

Aim:- Write a program to implement

- a. Create React refs
- b. How to access Refs
- c. Forward Refs
- d. Callback Refs.

LO Mapped:- LO5

Theory:-

React is a popular JavaScript library for building user interfaces, and it provides various tools and patterns to efficiently manage and manipulate the Document Object Model (DOM) elements. One of these tools is React Refs, which allow developers to access and interact with DOM elements directly. With the introduction of React Hooks, working with refs has become more accessible and intuitive. In this guide, we'll explore how to create and use React refs

effectively using functional components and hooks.

a. Creating React Refs:

React Refs are objects that provide a way to access and interact with DOM elements directly. In functional components, you can create refs using the `useRef` hook.

```
const myRef = useRef(null);
```

The `useRef` hook initializes the `myRef` variable with the `current` property set to `null`. You can attach this ref to a DOM element by assigning it as a `ref` attribute in the JSX.

b. How to Access Refs:

Once you've created a ref, you can access and manipulate the corresponding DOM element. To access the DOM element, you can use the `.current` property of the ref. For example:

```
const element = myRef.current;  
if (element) {  
  // Access and manipulate the DOM element  
}
```

It's important to check if the element exists before accessing and manipulating it because the `current` property may initially be `null`.

c. Forward Refs:

React allows you to forward refs from a parent component to a child component. This is particularly useful when you want to access a child's DOM element from a parent component. To create a forward ref, use the `React.forwardRef` function:

```
const ChildComponent = React.forwardRef((props, ref) => {  
  // JSX for child component  
  return <input ref={ref} />;  
});
```

In this example, the `ChildComponent` accepts a `ref` parameter and attaches it to the `input` element. You can then use this forwarded ref in a parent component:

```
function ParentComponent() {  
  const childRef = useRef(null);
```

// Attach ref to the child component

```
return <ChildComponent ref={childRef} />;  
}
```

Now, `childRef.current` refers to the `input` element inside `ChildComponent`.

d. Callback Refs:

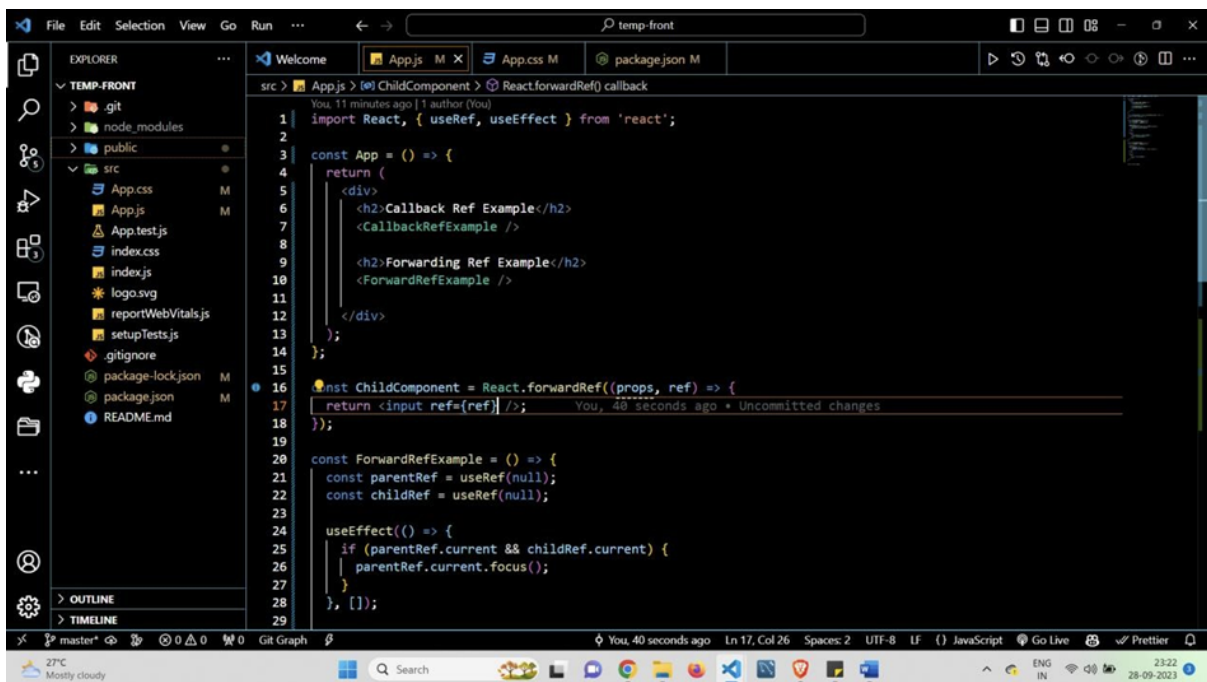
Callback refs are another way to work with refs in React. Instead of using the `useRef` hook, you define a function that receives the DOM element as an argument and stores it as a variable. Here's how to create and use a callback ref:

```
let myRef = null;  
const setMyRef = (element) => {  
  myRef = element;  
};
```

In this example, `setMyRef` is a callback function that receives the DOM element as an argument and assigns it to the `myRef` variable. You can then use `myRef` to access and manipulate the DOM element. Callback refs are typically defined inside the functional component.

React Hooks and functional refs provide a flexible and powerful way to work with DOM elements in your React applications. Whether you're creating refs, accessing DOM elements, forwarding refs to child components, or using callback refs, these techniques empower you to build dynamic and interactive user interfaces with ease.

Code :-

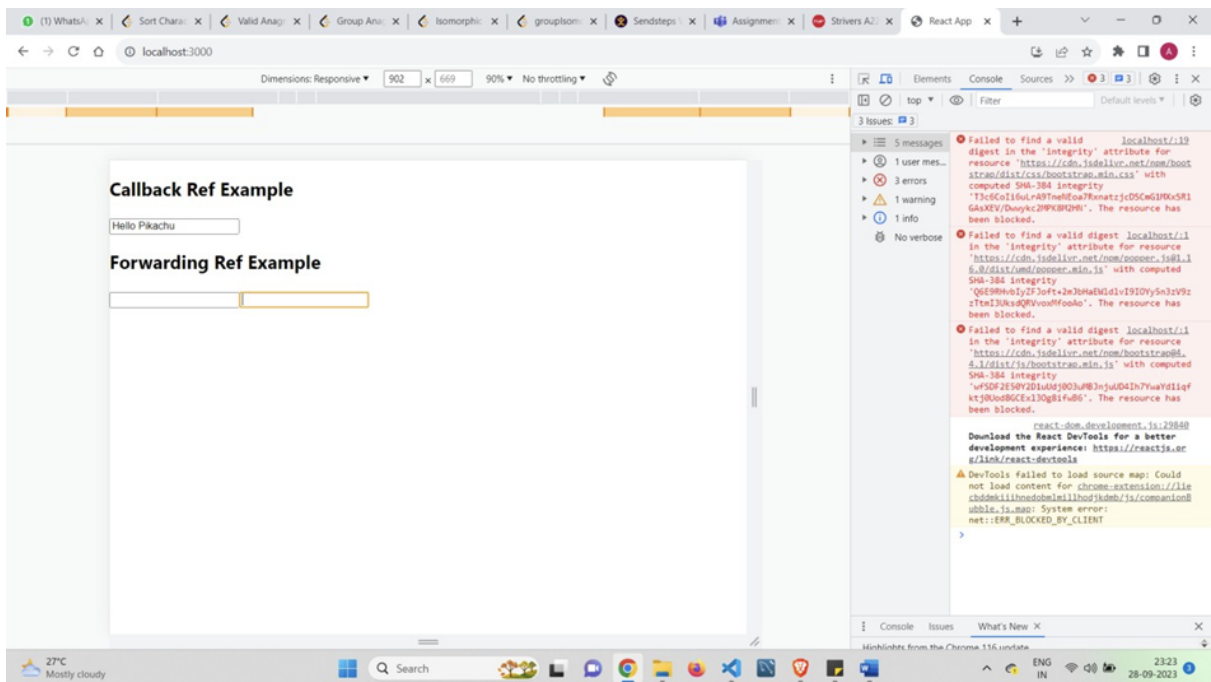


```
src > App.js > ChildComponent > React.forwardRef() callback  
You, 11 minutes ago [1 author (You)]  
1 import React, { useRef, useEffect } from 'react';  
2  
3 const App = () => {  
4   return (  
5     <div>  
6       <h2>Callback Ref Example</h2>  
7       <CallbackRefExample />  
8  
9       <h2>Forwarding Ref Example</h2>  
10      <ForwardRefExample />  
11    </div>  
12  );  
13 };  
14  
15  
16 const ChildComponent = React.forwardRef((props, ref) => {  
17   return <input ref={ref} />;  
18 });  
19  
20 const ForwardRefExample = () => {  
21   const parentRef = useRef(null);  
22   const childRef = useRef(null);  
23  
24   useEffect(() => {  
25     if (parentRef.current && childRef.current) {  
26       parentRef.current.focus();  
27     }  
28   }, []);  
29 }
```

The screenshot shows a VS Code editor with a project named 'temp-front'. The Explorer panel on the left shows the file structure: .git, node_modules, public, src (containing App.css, App.js, App.test.js, index.css, index.js, logo.svg, reportWebVitals.js, setupTests.js), .gitignore, package-lock.json, package.json, and README.md. The main editor displays the code for App.js, which is a ChildComponent using React.forwardRef(). The code defines a callback function that returns a JSX element with a child component and an input. It also defines a CallbackRefExample component that uses useRef and useEffect to update the input value to 'Hello Pikachu' when the component is mounted.

```
27 }
28 }, []);
29
30 return (
31   <div>
32     <ChildComponent ref={childRef} />
33     <input ref={parentRef} />
34   </div>
35 );
36
37
38 const CallbackRefExample = () => {
39   const inputRef = useRef(null);
40
41   useEffect(() => {
42     if (inputRef.current) {
43       inputRef.current.value = 'Hello Pikachu';
44     }
45   }, []);
46
47   return (
48     <div>
49       <input ref={inputRef} />
50     </div>
51   );
52 };
53
54
55 export default App;
```

Output :-



Learnt about react refs about how they can be created , how they can be accessed , about callback refs and about forward refs , implemented all these concepts practically and gained more knowledge about it .