Trainer guide
Examples and exercises (React)
React-7 – Handling events

Contents

React-7a	Example: Events in JSX
React-7b	Exercise: Event handling

React-7a | Example: Events in JSX

Link to environment

This example demonstrates how to add event handlers to JSX elements.

App.jsx

- React allows us to write code that responds to HTML events.
- For example, the `onclick` event, which fires when an element is clicked:
    - *Press the "Click me!" button and see the message in the console.*
- To respond to events, we can add an event handler prop to any JSX element, as long as that element corresponds to a regular HTML tag (think buttons, inputs, divs, imgs, etc. – any HTML tag).
- For example, let's look at the <Button> component:
    - This renders a <button> element.
    - The <button> element has an "onClick" handler, that we can use to handle the click event.
    - The value of the "onClick" prop is always a function. In this case, we pass the `handleClick` function. This function will be called when the button is clicked.
    - Note that when we pass the function as a prop, we have to make sure not to put brackets after the function name. If we did this, the function would be executed immediately when the component is rendered, but it would not be called on the `onClick` event. (*Demonstrate this)*
        - Instead, we pass the function itself as the prop value. When the event fires, React will run whatever function we've passed for us.
    - Also notice that in JSX, events are written in camel case, meaning that the first letter of each word is capitalised, except for the first word which starts with a lowercase letter. This is in contrast to native DOM events which are written in all lowercase. Like how we must use "className" instead of "class" in JSX, this is one of the slight differences between HTML and JSX syntax.
- Let's now look at the <TextInput> component.
    - The input element has an onFocus handler that will call the `handleFocus` function. This function will log the event to the console.
    - *Focus on the "Focus me!" input field and look at what appears in the console.*
    - To understand what's going on here, you need to know that under the hood, React uses something called "synthetic events". When the user focuses on this input element, a native DOM "focus" event is fired. This event is wrapped in a synthetic event object by React,

which normalises the event handling behaviour, ensuring it is consistent across different browsers. The synthetic event is passed as an argument to the handler function.
    - You can see that the event is passed as an argument to the `handleFocus` function which then logs the event to the console.
    - In the console, you can see that the event object is a `SyntheticBaseEvent`. This object holds all kinds of information about the event, like the timestamp and the target element. Because this is a synthetic event whose interface is defined by React, this will always be consistent across all browsers running our React app.
- Finally, let's look at the <HoverTarget> component.
    - This example is just to demonstrate that we can use arrow functions when writing event handlers.
    - Just as we passed pre-defined functions in the last two examples (`handleClick` and `handleFocus`), we can also define an anonymous function on the fly using arrow functions.
    - For example, here we've created an arrow function that logs a message to the console. Whenever the `onMouseOver` event fires, the arrow function will be called.
- Now that we've looked at event handling, we can start making our way towards adding interactivity to our React applications.

React-7b | Exercise: Event handling

Link to environment

This exercise gives learners practice on writing event handlers in JSX.

App.jsx

- Here is a very simple app that just contains a button.
- When the button is clicked, confetti is supposed to appear on the screen.
- But at the moment, the button is broken.

Exercise

- Your task is to make the confetti button work properly.
- There is an “addConfetti” function that is imported in App.jsx.
- You will need to make it so that the addConfetti function is called when the button is clicked.

Solve the exercise live before moving on.

Solution reference