



MISSION READY

DARE TO **DEVELOP**

Events in React, Conditional Rendering

Reuben Simpson

React events

- In HTML, the two main ways to listen to events are:
 - `onclick` attribute
 - `addEventListener()` method
- In React, these events work a little differently



React events

- Handling events with React elements is very similar to handling events on DOM elements. There are some syntax differences:
 1. React events are named using *camelCase*, rather than lowercase.
 2. With JSX you *pass a function* as the *event handler*, rather than a string.

```
<button onclick="activateLasers()">  
  Activate Lasers  
</button>
```

HTML

```
<button onClick={activateLasers}>  
  Activate Lasers  
</button>
```

React/JSX



Event listeners

- While it's still very possible to apply event listeners to elements in React, we usually don't
- Usually we apply events inline, meaning in the element itself
e.g.

```
<button onClick="updateCount()">add count</button>
```
- The above example shows this implementation in HTML but in JSX it looks slightly different
e.g.

```
<button onClick={updateCount}>add count</button>
```
- Notice that the event name is in camel case here and the function name is between curly brackets and does **not** include the ()



preventDefault() recap

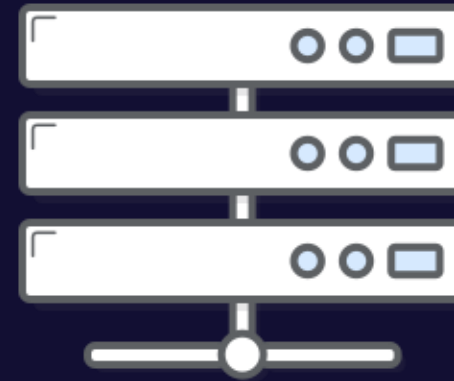
- There are a few HTML elements that have default functionality when they are used. For example,
 - When you click the **link**, it takes us to the **href**, like we would expect.
 - When someone submits a **form**, the submit event will fire and initiates its submission to the server, based on the form's submit action.
- The **preventDefault()** method is used to *cancel* an event if it is cancelable.
 - It says that the default action should **NOT** be taken as it normally would be.





FORM ELEMENTS

(FRONTEND HTML & CSS)

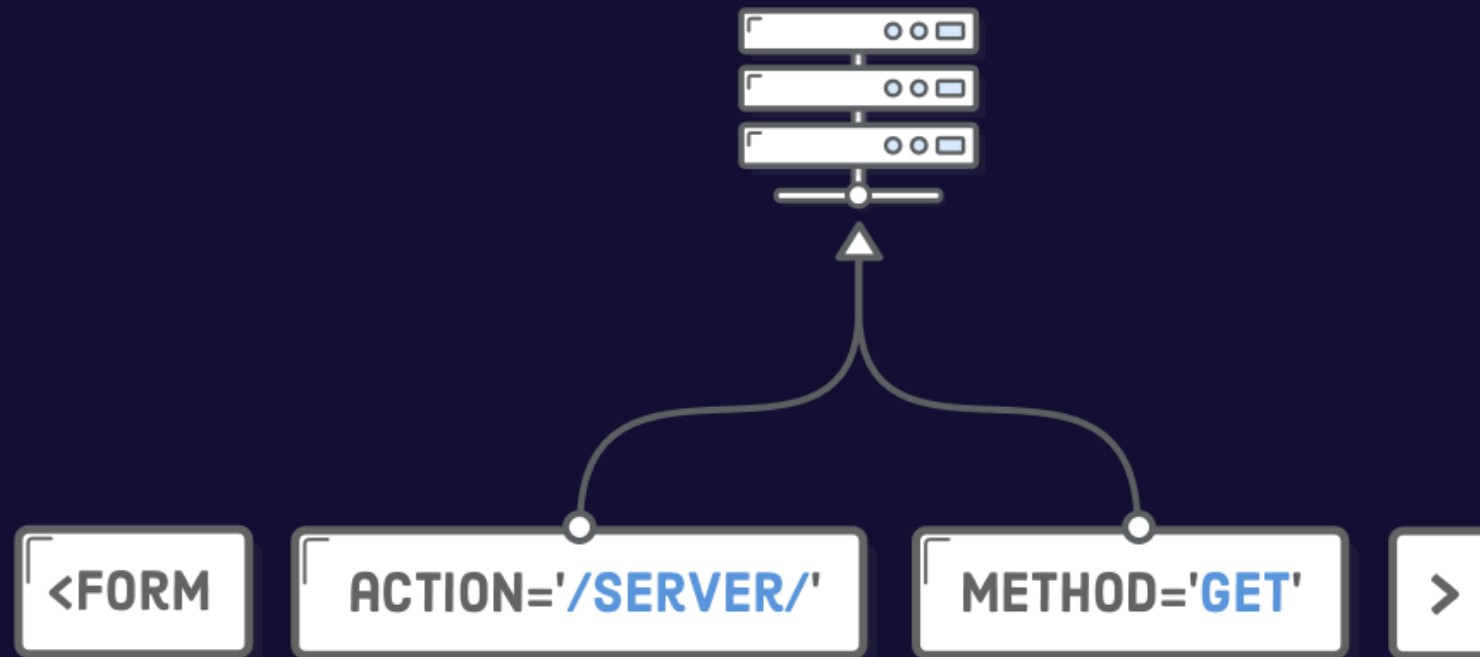


FORM PROCESSING

(BACKEND SERVER)



BACKEND SERVER



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Prevent Default Example</title>
  </head>
  <body>
    <a id="googleLink" href="https://www.google.com"> This should go to google.com </a>

    <script>
      "use strict";
      const googleLink = document.getElementById("googleLink");
      googleLink.addEventListener("click", (event) => {
        event.preventDefault();
        console.log("You have clicked the link. But nothing happens..hihihihi");
      });
    </script>
  </body>
</html>
```



Preventing default behaviour in React

- Let's look at how this could look in React
 - Camel case onSubmit
 - **e** (event object) automatically passed to the onSubmit function

```
export default function FormExample() {  
  
  const handleSubmit = (e) => {  
    e.preventDefault();  
  }  
  
  return (  
    <div>  
      <form onSubmit={handleSubmit}>  
        <input />  
        <input type="submit" />  
      </form>  
    </div>  
  )  
}
```



Input elements

- In HTML we used `getElementById` to select an input element then used `element.value` to get the text inside the input field.
- In React we do something slightly different
- We create a variable in the state that gets updated every time something is typed into the input field.

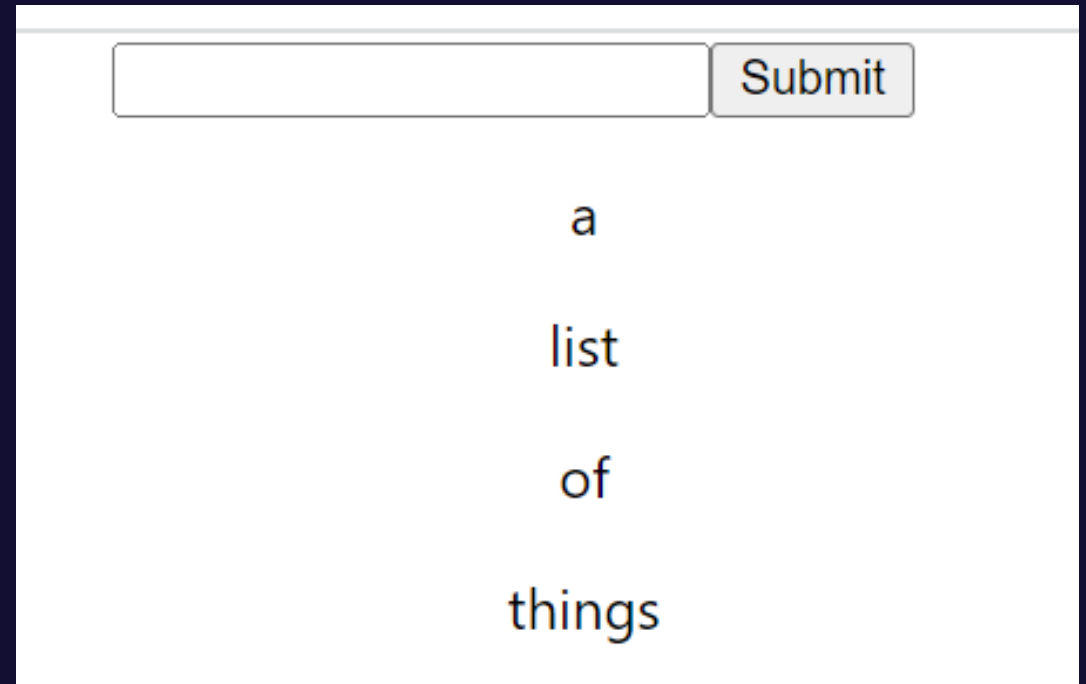
Copy this code...

```
export default function Form() {  
  const [val, setVal] = useState('')  
  
  const handleChange = (e) => setVal(e.target.value)  
  
  return (  
    <div>  
      <label>Input field</label>  
      <div><input onChange={handleChange} /></div>  
      {val}  
    </div>  
  )  
}
```



Exercise 1

- Let's use what we've learnt to create a form element
- This element should have an *input field* where the user can type something in
- When a user clicks **submit**, the text from the input field should be shown below the input box



A screenshot of a web form. At the top, there is a white rectangular input field with a thin border. To its right is a button labeled "Submit" in a light gray box. Below the input field, the text "a list of things" is displayed in a simple, black, sans-serif font, arranged in four lines: "a", "list", "of", and "things".



Updating Exercise 1

- Let's update our **form** from exercise 1 so that every time the form is **submitted** the value of the text in the form is added to **a list** that is shown below the input box
- We can start by adding our two state variables
 - **val** is going to store the value from the input field
 - **inputList** is going to be an array that will store our list of values

```
const [inputList, setInputList] = useState([]);  
const [val, setVal] = useState('');
```



Updating Exercise 1 continued...

- Next, we have our **handleChange** function which will update the *val* value, whenever we type something into our input field

```
const handleChange = (e) => setVal(e.target.value)
```

- Then we have our **handleSubmit** function, which will update our input list with the new value

- Since we can't update the value of `inputList` directly, we can assign it to a temporary variable to ***make a copy of it***
- We can then push a new value to that temp variable
- Then we can set the `inputList` to temp

```
const handleSubmit = (e) => {  
  e.preventDefault();  
  const temp = inputList;  
  temp.push(val);  
  console.log(temp);  
  setInputList(temp);  
}
```



Updating Exercise 1 continued...

- Now we can add our JSX
- We have a form with an onSubmit that will trigger **handleSubmit**
- An **input element** that has an onChange that will update the value of our input
- Then, our list. inputList is an **array** so we are able to use a **.map** to iterate through the array
 - For each item in our array we will wrap it with `` tags, so that we can render the list

```
return (  
  <>  
    <form onSubmit={handleSubmit}>  
      <div><input onChange={handleChange} /></div>  
      <button type="submit">Submit</button>  
    </form>  
    <ul>  
      {inputList.map(item => (  
        <li>{item}</li>  
      ))}  
    </ul>  
  </>  
)
```



Updating Exercise 1 continued...

- At this point you might've noticed that this kind of works but is a bit weird when it comes to updating our list.
- This is for a very important reason.
- This line in our handleSubmit function isn't exactly right
 - When we assign temp to have a value of inputList we are **not making a copy** of the array, temp in this case is **just referencing** the original array
 - When doing a **.push()** on our state variable this won't update our state which means the DOM also won't be updated

```
const handleSubmit = (e) => {  
  e.preventDefault();  
  const temp = inputList;  
  temp.push(val);  
  setInputList(temp);  
}
```



... spread operator

- To help us out here, we can use the spread operator which looks like ...
- This makes a copy of the array which we can then modify and then set to be the value of the state.
- This will look like

```
const handleSubmit = (e) => {  
  e.preventDefault()  
  const temp = [...inputList, val]  
  setInputList(temp)  
}
```

- Now everything should work as expected



Accessing the previous state

- Another way we could achieve this is by accessing a state variable's previous state value.
- Every state updating function gives us access to the previous state value, if we want it.
- We get access to it by passing in a function and specifying a parameter which represents the previous state.
- For example:

```
const handleSubmit = (e) => {  
  e.preventDefault()  
  setInputList(function (prevState) {  
    return [...prevState, val];  
  })  
}
```



Conditional Rendering

- Conditional rendering in React is the concept of only allowing elements to be rendered on the screen if certain conditions are met
 - We set conditional statements on parts of our application that we want to show/hide so that those elements will only be shown if the condition passes
- Let's look at an example



Conditional Rendering example

- Here we are using a ternary statement to show or hide some content

```
export default function CondRendering() {  
  const [showContent, setShowContent] = useState(false)  
  
  const changeContent = () => setShowContent(!showContent)  
  
  return (  
    <div>  
      <div>{showContent ? 'Congrats you found the hidden content!' : 'Click the button'}</div>  
      <button onClick={changeContent}>Click here</button>  
    </div>  
  )  
}
```



Conditional rendering with if statements

- We can also use if statements to do conditional rendering although you can't use them directly in the JSX.
- Let's look at an example

```
import { useState } from "react";
export default function IfElseRendering() {
  const [content, setContent] = useState("Click the button");
  const [showContent, setShowContent] = useState(false);

  const changeContent = () => {
    if (showContent) {
      setContent("Click the button");
      setShowContent(false);
    } else {
      setContent("Congrats you found the hidden content!");
      setShowContent(true);
    }
  };

  return (
    <div>
      <div>{content}</div>
      <button onClick={changeContent}>Click</button>
    </div>
  );
}
```



Logical &&

- We know that the && operator is used as an AND in conditions, but in our JSX it can act as an if statement.
- Let's look at the last example, we wanted to render 'Congrats you found the hidden content!' when the condition is true and nothing if it is false, then we can use the && operator to help us

```
export default function CondRendering() {  
  const [showContent, setShowContent] = useState(false)  
  const changeContent = () => setShowContent(!showContent)  
  return (  
    <div>  
      <div>{showContent && 'Congrats you found the hidden content'}</div>  
      <button onClick={changeContent}>Click</button>  
    </div>  
  )  
}
```



Exercise 2

- Create **two buttons**, one labeled 'Cat' and another labeled 'Dog'. Depending on the button clicked, the page should either **show a photo** of a cat or a dog (or both).



Exercise 3

- Add 1 more button for 1 more animal of your choice, to your code from Exercise 2. Make it so that it will only show one photo at a time.



why

won





MISSION READY

www.missionreadyhq.com

**DARE TO
DEVELOP**

Thank you

Reuben Simpson