22. Data Fetching & Other Side Effects

Github Repository | Vimeo Video Recording

Topics to cover

- [x] 1. Pure vs Impure Functions
- [x] 2. Side-Effects
- [x] 3. React Hooks
- [x] 4. The `useEffect` Hook

1. Pure Functions

It is a function that abides by these two rules:

1. Identical return values if given identical arguments
2. No side-effects

This would be considered pure:

```
// Given the same arguments, you will always receive the same result.
function sum(x, y) {
    return Number(x) + Number(y);
}
```

```
function greeting(name) {
    return `Hello, ${name}!`;
}
```

2. Side-Effects

If the function mutates any variable or value outside of its own blocked scope, it would be said to have a side-effect. This includes the following cases:

- Performing standard input-output operations
- Updating the value of a variable defined outside of itself
- Mutating an argument passed by-reference
- Envoking a separate function that, itself, has side-effects

These would be considered impure with side effects

```
let myNum = 1;

function addOneToNum() {
    myNum++; // Edits a globally-scoped variable.. impure!
    return myNum;
}
```

```
function sum(x, y) {
    console.log(Number(x) + Number(y)); // I/O side-effect... impure!
}
```

Let's look at another example, now with React components:

```
// This is a PURE COMPONENT/FUNCTION
function MovieCard (props) {
  return (
    <>
      <h1>{props.title}</h1>
      <h2>{props.description}</h2>
      <h3>{props.year}</h3>
      <h4>{props.genre}</h4>
      <h4>{props.runtime}</h4>
    </>
  )
}

// This is an IMPURE COMPONENT/FUNCTION
function MovieCard (props) {
  // useState makes this component impure
  const [title, setTitle] = useState('The Titanic');

  return (
    <>
      <h1>{title}</h1>
      <h2>{props.description}</h2>
      <h3>{props.year}</h3>
      <h4>{props.genre}</h4>
      <h4>{props.runtime}</h4>
    </>
  )
}
```

3. React Hooks

In React, Hooks are functions that allow us to tap into select React features like **state**. We have already made use of one: `useState`! If you're curious about the complete list of Hooks available to use, check the official Hooks API Reference.

React Hook Rules

There are rules we should follow when using React Hooks. There are two official ones:

1. Only call Hooks at the top level
2. Only call Hooks in React functions

Let's break these down.

Rule 1. Only call Hooks at the top level

Never call Hooks within nested functions, in conditions, or inside of loops. The intention is for them to be placed before the return in a React function. This rule must be upheld to ensure state, or other Hook data, is maintained between renders.

Rule 2. Only call Hooks in React functions

You may only call upon Hooks in your React function components, or in building your own Custom Hooks(though that is a topic for another day.) The existing Hooks, as well as any you design, will begin with `use` as a prefix in its name; consider our use of React's `useState` Hook so far—and our focus today: `useEffect`.

4. The `useEffect` Hook

Side-Effects in React

We'd discussed how side-effects are defined in general when speaking about programming, but how do we see this play out in our React applications? A few of the most common side-effects you may encounter in React include:

- Timers
- Setting up a subscription
- Direct updates to the DOM
- Fetching data from an external resource
- Console logging

```
const MyComponent = (props) => {
  const [user, setUser] = useState({});

  useEffect(() => {
    // retrieve user information from an API and update local state with
the response
    fetch(`/users/${props.userId}`)
      .then(res => res.json())
      .then(json => setUser(json));
  });

  return (
    <div className="my-component">
      <p>You are logged in as { user.username }</p>
    </div>
  );
};
```

Dependencies

The second argument to `useEffect` is a dependency array that lets you specify when you want the hook to run. The hook will run again anytime the value of a dependency changes - **NOTE:** It is possible to get stuck in an infinite loop if the *effect* hook updates a value in the dependency array

```
// will run every time the value of user.firstName changes
useEffect(() => {
  document.title = `${user.firstName}'s Home Page`;
}, [user.firstName]);

// infinite loop because it runs every time count gets updated
useEffect(() => {
  setCount(count + 1);
}, [count]);
```

Cleanup

Sometimes side effects need to be cleaned up (eg. socket connections terminated). To perform cleanup, return a function from your `useEffect`

```
const [timer, setTimer] = useState(0);

useEffect(() => {
  // set up an interval to increment a timer
  const myInterval = setInterval(() => {
    setTimer(timer => timer + 1);
  }, 1000);

  // declare a cleanup function
  const cleanup = () => {
    clearInterval(myInterval);
  };

  return cleanup;
}, []);
```

View on Compass | Leave Feedback