Styled Components

One of the biggest things you can do to make your components EVEN MORE reusable is to encapsulate styling with them. Something important to note is that we're not talking about just bundling a CSS sheet with them. We're talking about a component that has all of the styling (which can also be customized with themes or other properties) baked into it.

There are a lot of different ways to handle this from pre-made packages to utilizing some form of customizable library, but the concepts are the same. Let's take a look at using components WITHOUT the `styled components` pattern and then WITH.

Standard Components

Without using styled components, the design pattern is as basically what we've been doing thus far. Let's say we want to have a button that we want to have the same styling for across multiple components. Well we'd have a couple options:

1. Use the native `<button></button>` element
2. Add classes as needed wherever you use it
3. Add a css sheet either for that specific component that uses it (and then that same styling in ALL other components that use that button styling) OR add it to your index.css.

The above means that as you add more classes, your `index.css` would become bloated with a TON of different classes. What happens if you want 5 different color options for that button, or different sizes or something else. And then what happens if you want them all to have an onClick event etc. It can cause a lot of issues.

If we use Atomic design principles, it can clean it up a little bit but making a reusable component but you're passing in a TON of props for different classes to add, etc. That would look like:

1. Create a custom `<Button></Button>` component that takes in a lot of props.
2. Pass in all the event handlers, styling, classes, etc.
3. Use that wherever you need

This still doesn't fix the issue of where to put the CSS. Sure you could put it into a css file for the element itself but then EVERY atom sized component has it's own css file which can be a headache.

Styled Components

We'll look at what all this entails here in a moment but the general pattern is as follows:

1. Create a custom `<Button></Button>` component that gets passed its onClick and other information
2. That component is ACTUALLY a styled component which is a native button with styling built in.
3. With everything built in and created with JavaScript, you can share a theme across an entire application, pass a single prop that can change a ton of the styling of the element and a lot of other things.

Think of styled components as components that you could pass a ton of props to for in line styling but all of it can be handled with technically just a few props because all the needed styles would be calculated by the

element itself. In addition the props can be a LOT more semantic so instead of passing something like: `bgColor="red"` if it's a primary button (primary being whatever you call it for it being more important) or `bgColor="blue"` if its a secondary button you could just pass `primary` and let the component figure out the rest.

This probably sounds a little weird but let's take a look at it in action with the `@emotion/styled` package