React Fundamentals
React Hooks

DevelopIntelligence
A PLURALSIGHT COMPANY

Topics

- Introducing React Hooks
    - useState()
    - useEffect()
    - useRef()
    - useMemo()
    - useContext()
- Rebuild ToDo App / Jeopardy App using Hooks

Think, Discuss, and Share

When does the componentDidUpdate method run?

Think, Discuss, and Share

What is the difference between BrowserRouter and HashRouter?

React Hooks

What are Hooks?

What are Hooks?

- Hooks are a new feature introduced to React in version 16.8 (Feb 2019)
- Hooks allow functional React components to access state and other features previously restricted to classes
- Hooks *can* replace all use cases of classes, but classes aren’t going anywhere for the foreseeable future
- Hooks are really cool :)

React Hooks
useState

useState()

- Replaces use of **this.state** and **this.setState()**
- Returns a stateful value and a setter function to update that value using “array destructuring”
- Can use one or multiple state variables initialized by useState()

```
import { useState } from 'react'

function App() {
  const [input, setInput] = useState('')

  return (
    <div>
      <input
        value={input}
        onChange={(event) => setInput(event.target.value)}
      />

      <button onClick={() => console.log(input)}>
        submit
      </button>
    </div>
  )
}

export default App
```

React Hooks
useEffect

useEffect()

- Replaces **componentDidMount()**, **componentDidUpdate()**, and **componentWillUnmount()**

- Syntax to configure different lifecycle methods can be a bit obtuse (we'll go over it!)

```
import { useEffect } from 'react'

function App() {
  useEffect(() => {
    //runs on mount AND on every update


    return () => {
    // runs on unmount
  }
})

  return (
    <div>
      ...
    </div>
  )
}

export default App
```

React Hooks

useRef

useRef()

- Returns a mutable ref object
- Accepts an initial value as its argument
- Can be used to access and manipulate DOM elements directly (”control uncontrolled components”)
- Can be used with **useEffect()** to recreate **componentDidUpdate()** lifecycle method

```
import { useRef } from 'react'

function App() {
  const inputRef = useRef()

  const handleSubmit = () => {
    console.log(inputRef.current.value)

    inputRef.current.value = 'changed’
  }

  return (
    <div>
      <input ref={inputRef} />
      <button onClick={handleSubmit}>submit</button>
    </div>
  )
}

export default App
```

React Hooks
useMemo

useMemo()

- Accepts a function and a dependency array as arguments, returns a “memoized” value
- Used to avoid re-running expensive operations when unnecessary

```
import { useState, useMemo } from 'react'

function Counter() {
  const [countOne, setCountOne] = useState(0)
  const [countTwo, setCountTwo] = useState(0)

  const isEven = useMemo(() => {
    let i = 0

    // arbitrary slow-down
    while (i < 2000000000) i++

    return countOne % 2 === 0
  }, [countOne]) // runs again when this value changes

  return (
    <div>
      <div>
        <button onClick={() => setCountOne(countOne + 1)}>
          Count One - {countOne}
        </button>
        <span>{isEven ? 'Even' : 'Odd'}</span>
      </div>
      <div>
        <button onClick={() => setCountTwo(countTwo + 1)}>
          Count Two - {countTwo}
        </button>
      </div>
    </div>
  )
}
```

React Hooks
useContext

useContext()

- Accepts a ‘context object’, returns value stored in that context
- Used to store and access values without passing them through down the component tree
- Can use multiple contexts as needed

```
//Creating and Implementing UserContext
import { createContext } from 'react’
import NestedComponent from './NestedComponent’

const UserContext = createContext(null)

function App() {
  return (
    <UserContext.Provider
      value={{
        username: 'timk’,
        firstName: 'tim’,
        lastName: 'kellogg’
     }}>
      <NestedComponent />
    </UserContext.Provider>
  )
}
export default App

//Nested Component
import { useContext } from 'react'
import UserContext from './userContext'

function NestedComponent() {
  const user = useContext(UserContext)
  return (
    <div>{JSON.stringify(user)}</div>
  )
}

export default NestedComponent
```

Let’s try it!

DevelopIntelligence
A PLURALSIGHT COMPANY