ReactJS - State Management using React Hooks

React introduces an entirely new concepts called React Hooks from React 16.8. Even though, it is a relatively new concept, it enables React functional component to have its own state and lifecycle. Also, React Hooks enables functional component to use many of the feature not available earlier. Let us see how to do state management in a functional component using React Hooks in this chapter.

What is React Hooks?

React Hooks are special functions provided by React to handle a specific functionality inside a React functional component. React provides a Hook function for every supported feature. For example, React provides *useState()* function to manage state in a functional component. When a React functional component uses React Hooks, React Hooks attach itself into the component and provides additional functionality.

The general signature of *useState()* Hook is as follows −

```
const [<state variable>, <state update function>] = useState(<initial value>);
```

For example, state management in clock component using Hooks can be done as specified below −

```
const [currentDateTime, setCurrentDateTime] = useState(new Date());
setInterval(() => setCurrentDateTime(new Date()), 1000);
```

Here,

- *currentDateTime* − Variable used to hold current date and time (returned by *setState()*
- *setCurrentDate()* − Function used to set current date and time (returned by *setState()*)

Create a stateful component

Let us recreate our clock component using Hooks in this chapter.

First, create a new react application, *react-clock-hook-app* using *Create React App* or *Rollup bundler* by following instruction in *Creating a React application* chapter.

Next, open the application in your favorite editor.

Next, create *src* folder under the root directory of the application.

Next, create *components* folder under *src* folder.

Next, create a file, *Clock.js* under *src/components* folder and start editing.

Next, import *React library* and React state Hook, *setState*.

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

Next, create *Clock* component.

```
function Clock() {
}
```

Next, create state Hooks to maintain date and time.

```
const [currentDateTime, setCurrentDateTime] = useState(new Date());
```

Next, set date and time for every second.

```
setInterval(() => setCurrentDateTime(new Date()), 1000);
```

Next, create the user interface to show the current date and time using *currentDateTime* and return it.

```
return ( <div><p>The current time is {currentDateTime.toString()}</p></div> );
```

Finally, export the component using the code snippet −

```
export default Clock;
```

The complete source code of the *Clock* component is as follows −

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

function Clock(props) {
   const [currentDateTime, setCurrentDateTime] = useState(new
Date());    setInterval(() => setCurrentDateTime(new Date()), 1000);
return (
      <div><p>The current time is {currentDateTime.toString()}</p></div>
   );
}
export default Clock;
```

Next, create a file, *index.js* under the *src* folder and use *Clock component*.

```
import React from 'react'; import
ReactDOM from 'react-dom'; import
Clock from './components/Clock';

ReactDOM.render(
   <React.StrictMode>
      <Clock />
</React.StrictMode>,
document.getElementById('root') );
```

Finally, create a *public* folder under the root folder and create *index.html* file.

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="utf-8">
      <title>Clock</title>
   </head>
   <body>
      <div id="root"></div>
      <script type="text/JavaScript" src="./index.js"></script>
   </body>
</html>
```

Next, serve the application using npm command.

```
npm start
```

Next, open the browser and enter *http://localhost:3000* in the address bar and press enter. The application will show the time and update it every second.

```
The current time is Wed Nov 11 2020 10:10:18 GMT+0530 (India Standard Time)
```

The above application works fine. But, *setCurrentDateTime()* set to execute every second has to be removed when the application ends. We can do this using another Hook,*useEffect* provided by React. We will learn it in the upcoming chapter (*Component life cycle*).