Hema Coding School

YouTube Link: https://www.youtube.com/@HemaCodingSchool

useRef() in React:

useRef is a hook in React that allows you to create a mutable object that persists across re-renders. It is often used to access DOM elements directly or to keep a mutable variable that does not cause re-renders when changed.

```
import { useRef } from 'react';

const refContainer = useRef(initialValue);
```

initialValue: The initial value passed to useRef. This value will be assigned to the .current property of the ref object.

Accessing DOM Elements:

- You can use useRef to get a reference to a DOM element, which can be useful for manipulating or interacting with the element directly.

Storing Mutable Values:

- useRef can store any mutable value that you want to persist across renders without causing a re-render when the value changes.

```
function RefObject() {
   let h2Ref = useRef()
   let getRef =()=>{
      console.log("Ref");
      // document.getElementById('text').style.backgroundColor = "yellow";
      h2Ref.current.style.backgroundColor = "green";
   }
  return (
   <div className="train-info">
      <h2 id="text" ref={h2Ref}>This is Use Ref</h2>
      <button onClick={getRef}>Change</button>
   </div>
  )
}
```

When to Use useRef:

- When you need to interact directly with a DOM element.
- When you want to store a mutable value that does not need to trigger a re-render when changed.
- When you need to keep track of mutable values across renders without causing re-renders.

Hema Coding School

YouTube Link: https://www.youtube.com/@HemaCodingSchool

Controlled and Uncontrolled Components in React

In React, components can manage form data in two primary ways: controlled and uncontrolled components. Understanding the difference between these two approaches is essential for managing state and handling user input effectively.

Controlled Components:

Definition: A controlled component is a component where React controls the form data. The form data is handled by the state within the component, making the state the single source of truth.

Characteristics:

- **State Management:** The component's state holds the current value of the form element.
- **Real-time Updates:** Every change in the input value triggers a state update via an onChange event handler.
- **Single Source of Truth:** The value of the form element is always driven by the state, ensuring consistency.

```
import React,{useState} from 'react';
import "../../Component/ClassBased.css";


function MonthlySalesReport() {

   let [formData, setFormData] = useState({
      name:'',
      report:''
   });
   console.log(formData)

   let getReportData =(event)=>{
      event.preventDefault();
      console.log(formData)
      console.log(`Custumer name ${formData.name} and report value ${formData.report}`)

   }
  return (
    <div className="train-info">
        <form>
          <div>
             <input type="text" placeholder="Enter Name"
onChange={(e)=>setFormData({...formData,name: e.target.value})}/>
          </div>
          <div>
             <input type="number" placeholder="Enter Report Value"
onChange={(e)=>setFormData({...formData,report: e.target.value})}/>
```

```
        </div>
        <button onClick={getReportData}>Submit</button>
      </form>
        <div>
        <div>Name: {formData.name}</div>
        <div>Report: {formData.report}</div>

        </div>
   </div>
  )
}

export default MonthlySalesReport
```

Benefits:

- **Predictability:** The state always reflects the current value of the input, making the component's behavior predictable.
- **Validation:** Easier to validate form data since the state holds the input values.
- **Data Binding:** Simplifies data binding between the form and the component state.

Drawbacks:

- **Verbosity:** Requires more boilerplate code to manage state and event handlers.
- **Performance:** May cause performance issues with many form elements due to frequent re-renders

Uncontrolled Components:

Definition: An uncontrolled component is a component where the form data is handled by the DOM itself. React does not manage the form data directly, instead relying on refs to access the form values.

Characteristics:

- **Refs:** Use React.createRef or useRef to create references to the form elements.
- **Direct DOM Access:** The form data is accessed directly from the DOM using refs.
- **Less State Management:** Less reliance on React state, reducing the need for frequent state updates.

```
function SalesReport() {
      let nameRef = useRef();
      let reportRef = useRef();
   console.log(nameRef, reportRef)
   let getReport =(event)=>{
      event.preventDefault();
      console.log(`Custumer name ${nameRef.current.value} and report value
${reportRef.current.value}`)
   }
```

```
return (
  <div className="train-info">
    <h1>Un Controlled Component</h1>
    <form>
      <div>
        <input type="text" placeholder="Enter Name" ref={nameRef}/>
      </div>
      <div>
        <input type="number" placeholder="Enter Report Value" ref={reportRef}/>
      </div>
      <button onClick={getReport}>Submit</button>
    </form>
  </div>
 )
}
```

Benefits:

- **Simplicity:** Less code is needed since there are no state updates for each input change.
- **Performance:** Better performance for forms with many elements as it avoids frequent re-renders.

Drawbacks:

- **Validation:** Harder to perform validation since the form data is not stored in the state.
- **Testability:** More challenging to test as you need to interact with the DOM directly.

When to Use Which:

- **Controlled Components:** Use when you need real-time validation, dynamic input updates, or when the form data needs to be shared or manipulated frequently.
- **Uncontrolled Components:** Use for simple forms, especially when performance is a concern, or when form data does not need to be validated or manipulated in real-time.

Interview Questions:

Q1: What is useRef in React and how does it differ from useState?

A1: useRef is a hook that allows you to create a mutable object which persists for the lifetime of the component. It is often used to reference DOM elements or store mutable values that do not cause a re-render when updated. Unlike useState, which triggers a re-render of the component when the state changes, updating a useRef value does not cause a re-render.

Q2: What is a controlled component in React?

A2: A controlled component is a component where React controls the state of the form elements. The form data is handled by the component's state, making the state the single source of truth. Changes in the input elements trigger state updates via event handlers like onChange.

Q3: What is an uncontrolled component in React?

A3: An uncontrolled component is a component where the form data is handled by the DOM itself. React does not manage the form data directly, instead relying on refs to access the form values. This approach is useful when you want to avoid managing state for form inputs

Q4: What are the main differences between controlled and uncontrolled components in React?

A4:

- **State Management:** Controlled components use React state to manage form data, whereas uncontrolled components rely on the DOM to manage form data.
- **Event Handling:** Controlled components use event handlers (e.g., onChange) to update state, while uncontrolled components use refs to access form values.
- **Validation:** Controlled components make it easier to validate form data in real-time since the state holds the input values. Uncontrolled components require manual validation.
- **Code Complexity:** Controlled components may require more boilerplate code to manage state, while uncontrolled components are simpler but can be harder to validate and test.