React Hooks II

INFO 253A: Frontend Web Architecture

Kay Ashaolu

Rules of Hooks

- Must use hooks in functional React components
- Must use hooks at the top level of your components, or in your custom hooks
    - This means don't use hooks inside conditionals either
    - This is because React relies on the order of hooks to determine functionality
    - If the order of hooks executed changes dynamically during execution, very hard to figure out bugs will appear

Custom Hooks

- React provides the ability to write your own hooks
- Custom Hooks provide another way to share stateful logic across components
- What is this stateful logic you speak of? Or what does that even mean?

Custom Hooks

- As your front end application becomes more complex, it becomes harder to manage all of the state variables as well as all of the logic that modifies those state variables
- What happens when you want to have some state that affects multiple components?
- React has provided a few ways of accomplishing this task, but typically involve creating more components that contain the shared state at a higher level.
- Hooks provide an alternative path that does not necessiate these "higher order components"

Setup Code

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

function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);
  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }
    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

Custom Hooks Example

- We have a component called FriendStatus that displays "Online" if the Friend was online and "Offline" if not
- Note the use of the useEffect hook to define what should happen when the component is created or is updated, and what should happen when it unsubscribes.

Custom Hooks Example

- However imagine if we had another component, a contact list, where we wanted to highlight a person's name if they were online.
- We would write out the logic, but it would require repeating a lot of the same code

Setup code 2

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

function FriendListItem(props) {
  const [isOnline, setIsOnline] = useState(null);
  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }
    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  return (
    <li style={{ color: isOnline ? 'green' : 'black' }}>
      {props.friend.name}
    </li>
  );
}
```

Refactoring?

- In previous classes I've shown the benefits of putting common code into functions and using that function instead
- Benefits include
  - not repeating code which can be error prone
  - once you update behavior for the shared function it is available everywhere
  - code is easier to read

Let's look at a custom hook

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

function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChar
    };
  });
```

What's happening here?

- The amazing thing is that a custom hook is also just another function: inputs, outputs, and logic
- You can use other hooks inside custom hooks
- You are writing stateful logic that can be shared across components
- You can also fully control the inputs and outputs of your hook. In this case, we pass in a friend ID and return whether that friend is online

How to use custom hook

```
function FriendStatus(props) {
  const isOnline = useFriendStatus(props.friend.id);

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

```
function FriendListItem(props) {
  const isOnline = useFriendStatus(props.friend.id);

  return (
    <li style={{ color: isOnline ? 'green' : 'black' }}>
      {props.friend.name}
    </li>
  );
```

That was elegant

- Wasn't it? Those two components are now using the useFriendStatus custom hook, and thus reduced a lot of repeated code.
- If anything were to change with the API or the handling of the API results, we could simply update the useFriendStatus hook and all components using it would be updated

Demo