



## Section 8: Working with Refs & Portals

21 September 2024 15:22





# Refs & Portals


### Advanced DOM Access & Value Management

- ▶ Accessing **DOM Elements** with **Refs**
- ▶ **Managing Values** with Refs
- ▶ **Exposing API Functions** from Components
- ▶ **Detaching DOM Rendering** from JSX Structure with **Portals**

```
export default function Player() {  
  const playerName = useRef();  
  
  const [enteredPlayerName, setEnteredPlayerName] = useState(null);  
  
  function handleClick() {  
    setEnteredPlayerName(playerName.current.value);  
    playerName.current.value = '';  
  }  
  
  return (  
    <section id="player">
```

React is about writing declarative code. It's not about directly manipulating the DOM yourself. Instead, you wanna let React do that.

But here we are crossing the line little bit. here we are just reading a value. We are not changing anything in the DOM.



but here we are changing the code. I mean directly we are manipulating the DOM. Therefore we are basically violating the rule or that idea that React should handle all those DOM interactions. For a use case like this, for clearing a input which is not connected to any other state or anything like that, you can definitely consider writing code like this.

## **Refs vs State Values**

```

Player.jsx 1, M X
1  import { useState, useRef } from 'react';
2
3  export default function Player() {
4    const playerName = useRef();
5
6    //const [enteredPlayerName, setEnteredPlayerName] = useState(null);
7
8    function handleClick() {
9      //setEnteredPlayerName(playerName.current.value);
10     playerName.current.value = '';
11   }
12
13   return (
14     <section id="player">
15       <h2>Welcome {playerName.current ? playerName.current.value : 'unknown entity'}</h2>
16       <p>
17         <input
18           ref={playerName}
19           type="text"

```

ACADE  
MIND

# State vs Refs

## State

Causes component re-evaluation (re-execution) when changed

Should be used for values that are directly reflected in the UI

Should not be used for “behind the scenes” values that have no direct UI impact

## Refs

Do not cause component re-evaluation when changed

Can be used to gain direct DOM element access (→ great for reading values or accessing certain browser APIs)

## Closing the Modal via the ESC (Escape) Key

The `<dialog>` element allows website visitors to close the opened dialog by pressing the **ESC** (Escape) key on their keyboard.

Currently, this will **not** trigger the `onReset` function though (unlike closing the dialog with a

button click).

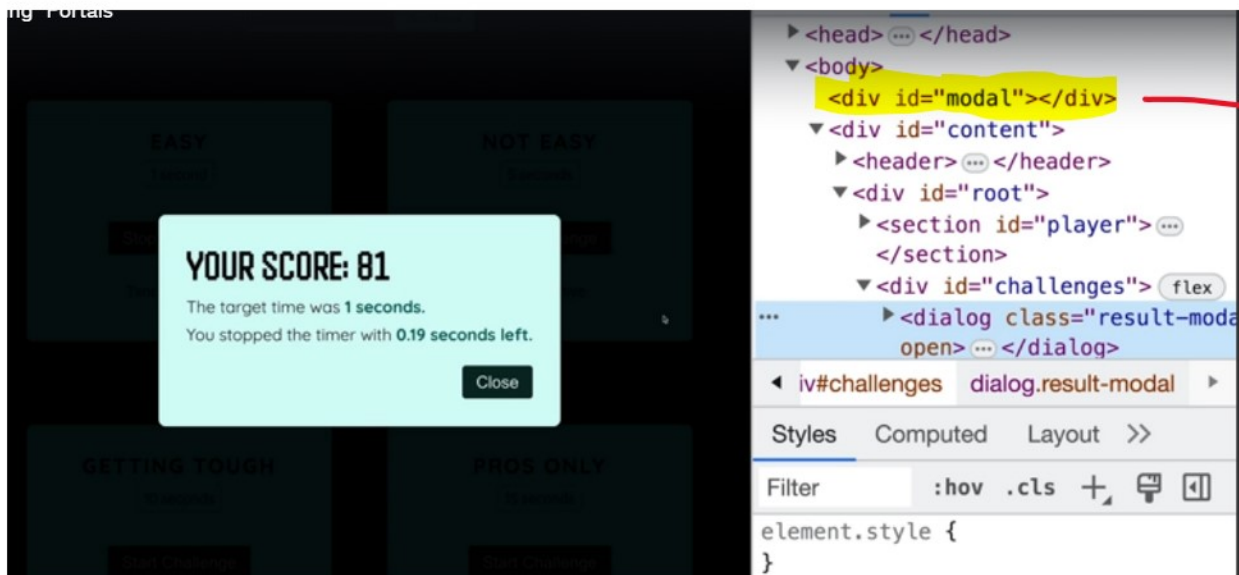
To make sure that `onReset` gets triggered when the dialog is closed via the escape key, you should add the built-in `onClose` prop to the `<dialog>` element and bind it to the `onReset` prop value.

Like this:

1. `<dialog ref={dialog} className="result-modal" onClose={onReset}>`
2. ...
3. `</dialog>`

From <https://www.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/39836564#learning-tools>

## Introducing & Understanding Portals



Now visually, this actually all works here. There's nothing wrong with it, but technically it would make more sense if this overlay element which visually sits on top of the entire page would be output directly inside of the body or maybe inside of this div which sits here with an ID of modal

which I did not add to the starting project by accident. And it would make sense to have the dialogue on such a higher level because that would map its visual appearance to its location in the HTML structure which can be better for accessibility reasons and which can also help you avoid styling problems. Here we don't have any, but a deeply nested element could be hidden by other elements above it in certain circumstances. And therefore we might wanna control and kind of output