Step 6: Rendering data submitted by a form

# 

# Introduction:

In this exercise we’re going to create a component that receives data submitted through a form and **renders the content in a list**. The data is stored in an array. This is a **two-part** component. In the next exercise we will add buttons to clear and delete individual rows from the list. But for now, we’ll just focus on the rendering of the items.

# Form:

Start by creating a file called **List.jsx**, create a base to the component and import React and useState:

import React from "react";

import { useState } from "react";

export const List = () => {

    return (

        <div>

        </div>

    )

}

**Remember to import and add a route for the component in the App.js file and a link in the Router.jsx file so we can see our progress during the exercise.**

Let’s start by adding a form, create a new div element and add a normal HTML form with a **text input field** and a **submit button**:

import React from "react";

import { useState } from "react";

export const List = () => {

    return (

        <div>

            <div>

                <form>

                    <input type="text" required/>

                    <input type="submit" value="Submit"/>

                </form>

            </div>

        </div>

    )

}

Graphical user interface, text, application

Description automatically generated

You should now have rendered the simple **form and a button**.

# Adding arrays using useState:

Next, we will create our array that we will submit the data to. Data being the **string** that we write in the forms input field. We’re going to do it with useState, let’s name the state content, **initial state** will be an **empty array**:

const [content, setContent] = useState([]);

**Remember that useState usually goes on top of everything else, right under the component definition!**

We will also need a **second state**, which will be the **input fields data**. Let’s name its state to inputValue, initialstate will be **an empty string**:

const [inputValue, setInputValue] = useState('');

Let’s add a couple of things to our input field. We’ll add a placeholder and a value. Set the placeholder to say something like “**Type here**”. The placeholder will display text on our **empty** input field:



For the value, let’s set it as our inputValue state. Also make the field **required** to submit the form.

<input

    type="text"

    placeholder="Type here.."

    value={inputValue}

    required

/>

# onSubmit, preventDefault and spread:

Now, add an **onSubmit** event to the form so we can **add functions that execute when submitted**. We will also add a preventDefault method here, so submitting the form **doesn’t refresh** the page. If you’re not familiar with the JavaScript method, you can learn more about it [here](https://www.w3schools.com/jsref/event_onsubmit.asp).

<form onSubmit={(event) => {

event.preventDefault()

\*\*We will add our functions here\*\*

}}>

We’ll use our content state here, so our component will update it when the form is submitted. We’re going to use JavaScript spread operator to update our state.

Spread syntax (...):

The JavaScript **spread syntax** can be used to add elements from an array to function arguments or array elements. For example, in a case like this where we want to combine two arrays, we can't do it by just placing two arrays inside an array.

const arrayA = [1, 2, 3];

const arrayB = [4, 5, 6];

const combinedArray = [arrayA, arrayB];

This would result in the combinedArray to be an array that holds the two original arrays like this:

combinedArray = [[1, 2, 3], [4, 5, 6]];

This is where we can use the spread syntax to add all the elements to the combined array:

const arrayA = [1, 2, 3];

const arrayB = [4, 5, 6];

const combinedArray = [...arrayA, ...arrayB];

Now the combinedArray will look like this: [1, 2, 3, 4, 5, 6]. So, in the above example the spread syntax effectively does the same this as doing this:

const arrayA = [1, 2, 3];

const arrayB = [4, 5, 6];

const combinedArray = [arrayA[0], arrayA[1], arrayA[2], arrayB[0], arrayB[1], arrayB[2]];

So, the spread syntax "spreads" the elements from the original array to the combinedArray array rather than adding the whole array element.

# Updating states:

Because we want to add multiple items to our array we **can’t** just set the content state to something because it will **replace the whole array** with every execution, leaving **one item** every time.

<form onSubmit={(event) => {

    event.preventDefault()

    setContent(["New item"]);

    \*\*NOT LIKE THIS\*\*

}}>

When working with useState arrays, it is important to understand that useState does not render the changes **if you do not create a completely new array** and replace the old one with setState. That’s why we can’t add items to our array with JavaScript’s push method since it only pushes a new item to the original array.

So, we will create a new array called **temp** and using the spread operator, copy both, the inputValue string and our existing content array to it. After that we will use the setContent to update.

const temp = [inputValue, ...content];

setContent(temp);

Now we have added the item from the input field to the array instead of setting the whole state to one item on every submit.

Let’s also reset the input field here with the setInputValue state, so every time we submit the form the field will be cleared.

<form onSubmit={(event) => {

    event.preventDefault()

    const temp = [inputValue, ...content];

    setContent(temp);

    setInputValue('');

}}>

We have to update the input field a bit. We have now set the value to our inputValue state, but it is an empty string. We haven’t set the state to anything yet so we’ll do that next, we will use something called onChange handler.

The onChange event in React detects when the value of an input element **changes**, after that it will execute something. In our case, it will set the inputValue state to whatever we input in to our form (**e.target.value is the string that the user inputs in the input field**).

onChange={e => setInputValue(e.target.value)}

Your form should now look like this:

<div>

    <form onSubmit={(event) => {

        event.preventDefault()

        const temp = [inputValue, ...content];

        setContent(temp);

        setInputValue('');

    }}>

        <input

            type="text"

            placeholder="Type here.."

            value={inputValue}

            onChange={e => setInputValue(e.target.value)}

            required

        />

        <input type="submit" value="Submit"/>

    </form>

</div>

# Mapping the array:

Now our form should be good and working, we just don’t see the list yet.

To show the items in the array we will use the .map method that we learned in the previous exercise. Let’s add squirrely brackets and inside those, the content map, the map will be placed **under the forms submit button**:

{

    content.map((item) => (

    )

)}

**You have to use the brackets every time you want to write JavaSript inside our HTML divs!**

This time, we will display the data in a **list**. Add a normal HTML list with one list item and display the content with h2 tags:

content.map((item) => (

    <ul>

        <li>

            <h2>{item}</h2>

        </li>

    </ul>

)

Now, in your browser you should see a list that updates every time you submit a new string to it. Notice that the list will show the most recent item first and oldest at the bottom:

Graphical user interface, text, application

Description automatically generated

In the next exercise we’ll be staying on the same component. We’ll add an individual **item remover** and a **reset list function**. **See you there!** 😊