Manageable Styling for Reusable Components

Let's style our expandable component in this lesson!

WE'LL COVER THE FOLLOWING

Love it or hate it, styling (or CSS) is integral to how the web works.

While there's a number of ways to style a `React` component - and I'm sure you have a favorite - when you build reusable components, it's always a good idea to expose a frictionless API for overriding default styles.

Usually, I recommend having your components be styled via both `style` and `className` props.

For example:

```
// this should work.
<MyComponent style={{name: "value"}} />
// and this.
<MyComponent className="my-class-name-with-dope-styles" />
```

Now, our goal isn't just styling the component, but to make it as reusable as possible. This means letting the consumer style the component whichever way they want, whether that be using inline styles via the `style` prop, or by passing some `className` prop.

Styling the `Header` Component

Let’s begin with the `Header` child component; have a look at `Header.js`.

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

First, let’s change the rendered markup to a `button`. It’s a more accessible alternative to the `div` used earlier.

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

Adding CSS

We will now write some default styles for the `Header` component in a `Header.css` file.

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

I’m sure you can figure out the simple CSS above. If not, don’t stress it. What’s important is to note the default `className` used here, `.Expandable-trigger`.

To apply these styles, we need to import the `CSS` file and apply the appropriate `className` prop to the rendered `button`.

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

Allowing Changes in Default CSS

This works great, however, the `className` is set to the default string `Expandable-trigger`.

This will apply the styling we've written in the `css` file, but it doesn't take into account any `className` prop passed in by the user.

It's important to accommodate passing this `className` prop as a user might like to change the default style you've set in your `css`.

Here's one way to do this:

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

Now, whatever `className` is passed to the `Header` component will be combined with the `default` `Expandable-trigger` before being passed on to the rendered `button` element.

Let's consider how good the current solution is.

Passing a Default `className`

First, if the `className` prop is `null` or `undefined`, the `combinedClassName` variable will hold the value `"Expandable-trigger null"` or `"Expandable-trigger undefined".`

To prevent this, be sure to pass a default `className` by using the ES6 default parameters syntax as shown below in `Header.js`:

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

Having provided a default value, if the user still doesn't enter a `className`, the `combinedClassName` value will be equal to `"Expandable-trigger "`. Note the empty string appended to the `Expandable-trigger`. This works because of the way template literals work.

My preferred solution would be to do this:

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

The JavaScript `join()` method concatenates an array of strings into one single array separated by a 'separator' passed as an argument to `join()`. So on **line 8**, we concatenate the string `Expandable-trigger` with the passed `className` not separated by anything.

Removing False Values

This solution handles the previously discussed edge cases. If you also want to be explicit about removing `null`, `undefined` or any other false values, you can do the following:

```
const combinedClassName = ['Expandable-trigger',className].filter(Boolean).join('')
```

I'll stick with the simpler alternative, and provide a default for `className` via default parameters.

Having said that, here's the final implementation for `Header`.

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

So far, so good.

In case you were wondering, `combinedClassName` returns a string. Since strings are compared by value, there's no need to memoize this value with `useMemo`.

Quick Quiz!

1 Why do we want the component to be stylable via both `style` and `className` props?

COMPLETED 0%

1 of 7

[illegible]

So far, we've covered the `className` prop. What about the option to override default styles by passing a `style` prop? Let's write code to allow that in the next lesson!