Animesh Parab T2-T21 88

Assignment : 9

Aim : WAP to implement Refs in React.

Theory :

Refs in React are a way to directly access and interact with DOM elements or React components. They provide a way to reference a specific object or object in your application, so that you can read values, trigger actions, or perform other tasks that would be difficult to achieve using React's standard dataflow and state scheduling methods role can help in situations such as access .
Use comments for React refs:
1. DOM Access:
Often Refs are used to interact directly with DOM elements. For example, you can access input values, set focus, or trigger animations.
2. Managing Third Party Libraries:
When integrating React with third-party libraries that require direct DOM manipulation, refs are needed. For example, if you use a charting library or a video player, you can use refs to interact with their API.

1. **Animating Elements:**

   You can use refs to trigger animations by changing CSS classes or properties. This is useful when you need precise control over animations.

2. **Scrolling and Measuring Elements:**

   You can use refs to scroll to specific elements on the page or measure their dimensions. This is helpful for building features like smooth scrolling or lazy-loading content.

3. **Integration with Non-React Code:**
   If you need to integrate React with non-React code, such as legacy JavaScript or libraries like D3.js, you can use refs to bridge the gap between React components and external code.

4. **Forms and Input Validation:**
   Refs can be useful for form handling and input validation. You can access input elements directly to check and manipulate their values or apply custom validation logic.

Creating Refs:

You can create a ref using the **React.createRef()** method (for class components) or the **createRef** hook (for functional components).

Accessing Refs:

You can access the DOM element or React component associated with a ref by using the **current** property of the ref object. This property holds the reference to the underlying element or component. You typically access the ref inside lifecycle methods
(for class components) or within the functional component itself.

Forwarding Refs:

Sometimes, you may need to pass a ref from a parent component to a child component. This can be achieved using the "forwarding refs" technique. React provides the **forwardRef** function for this purpose. It allows you to pass a ref down the component hierarchy and access the child component's ref from a parent component.

Callback Refs:

Callback refs are another way to work with refs, especially in cases where you need to perform additional operations when a ref is set. Instead of using the **createRef** or **useRef** functions, you define a callback function that will be called when the ref is attached to an element or component.

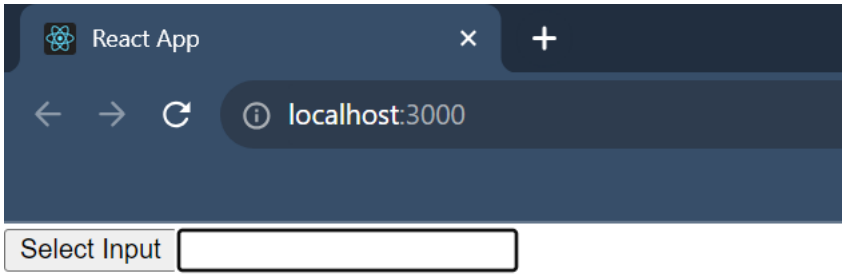
Code:

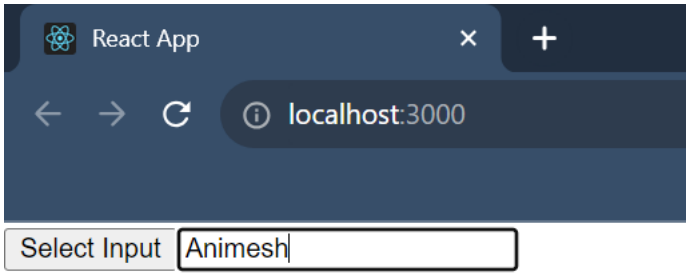
App.js

File Edit View

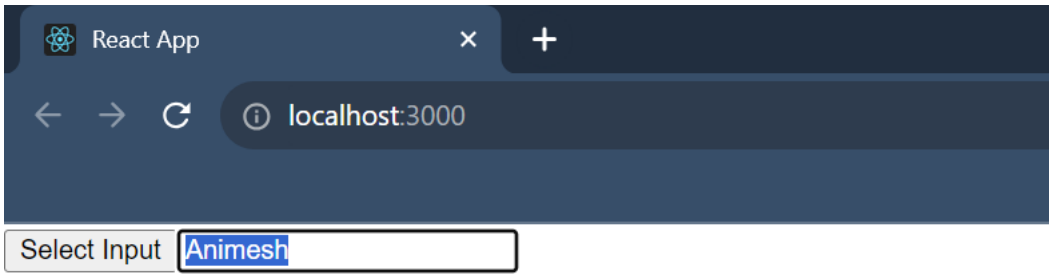
```
import React, { useRef } from 'react';

const MyComponent = () => {
        const formInput = useRef(null);


        const inputSelection = () => {
                const input = formInput.current;

                if (input.value) {
                        input.select();
                }
              };

                return (
                   <div>
                      <button type="button" onclick={inputSelection}>
                          Select Input
                      </button>
                      <input ref={formInput} />
                   </div>
                );
            };

export default MyComponent;
```

Output:





After clicking the button:



Conclusion :

We successfully demonstrated how React refs can be used to create, access, and manipulate DOM elements and components, showcasing their versatility within a React application.