Step 2: Buttons

# Introduction:

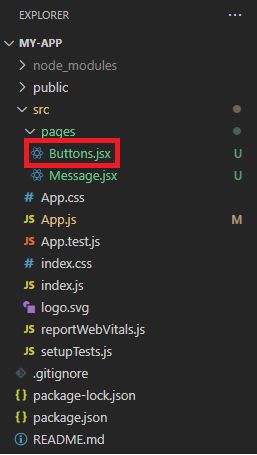
New features in this exercise:

* [useState hook](https://www.w3schools.com/react/react_usestate.asp)
* [onClick event handler](https://www.w3schools.com/JSREF/event_onclick.asp)

Now that you have a general understanding of how components work, we will create our second one which contains a couple of buttons and more advanced logic. In this step we’re also introducing **useState** and **onClick** features.

# Button component:

Start by creating a file called **Buttons.jsx** and importing React as usual. Also create base to your component, like shown below:



import React from 'react';

export const Buttons = () => {

    return (

        <div>

        </div>

    );

}

We’ll start by adding and rendering one normal HTML button **inside** our components div element. It will have no purpose yet.

<button>Button</button>

Nothing special here, lets move to the App.js and **import the component**. Just like in the previous step, render the Buttons component. It should now be visible on your browser.

Graphical user interface, text, application, chat or text message

Description automatically generated

By now your App.js file should look something like this:

import React from "react";

import { Message } from "./pages/Message";

import { Buttons } from "./pages/Buttons";

function App() {

    return (

        <div>

            <div>

                <Message />

                <Buttons />

            </div>

        </div>

    );

}

export default App;

# useState introduction:

Let’s move back to the Buttons.jsx file. Now we’re going to add a **counter** to our button that counts the number of clicks. To do so, we’re going to need **useState hook**. **Import useState** to your file using the statement below:

import { useState } from "react";

What is useState?

The React **useState hook** allows us to **track a state** in a function component. State generally refers to data or properties that need to be tracked in an application, useState can keep track of strings, numbers, boolean values, arrays, objects, and any combination of these.

useState has 2 values, the first one is the **current state** and the second is the function that is used to **update the state**, usually named as the first value with set in front of it (Example **color**, **setColor**). The value of the first state comes from an **initial value** that is located **at the end of the row**.

const [color, setColor] = useState(“”)

function Example () {

    setColor(“blue”);

}

In the example above, we’ve **set** the value of the color state to an empty string. In the function we use the setColor function to update our color state to blue.

# Adding useState to the component:

Now that we have learned a bit about useState, we can use it to **add a function** to our Buttons component. The usual convension is to put the useState statements on **top of the component**. Let’s simply name the values to count and setCount, **initial value** will be **0**.

Now we’ll make a function inside the component to **update the state**, let’s name that to countClicks. This function will be as simple as adding 1 to our count state with every execution.

import React from 'react';

import { useState } from 'react';

export const Buttons = () => {

    const [count, setCount] = useState(0);

    function countClicks() {

        setCount(count + 1);

    }

    return (

        <div>

            <button>Button</button>

        </div>

    );

}

Now, this function should work, but the countClicks function is **never called**, so we won’t see it anywhere. To display the count of clicks we have to add the function to our button with something called **onClick**.

# Adding onClick to the component:

What is onClick?

**onClick** is an **event handler** that allows you to **pass a function** to a component, which will be **executed when it’s clicked**. onClick is a basic feature of button element so it will not be imported like we did with useState.

<button onClick={function}>Button</button>

In the example above you’ll see how to pass a function to your button using onClick.

Now we’ll use the onClick event handler to add our countClicks function to our button like shown below:

<button onClick={countClicks}>Button</button>

Now our button clicks should be tracked, however at the moment we have nothing to render the number with, so let’s add that next. We’ll do this by simply **rendering the count state in our button**.

<button onClick={countClicks}> Clicked: {count} times </button>

Now you should see the count change **with every click**.

Graphical user interface, text, application, chat or text message

Description automatically generated

# Adding a reset function:

Next, we’ll add something to reset the count with. Because of useState, we can do that very, very easily, we’ll just use **the same state** that we used to update the number of clicks in our countClicks function. Let’s name our reset function resetCount. And just as we updated our state in the previous function, we’ll do the same in this one, only exception being **setting the count to 0** instead of adding one to it.

function resetCount() {

    setCount(0);

}

To execute this function, we have to add a button and pass it the function with onClick like before.

<button onClick={resetCount}>Clear count</button>

Now you should see the reset button on your browser, it should reset the count next to our first button.

It should look something like this:

Graphical user interface, text, application, chat or text message, website

Description automatically generated Graphical user interface, text, application, email, website

Description automatically generated

# Adding a second button with separate counter and a shared reset:

Now to learn a bit more about useState, let’s add a second button. This will be exactly like the previous one, but each button will have their own counter. To do so we need to add new useState values to the new button. Let’s just name them simply count2 and setCount2.

export const Buttons = () => {

    const [count, setCount] = useState(0);

    const [count2, setCount2] = useState(0);

    function countClicks() {

We will also need another function to count the clicks with, exactly like the first one. Again, you can simply name it countClicks2 and repeat the code with our new values, like shown below:

    function countClicks2() {

        setCount2(count2 + 1);

    }

Add a **new button** that we can pass the new function to:

<button onClick={countClicks2}>Button</button>

And render the clicks:

<button onClick={countClicks2}> Clicked: {count2} times </button>

Now, on your browser you should have two identical buttons that count their own clicks. But you’ll see that the reset button only works on the first one. Let’s make the reset button **work on both buttons** at the same click, so we’ll have counters that update separately but reset together.

We will do this simply by adding one line to our resetCount function, and that would be to also **set our second count to 0.**

    function resetCount() {

        setCount(0);

        setCount2(0);

    }

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Now we should have two separate counters, and a shared reset. Hope you got the hang of useState! **See you in the next exercise!** 😊