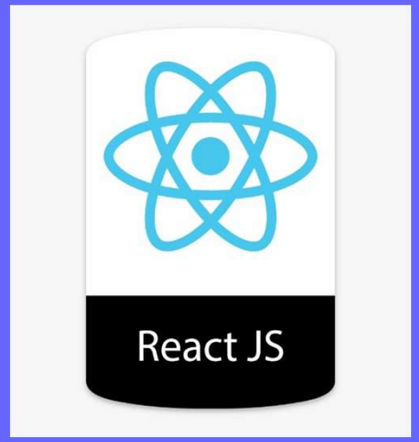
12

React Hooks

Objectives

After completing this lesson, you should be able to:

- React Hooks
- useState
- useEffect
- useContext



Introduction to React Hooks

- Hooks were added to React in version 16.8.
- Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

What is a Hook?

- Hooks allow us to "hook" into React features such as state and lifecycle methods.

```
import React, { useState } from "react";
import ReactDOM from "react-dom/client";

function FavoriteColor() {
  const [color, setColor] = useState("red");

  return (
    <>
      <h1>My favorite color is {color}!</h1>
      <button
        type="button"
        onClick={() => setColor("blue")}
      >Blue</button>
      <button
        type="button"
        onClick={() => setColor("red")}
      >Red</button>
      <button
        type="button"
        onClick={() => setColor("pink")}
      >Pink</button>
      <button
        type="button"
        onClick={() => setColor("green")}
      >Green</button>
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<FavoriteColor />);
```

- You must import Hooks from react.
- Here we are using the useState Hook to keep track of the application state.
- State generally refers to application data or properties that need to be tracke

Hook Rules

- There are 3 rules for hooks:

1. Hooks can only be called inside React function components.
2. Hooks can only be called at the top level of a component.
3. Hooks cannot be conditional

React useState Hook

- The React useState Hook allows us to track state in a function component.

- State generally refers to data or properties that need to be tracking in an application.

Import useState

To use the `useState` Hook, we first need to `import` it into our component.

At the top of your component, `import` the `useState` Hook.

```
import { useState } from "react";
```

Initialize `useState`

We initialize our state by calling `useState` in our function component.

`useState` accepts an initial state and returns two values:

- The current state.
- A function that updates the state.

Example:

Initialize state at the top of the function component.

```
import { useState } from "react";

function FavoriteColor() {
  const [color, setColor] = useState("");
}
```

Notice that again, we are destructuring the returned values from useState.

The first value, color, is our current state.

The second value, setColor, is the function that is used to update our state.

Read State

We can now include our state anywhere in our component.

Example:

Use the state variable in the rendered component.

```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function FavoriteColor() {
  const [color, setColor] = useState("red");

  return <h1>My favorite color is {color}!</h1>
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<FavoriteColor />);
```

Update State

Use a button to update the state:

```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function FavoriteColor() {
  const [color, setColor] = useState("red");

  return (
    <>
      <h1>My favorite color is {color}!</h1>
      <button
        type="button"
        onClick={() => setColor("blue")}
      >Blue</button>
    </>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<FavoriteColor />);
```

- To update our state, we use our state updater function.
- We should never directly update state. Ex: color = "red" is not allowed.

What Can State Hold ?

```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function Car() {
  const [brand, setBrand] = useState("Ford");
  const [model, setModel] = useState("Mustang");
  const [year, setYear] = useState("1964");
  const [color, setColor] = useState("red");

  return (
    <>
      <h1>My {brand}</h1>
      <p>
        It is a {color} {model} from {year}.
      </p>
    </>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

- The useState Hook can be used to keep track of strings, numbers, booleans, arrays, objects, and any combination of these!
- We could create multiple state Hooks to track individual values.

```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function Car() {
  const [car, setCar] = useState({
    brand: "Ford",
    model: "Mustang",
    year: "1964",
    color: "red"
  });

  return (
    <>
      <h1>My {car.brand}</h1>
      <p>
        It is a {car.color} {car.model} from {car.year}.
      </p>
    </>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

- we can just use one state and include an object instead!

Updating Objects and Arrays in State

- When state is updated, the entire state gets overwritten.
- What if we only want to update the color of our car?
- If we only called setCar({color: "blue"}), this would remove the brand, model, and year from our state.
- We can use the JavaScript spread operator to help us.

```jsx
import { useState } from "react";
import ReactDOM from "react-dom/client";

function Car() {
  const [car, setCar] = useState({
    brand: "Ford",
    model: "Mustang",
    year: "1964",
    color: "red"
  });

  const updateColor = () => {
    setCar(previousState => {
      return { ...previousState, color: "blue" }
    });
  }
```

```jsx
  return (
    <>
      <h1>My {car.brand}</h1>
      <p>
        It is a {car.color} {car.model} from {car.year}.
      </p>
      <button
        type="button"
        onClick={updateColor}
      >Blue</button>
    </>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

React useEffect Hooks

- The useEffect Hook allows you to perform side effects in your components.
- Some examples of side effects are: fetching data, directly updating the DOM, and timers.
- useEffect accepts two arguments. The second argument is optional.
- useEffect(<function>, <dependency>)

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom/client";

function Timer() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  });

  return <h1>I've rendered {count} times!</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Timer />);
```

- But Here It keeps counting even though it should only count once!
- useEffect runs on every render. That means that when the count changes, a render happens, which then triggers another effect.
- This is not what we want. There are several ways to control when side effects run.
- We should always include the second parameter which accepts an array. We can optionally pass dependencies to useEffect in this array.

1. No dependency passed:

```
useEffect(() => {
  //Runs on every render
});
```

Example

2. An empty array:

```
useEffect(() => {
  //Runs only on the first render
}, []);
```

Example

3. Props or state values:

```
useEffect(() => {
  //Runs on the first render
  //And any time any dependency value changes
}, [prop, state]);
```

Only run the effect on the initial render:

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom/client";

function Timer() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  }, []); // <- add empty brackets here

  return <h1>I've rendered {count} times!</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Timer />);
```

- A useEffect Hook that is dependent on a variable. If the count variable updates, the effect will run again:

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom/client";

function Counter() {
  const [count, setCount] = useState(0);
  const [calculation, setCalculation] = useState(0);

  useEffect(() => {
    setCalculation(() => count * 2);
  }, [count]); // <- add the count variable here

  return (
    <>
      <p>Count: {count}</p>
      <button onClick={() => setCount((c) => c + 1)}>+</button>
      <p>Calculation: {calculation}</p>
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Counter />);
```

React useContext Hook

React Context

- React Context is a way to manage state globally.

- It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

The Problem

- State should be held by the highest parent component in the stack that requires access to the state.

- To illustrate, we have many nested components. The component at the top and bottom of the stack need access to the state.

- To do this without Context, we will need to pass the state as "props" through each nested component. This is called "prop drilling".

Passing "props" through nested components:

```jsx
import { useState } from "react";
import ReactDOM from "react-dom/client";

function Component1() {
  const [user, setUser] = useState("Jesse Hall");

  return (
    <>
      <h1>{`Hello ${user}!`}</h1>
      <Component2 user={user} />
    </>
  );
}

function Component2({ user }) {
  return (
    <>
      <h1>Component 2</h1>
      <Component3 user={user} />
    </>
  );
}
```

```jsx
function Component3({ user }) {
  return (
    <>
      <h1>Component 3</h1>
      <Component4 user={user} />
    </>
  );
}

function Component4({ user }) {
  return (
    <>
      <h1>Component 4</h1>
      <Component5 user={user} />
    </>
  );
}
```

```
function Component5({ user }) {
  return (
    <>
      <h1>Component 5</h1>
      <h2>{`Hello ${user} again!`}</h2>
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Component1 />);
```

- Even though components 2-4 did not need the state, they had to pass the state along so that it could reach component 5.

The Solution

The solution is to create context.

- Create Context
- To create context, you must Import createContext and initialize it:

```
import { useState, createContext } from "react";
import ReactDOM from "react-dom/client";

const UserContext = createContext()
```

- We'll use the Context Provider to wrap the tree of components that need the state Context.
- Context Provider
- Wrap child components in the Context Provider and supply the state value.

```
function Component1() {
  const [user, setUser] = useState("Jesse Hall");

  return (
    <UserContext.Provider value={user}>
      <h1>{`Hello ${user}!`}</h1>
      <Component2 user={user} />
    </UserContext.Provider>
  );
}
```

- All components in this tree will have access to the user Context.

- Use the useContext Hook
- In order to use the Context in a child component, we need to access it using the useContext Hook.

- First, include the useContext in the import statement

```
import { useState, createContext, useContext } from "react";
```

Then you can access the user Context in all components:

```
function Component5() {
  const user = useContext(UserContext);

  return (
    <>
      <h1>Component 5</h1>
      <h2>{`Hello ${user} again!`}</h2>
    </>
  );
}
```

```jsx
import { useState, createContext, useContext } from "react";
import ReactDOM from "react-dom/client";

const UserContext = createContext();

function Component1() {
  const [user, setUser] = useState("Jesse Hall");

  return (
    <UserContext.Provider value={user}>
      <h1>{`Hello ${user}!`}</h1>
      <Component2 />
    </UserContext.Provider>
  );
}
```

```jsx
function Component2() {
  return (
    <>
      <h1>Component 2</h1>
      <Component3 />
    </>
  );
}

function Component3() {
  return (
    <>
      <h1>Component 3</h1>
      <Component4 />
    </>
  );
}
```

```jsx
function Component4() {
  return (
    <>
      <h1>Component 4</h1>
      <Component5 />
    </>
  );
}

function Component5() {
  const user = useContext(UserContext);

  return (
    <>
      <h1>Component 5</h1>
      <h2>{`Hello ${user} again!`}</h2>
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Component1 />);
```

Summary

In this lesson, you should have learned that:

- React Hooks
- useState
- useEffect
- useContext