COURSEWARE

Professional Skills
Agile Fundamentals
Jira
Git
Databases Introduction
Java Beginner
Maven
Testing (Foundation)
Java Intermediate
HTML
CSS
Javascript
Spring Boot
Selenium
Sonarqube
Advanced Testing (Theory)
Cucumber
MongoDB
Express
NodeJS
React
- Introduction
- JSX
- Babel
- Component Hierarchy
- Components
- Props
- Lifecycle
- State
- Lifting State
- Hooks
- React Routing

Hooks

Contents

Overview

In this module, we will be looking at Hooks in React.

What are Hooks?

Hooks are a new addition in React 16.8 which let you use state and other React features without writing a class - or *hook* into them.

This helps solve numerous problems encountered by the React team:

- Re-use of stateful logic between components
- Splitting of complex components into smaller functions based on a relationship
- Simplifies syntax of class components to functions – which have to be transpiled anyway

State Hooks

State Hooks allow for the addition of a local state to a function component.

It requires useState to be used (this is the hook); it takes an argument of the initial state and returns the current state with a function to update it.

This is similar to `this.setState` in a class; calling the setting function causes a re-render of the component.

```
import { useState } from `react`;
const ExampleWithManyStates = () => {
    // Declare state variables as needed
    const [name, setName] = useState(``);
    const [count, setCount] = useState(10);
    const [myObj, setMyObj] = useState({ myKey1: `myVal1`, myKey2: true});
    //…
```

Effect Hooks

Effect Hook
The Effect Hook lets you perform side effects in function components, replacing the `ComponentDidMount`, `ComponentDidUpdate` and `ComponentWillUnmount` lifecycle methods.

Side effects include fetching data, subscriptions or manually changing DOM from React components.

This runs after React flushes changes to DOM – after every render, including the first.

It's declared inside a component so it can have access to props and state.

All of the clean-up is done by adding a return callback function.

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

const Counter = () => {
    const [count, setCount] = useState(0);
    useEffect(() => {
        // Replaces to CDM and CDU
        document.title = `Clicked ${count} times`;
        // Replaces CWU
        return(() => console.log(`Final: ${count}`));
    });
    return (
        <>
            <p>You have clicked the button {count} times</p>
            <button onClick={()=>setCount(count=>count+1)}>
            Click Me!
            </button>
        </>
    );
}
```

Other hooks can be used, including:

- useContext
- useReducer
- useCallback
- useMemo
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue

Tutorial

In this tutorial, we will look at how to add an item to a shopping list

1. Create a component called `Shopping`:

```
const Shopping = () => {
    return();
}
export default Shopping;
```

2. Import the 'useState' hook from React.
3. Create a state named `items` which has an initial state value of an empty array:

```
const [items, setItems] = useState([]);
```

4. Create another state called 'itemName' which has an initial state value of an empty string:

```
const [itemName, setItemName] = useState("");
```

5. In the return of the component, create a form which has an input text field and a submit button:

```
return(
    <form>
        <input type="text" name="Item" placeholder="Enter an item"/>
        <button type="submit">Add </button>
    </form>
);
```

6. Add a value to the input text field which is assigned to the `itemName` state, as well as `onChange` function that calls `setItemName` and assigns the value of the user input to the state:

```
return(
    <form>
        <input type="text" name="Item" placeholder="Enter an item" value=
{itemName} onChange={(e) => setItemName(e.target.value)}> />
        <button type="submit">Add </button>
    </form>
);
```

7. Create a function `addItem` which takes in an `event` parameter, then in the body of the function, prevent the default behaviour of the event, followed by adding to the `Items` array an object which contains an item id and item name:

```
const addItem = event => {
    event.preventDefault();
    setItems([...items, {id: item.length, name: itemName}]);
    setItemName("");
}
```

8. Assign the function to the form, which is to be called when the form is submitted:

```
return(
    <form onSubmit={addItem}>
        // Form elements
    </form>
)
```

9. After the form, in a list element, loop through the `items` array and print the item name to the screen:

```
return(
    <>
        //form

        <ul>
            {items.map(item => (
                <li key={item.id}>{item.name} </li>
            ))}
        </ul>
    </>
)
```

10. Import the component into `App.js` and run `npm start`

▶ Code

Exercises

1. Incorporate the `useState` hook and create a component that displays some text with a 'read more' link at the end, and will expand to show the rest of the text when the link is clicked.

   ▶ Solution

2. Create a component which has a name passed in as property then displays the name in the return of the component, incorporate a useEffect hook so that the document title updates only when the prop `name` changes:

   ▶ Solution