After, If you are accessing the object properties like this you will get a error.

<h2>name : {user.name}</h2>

The error will say this:

Text

Description automatically generated

You can do two changes to make this write.

1. Add && before access

<h2>name : {user && user.name}</h2>

2. Add ?. when accessing

<h2>name : {user?.name}</h2>

But, I recommend you to add a default value when initialization. You can add expected properties with an empty value.

const [user, setUser] = useState({  
 name: "",  
 age: "",  
 friends: [],  
});

**Updating an object**

Here we are going to talk about how to update an object in a proper way. Let’s take the previous user object with initial values.

const [user, setUser] = useState({  
 name: "kushan",  
 age: 24,  
 friends: ["john", "Anne", "will"],  
});

Add an input field with a button to change the nameof the userobject. I have displayed the object properties separately.



Graphical user interface, application

Description automatically generated

First of all, I will show some wrong ways developers do this.

setUser(input);orsetUser({name : input});

This will replace the object and assign the input we enter here. That is not what we want. This is how it updates correctly.

setUser({...user, name : input});orsetUser((prevState) => ({ ...prevState, name: input }));

**Create an object instead of separate states**

In every web application, we use objects. Some developers define separate states for the property of an object. It takes lots of time and reduces the code readability. Think, we have a user in our application and we create a form to get user details. what are you going to use to store these user details ?

In separate states like this:

const [fName, setFName] = useState("");  
const [lName, setLName] = useState("");  
const [email, setEmail] = useState("");  
const [age, aetAge] = useState("");  
const [city, setCity] = useState("");  
const [gender, setGender] = useState("");  
const [password, setPassword] = useState("");

Nooo…You should create this using one state:

const [user, setUser] = useState({  
 fName: "",  
 lName: "",  
 email: "",  
 age: "",  
 city: "",  
 gender: "",  
 address: "",  
 password: "",  
});

**Create a single onChage function for all inputs in a form**

Let’s take the above scenario. We can create a form to get that user details. After how you are going to create onChange functions for those input fields.

<input type='text' name="fName" />  
<input type='text' name="lName" />  
<input type='text' name="email" />  
<input type='text' name="age" />  
<input type='text' name="city" />  
<input type='text' name="gender" />  
<input type='text' name="address" />  
<input type='password' name="password" />

Are you gonna create separate onChnage functions?

onChange={(e) => setUser((prevState)=>({ ...prevState, fName: e.target.value }))}onChange={(e) => setUser((prevState)=>({ ...prevState, lName: e.target.value }))}onChange={(e) => setUser((prevState)=>({ ...prevState, email: e.target.value }))}....likewise

This is not good. You can use a better way to do this. Create a one onChange function for all input fields. You need to add the same property name of the user to the name property of the input tag.

const handleChange = (e) => {  
setUser((prevState) => ({...prevState,[e.target.name]:e.target.value }));  
};

Here full code for this:



**Direct DOM manipulation**

Have you written code like this before:

function App() {  
 const handleClick = () => {  
 const menu = document.querySelector(".menu");  
 menu.classList.add("color");  
 }  
 return (  
 <>  
 <div className="menu">Sample Text</div>  
 <button onClick={handleClick}>Click</button>  
 </>  
 );  
}

The problem?

This is very common among new developers. We Shouldn't do direct DOM manipulation in react. Instead of accessing DOM elements and adding classes to them, we can use state in our react component for this purpose. It will get harder to test, harder to debug, and harder to reason out if the application mixes up the state inside of the DOM.

Here, a possible solution for this. I have done the task without using document.querySelector .

import { useState } from 'react';  
import './App.css'function App() {  
 const [isColored, setIsColored] = useState(false);  
 const handleClick = () => {  
 setIsColored(true);  
 }  
 return (  
 <>  
 <div className={isColored ? "color": "" + "menu" }>Sample Text</div>  
 <button onClick={handleClick}>Click</button>  
 </>  
 );  
}

**useState vs useReducer (When to use?)**

Does it a constant to use useState everywhere when we need a state? The answer is no. Sometimes we have a better option instead of using useSate. It’s useReducer . Of course, there is no rule that you should use useReducer instead of useState or it’s not a mistake if you still use it. But in some specific situations, useReducer is more advantageous.

For example, think you have a more complex object that has different properties like arrays, and nested objects.

const [post, setPost] = useState({  
 title: "",  
 description: "",  
 date: "",  
 category: "",  
 image: [],  
 tag: [],  
 owner: {  
 name: "",  
 email: "",  
 }  
});

Then, it’s better to use a reducer because when you update different properties, it’s gonna be really messy here. Because all elements here are not strings or numbers like in the previous examples. So we cant update them in one function. So, creating a useReducer and using different actions for each property, might be more useful here. Since It’s better to use useReducer .

**Be careful about React-derived states**

To understand this I have created a small scenario. Think we have a state called classRooms and it’s an array of objects.

const [classRooms, setClassRooms] = useState([  
 { id: 1, className: "Sons of Sun", studentCount: 12 },  
 { id: 2, className: "Blooming Volcanoes", studentCount: 10 },  
 { id: 3, className: "Pillaging Pirates", studentCount: 20 },  
 { id: 4, className: "Ping Bombs", studentCount: 8 },  
]);

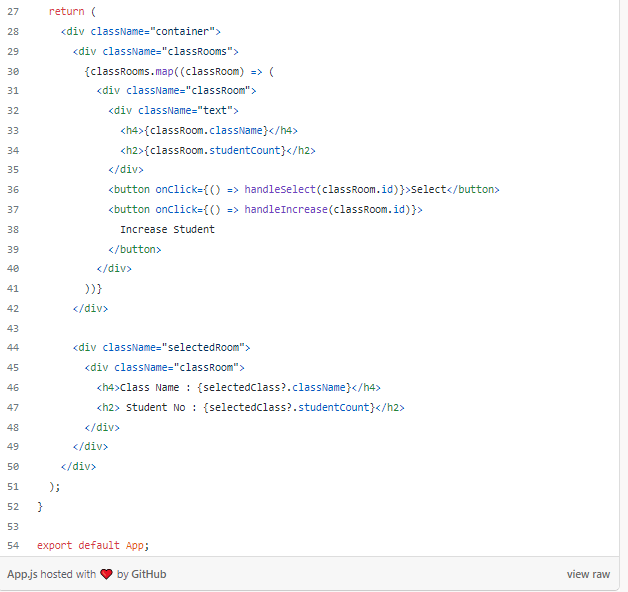
What we are gonna do is, we will display this array on the screen and give a selection button and an increase button (to increase students). The selected class also will be displayed.

Graphical user interface, application

Description automatically generated

This is the code. But I have done it the wrong way.





The problem?

This will increase the student count and select the relevant class as expected. But If you try to increase the student count after selecting the classroom, the student count will be increased. But student count in the selected area will not be updated. The way we have implemented the code is the reason for that. Most developers do this error. How we are going to fix this ?

You can store the only id of the selected classroom in the state. Then take a variable called selectedClassID and assign the selected classroom to it.

const [selectedClass, setSelectedClass] = useState(null);// not thisconst [selectedClassID, setSelectedClassID] = useState(null);//do like this

Declare and assign a value to selectedClassID

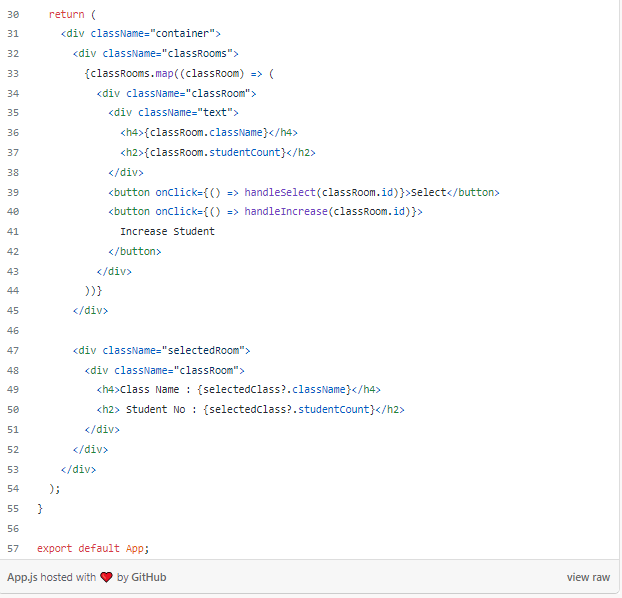
const selectedClass = classRooms.find((classRoom) => classRoom.id === selectedClassID);

Next, change the handleSelect function

const handleSelect = (id) => {  
 setSelectedClassID(id);  
};

This is the full code after correction:





So, this is the end of this article. I have discussed 8 points that will make mistakes. Most beginner developers do these wrong ways all the time. Therefore, I think this will help you to correct your mistakes and go to the next level.