Events and Listeners

Events are typically handled using React's ***useEffect*** hook.

useEffect

React has a tutorial as well as documentation that can be found here.

Use effect allows you to run a function every time a state or context property changes.

The following example will run a function every time the ***user*** object is updated,

```
useEffect(() => {
    if (!!user) {
        // do something here
    }
}, [user]);
```

Lifting Up State

In some cases, you may not even need to use ***useEffect***.

You can lift up state as described in this article.

When a parent component is controlling state, and passing in handler functions as props to children components, may eliminate the need for an event altogether.

Window Event Listeners

We use ***window.addEventListener*** is several places. For example, listening for posted messages from the Utopia iframe or listening for resize events to know when we the screen size has reached a new bootstrap breakpoint. Functions that run as callbacks for these event listeners cache state variables, so you can do a couple things to account for this.

1. Re-setup the event listener every time a state variable changes

```
useEffect(() => {
    window.addEventListener('resize', onScreenResized);
    return () => {
        return window.removeEventListener('resize', onScreenResized);
    };
}, [screenSize]);
```

2. Use the functional version of the setter to get the current value of the state variable

```
setScreenSize(currentScreenSize => {
  if (currentScreenSize != newScreenSize) {
    return newScreenSize;
  }
  else {
    return currentScreenSize;
  }
});
```

> The top answer to this stack overflow question describes some the different ways you can account for this.

Revision #6
Created 1 November 2022 20:13:57 by Quinn Godfrey
Updated 24 June 2024 05:31:56 by Quinn Godfrey